Actas de las
XXIV Jornadas de Paralelismo

Editores
Guillermo Botella y Alberto A. Del Barrio

Organiza
Sociedad de ARquitectura y TEcnología de COmputadores (SARTECO)
Universidad Complutense de Madrid

Madrid (Madrid), 17-20 Septiembre 2013

Actas de las XXIV Jornadas de Paralelismo

Editores: Guillermo Botella y Alberto A. Del Barrio

ISBN: 978-84-695-8330-2

Servicio de Publicaciones. Universidad Complutense de Madrid, Madrid, 2013

Edición: 1^a
Impresión: 1^a
N^o **de páginas:** 435
Formato: 21 x 29.7
Materia CDU: 004 Ciencia y Tecnología de los ordenadores. Informática.

Actas de las XXIV Jornadas de Paralelismo

1^a Edición, 1^a Impresión

ISBN: 978-84-695-8330-2

Publicado por:
Universidad Complutense de Madrid
http://www.jornadassarteco.org/

Créditos:
Diseño de portada: los editores.
Maquetación: los editores (utilizando el paquete LaTeX 'confproc', versión 0.7 de V. Verfaille).

Impreso en Madrid (España) por Limencop, S.L. — Septiembre 2013

Comités

Comité Director

Francisco Tirado Fernández (SARTECO)(UCM)
Ramón Doallo Biempica (SARTECO)(UDC)
Katzalin Olcoz Herrero (JP2013)(UCM)
José Ignacio Martínez Torre (JCRA2013)(URJC)
Jesús González Peñalver (JCE2013)(UGR)
Domingo Giménez Cánovas (CPP2013)(UM)

Comité Organizador

José Luis Ayala Rodrigo (UCM)
Alberto Del Barrio García (Editor)(UCM)
Guillermo Botella Juan (Editor)(UCM)
Carlos García Sánchez (UCM)
José Ignacio Gómez Pérez (UCM)
José M. Moya Fernández (UPM)
Juan Carlos Sáez Alcaide (UCM)
Katzalin Olcoz Herrero (Coordinadora)(UCM)

Entidades

Entidades Patrocinadoras

NVIDIA
Texas Instruments

Entidades Organizadoras

Departamento de Arquitectura de Computadores y Automática (DACYA)
Universidad Complutense de Madrid (UCM)
Sociedad de Arquitectura y Tecnología de Computadores (SARTECO)

Presentación

Es un honor para nosotros daros la bienvenida a las **XXIV Jornadas de Paralelismo**, que tienen lugar en Madrid entre los días **17 y 20 de septiembre de 2013**, en el marco del **Congreso Español De Informática (CEDI)**. La celebración de este gran evento favorece particularmente el encuentro de numerosos investigadores procedentes de universidades y centros de investigación de España y América Latina, para **intercambiar** experiencias, presentar y **debatir** resultados de investigación, **fomentar** la coordinación entre grupos, y poner en común ideas sobre tendencias relacionadas con los distintos campos del conocimiento dentro de la Informática.

En esta edición se han aceptado un total de **72 artículos**, estando prevista la participación de aproximadamente **100 investigadores** en las diferentes actividades propuestas. Finalmente el programa de ponencias ha quedado estructurado en 18 sesiones que se reparten entre los siguientes temas: Arquitecturas del Procesador, Multiprocesadores y Chips Multinúcleo, Arquitecturas del Subsistema de Memoria y Almacenamiento Secundario, Redes y Comunicaciones, Tecnología Grid, Clúster y Plataformas Distribuidas, Algoritmos y Técnicas de Programación paralelas, Eficiencia Energética, Evaluación de Prestaciones y Aplicaciones de Computación de Altas Prestaciones.

Contamos en esta edición con **cuatro conferencias plenarias** de temas de gran actualidad y que esperamos sean de su interés. Por una parte, dentro del programa del CEDI intervendrán el **Dr. Markus Gross** del departamento de Computer Science del ETH (Zurich) y el **Prof. Giovanni De Micheli**, director del Institute of Electrical Engineering y del Integrated Systems Centre en la EPFL (Lausana).

Por otra parte, específicamente dentro del programa de las **XXIV Jornadas de Paralelismo**, el **Dr. Arnon Friedmann**, de Texas Instruments, nos hablará del nuevo papel de los DSPs como infraestructura de computación para las aplicaciones de alto rendimiento. Adicionalmente, el **Prof. Manuel Ujaldón**, del Departamento de Arquitectura de Computadores de la Universidad de Málaga, nos mostrará las perspectivas de uso de las GPUs para computación de altas prestaciones.

El programa de las Jornadas incluye también **dos mesas redondas** con ponentes de gran relevancia. En la primera mesa redonda se debatirá sobre la **Informática en la Educación Secundaria** y en la segunda sobre los **doctorados en el ámbito de Ingeniería Informática**. Además, tendremos la oportunidad de asistir a las **sesiones plenarias** que se han organizado dentro del CEDI y a la asamblea de la Sociedad de Arquitectura y Tecnología de Computadores (**SARTECO**).

No queremos acabar esta presentación sin mostrar nuestro **agradecimiento** a todos los organismos y entidades, públicos y privados, que han colaborado en el desarrollo de estas Jornadas, en particular a **Texas Instruments** y a **NVIDIA**, que han patrocinado nuestras conferencias invitadas.

Finalmente, queremos agradecer vuestro interés y participación en estas **XXIV Jornadas de Paralelismo**, que esperamos sean un éxito y cumplan con vuestras expectativas. Una vez más, bienvenidos a Madrid.

Comité Organizador de las XXIV Jornadas de Paralelismo
Madrid (Madrid), 17-20 Septiembre 2013

Índice de las Actas JP2013

Arquitecturas del Procesador, Multiprocesadores y Chips Multinúcleo

Dynamic Cache Blocks Mapping to Reduce Last-Level Cache Access Latency

Mario Lodde[1] Jose Flich[2] Manuel E. Acacio[3]

Abstract— **In tiled Chip Multiprocessors (CMPs) the banks of the built-in last level cache (LLC) are usually distributed among the tiles and logically shared. A static mapping of cache blocks to the LLC banks leads to poor efficiency since a block can be mapped to a bank far away from the tiles which actually access it. Partially dynamic policies have been proposed, which however rely on the static mapping of blocks to a set of banks (D-NUCA) or rely on the OS to dynamically load pages to statically mapped addresses (first-touch).**

We propose a new dynamic approach where the LLC home bank is determined at runtime in hardware, with the memory controller in charge to perform the block mapping when fetched from main memory. To speed up the home bank lookup process, we use simple and lightweight NoC optimizations. When compared to alternative solutions (S-NUCA, D-NUCA, first touch, private LLCs) results with PARSEC and SPLASH-2 applications indicate improvement in locality of LLC blocks in the same tile (56.2% from 5.8%) and more than 33% reduction in load and store miss latencies. This leads to an average reduction of 24% in application's execution time compared to static mapping.

Keywords— **CMPs, cache hierarchy, NoC**

I. Introduction

Chip multiprocessor systems (CMPs) usually employ a shared memory programming model, thus requiring a cache coherence protocol to keep data consistency along the cache hierarchy. The on-chip cache is organized hierarchically, with small low-latency caches at the highest level and larger caches with higher access times at the lower levels. This provides high on-chip storage capacity without the high access latency a single, large cache would have. Without losing generality, in this work we assume the tiled CMP system shown in Figure 1 with a two-level cache. Each tile includes a core, separate L1 caches for instructions and data, a bank of L2 cache and a switch to connect the tiles through a 2D mesh.

L1 caches are private to the core in the tile. For the L2 cache, different policies can be implemented, but the common choices are two. The first one is to use each L2 cache bank as a private cache to the tile, extending its private cache capacity. This is the best option if the working set of the application fits in the L2 cache bank, since all cached data can be accessed without sending requests over the NoC. If the working set does not fit, this policy generates many L2 cache line replacements, and therefore, off-chip requests. Furthermore, shared blocks are replicated in different L2 cache banks. The second option is to consider the L2 banks as a shared but distributed L2 cache. Data replication is avoided and cache resources are used efficiently, but the latency of retrieving a cached data in case of L1 miss will be higher and variable depending on the location of the L1 cache and the L2 bank. Thus, the mapping of blocks to L2 banks is a crucial design parameter for this approach. In this paper we follow this policy.

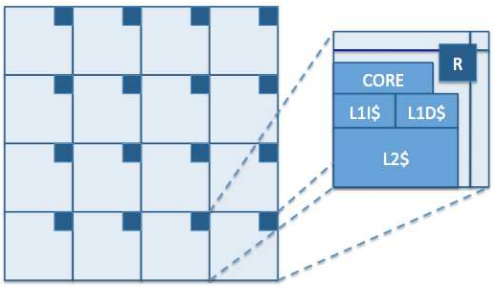


Fig. 1. Tiled CMP system.

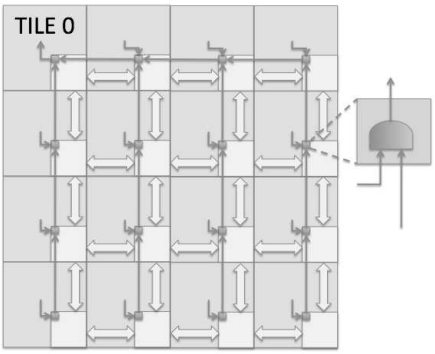


Fig. 2. Gather control network for tile 0.

The L2 bank that hosts a block is called the *home* bank. There are two main design options when deciding which L2 bank is the *home* for a block. On the one hand, block mapping can be done statically (S-NUCA): the address space is divided in subsets and all the blocks of a subset are statically mapped to a bank. This policy is very simple to implement but can be inefficient as blocks may be mapped to banks which are far away from L1 requestors. The second option is to perform the mapping dynamically (D-NUCA) [1], where each subset of blocks is mapped to a group of banks, or bank set, and blocks can migrate within a bank set to move as close as possible to the requestor's tile. This policy has lower miss latencies but is more complex to implement. Furthermore, the process of finding a block within a bank set leads to a tradeoff between access time and NoC traffic since all the banks of a bank set must be accessed, leading to either high latency (sequential search) or more traffic (parallel search).

[1]Grupo de Arquitecturas Paralelas, Universitat Politècnica de València , e-mail: mlodde@gap.upv.es
[2]Grupo de Arquitecturas Paralelas, Universitat Politècnica de València , e-mail: jflich@disca.upv.es
[3]Universidad de Murcia, e-mail: meacacio@ditec.um.es

In this work we propose Runtime Home Mapping (RHM), a new dynamic approach where the LLC home bank is determined at runtime in hardware by the memory controller. While in D-NUCA the mapping is partially static, in RHM a block can be mapped to any L2 cache bank, enabling future optimization strategies such as virtualization, where the *home* banks can be bounded to the region of the chip running a given application, and increased efficiency of thread migration, where migrated threads can attract data to their closest L2 cache banks. Since the *home* bank is not known a priori, a search must be performed each time an L1 miss occurs. Conversely to previous approaches to home location, based on the use of limited-size tables, and therefore prone to costly overflows [2], we employ NoC-level mechanisms to optimize the search phase. In addition, we propose a migration strategy similar to the one of D-NUCA caches but without the bank set constraint: a block can migrate to any bank. Results show RHM is effective in placing blocks near the core using them, reducing the average number of hops per request by 60% on average compared to static mapping, which leads to reductions of more than 30% in terms of cache miss latency and 24% in execution time.

The rest of the paper is organized as follows: in Section II we show RHM and its NoC-level support. In Section III we show the evaluation results. In Section IV we describe the related work. In Section V we describe future work and conclusions.

II. Runtime Home Mapping with NoC-level support

RHM aims to map blocks to L2 banks at runtime, in order to allocate them as close as possible to the requesting cores, possibly in the L2 bank of the local tile. The mapping is performed by the memory controller each time it receives a request. Thus, if a block is removed from the chip due to an L2 cache replacement, it can be mapped to a different L2 bank the next time depending on the requestor and L2 cache availability.

In case of an L1 miss, a request is sent to the local L2 bank in the same tile. On a miss in the local bank, a broadcast is sent to all other L2 banks. When a bank receives this broadcast request, it checks its tag array. In case of a hit, it sends the data back to the requestor. In case of a miss, an acknowledgement (ACK) is sent to the L2 which issued the broadcast. If all the L2 banks send an ACK to that L2 bank, it means the block is not cached on chip, so the bank sends a request to the memory controller (MC), which in turn fetches the block from main memory. The MC keeps track of the utilization of each L2 bank, so while waiting for the data it decides which L2 bank will be its *home* depending on the requestor location, utilization statistics and the mapping policy (explained in Section II-B).

Once a bank is chosen as the *home* for a particular block, the MC notifies the bank so it can start replacing a cache line, if needed, and allocating a line to the incoming block while the MC is still waiting for the block. When the block is received at the MC, it is sent to the chosen *home* bank, which in turn will send the block to the requestor L1 cache.

The L2 home search policy just described above has high network resources demand: every time a request misses in the local L2 bank, a broadcast is issued. Also, all other banks must answer to the broadcast with an ACK or with the data. The first problem can be attenuated by implementing a tree-based broadcast mechanism within the NoC: a broadcast is sent injecting a single message, that replicates at switches to reach every L2 bank. This reduces NoC traffic and eliminates the serialization of multiple copies of the same request (one per destination). ACKs however still represent a big problem: they are indeed sent roughly at the same time and will probably serialize in the network and, most important, all of them must reach the same L2 cache bank (i.e., the one that initiated the broadcast).

A. Gather Control network

To solve the problems introduced by ACK messages our proposal uses a simple and fast dedicated control network. [1] This network, called Gather Control Network (GCN), can be logically seen as 16 one-bit wide subnetworks, one per tile. Each subnetwork is a tree of AND gates, connecting the destination tile (the root) with all other tiles (located at the leaves of the AND tree).

Figure 2 shows the subnetwork with root in tile 0. A one-bit subnetwork (darker arrows) is added to the regular NoC (bidirectional arrows). If a request misses in the L1 and L2 caches of tile 0 (L1-0 and L2-0 from now on), L2-0 broadcasts the request to all other L2 banks through the regular NoC. When an L2 bank receives this request, it triggers the output signal of the GCN for tile 0. Once all L2 banks have received the broadcast and triggered their output signals, the output of the AND tree will notify the L2-0 and thus acts as a global ACK. Sending ACKs through the GCN has three major advantages: (1) NoC traffic is highly reduced; (2) power consumption is highly reduced; (3) delay in transmitting ACKs is hugely reduced. Indeed, the GCN does not require routing, flow control, nor arbitration at each hop and eliminates message serialization at destination.

The GCN we assume in this work has the same logic behavior of the AND trees but is implemented with sequential logic, which reduces wiring requirements. An extensive discussion on the sequential GCN is published in [5].

B. Mapping Algorithm

Each time the MC receives a request, a mapping algorithm chooses the *home* bank for the requested block. Particularly, the *home* is chosen depending on the requestor's tile and current L2 banks utilization.

[1] This control network is similar to the one proposed in previous works [3],[4] to collect ACKs generated by Hammer and Directory coherence protocols.

The MC collects statistics about cache utilization, which are stored in a table (*alloc* table) with $N \times M$ entries, where N is the number of L2 cache banks and M the number of L2 sets. Each entry contains the number of allocations performed in set m of L2 bank n. If the associativity of L2 sets is Z, the table has to store at minimum $N \times M \times \log_2 Z$ bits. However, the table will double the bits in each entry. For a 4×4 tile system with 16-way 256KB bank sets, the minimum memory requirements for this single table is 2KB ($m = 16$, $M = 256$, $Z = 16$). With the increased size, the table will grow to 4KB (to allow 256 allocations per set). The following pseudocode describes the simple algorithm we implemented:

```
int function allocate(int r, address a) {
 banklist n; bank b; set s; bank h;

 s = get_set(a);
 if (alloc[r,s]<num_ways)
   {alloc[r, s]++; return r;}

 for(int h = 1; h <= MaxHops; h++){
   n = BanksReachable(r, h);
   for (int i = 0; i < size(n); i++){
     b = SelectBankClockWise(n, i);
     if (alloc[b,s]<num_ways)
       {alloc[b,s]++; return b;}
   }
 }

 for(int h = 1; h <= MaxHops; h++){
   n = BanksReachable(r, h);
   for (int i = 0; i < size(n); i++){
     b = SelectBankClockWise(n, i);
     if (alloc[r,s] - alloc[b,s] > UtilThr)
       {alloc[b,s]++; return b;}
   }
 }

 alloc[r, s]++; return r;
}
```

If there is room in the set in the local L2 bank (r tile), then the *home* is the local tile of the requestor. Otherwise, the algorithm scans the neighbor banks in distance order (first *for* loop). This search is performed until the threshold $MaxHops$ is reached, which can be equal to the physical threshold forced by the system size (number of hops from the requestor to the furthest tile) or lower.

If all the L2 banks are full (*alloc* higher than *num_ways*), the algorithm tries to balance the number of allocations (thus, replacements) in all banks (second *for* loop). A threshold (*UtilThr*) is used. If the difference between the number of allocations in the local tile's bank and a neighbor bank is higher than the threshold, then the neighbor bank is selected as the *home* bank.

If all the banks are balanced, then the block is mapped to the requestor's tile. Notice that this does not imply that RHM defaults to private L2 caches.

TABLE I
NETWORK PARAMETERS.

Routing	XY
Flow control	credits
Flit size	8 byte
Switch model	4-stage pipelined
Switching	virtual cut-through
Buffer size:	9 flit deep
Virtual channels:	4
GCN delay	2 cycles

With private caches all the data accessed by a core must be present in the L2 bank of the same tile, while in RHM this does not apply. For instance, a shared block will be replicated in all L2 caches if they are private, while in RHM it will be present only in the *home* tile. The proposed policy defaults to private caches only if all L2 banks are full, each core is requesting private blocks and all banks are uniformly used. In this case, very unlikely in a parallel application, all blocks are allocated in the requestor tile, which indeed is the best choice since it minimizes the data access latency.

C. Block Migration

If the initial *home* allocation performed by the MC results sub-optimal, block migration can be enabled to further reduce the number of hops between an L1 cache and the L2 bank. Notice that a sort of migration mechanism is implicit in RHM, since each time a block is replaced from an L2 bank and then requested again it may be mapped to another L2 bank, but this may not always be effective.

We propose a migration scheme similar to the one used in D-NUCA but without the constraint of being limited within a bank set: in RHM a block is allowed to migrate to any L2 bank. However, since the migration process introduces an overhead in terms of traffic and energy, it should be performed only if it actually leads to a benefit in terms of miss latency reduction.

Solutions in D-NUCA reduce unnecessary migrations and avoid the ping-pong effect by using a saturating counter for each direction to which a block can move. A counter is updated each time a request comes from a node located in the counter's direction; when the counter saturates, the migration process towards that direction is triggered. In our case a block may migrate in any direction, so four counters are needed, one per direction. Each time a request is received from a tile, the counters are updated adding the distance in hops from the requestor. When a counter is incremented, the one in the opposite direction is decremented. When a counter saturates, it starts the migration process: the block migrates to the L2 bank located in the same tile of the L1 which sent the request that triggered the migration.

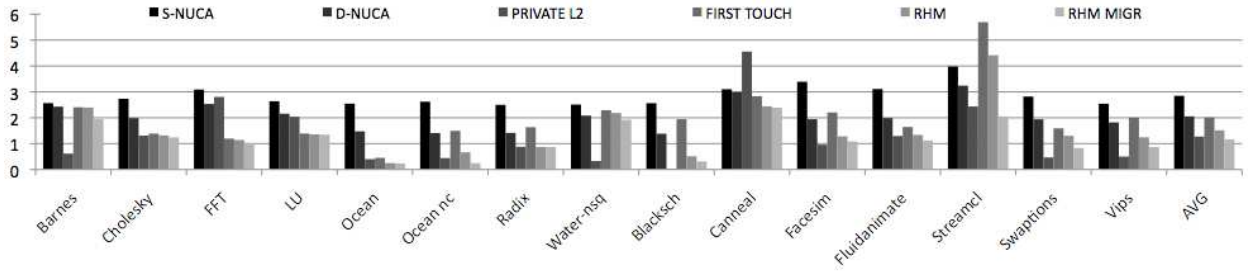


Fig. 3. Avg hop distance between L1 requestors and the tile where the data is found.

TABLE II
CACHE PARAMETERS.

Coherence protocol	Directory (MESI)
L1 cache size	16 + 16 kB (I + D)
L1 tag access latency	1 cycle
L1 data access latency	2 cycles
L2 bank size	256 kB
L2 tag access latency	1 cycle
L2 data access latency	4 cycles
Cache block size	64 B

III. EVALUATION

We evaluate RHM and compare it with other proposed NUCA configurations. In the baseline (S-NUCA) blocks are statically mapped to L2 banks using the less significant bits of the block address. In D-NUCA, blocks are statically mapped to a bank-set depending on their addresses. The matrix of L2 banks is divided in four bank-sets, one per column of tiles. Blocks are inserted in the L2 bank located in the same row of the requestor and then can migrate within the bank-set, one hop each time a migration is triggered. A third configuration uses private LLCs. Finally, we consider an S-NUCA configuration in which the blocks are mapped to the L2 banks using a first touch policy [6]. These configurations are compared to RHM, with and without block migration.

The cache coherence protocol for each configuration, the NoC with broadcast support and the GCN have been implemented and simulated using our flit-level cycle-accurate network and cache hierarchy simulator. We embedded it in Graphite[7], capturing the memory accesses of Graphite's simulated cores and using our tool for cache hierarchy and NoC timing. Different applications of the SPLASH-2 and PARSEC benchmark suites have been run on the 16-core system considered throughout this paper. Network and cache parameters are shown in Table I. Cache latencies have been obtained using Cacti [8]. One memory controller is placed at the top left corner of the chip. For the sake of fairness, we have used also the broadcast support and the GCN in the D-NUCA approach.

Figure 3 shows the average hop distance from the requestor to the *home* tile. For S-NUCA, the block is found on average at a distance of 2.85 hops. This distance is roughly the same for most applications as blocks are uniformly distributed among the L2 banks. With other configurations, however, since blocks are dynamically mapped and/or moved from a bank to another, the distance is quite variable depending on the application. For Barnes, dynamic techniques are not so effective, and the average value is always higher than 2 hops, while for other applications, e.g. Ocean, those techniques achieve a large reduction in the average number of hops. On average, RHM locates the data closer to the requestor than the other configurations, and this distance is further reduced if block migration is enabled. Indeed RHM MIGR achieves a locality close to that of PRIVATE L2

Figure 4 shows the percentage of requests which hit in the L2 bank located in the same tile of the requestor. Again, results when using S-NUCA do not depend on the application due to the uniform mapping of the blocks, and this percentage is quite low (6% on average). This percentage increased to 16% for D-NUCA, but is still much lower when compared to First Touch (33%), RHM (49%), RHM with block migration (56%) and Private L2 (72%). Thus, the most effective dynamic method is RHM.

Figures 5 shows the normalized execution time with the six different configurations (PTA is not enabled in this evaluation). We can observe how execution time is largely reduced with average factors ranging between 12% and 24% when using First Touch and RHM (with and without migration enabled). RHM achieves lower execution time due to its achieved higher locality in L2. Also, the migration policy helped in further reducing execution time. Contrary to this, D-NUCA is not able to achieve large reductions when compared to S-NUCA. The use of private caches achieves large execution time reductions but its effectiveness depends on the size of the working set of every application.

Figure 6 shows the average load and store miss latency, respectively, for the evaluated configurations, normalized to the S-NUCA approach. RHM reduces these latencies more than 35% on average and up to 80% (FFT store miss latency). Again, the effectiveness of RHM in reducing the miss latency depends on the memory access pattern of each application. Streamcluster shows a high percentage of blocks which are first accessed by a tile and then by different tiles during different phases of the ap-

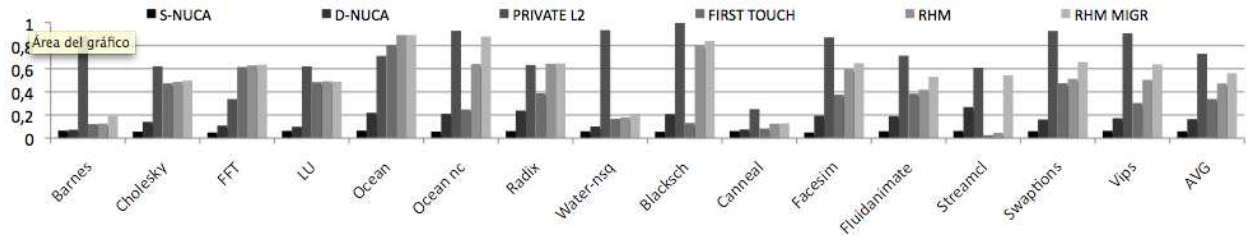


Fig. 4. Percentage of hits in the L2 bank located in the tile's requestor.

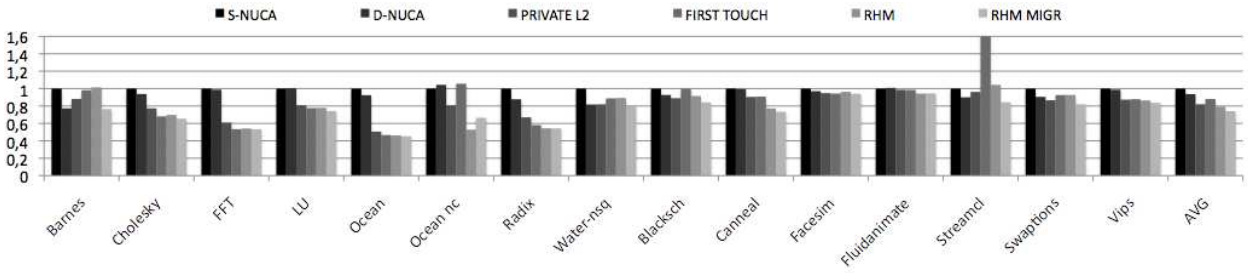


Fig. 5. Normalized execution time (L2 256KB L1 16KB).

plication. In this case, a first touch policy has the negative effect of overloading the tile where blocks are mapped, and the migration mechanism can effectively move the blocks to the correct tiles.

To conclude performance analysis, it should be observed that RHM performs better than First-Touch. Although FirstTouch is a simple mechanism not requiring any hardware assistance, it should be noted RHM allows finer-grained assignments (blocks vs pages) and also more effective thread migration as blocks can be effectively migrated along with threads.

IV. Related Work

To overcome the wire-delay problem [9], the LLC in CMP systems is usually banked, thus offering a wide spectrum of design choices: each bank indeed can expand the private cache of a core or be part of a globally shared LLC. The latter configuration is commonly called a Non-Uniform Cache Access architecture (NUCA), initially proposed by Kim *et al.* for a single core system [1] and then extended to many cores and CMPs [10], [11], and in turn offers many options when implementing the mapping of the blocks on each bank, the home bank search policy [12] [13] and the potential migration [1] or replication [14] of blocks. Both private and shared LLCs have their advantages and drawbacks, so hybrid configurations have been proposed to exploit the benefits of both design choices, such as ESP-NUCA [15] and CloudCache [16]. CMP-NuRAPID [17] decouples tags and data to allow data placement and replication in any LLC bank. Reactive-NUCA [14] also allows block replication. CMP-NuRAPID however requires an additional bus, while Reactive-NUCA is based on a 2D torus, so they can't be implemented in a 2D mesh-based system. OS-based techniques to achieve a better mapping of the cache blocks to the LLC banks have been proposed by Cho et al.[6], Ros et al. [18], Das *et al.* [19] to achieve dynamic mapping through OS-level page allocation. Cuesta et al. [20] deactivate the coherence protocol for blocks which are detected as private by the OS. Compile-time and data-based techniques have also been proposed in [21] and [22]. OS- and compiler-based techniques however rely on static mapping at hardware-level and can't support block migration or replication. Finally, Hammoud *et at.* [2] propose to implement blocks placement strategies at the memory controller(s) to prevent placing a block at an exceedingly pressured local set. To locate cache blocks at the LLC a CTCT [23] policy is assumed, which introduces 3-way communications in some cases, thus increasing the latency of L1 misses. RHM, differently from previous proposals, allows efficient block search between L2 banks in the whole chip. The optimizations/support at the NoC level allow for a aggressive data placement policy requiring only a small table at the memory controller, and avoiding the 3-way communication of some of the previous solutions, or the static assumption of private caches or OS-level solutions.

V. Conclusions and Future Work

In this work we have proposed Runtime Home Mapping (RHM) of the cache blocks to the LLC banks performed at the memory controller with NoC level support. Different improvements and designs at the NoC level enable fast and efficient location of data. The aim is to allocate L2 home blocks as close as possible from requestors. Results indicate a large span of improvement both in execution time and in reduced miss latencies. The current work can be extended in many directions, potentially leading to further improvements. Indeed, in this paper we applied baseline methods for different critical design choices of the method. As a first thing, we have plans to evaluate how the search phase behaves with differ-

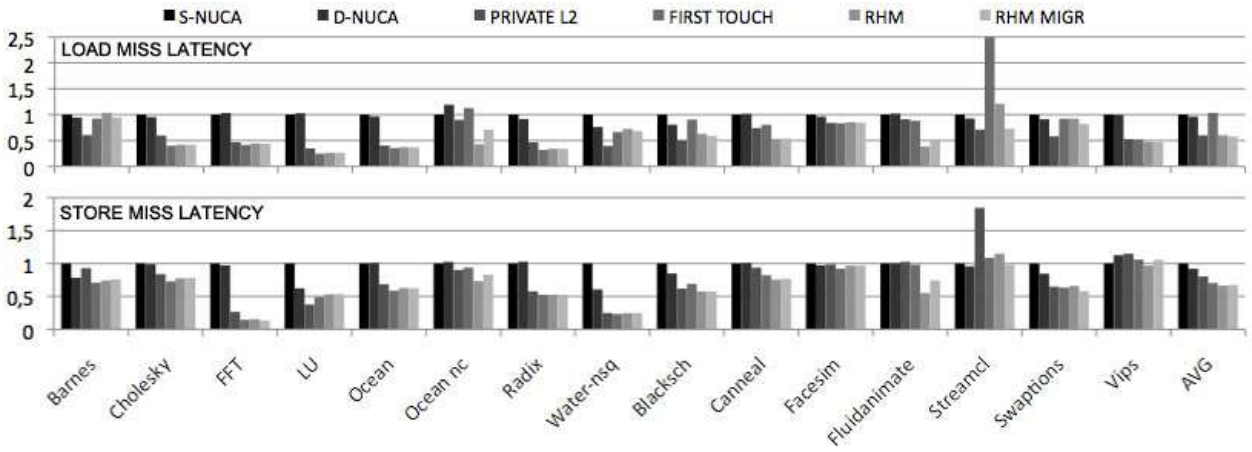


Fig. 6. Normalized load and store miss latency.

ent broadcast implementations on the search method of home nodes and different network topologies. Second direction, linked to the broadcast operation, is the definition of smart mapping strategies located in the memory controller. Finally, we would like to evaluate the performance of dynamic home mapping combined with virtualization where the memory controller is aware of the partition of chip resources to applications.

Acknowledgements

This work has been supported by the VIRTICAL project (grant agreement n 288574) which is funded by the European Commission within the Research Programme FP7.

References

[1] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Proc. of the 10th Intl Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[2] M. Hammoud, S. Cho, and R.G. Melhem, "A dynamic pressure-aware associative placement strategy for large scale chip multiprocessors," *IEEE Computer Architecture Letters*, vol. 9, no. 1, pp. 29–32, January-June 2010.

[3] M. Lodde, J. Flich, and M.E. Acacio, "Heterogeneous noc design for efficient broadcast-based coherence protocol support," in *Proc. of the 6th Intl, Symp on Networks on Chip*, 2012.

[4] M. Lodde, T. Roca, and J. Flich, "Heterogeneous network design for effective support of invalidation-based coherency protocols," in *Proc. of the 2012 Interconnection Network Architecture: On-Chip, Multi-Chip Workshop.*

[5] M. Lodde, T. Roca, and J. Flich, "Built-in fast gather control network for efficient support of coherence protocols," in *IET Computers and Digital Techniques INA-OCMC 2012 Special Issue.*

[6] Sangyeun Cho and Lei Jin, "Managing distributed, shared l2 caches through os-level page allocation," in *Proc. of the 39th IEEE/ACM Int'l Symp on Microarchitecture (MICRO-39)*, 2006.

[7] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald III, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *The 16th IEEE Intl. Symp. on High-Performance Computer Architecture*, 2010.

[8] "Cacti 5 technical report, available at http://www.hpl.hp.com/techreports/2008/hpl-2008-20.html," .

[9] D. Matzke, "Will physical scalability sabotage performance gains?," *Computer*, vol. 30, no. 9, pp. 37–39, 1997.

[10] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Bourger, and S.W. Keckler, "A nuca substrate for flexible cmp cache sharing," in *Proc. of the 19th Intl Conference on Supercomputing*, 2005.

[11] B.M. Beckmann and D.A. Wood, "Managing wire-delay in large chip-multiprocessors caches," in *Proc. of the 37th Intl Symp on Microarchitecture*, 2003.

[12] J. Lira, C. Molina, and A. Gonzales, "Hk-nuca: Boosting data searches in dynamic non-uniform cache architectures for chip multiprocessors," in *Proc. of the 2011 IEEE Intl Parallel and Distributed Processing Symp.*

[13] R. Ricci, S. Barrus, and R. Balasubramonian, "Leveraging bloom filters for smart search within nuca caches," in *Proc. of the 7th Workshop on Complexity-Effective Design*, 2006.

[14] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive nuca: near-optimal block placement and replication in distributed caches," in *Proc. of the 36th Intl Symp on Computer Architecture*, 2009.

[15] J. Merino, V. Puente, and J.A. Gregorio, "Esp-nuca: A low cost adaptive non-uniform cache architecture," in *Proc. of the Intl Conference on High Performance Computer Architectures*, 2010.

[16] H.Lee, S. Cho, and B.R. Childers, "Cloudcache: Expanding and shrinking private caches," in *Proc. of 44th Int'l Symp on High-Performance Computer Architecture*, 2011.

[17] Z. Christi, M.D. Powell, and T.N. Vijaykumar, "Optimizing replication, communication and capacity allocation in cmps," in *Proc. of Intl Symp on Computer Architecture*, 2005.

[18] A. Ros, M. Cintra, M.E. Acacio, and J.M. Garcia, "Evaluation of low-overhead organizations for the directory in future many-core cmps," in *Proc. of the Intl Conference on High Performance Computing*, December 2009.

[19] A. Das, M. Schuchhardt, N. Hardavellas, G. Memik, and A. Choudhary, "Dynamic directories: A mechanism for reducing on-chip interconnect power in multicores," in *Proc. of Design, Automation and Test in Europe*, 2012.

[20] B. Cuesta, A. Ros, M.E. Gomez, A. Robles, and J. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *Proc. of the 38th Intl Symp on Computer Architecture*, 2011.

[21] Y.Li, A.Abousamra, R.Melhem, and A.K. Jones, "Compiler-assisted data distribution for chip multiprocessors," in *Proc. of 19th Intl Conference on Parallel Architectures and Configuration Techniques*, 2010.

[22] Y.Zhang, W.Ding, M.Kandemir, J. Liu, and O. Jang, "A data layout optimization framework for nuca-based multicores," in *Proc. of 44th Intl Symp on Microarchitecture*, 2011.

[23] M. Hammoud, S. Cho, and R.G. Melhem, "Acm: An efficient approach for managing shared caches in chip multiprocessors," in *Proc. of the 4th High Performance Embedded Architectures and Compilers Int'l Conference (HiPEAC-09)*, 2009.

Interfaces de Red con afinidad para la comunicación eficiente en arquitecturas multinúcleo

Andrés Ortiz[1] Julio Ortega, [2] Antonio F. Díaz, [3] Alberto Prieto[4]

Resumen— **Las aplicaciones con gran demanda de ancho de banda de comunicación (algunas aplicaciones de tiempo real, multimedia, computación de altas prestaciones, etc.) y la disponibilidad de enlaces de comunicación con anchos de banda de decenas de gigabits por segundo hacen necesaria la mejora de las prestaciones de las interfaces de red para evitar que los procesadores deban dedicar un número considerable de ciclos a tareas de comunicación, en lugar de a la propia aplicación. Aunque las arquitecturas multinúcleo ofrecen nuevas oportunidades para aprovechar el paralelismo disponible en el diseño de arquitecturas de comunicación eficientes, y los núcleos de los sistemas operativos utilizan varias hebras que ejecutan tareas de red concurrentemente, las implementaciones paralelas del procesamiento de paquetes o conexiones no son triviales debido al coste de sincronización en el acceso a los recursos compartidos y a la eficiencia de uso de las caches. Recientemente se está explorando la posibilidad de asignar las interrupciones de red, junto con el procesamiento del protocolo de comunicación y la aplicación de red correspondientes al mismo núcleo (scheduling con afinidad), para así reducir la pugna por los recursos compartidos y los fallos de cache. En este artículo se proponen algunas alternativas para distribuir la interfaz de red entre los núcleos existentes en el servidor de acuerdo con la afinidad de las tareas de comunicación: su proximidad a las memorias que almacenan las distintas estructuras de datos utilizdas y las características de los nodos. Para evaluar las prestaciones de las interfaces de red propuestas se han utilizados cargas de trabajo de MPI, para las que se han obtenido mejoras de alrededor del 35% en ancho de banda y del 23% en latencia.**

Palabras clave— **Afinidad de interrupciones, Afinidad de procesador, Interfaces de red, SIMICS.**

I. Introducción

LOS servidores necesitan interfaces de red optimizadas, que permitan aprovechar los anchos de banda de varias decenas de gigabits por segundo usuales en los enlaces de red (multigigabit Ethernet, Infiniband, etc.) [1], sin comprometer la disponibilidad de los procesadores en la ejecución de las aplicaciones por su dedicación a tareas de comunicación. Así, el diseño de interfaces de red (NI, Network Interfaces) para reducir el sobrecosto (overhead) de comunicación debido a los cambios de contexto, copias de datos, e interrupciones ha constituido una línea de investigación relevante desde hace tiempo. La disponibilidad de varios procesadores en los computadores, en particular la generalización de las arquitecturas multinúleo, ha dado lugar a propuestas en las que alguno, o incluso algunos de los procesadores del servidor, ejecutan las tareas de comunicación. En esta tendencia se pueden incluir las técnicas de offloading y onloading [10], [2], [3], [20], y las estrategias de paralelización de la pila de protocolos de comunicación para el procesamiento simultáneo de paquetes de la misma o de distintas conexiones [19], [4], [21]. Si bien las arquitecturas multinúcleo refuerzan el interés en mejorar las prestaciones de la NI mediante la paralelización de los protocolos, existen dificultades importantes al paralelizar protocolos ampliamente utilizados, como TCP/IP. Estas dificultades están relacionadas con los costes de sincronización y acceso exclusivo a los recursos, y con la eficiencia de las caches [4], [7], [5]. Una aproximación para reducir estos costes es la asignación coordinada del procesamiento de los protocolos y las aplicaciones de red, y dirigir las interrupciones y flujos de datos de red, a los mismos núcleos (afinidad). Se pueden distinguir dos alternativas complementarias para conseguir interfaces de red con afinidad: las tecnicas NIC (Network Interface Card) [6], [8], [9], y las técnicas de software de sistema operativo [4], [7], [13]. Este trabajo se circunscribe dentro de éstas últimas y consiste en la distribución de las tareas de comunicación de la NI entre los n'ucleos del servidor de acuerdo con su afinidad (proximidad entre los nodos donde se está procesando la aplicación y la memoria donde se almacenan las estructuras de datos utilizadas). Se trata de evitar que haya núcleos muy cargados, donde se ejecuta no solo la aplicación de red sino también el procesamiento de la comunicación, y para eso se distribuyen las tareas, pero manteniendo la localidad de los accesos a memoria mediante una planificación que refuerce la afinidad. Aunque nuestra aproximación permite la mejora de prestaciones de las conexiones individuales, también podría aplicarse en el aprovechamiento del paralelismo de paquetes o de conexiones, de la misma forma que las técnicas RPS (Receive Packet Steering) [15] y RFS (Receive Flow Steering) [16], recientemente propuestas. A continuación, la Sección 2 presenta la interfaz de red con afinidad que proponemos, y la Sección 3 describe el trabajo experimental realizado y los resultados obtenidos para algunas cargas de trabajo de aplicaciones MPI y distintas alternativas de NI. Para terminar, la Sección 4 proporciona las conclusiones del artículo.

[1]Dpto. de Ingeniería de Comunicaciones. Univ. de Málaga, e-mail: aortiz@ic.uma.es
[2]Dpto. de Arquitectura y Tecnología de Computadores, Univ. de Granada, e-mail: jortega@ugr.es
[3]Dpto. de Arquitectura y Tecnología de Computadores, Univ. de Granada, e-mail: afdiaz@ugr.es
[4]Dpto. de Arquitectura y Tecnología de Computadores, Univ. de Granada, e-mail: aprieto@ugr.es

II. AbNI: una interfaz de red con afinidad

En esta sección se presenta la interfaz de red AbNI (Affinity-based Network Interface), que implementa técnicas relacionadas con el concepto de afinidad para, de esta forma, aprovechar las características de la arquitecturas multinúcleo. La interfaz AbNI puede aprovechar las ventajas de las previamente propuestas técnicas de onloading y, en algunas arquitecturas, las técnicas de offloading, evitando sus inconvenientes. Para mejorar las prestaciones de comunicación, la interfaz realiza una distribución adecuada de la carga de comunicación sin realizar ninguna modificación en la pila TCP/IP. De hecho, se ha usado la implementación de TCP/IP del núcleo 2.6 de Linux.

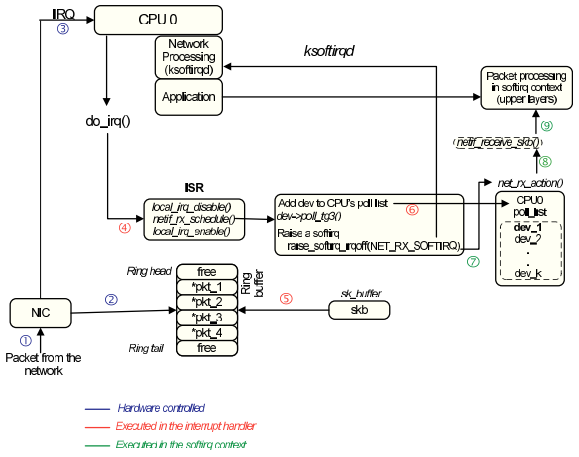


Fig. 1. Recepción de paquetes en la interfaz de red base utilizada como punto de partida: funciones del núcleo de Linux utilizadas en el proceso (Núcleo de Linux con driver NAPI.)

A. Recepción de paquetes

La Figura 1 describe las etapas principales de la recepción de paquetes en una interfaz de red en la que los protocolos de comunicación se ejecutan en el mismo núcleo de procesamiento que la aplicación y el SO, y en la que no se ha utilizado ninguna de las estrategias de optimización que se consideran en este artículo. Se considera que la tarjeta de red (NIC) soporta NAPI [3] y que el driver de dispositivo utiliza un método basado en sondeo para reducir el efecto de las interrupciones en el overhead asociado a la recepción de paquetes. Esta será la interfaz de red base que se utilizará en el artículo para evaluar las mejoras que aportan nuestras propuestas. En la Figura 1, cada vez que un paquete llega a la NIC ①, se copia mediante DMA desde la memoria de la NIC a un área de DMA denominada buffer en anillo (ring buffer) y situada en la memoria principal,②. Este buffer en anillo consta de un conjunto de descriptores de paquetes que mantiene el driver de la NIC, e incluye punteros a buffers y a ciertos bits de estado (Recepción de paquete OK, error CRC, etc.). Para recibir un paquete, cada descriptor de paquetes debe inicializarse y pre-asignársele una estructura *sk_buff* (la que se utiliza en Linux para almacenar los paquetes entrantes) vacía. Una estructura *sk_buff* contiene 240 bytes para almacenar el paquete de metadatos correspondiente a los protocolos utilizados por las capas superiores (TCP por ejemplo), y 2KBytes para almacenar el paquete en si. El tamaño del buffer en anillo depende del MTU, unidad de transferencia máxima, fijada por defecto a 1500, aunque en nuestro caso se utilizan 9000 bytes, correspondiente al tamaño de las tramas Jumbo, para reducer el número de interrupciones. Cada descriptor de paquete se ubica en el espacio de direcciones accessible a la NIC que, antes de copiar los paquetes en el area de DMA, también tiene que leer los descriptores de paquete listos para recibirse con el fin de determinar las direcciones de memoria en las que deben ubicarse los paquetes recibidos. Así, después de ③, y una vez que se llena el buffer en anillo o termina un periodo de tiempo preestablecido (no se reciben más paquetes), la NIC actualize los correspondientes descriptores con las longitudes de los paquetes, los marca como usados, y envía una interrupción hardware a la CPU0 del sistema a través de la APIC (Advanced Programmable Interrupt Controller), para informar que hay nuevos paquetes listos para procesarse, ④. Cuando la CPU0 recibe la interrupción, el gestor de interrupciones indicado en el driver de la NIC se inicia, ⑤, comprueba la causa de la interrupción leyendo en el registro correspondiente de la NIC, determina la dirección y las longitudes de los paquetes, y mapea las estructuras *sk_buff* para hacerlas accesibles a la pila de protocolos, ⑥. En la implementación que consideramos, el mapeo se realize a través de punteros a los paquetes del buffer en anillo como se explica a continuación. Una vez ubicada la estructura *sk_buff* y puede accederse a los paquetes del buffer en anillo, CPU0 llama a la función *netif_rx_schedule()* para incluir al dispositivo en la lista de sondeo (poll list) de la CPU, ⑦. Lo que resta del procesamiento del paquete se lleva a cabo con la rutina softirq [10], que es planificada por el SO para el procesamiento del protocolo correspondiente (IP y TCP/UDP. La rutina softirq se activa para ser ejecutada mediante la función *cpu_raise_softirq()*,⑧. Así, la función *do_softirq()*, utilizando *softirq_pending()*, encola *softirq* y, al mismo tiempo, la function *softirq_pending()* devuelve una mascara de 32 bits que indica el estado de las rutinas softirq pendientes a las que señala el punter *softirq_vec*. A partir de aquí, el paquete almacenado en *sk_buffer* comienza a ser procesado por los niveles de protocolo superiors a través de la función *netif_receive_skb()*, ⑨. Una vez softirq ha procesado los protocolos, CPU0 mapea los datos desde *sk_buff* a los buffers del espacio de memoria de usuario donde se encuentran accesibles para la aplicación, ⑩.

Las aplicaciones de red implican a muchos procesos. Por ejemplo, las aplicaciones web involucran a un servidor web que extrae los datos del disco o de la memoria y realiza cierto procesamiento para construir la respuesta a los procesos clientes, a los procesos de la pila de protocolos, y a los procesos

del SO. Así, si el servidor se sobrecarga en el caso de un número elevado de conexiones y la necesidad de realizar calculos complejos, la arquitectura del sistema es crucial para el rendimiento del servidor. En las versiones del núcleo de Linux anteriores a la 2.4, el procesamiento de los paquetes se realizaba en el denominado contexto *bottom half*, que no podia ejecutarse concurrentemente, incluso existiendo varias CPU disponibles. Esta limitación hacía que los *bottom half* no fueran adecuados para enlaces de anchos de banda elevados en los que los paquetes deben ser procesados a gran velocidad. No obstante, desde la version 2.5 del núcleo de Linux, el mecanismo basado en el uso de textitsoftirq permite reemplazar los *bottom half* haciendo posible la ejecución diferida de las funciones asociadas al correspondiente evento software [10], [11]. Aunque *softirqs* del mismo tipo (por ejemplo softirq de recepción de paquetes) no pueden ejecutar concurrentemente sobre la misma CPU, esto si es posible si hay varias CPU en el sistema. En este caso distintas softirqs pueden lanzarse en procesadores diferentes y varias hebras de procesamiento de TCP/IP pueden ejecutarse en paralelo en esos procesadores. Esta circunstacia constituye la base de la interfaz AbNI que proponemos, y que se describe a continuación.

B. Recepción de paquetes con AbNI

La recepción de paquetes con la interfaz AbNI (Affinity-based Network Interface) se resume en la Figura 2. Esta interfaz de red requiere ciertas modificaciones en el código del núcleo del SO junto con el uso de ciertas funciones del núcleo de SMP que permiten aprovechar la afinidad de interrupciones. De esta forma, en la recepción de paquetes, las instrucciones generadas en la NIC se pueden redirigir a la CPU1, como se muestra en la Figura 2. Para eso se puede utilizar *smp_affinity*, disponible a partir de la version 2.6 del núcleo de Linux. Al mismo tiempo, es necsario iniciar los textitsoftirqs en una CPU distinta a la que ejecuta la rutina de servicio de interrupción (ISR) correspondiente a la interrupci'on generada por la NIC (NIC IRQ). A continuaci'on de describen las modificaciones del c'odigo del n'ucleo y del mecanismo de interrupciones que modifica el comportamiento de la interfaz de red, en el caso de la recepci'on de paquetes.

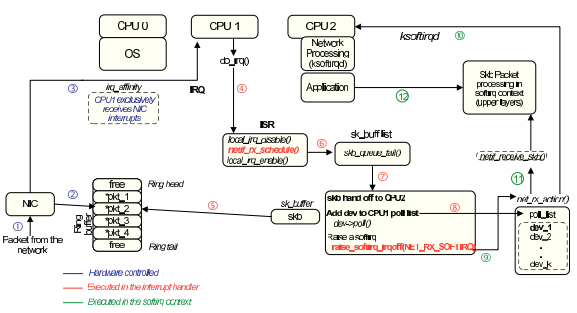


Fig. 2. Recepción de paquetes en la interfaz AbNI (las funciones del núcleo que se han modificado se indican en negrita)

Cuando llega un paquete,①, la NIC lo transfiere al buffer en anillo, como en el caso anterior, utilizando DMA, ②. Cuando termina la transferencia mediante DMA, la NIC interrumpe a la CPU (CPU1 en la Figura 2), ③. Como se indica en la Figura 2, hay que modificar el funcionamiento de la interrupci'on dado que, por defecto, la APIC env'ia todas las interrupciones a la CPU0 (incluyendo la eth0 proveniente de la NIC). As'i, si la velocidad de recepci'on de paquetes es elevada, como puede ocurrir en los enlaces multigigabit, se generar'ia un n'umero de interrupciones por segundo considerable. Esta situaci'on se puede resolver mediante la máscara *smp_affinity*, que permite distribuir las peticiones de interrupción (IRQs) entre las CPUs del sistema: las interrupciones de la NIC pueden redirigirse a CPU1, ③, mientras que las restantes se dirigen a CPU0. Esto se ha implementado a través del sistema de ficheros virtual */proc/irq*, modificando la mascara de afinidad y configurando la APIC para que la NIC interrumpa exclusivamente a CPU1. Una vez aceptada la interrupción, CPU1 empieza la ejecución del driver correspondiente a la ISR ④, que mapea los paquetes almacenados en el buffer en anillo a las estructuras *sk_buffe*, ⑤, se llama a la función *netif_rx_schedule()* y se incluyen los *sk_buffers* no procesados en una lista, ⑥, y el dispositivo en la lista de sondeo de CPU1, ⑦.

Todas las funciones que se han indicado antes se ejecutan en la CPU que ha recibido la interrupción (CPU1). As; los paquetes recibidos se almacenan en la memoria direccionada por CPU1. Es más, las estructuras *sk_buff* que alojan los paquetes son mapeadas por CPU1 (que ejecuta el gestor de interrupción) y se transfieren (*sk_buff handoff*) para permitir el procesamiento de los paquetes en otras CPUs, bajo un contexto *softirq*. Para ello se utilizan las variables por CPU [10], [11] que hacen posible que cada CPU del sistema pueda disponer de su propia copia de la variable por-CPU. Al mismo tiempo, estas variables por-CPU pueden mantenerse en las caches de las respectivas CPU en donde se procesan para minimizar así, en el bus del sistema, el tráfico debido a actualizaciones de variables (aunque podría aparecer algún tráfico para mantener la coherencia de las caches). Este es el caso de las estructuras *softnet_data* por-CPU que se definen en *netif_rx_ schedule()*. Como se ha indicado, los paquetes recibidos se añaden a la lista de *sk_buffer* que puede enviarse a una CPU específica para su procesamiento. No obstante, lanzar la rutina softirq en una CPU distinta a CPU1, ⑧ (ejecutaría la ISR correspondiente a la IRQ de la NIC) necesita que se hagan algunas modificaciones en la función *netif_rx_schedule()*, llamada por la interfaz NAPI bajo la rutina *softirq*.

Como se ha dicho, *softirq* procesa los paquetes. Aunque la CPU1 interrumpida es responsable de sondear (*polling*) el dispositivo, la rutina *softirq* debe ejecutarse en la misma CPU que ejecuta la aplicación (CPU2 en la Figura 2). Mediante esta estrategia se intenta evitar los fallos de cache de-

bido a que los datos que llegan son compartidos por procesos/hebras ejecutados en distintas CPUs. El código de la ISR del driver de la NIC también marca un procedimiento *softirq* como pendiente de ejecución (es decir, lo inserta en la cola de ejecución de *softirq*) y establece las condiciones iniciales para la ejecución de *softirq* mediante *raise_softirq()* (es decir, *NET_RX_SOFTIRQ* para la recepción de paquetes) ⑨. Hemos modificado la rutina *raise_softirq()* para que se pueda ejecutar *softirq* a través de la rutina *ksoftirqd* correspondiente a una CPU específica, ⑩. El uso de hebras (kernel threads) *ksoftirq*, introducido desde el núcleo 2.6, evita tener que comprobar continuamente los *softirqs* pendientes, lo que implicaría la ejecución de *do_softirq()* en un bucle que se repetirá continuamente y ocasionaría la falta de atención a las hebras de usuario. No obstante, el uso de hebras (kernel threads) para ejecutar las *softirq* es una solución de compromiso puesto que las hebras *ksoftirqd* (también llamadas *ksoftirqd_CPUX*) tienen baja prioridad (las hebras de usuario tienen una prioridad mayor). Entonces, después de la primera ejecución de *do_softirq()* para procesar el número de paquetes *netdev_max_backlog*, el siguiente paquete en entrar no se procesará hasta que *ksoftirqd* alcance el tiempo de CPU correspondiente, y las hebras de usuario no sean procesadas con una mayor prioridad. Dado que nuestro propósito es utilizar un núcleo específico para el procesamiento de paquetes, se ha incrementado la prioridad del *ksoftirqd* que se ejecuta en ese núcleo para la ejecución de *NET_RX_SOFTIRQ* softirq. Esto se ha conseguido a través de modificaciones en la function *ksoftirqd* (en /linux/kernel/softirq.c). Una vez que el paquete almacenado en la estructura *sk_buffer* se ha procesado en las capas inferiores, la rutina *netif_receive_skb()* hace possible que el paquete sea procesado por las capas superiores (IP), y los datos sean copiados por CPU2 desde los *sk_buff* al buffer de la aplicación mapeado en la memoria de usuario ⑪, estando disponible para la aplicación ⑫.

Algorithm 1: Recepción de paquetes con modificaciones en el núcleo

1. Se recibe el paquete en el NIC.
2. Después de la transferencia de DMA al buffer en anillo se envía IRQ a CPU_i
3. *dev_alloc_skb()* mapea paquetes del buffer en anillo en sk_buff.
4. *netif_rx_schedule()* comienza el procesamiento de protocolos de alto nivel.
4.1 *skb_queue_tail()* encola estructuras *sk_buff* a procesar (softirq context: packet processing) y transfiere sk_buff a CPU_j.
4.2 La NIC se incluye en la lista de sondeo de CPU_j ($dev \rightarrow poll()$).
4.3 *raise_softirq_irqoff(NET_RX_SOFTIRQ)* se ejecuta en CPU_j para activar el flag NET_RX_SOFTIRQ (softirq pendiente).
4.4 *net_rx_action()* ejecuta la softirq.
4.5 *netif_receive_skb()* continua procesando los paquetes en los protocolos de nivel superior (IP).
5. La aplicación, en el espacio de usuario, toma los datos de las estructuras *sk_buf*.

El Algoritmo 1 describe la operación del driver de dispositivo después de recibir un paquete en la NIC y las funciones incluidas en la ISR y el mecanismo de interrupción utilizados en nuestra propuesta. Algunos detalles adicionales sobre las funciones indicadas se pueden encontrar en [10], [11], [12].

C. Afinidad y localidad de los datos

Las Figuras 3 y 4 muestran dos arquitecturas de multiprocesador alternativas para analizar los aspectos relacionados con la afinidad. En dichas figuras también se indica la forma en que se han distribuido los procesos para equilibrar la carga entre las CPUs del nodo. En la alternativa de la Figura 3, que se ha denominado *hybrid(1)*, CPU0 y CPU1 comparten la cache L2 mientras que la CPU2, que como se ha indicado en la Figura 2 ejecuta la pila de protocolos y la aplicaci'on que consume los paquetes (las tareas MPI que utilizaremos como benchmarks en la sección 3), tiene su propia cache L2. En esta configuración de la Figura 3, las interrupciones se redirigen a CPU1, que ejecuta el driver de la NIC. En esta alternative se puede producer un número de fallos de cache relativamente alto, y por tanto, se perderán ciclos al acceder a los datos de memoria, especialmente si el servidor está muy cargado. La configuración de la Figura 4 se ha denominado *hybrid(2)*. En este caso se utiliza una configuración de caches distinta de la que utiliza *hybrid(1)* (Figura 3). Las interrupciones también se redirigen a CPU1, como corresponde a la interfaz AbNI que proponemos, para evitar que la NIC interrumpa a la CPU que ejecuta los procesos de la aplicación (CPU2), pero se utiliza una cache L2 unificada. Mediante esta configuración se pueden mejorar los problemas de localidad de datos de las caches puesto que la cache utilizada por CPU2, que como se ha indicado procesa los protocolos y la aplicación, también está compartida con CPU1, y se ocasionarán menos fallos en la cache L2.

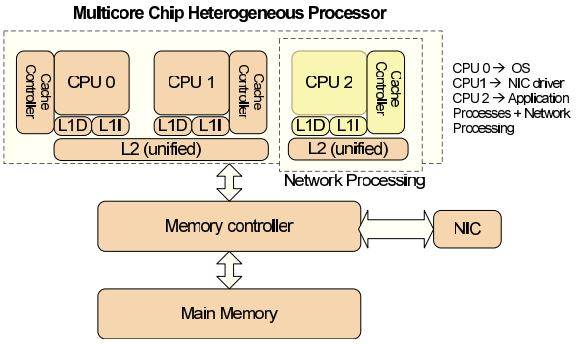


Fig. 3. Configuració (hybrid(1))

III. Resultados experimentales

En esta sección se describen los resultados obtenidos mediante el simulador de sistema completo (full-system simulator) SIMICS [17], que permite no solo simular los detalles del hardware del sistema sino también el sistema operativo y la aplicación que se ejecuta sobre él. De esta forma, las configuraciones *hybrid(1)* e *hybrid(2)*, descritas en las Figuras 3 y 4, se han modelado en SIMICS para permitir la simulación del hardware del sistema y la versión 2.6 del núcleo de Linux, se ha instalado sobre estos modelos

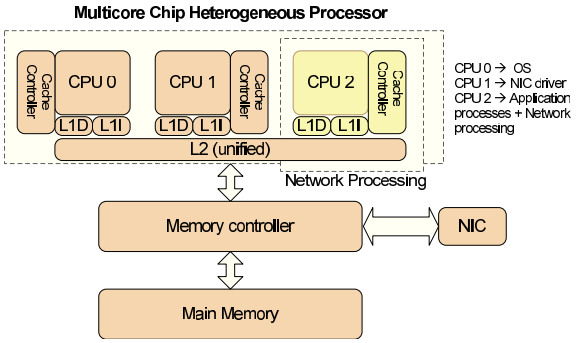
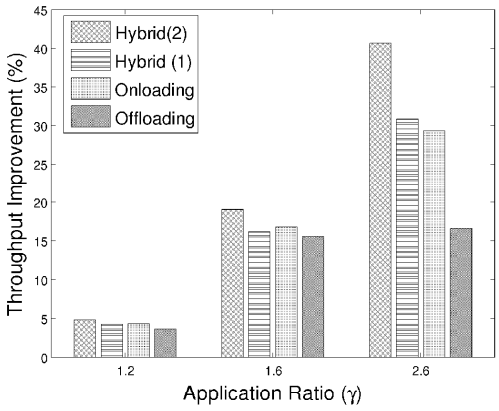
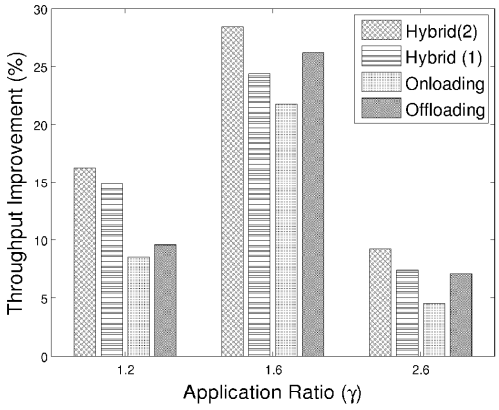


Fig. 4. Configuración (hybrid(2))



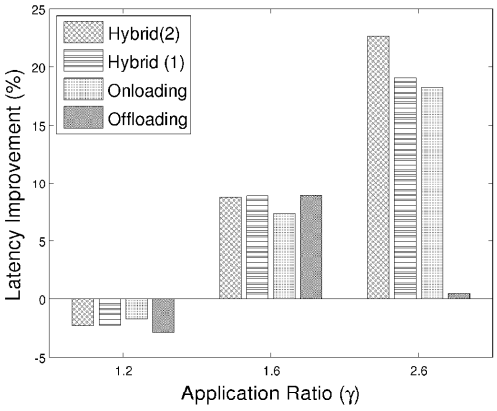
(a)



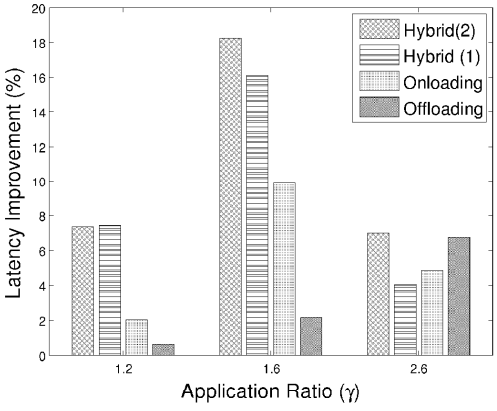
(b)

Fig. 5. Mejoras de ancho de banda (throughput) para mensajes MPI de (a) 2KB y (b) 64 KB

incluyendo las modificaciones realizadas en el núcleo, correspondientes a la interfaz AbNI propuesta, que se han descrito en la Sección 2. Como benchmarks para nuestros experimentos se han utilizado servidores web y códigos MPI (Message Passign Interface) [18] ejecutados en clusters. Por razones de espacio, aquí analizaremos las prestaciones de AbNI en la comunicación mediante paso de mensajes con MPI utilizando distinto tamaño de mensajes y cargas en los nodos de los clusters. MPI utiliza TCP como protocolo de transporte y es prácticamente un estándar para procesamiento paralelo. Los resultados obtenidos para servidores web se pueden consultar en [22].



(a)



(b)

Fig. 6. Mejoras de latencia para mensajes MPI de (a) 2KB y (b) 64 KB

En los experimentos realizados, los nodos del cluster implementan las distintas configuraciones para AbNI que se han descrito en la Sección 2. La Figura 5 muestra las mejoras en ancho de banda que proporcionan las distintas configuraciones de la interfaz de red consideradas, incluyendo las implementaciones de AbNI para las alternativas, *hybrid(1)* e *hybrid(2)*, descritas en las Figuras 3 y 4, en relación con la interfaz de red base. También se han incluido modificaciones de la interfaz de red base correspondientes al uso de las técnicas de offloading y de onloading [12]. En la Figura 6 se muestran las mejoras en la latencia para las mismas configuraciones consideradas en la Figura 5. Las mejoras obtenidas se han evaluado en experimentos realizados para distintos valores del parámetro γ, denominado razón de aplicación (*application rate*) en el modelo LAWS, que tiene en cuenta la relación entre la carga de la aplicación y la de comunicación.

Como se puede ver en las Figuras (a) y (b), tanto para mensajes de pequeño tamaño (2Kbytes), como de tamaño mayor (64 KBytes) se producen mejoras considerables por parte de *hybrid(2)* con respecto a las demás alternativas de mejora consideradas, aunque todas proporcionan mejoras en relación a la interfaz base. En el caso de los mensajes de tamaño pequeño, esas mejoras crecen a medida que γ aumenta (más carga de aplicación en relación con la de

comunicación). Las Figuras (a) y (b) muestran las mejoras en latencia. Para valores bajos de γ el cuello de botella está en la comunicación y puesto que las técnicas propuestas incrementan el número de ciclos disponibles para la aplicación en las CPUs de los nodos, las mejoras son más reducidas o incluso no las hay para paquetes pequeños (2 KBytes). A medida que crece γ (la carga debida a la aplicación crece más que la carga de comunicación), y se observan mejoras mayores puesto que tiene más efecto el hecho de que haya ciclos libres para la aplicación. En el caso de mensajes más largos (64 Kbytes), el cuello de botella podría desplazarse a los buses internos del nodo, sobre todo en nodos de capacidades intermedias y la mejora que proporcionan las distintas técnicas disminuyen al crecer γ, excepto en el caso del offloading (la carga de comunicación se desplaza a la NIC). En cualquier caso, las configuraciones *hybrid(1)* e *hybrid(2)* que implementan AbNI siempre proporcionan mejoras mayores que el resto de las alternativas consideradas tanto en ancho de banda como en latencia y se ven menos afectadas por variaciones de γ (carga relativa de computación y comunicación).

IV. Conclusiones

En este artículo se han considerado las distintas cuestiones relacionadas con el uso de la afinidad en la mejora de las prestaciones de las aplicaciones de red en arquitecturas de nodo con varios procesadores/núcleos. Se ha propuesto la interfaz AbNI basada en el uso de la afinidad entre procesamiento de las tareas, y ubicación de los datos y de la atención a las interrupciones asociados a dichas tareas, para mejorar las prestaciones de las aplicaciones de red. Las prestaciones de AbNI se han analizado para dos configuraciones del hardware de los nodos (*hybrid(1)* e *hybrid(2)*, que presentan distintas alternativas de organización de las caches, y se han comparado con interfaces de red que implementan estrategias de onloading y offloading, y con una interfaz de red base. Los resultados experimentales obtenidos mediante simulaciones con el simulador de sistema complete SIMICS, y con cargas de trabajo de paso de mensajes con MPI en clusters, ponen de manifiesto las mejoras que la interfaz AbNI proporciona tanto en ancho de banda como en latencia cuando se compara con la configuración base y con las técnicas de offloading y onloading. Así, se han obtenido mejoras de alrededor del 35% en ancho de banda y del 23% en latencia. Todavía quedan bastantes aspectos que analizar a nivel de sistema operativo para mejorar las prestaciones de comunicación en las plataformas multinúcleo.

Agradecimientos

Este trabajo ha sido financiado parcialmente por el proyecto TIN2012-32039 (Ministerio de Economía y Competitividad de España) y por la Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

Referencias

[1] Balaji P, Feng W, Panda D.K. *Bridging the Ethernet-Ethernot performance gap*, IEEE Micro, 2006, 1:24–40.

[2] Bhoedjang R, Rúhl T, Bal H.E. *User-level Network Interface Protocols* IEEE Computer, 1998, 1:53–60.

[3] Binkert, N.L., Hsu, L.R., Saidi, A.G., Dreslinski, R.G., Schultz, A.L., Reinhardt, S.K. *Analyzing NIC Overheads in Network-Intensive Workloads*, In Proc. 8th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW), 2005.

[4] Cgroup Documentation. *http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt*

[5] Gilfeather P, Maccabe A. Modeling protocol offload for message-oriented communication, In Proceedings of the 2005 IEEE International Conference on Cluster Computing, September 2005, pp.1–10.

[6] Hansen T, Mainkar V, Reeser, P. *Performance Comparison of Dyamic Web Platforms*, Computer Communications. 2002, 26(8):888–898.

[7] Foong A, Fung J, Newell D, Abraham S, Irelan P, Lopez-Estrada, A. *Architectural characterization of processor affinity in network processing*, In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, March 2005, pp.207–218.

[8] Broadcom webpage. *http://www.broadcom.com/*, 2007.

[9] Intel 82599 10GbE Controller Datasheet *www.intel.com/content/www/us/en/ethernet-controllers/82598-10-gbe-controller-datasheet.html*, 2009.

[10] Benvenuti, C. *Understanding Linux Network Internals*, 2nd Edition. OReilly Media Inc., 2005.

[11] Love R. Linux Kernel Development. *Novell Press, Second Edition*, 2005.

[12] Ortiz A, Ortega J, Diaz A, Prieto, A. *Network interfaces for programmable NICs and multicore platforms*, Computer Networks. 2010, 54(3):357–376.

[13] GadelRab S. *10-Gigabit Ethernet Connectivity for Computer Servers*, IEEE Micro, 2007, 1:94–105.

[14] Ortiz A, Ortega J, Díaz A, Cascón A, Prieto A. *Protocol offload analysis by simulation*, Journal of Systems Architecture, 2009, 55(1):25–42.

[15] Jonathan Corbet. *Receive Packet Steering*, http://lwn.net/Articles/362339/, 2009.

[16] Jake Edge. *Receive Flow Steering* , http://lwn.net/Articles/382428/, 2010.

[17] Magnusson, P. S. et al. *Simics: A Full System Simulation Platform*, IEEE Computer, 2002, pp.50–58.

[18] MPICH2: a high performance and widely portable implementation of the Message Passing Interface (MPI) standard. *http://www.mpich.org/*, October 2012.

[19] Bruijn W, Bos H. *Model-T: Rethinking the OS for terabits speeds. In Proceedings of the INFOCOM workshop on High-Speed Networks*, April 2008, pp.1–6.

[20] Brecht T. et al. *Evaluating network processing efficiency with processor partitioning and asynchronous I/O*, In Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, April 2006, pp.265–278.

[21] Goglin B. *NIC-assisted Cache-Efficient Receive Stack for Message Passing over Ethernet*, Concurrency and Computation: Practice and Experience, 2011, 23(2):199–210.

[22] Ortiz A, Ortega J, Díaz A, Prieto A. *Affinity-Based Network Interfaces for Efficient Communication on Multicore Architectures* Journal of Computer Science and Technology, 2013, 28 (3): 508–524

Increasing the Endurance of Phase-Change Memories with Cache Replacement Policies

Roberto Rodríguez-R, Fernando Castro, Daniel Chaver, Luis Piñuel y Francisco Tirado[1]

Resumen— **The different performance evolution between the microprocessor and main memory is one of the greatest challenges that current designers face in order to develop more powerful computer systems. In addition to this problem, called memory gap, the scalability of the Dynamic Random Access Memory (DRAM) technology is very limited nowadays, leading to consider new memory technologies as possible candidates for the replacement of conventional DRAM. Phase-Change Memory (PCM) is currently postulated as the best alternative.**

PCM exhibits significant advantages over DRAM, but also some drawbacks, like its low endurance – limited by the number of write cycles that can be performed on each cell– that need to be mitigated before it can be used as the main memory technology for the next computers generation. This work presents a behavior analysis, in terms of number of writes to main memory, of some conventional cache replacement policies. Besides, new last level cache (LLC) replacement algorithms are exposed, aimed at reducing the number of writes to PCM and hence increasing its lifetime, without significantly degrading the system performance. In this paper we target embedded processors with PCM main memory. Experimental results show that on average, compared to a conventional Least Recently Used (LRU) algorithm, our policies manage to reduce the amount of writes to main memory for MiBench and SPEC CPU2006 suites by around 30% and 12% respectively, reducing significantly the energy consumption of the memory system and without degrading performance.

I. Introduction

ALTHOUGH Dynamic Random Access Memory (DRAM) has been the prevalent building block for main memories during many years, current research is focused on exploring other technologies for designing future memory systems in response to the scaling constraints observed when DRAM is used with small feature sizes. Among these technologies, PCM is one of the prime contenders.

PCM is a low-cost and non-volatile memory that avoids the need of refreshing the content of the cell, reducing the static power consumption. Furthermore, it provides higher density than DRAM and therefore much higher capacity for the memory system within the same budget. A PCM cell consists of two enveloping electrodes, a thin layer of chalcogenide and a heating element. The heater is just a material that produces Joule heat when an electrical current is driven through, warming the chalcogenous material. In order to write a logical value in the cell, the heating element is employed to apply electrical pulses to the chalcogenide, changing the properties of the material and leading to two different physical states: amorphous (high electrical resistivity) and crystalline (low resistance). Notably, if a high-intensity current pulse is applied, the material reaches over 600^oC and melts. Then, it is cooled down quickly, making it amorphous (RESET process). If the pulse is longer and with lower intensity, the material goes through an annealing process allowing the molecules to re-crystallize, lowering the electrical resistance (SET process). Thus, the chalcogenide switches easily, rapidly and in a reliable way between both states. The process for reading the stored value just consists in applying a low current to the cell in order to measure the associated resistance. The limited endurance of PCM relies on the fact that after a certain number of writes on a PCM cell (around 10^8), the heating element is detached from the cell as a consequence of the continuous expansions/contractions derived from the writing process, leaving the cell in a *stuck at failure* state; from that moment on, although the cell is still readable, the stored valued can not be changed any more in this failing cell. Thus, the write traffic must be cut in order to not limit the lifetime of PCM-based systems. For this purpose, several architectural techniques have been proposed recently [1], [2], [3], [4], [5].

In this work, we deal with this problem by focusing on the LLC replacement policy. The goal is to efficiently design a policy that retains in the LLC the most frequently written blocks. Thus, in this context, the LLC replacement policy is not just oriented to provide a low miss rate and to increase system performance, but also to reduce the amount of writes to main memory. We analyze the behavior of classical and current cache replacement policies [6], [7], [8] regarding the amount of writes to the memory system, and we propose new policies in order to reduce this write traffic without significantly impacting performance.

The rest of the paper is organized as follows: Section 2 describes the work related to our research. Section 3 presents the algorithms proposed to increase the lifetime of PCM-based memory systems. Section 4 and Section 5 details the experimental framework used and analyzes the obtained results respectively. Finally, Section 6 concludes.

II. Related work

A. Techniques for reducing writes to PCM

During the last years, several techniques have been proposed for extending PCM lifetime. As we will see during the paper, the mechanisms that we propose

[1]Grupo ArTeCS, Dpto. de Arquitectura de Computadores y Atomática, Universidad Complutense de Madrid, e-mail: rrodriguezr, fcastror, dani02, lpinuel, ptirado@ucm.es.

are completely orthogonal to these ideas.

- Eliminating redundant bit writes [9], [1]: As reads are much faster and less power consuming than writes in PCM, every write is preceded by a read and a bitwise comparison, writing only those bits that change.
- *Flip-N-Write* [10]: Before performing a write to PCM, both the data to write and its bitwise inverse are compared with the data stored in the row, writing the one that involves less bit flips.
- *Wear Leveling* [9]: These techniques try to even out the number of writes performed to each PCM cell.
- Hybrid memory [4]: The idea here is to avoid writes to PCM by inserting an extra memory level that filters most of such accesses, and which is built on a memory technology not so sensible to writes.
- Write-back impact reduction [11], [12]: The amount of writebacks to PCM is reduced extending [13] by including this goal in the proposed LLC partitioning algorithm and changing the LRU policy in order to consider also dirtiness information [11]. The writebacks impact is also reduced by evenly distributing them among write queues.

B. Cache replacement policies

When the insertion of a new block in cache involves an eviction, the cache replacement policy must decide which block to replace. In general, as Bélády established in [14], the best decision is to replace the block that will not be referenced again for the longest time. Since knowing the future is not possible, the different policies proposed in the literature try to predict, based on the analysis of past information, which one is such block. Among all the classic and modern mechanisms, in this work we make use of the following:

- LRU (Least Recently Used) [6]: It discards the least recently used block, under the philosophy that, due to temporal locality, it is also the block that will not be required for the longest time in the future.
- SRRIP, BRRIP and DRRIP (Static, Bimodal and Dynamic Re-Reference Interval Prediction respectively) [8]: Here the recency stack is thought of as a Re-Reference Interval Prediction (RRIP) Stack that represents the order in which blocks are predicted to be re-referenced. The block at the head of the RRIP Stack is predicted as *near-immediate* (i.e. the block will be re-referenced sometime soon) while the block at the tail of the RRIP Stack is predicted as *distant* (i.e. the block will be re-referenced in the distant future). On a cache miss, the block at the tail of the RRIP chain (i.e., the block predicted to be referenced most far into the future) is replaced.
  The authors first propose SRRIP, which uses M bits per cache block to store one of 2^M possible RRPV (Re-Reference Prediction Values). On cache fills, SRRIP predicts that the missing block will have an intermediate state denoted as *long* re-reference (RRPV=2^M-2). On re-reference, SRRIP may employ two different policies, HP (Hit Priority) and FP (Frequency Priority): SRRIP-HP updates the RRPV of the associated block to zero while SRRIP-FP just decrements it. On a cache miss, SRRIP randomly selects as the victim block one with a *distant* re-reference interval prediction (RRPV=2^M-1). If such a block does not exist, SRRIP updates the re-reference predictions by incrementing the RRPVs of all blocks in the cache set and repeats the search.
  SRRIP inefficiently utilizes the cache when the re-reference interval of all blocks is larger than the available cache. In such scenarios, SRRIP causes cache thrashing and results in no cache hits. To avoid it, the authors propose BRRIP that inserts majority of cache blocks with a *distant* RRIP and infrequently (with low probability) inserts new cache blocks with a *long* RRIP. This policy helps to preserve some of the working set in the cache. For non-thrashing access patterns, always using BRRIP can significantly degrade cache performance. In order to be robust across all cache access patterns, a third policy (DRRIP) proposes to use Set Dueling [15] to identify which replacement policy is best suited for the application, choosing between SRRIP and BRRIP as Fig. 1 illustrates.
- CLean-Preferred victim selection policy CLP [11]: This policy gives preference to clean blocks when choosing a victim. CLP aims to maintain dirty blocks in the cache to increase the probability that writes are coalesced. The authors propose a family of clean-preferred replacement policies, called *N-Chance*. The N parameter reflects how much preference is given to clean blocks. The algorithm selects the oldest clean block among the N least recently used ones as the victim. If such a block does not exist, the LRU block is used. In this paper we evaluate CLP with N equal to the cache associativity, since it is the policy reporting the highest write reduction.

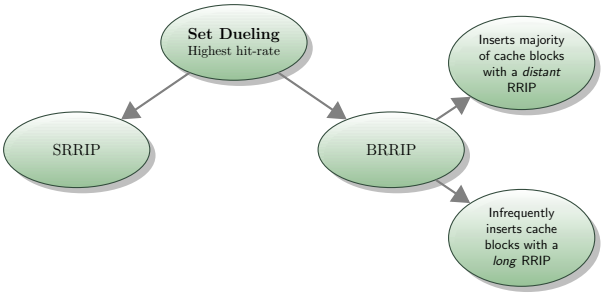


Fig. 1. Insertion of a new block under DRRIP.

III. Proposed Policies

The replacement policy determines how to manage insertion, promotion, and victimization of blocks in cache. Usually, decisions are only performed with the goal of increasing hit rate. However, when the LLC is backed up with a PCM, decreasing the amount of LLC writebacks may become even more important than reducing the number of misses. Hence, we propose several modifications to the insertion, promotion and victimization policies that incorporate this objective. Information such as the block dirtiness state or the kind of access performed (read or write), are used by our policies to predict whether the block will generate future writebacks.

In some cases a hybrid mechanism that combines two or more replacement policies is employed, selecting in each replacement the policy that should be used. The most common mechanism used for making this decision is the Set Dueling technique proposed in [15]. Generally, the decision is guided only by hit rate information. However, in the case of a PCM-based system, the Set Dueling mechanism should also consider information concerning the amount of writebacks that the LLC will generate.

Next, we describe the replacement policies proposed in order to reduce the amount of PCM writes. Although we have tried many different options, we only explain and evaluate those that report interesting results. To denote them, we add a *W* to the acronym of policy they are based on:

- *DRRIPW4*: This policy modifies the Set Dueling mechanism as follows: it selects the replacement policy that reports the lowest number of writebacks to main memory, instead of the one that achieves the highest hit rate.
- *DRRIPW5*: We augment the number of bits in the RRPV from 2 (original DRRIP) to 3, enhancing the granularity of the re-reference prediction. Besides, we employ a modified HP promotion policy, in which clean blocks are promoted to *long* RRIP instead to *near-immediate*.
- *DRRIPW7*: This policy operates like DRRIPW4, but instead of using a static promotion policy it uses a dynamic one: when the Set Dueling mechanism selects the SRRIP component, HP is employed, whereas when BRRIP is selected, FP is used.
- *DRRIPW8*: Like the previous policy, it operates as DRRIPW4 and changes the promotion policy to a dynamic one: clean blocks are promoted with FP while dirty blocks use HP. This way, dirty blocks (which generate writebacks), are given more opportunities to stay in the cache than clean blocks.
- *DRRIPW9*: This algorithm combines the two previous policies. It operates like DRRIPW4, and changes the promotion policy as follows: when the Set Dueling mechanism selects the SRRIP component, HP is employed for dirty blocks and FP for clean blocks, whereas when BRRIP is selected, FP is always used.
- *DRRIPW10*: This algorithm modifies the victimization policy of DRRIPW8: Among all blocks with a *distant* RRIP, it selects a clean one; if not found, a dirty block is victimized. Like in the original DRRIP policy, if no blocks with a *distant* RRIP exist, it increments RRPV of all blocks and repeats the process.
- *DRRIPW11*: In this case, the victimization policy of DRRIPW4 is modified: When a replacement is required, this algorithm looks for a candidate satisfying two criteria: 1) RRPV with the maximum value, 2) clean block. If such a block is not found, the RRPV of all blocks in the cache set is augmented by one and a new search starts. If the RRPV of all blocks are set to the maximum values and no clean block exists, the algorithm evicts the first block in the array.
- *DRRIPW12*: This algorithm is very similar to the previous one, but with an important difference: during the first stage, like before, the clean block with the highest RRPV is victimized, but in this case without changing the RRPV state of the blocks. During the second stage, a common search is performed.

IV. Experimental Framework

As simulation infrastructure we use Pin [16] –a dynamic instrumentation tool that allows us to inject C or C++ code at arbitrary points of an executable during runtime– and a modified version of SESC [17] (a cycle-accurate microprocessor architectural simulator), called MultiCacheSim [18], to model the cache hierarchy. This infrastructure operates as follows: First, we instrument the application with Pin, obtaining a list with the memory accesses that it performs. Then, we use this list as input for the cache simulator, obtaining information about hits, misses, writebacks, etc. that take place in each level of the hierarchy. With this information we can easily extract the number of accesses and writes to main memory, which constitute the main metrics for our purposes. Fig. 2 illustrates this experimental environment.

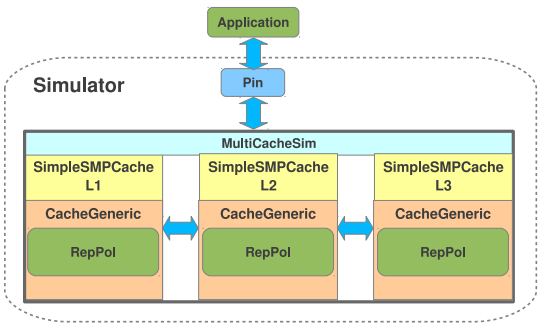


Fig. 2. Simulator architecture.

The cache hierarchy employed to evaluate our policies is based on an Intel Atom [19] that provides two levels, being the LLC a non-inclusive (also non-

exclusive) cache. Their sizes, as well as other important parameters, are shown in Table I. Since we target a processor in the frontier between embedded and general purpose, we use both the MiBench [20] and the SPEC CPU2006 [21] suites with *large* and *train* entries respectively. All applications are executed until completion. In the case of the SPEC CPU2006 suite, results of 4 out of 29 benchmarks are not considered in the evaluation section due to experimental framework constraints.

Regarding the energy model employed, we use CACTI 6.5 [22] to determine the energy consumption per access in each cache level, whereas for computing the energy consumption associated with accesses to main memory we follow [12], employing 1J/GB and 6J/GB per PCM read and PCM write respectively. In Table II we show data regarding latencies and energy consumption per memory level.

TABLA I
CACHE CONFIGURATION

Cache level	Size
L1	24KB, 4 ways, 64-byte line size
L2	512KB, 8 ways, 64-byte line size

TABLA II
LATENCIES AND ENERGY CONSUMPTION

Level	Latencies (cycles)	Energy (Read/Write/Tag) (nJoules)
L1	1	0.092/0.066/0.001
L2	15	1.102/1.166/0.010
PCM read	200	59.604
PCM write	4000	357.627

Finally, it is worth to note that we employ the Average Memory Access Time (AMAT) in order to estimate the performance of each evaluated algorithm. Although AMAT just considers the latencies of each level and not the overlapping between memory accesses, it is commonly accepted as a valid metric to estimate system performance. Next we show the equations employed to determine the memory hierarchy energy consumption (1) and the AMAT (2):

$$Energy = \sum_{i=1}^{2}(RHL_i * REL_i + WHL_i * WEL_i + \\ +(RML_i + WML_i) * (TEL_i + WEL_i)) + \\ +RPCM * REPCM + WPCM * WEPCM \quad (1)$$

where RHL_i and WHL_i denote read and write hits in cache level i respectively, RML_i and WML_i denote read and write misses in cache level i respectively, RPCM and WPCM correspond to the amount of reads and writes to PCM respectively, REPCM and WEPCM denote the energy consumption per read and write to PCM respectively and finally REL_i, WEL_i and TEL_i correspond to the energy consumption associated to a read, a write and a tag array consult in the cache level i respectively.

$$AMAT = \frac{HL_1 * LL_1 + RHL_2 * LL_2 + RPCM * LRPCM}{AccL_1} \quad (2)$$

where HL_1 and $AccL_1$ denote the total number of hits and accesses respectively to the first level cache, LL_i refers to the latency of i cache level and LRPCM denotes the latency in the access to PCM.

V. EVALUATION

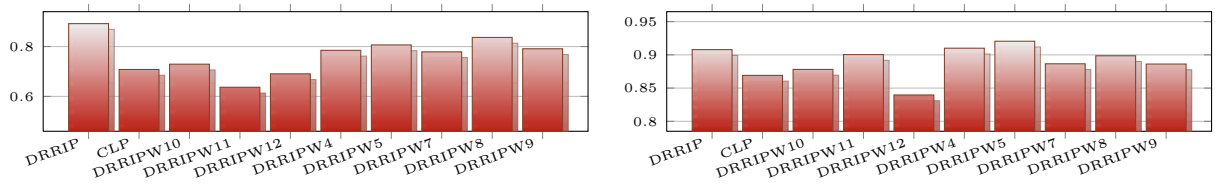
In this section we present a detailed experimental study of the different LLC replacement policies that we have described previously. In subsection A we report the main results derived from our policies. Then, for the sake of clarity, in subsection B we compare our best performing policies in terms of writes reduction against an optimal policy.

A. Main Results

Given that the main goal of this work is to reduce the amount of writebacks from LLC to PCM without significantly degrading performance, we report results about the number of writebacks, the LLC miss rate and the AMAT. Besides, being the energy consumption one of the main motivations for adopting PCM as the main memory technology, we also include this metric in our experiments. Note that in all the figures and for each evaluated policy, we show the arithmetic mean of the normalized metrics respect to LRU, considering all the selected applications of each suite (MiBench and SPEC 2006). It is worth noting that L1 always employs the LRU algorithm, whereas the policies under evaluation are used in the second level of the cache. Note also that results shown for original DRRIP and for our proposed DRRIP-based algorithms that do not specifically change the promotion policy, are derived from using an *HP promotion policy*.

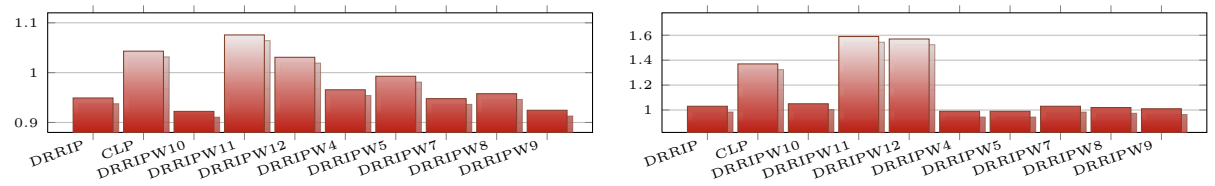
Fig. 3 illustrates the average number of writes from LLC to PCM that generate the different policies. Focusing on MiBench suite, we observe that DRRIPW11 is the policy that gets the highest reduction with around 37%. In the SPEC 2006 scenario, DRRIPW12 exhibts the highest reduction (around 13%), followed by CLP (almost 12% reduction), and DDRIPW10 that achieves a 10% reduction. Note that the percentage of avoided writes is higher in the MiBench suite due to the lower amount of writes that take place in this scenario (between 2 and 3 orders of magnitude compared to the SPEC 2006 scenario).

Being writeback reduction to PCM the main objective of this work, we should not neglect performance. Fig. 4 and Fig. 5 illustrate the miss rate and the AMAT respectively for the different policies evaluated. In some policies a high write reduction comes at the expense of a significant miss rate increment and consequently a performance drop. Although in the MiBench scenario this AMAT degradation may be tolerable, like occurs with DRRIPW11 and CLP



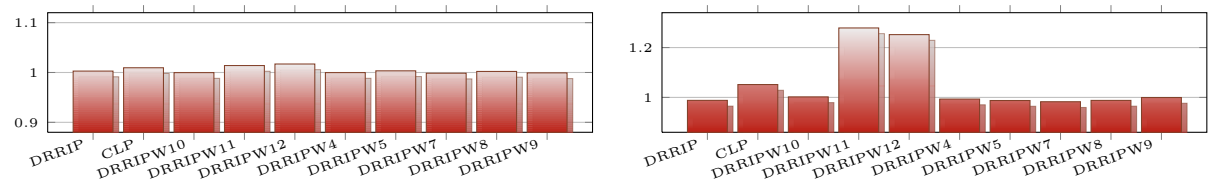
(a) MiBench (b) SPEC2006

Fig. 3. Writes to main memory normalized to LRU.



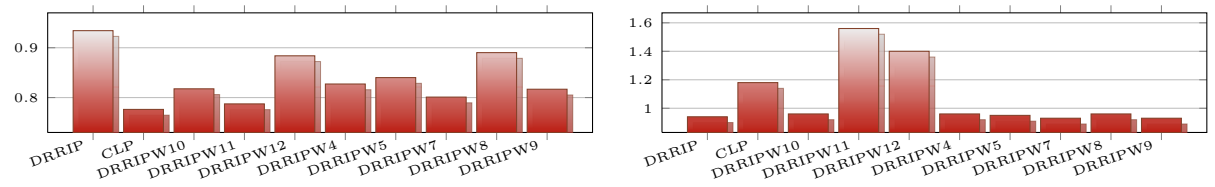
(a) MiBench (b) SPEC2006

Fig. 4. LLC miss rate normalized to LRU.



(a) MiBench (b) SPEC2006

Fig. 5. AMAT normalized to LRU.



(a) MiBench (b) SPEC2006

Fig. 6. Memory energy consumption normalized to LRU.

algorithms, in the SPEC 2006 scenario this performance drop rises turning unacceptable, invalidating this kind of proposals (DRRIPW11, DRRIPW12 and CLP). Nevertheless, DRRIPW10 (or other policies like DRRIPW4 and DRRIPW7), achieves a significant write reduction (28% and 11% for MiBench and SPEC 2006 respectively in DRRIPW10), maintaining a reduced miss rate and therefore without impacting performance.

Finally, Fig. 6 illustrates energy consumption on the memory system for the different evaluated policies. In the case of MiBench, all policies reduce the energy consumption with respect to LRU, being CLP and DRRIPW11 the less consuming ones (around 20% energy reduction). Instead, in the case of SPEC 2006, these two policies and DRRIPW12 exceed the consumption of the others due to the high miss rate they exhibit. For this suite, most remaining policies oscillate between 5 and 10% reduction.

In summary we can conclude that the optimal policy depends on the particular requirements of the system and the user. For our configuration, DRRIPW10 (and other DRRIP-based policies) constitutes a good trade-off, showing a considerable PCM endurance improvement without degrading performance in both scenarios. Besides, in the case of SPEC 2006, most DRRIP based policies achieve a good trade-off between impact on PCM write traffic and performance.

B. *Comparison with an optimal policy*

In order to evaluate the capacity of our proposed policies to cut the number of writes to main memory, we compare them to a straightforward optimal policy that operates as follows: Starting from the obtained traces containing the writes from L1 to L2, we allocate an array per cache set with an amount of entries (n) matching the associativity of the last level cache. We fill each array with the first n written blocks. When a different blocks is written, it does not fit in the array, so we analyze the trace onward to find the block (among the n blocks in the array and the new written block) that will be re-written longer, evicting this block from the array. In the case that the own new written block was the candidate, we just bypass this block and it is not filled in the ar-

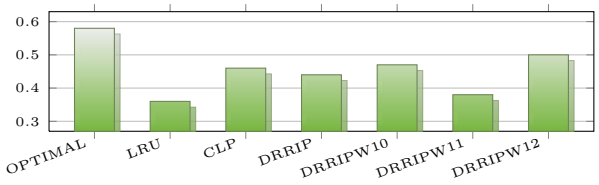


Fig. 7. Comparison between the reduction of writes to main memory reported by an optimal policy and ours.

ray. Each block evicted from the array counts as one write to main memory. Finally, we obtain the total number of writes to PCM with an optimal policy for the selected cache set.

We calculate the amount of writes to main memory according to this optimal policy for several cache sets and we compare these number to those reported by our proposed policies. We observe that the differences between the various analyzed cache sets are negligible. Thus, data derived from cache set 0 are illustrated in Figure 7 in order to infer how near of the maximum reduction in the traffic to main memory some of our proposed policies are.

VI. Conclusions

In this paper we have evaluated the operation of some classical cache replacement algorithms in terms of writes to main memory. As the trend today leads to search for new memory technologies that may replace conventional DRAM, we target PCM and propose different new LLC replacement policies with the goal of minimizing the number of writes to this kind of new memory, thus mitigating the PCM endurance drawback while still taking advantage of all other features derived from this technology.

The obtained results show that most of our policies manage to significantly reduce the amount of writes to PCM. Among them, DRRIPW10 postulates as the best alternative for obtaining a trade-off between write traffic reduction (28% and 11% for MiBench and SPEC 2006 respectively) and performance (0.1% and 0.4% for MiBench and SPEC 2006 respectively) respect to LRU. Furthermore, the energy reduction achieved reaches 18% and 4% for MiBench and SPEC 2006 respectively.

Acknowledgment

This work has been supported by the Spanish government through the research contract CICYT-TIN 2008/508, TIN2012-32180, Consolider Ingenio-2010 CSD2007-0050, and the HIPEAC-3 European Network of Excellence. It also has been supported by a grant scholarship from the University of Costa Rica and Costa Rican Ministry of Science and Technology.

Referencias

[1] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 14, June 2009.

[2] Wangyuan Zhang and Tao Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, p. 2, 2009.

[3] MK Qureshi and MM Franceschini, "Morphable memory system: a robust architecture for exploiting multi-level phase change memories," *Computer Architecture*, 2010.

[4] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24, June 2009.

[5] MK Qureshi and S Gurumurthi, "Phase Change Memory: From Devices to Systems," *Synthesis Lectures on*, 2011.

[6] JL Hennessy and DA Patterson, *Computer architecture: a quantitative approach*, Morgan K. Pub, 2011.

[7] CS Kim, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, 2001.

[8] Aamer Jaleel, KB Theobald, SC Steely Jr., and Joel Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Computer*, 2010.

[9] Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, pp. 143, 2010.

[10] S Cho, "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," *, 2009. MICRO-42. 42nd Annual IEEE/ACM*, 2009.

[11] Alexandre Peixoto Ferreira, Miao Zhou, Santiago Bock, Bruce R. Childers, Rami G. Melhem, and Daniel Mossé, "Increasing pcm main memory lifetime," in *DATE*, 2010, pp. 914–919.

[12] Miao Zhou, Yu Du, Bruce Childers, Rami Melhem, and Daniel Mossé, "Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems," *ACM Transactions on Architecture and Code Optimization*, vol. 8, no. 4, pp. 1–21, Jan. 2012.

[13] Moinuddin K. Qureshi and Yale N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO*, 2006, pp. 423–432.

[14] Laszlo A. Belady, "A study of replacement algorithms for virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[15] MK Qureshi, Aamer Jaleel, YN Patt, and SC Steely, "Adaptive insertion policies for high performance caching," *ACM SIGARCH*, 2007.

[16] Chi-keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Steven Wallace, Vijay Janapa, and Geoff Lowney, "Pin: Building Customized Program Analysis Tools," *Design*, 2005.

[17] I-acoma/University of Illinois at Urbana-Champaign, "http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/," .

[18] Brandon Lucia, "https://github.com/blucia0a/ MultiCacheSim," .

[19] "http://www.intel.com/content/www/us/en/processors/atom/atom-processor.html," .

[20] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Workload Characterization, 2001. WWC-4.*, 2001, WWC '01, pp. 3–14.

[21] "http://www.spec.org/cpu2006/," .

[22] "http://www.hpl.hp.com/research/cacti/," .

Congestion Isolation in Networks-on-chip

José Vicente Escamilla, José Flich [1] and Pedro J. García[2]

Abstract— **CMPs and MPSoCs represent the main trend for high-performance computing. However, as the number of interconnected elements grows, the network performance may be degraded due to aggressive traffic patterns and power-saving mechanisms. In this paper we propose a mechanism for dealing with congestion caused by network performance degradation. This mechanism detects congestion at routers and isolates harmful traffic in order to avoid the HoL-blocking caused by congested traffic.**

Keywords— **congestion; ICARO; contention; network-on-chip; MCSL**

I. Introduction

NOWADAYS , CMPs and MPSoCs are becoming the main trend for high performance computing. As the integration scale goes further, more cores, nodes, or processing units are included in the same chip. Examples of the many-core integrated Intel CMPs are: The Xeon Phi coprocessor [1] with 60 cores, and the single chip cloud computer (SCC) [2] with 48 cores. The Tile-Gx [3] from Tilera, with up to 72 cores is another example. Both platforms, CMP and MPSoCs, implement an interconnection network infrastructure that provides a communication channel for data exchange between all nodes. This network must be able to support high data rates with low latencies to avoid slowing down nodes waiting for remote data to arrive. However, due to its constraints and manufacturing process, in the NoCs framework congestion may arise by several fronts.

As technology advances, integration scale allows to insert more and more transistors on the same chip. However, current high integration scales may lead to faults in critical chip components or variability[4] that may affect to the on-chip-network degrading its performance or simply making links or routers inoperable. On-chip-networks may be reconfigured in order to support these issues by redirecting traffic through an alternative path, however this implies overloading links and routers, thus creating congestion-prone spots.

Currently, heterogeneity[5] is an emerging design policy. Searching for the optimal performance drives the designers to incline for specific-purpose nodes like prefetchers[6], specific modules for complex arithmetic operations[7], etc, instead of using general-purpose modules, therefore the network must be adapted[8] in order to satisfy the communications needs of such devices. Most of these devices show patterns characterized by unpredictable high data rate bursts that may generate hotspots temporarily, thus degrading network performance.

In addition, power-saving in CMPs and MPSoCs is becoming an important design factor. Designing energy efficient devices provides several benefits such as less heat dissipation which minimizes electronics failures and increases batteries life for handheld devices, an essential property for devices like smartphones. As the number of interconnected devices in a CMP or MPSoCs grows the network must grow in accordance. This growth leads to more energy consumption and increases the energy proportion consumed by the network. For example, the Alpha 21364 NoC consumes about 20% [9] of the total power consumption. Several works have been carried out in order to decrease the energy consumption of the network but all techniques imply performance loss to a greater or lesser extent in the NoC. A very extended and effective technique for power-saving consists in decreasing voltage and frequency of the network (DVFS) when workload is low. This technique is very effective in reducing power consumption, however, reducing voltage and frequency can lead to a performance penalty due to congestion. Even when the clock is decreased at the correct moment, if the workload is quickly increased the frequency needs to be rapidly increased due to the high demand and this may take some time for increasing and stabilize the clock. During this time lapse, network is unusable and congestion quickly appears.

Congestion is not a problem by itself. This view of the problem is recently new [12] and opposed to the common belief where efforts were directed towards detecting and eliminating congestion spots or even avoiding the appearance of congestion spots. The real problem that impacts performance is the Head-of-line blocking effect that congestion spots cause. Head-of-line blocking (HoL-blocking) occurs when a packet at the head of a buffer is blocked (because its requested output port/resource is unavailable, that is, congested) and this packet blocks other packets allocated in the same queue, even if those packets request free output ports/resources. If HoL-blocking originated from congestion spots is removed, then the congestion becomes harmless and there is no need to remove the congestion spot. In this paper we follow the same approach and focus on head-of-line blocking effects.

To summarize, power-saving techniques are necessary mechanisms and an important design trend nowadays, but their use may cause congestion problems that need to be addressed. For that purpose, in this paper we present ICARO (Internal Congestion Aware hol-blocking RemOval), a mechanism that dynamically detects bursty and congested traffic respectively in the network, then isolating them and thus guaranteeing that non-harmful traffic is unaffected. This proposal is made by taking into account the strict limitations in area, power, and delay imposed by the networks environment .

II. Related Work and Contribution

There is a plethora of publications for congestion management in off-chip networks, and a increasing

[1]Grupo de Arquitecturas Paralelas, DISCA, Universitat Politècnica de València, Valencia, Spain, e-mail: joseslo@gap.upv.es, jflich@disca.upv.es

[2]Escuela Superior de Ingeniería Informática, Universidad de Castilla-La Mancha, Albacete, Spain, e-mail: PedroJavier.Garcia@uclm.es

number for on-chip networks. However, as stated previously, our approach to deal with congestion is different and relatively new. The idea is, instead of detecting congestion, notifying sources, and then removing congestion (by halting injection), we follow the approach of separating congestion by using dedicated resources. Indeed, our approach followed in ICARO is the application of the RECN mechanism that was proposed for off-chip networks [12]. However, due to the tight limitations in area and power in on-chip networks, we need to follow a different way of implementing it. This is the main property of ICARO in comparison with RECN.

In [13] a mechanism called RCA is proposed for congestion avoidance in NoCs with adaptative routing. RCA uses a composition of multiple global metrics collected from the whole network for conditioning the selected output port through which messages are forwarded in order to avoid congested paths. These metrics are: the count of free virtual channels, the count of free buffers and the crossbar demand. In order to collect the metrics from the whole network, such metrics are aggregated (piggybacking) from a router to the next an so on. In a heavy-congestion situation this mechanism may collapse since the information used to avoid the congestion is aggregated in the same messages that are congested so a vicious cycle may be created.

In [14] authors propose PARS, an adaptative routing policy for avoiding congestion. Authors use a dedicated subnetwork for sending congestion metrics based on the buffers state from certain routers. Then, such metrics are used to select the proper path in order to avoid congested spots.

A predictive-based flow control mechanism is proposed in [15]. Authors propose an end-to-end flow control mechanism based on prediction-models to control the injection rate at the source node. Predictions are computed in every swich using its state and its neighbours state. In order to exchange the necessary data for computing the prediction, routers implement additional wires interconnecting them.

As we start with the premise that congestion is not a problem by itself, we propose ICARO from a different point of view. ICARO is a mechanism for facing harmful effects (HoL-blocking) derived from congestion in NoCs. ICARO separates harmful traffic from the non-harmful one making use of virtual network queues. For carrying this out, congestion is detected at network routers composing the network. Later, end-nodes are notified and thus react by isolating traffic belonging to the congestion into the special virtual network queue.

ICARO is intended to be used for dealing congestion scenarios in challenging environments, specially in those where multiple domains of voltage and frequency islands will be common. In this document, however, we do not evaluate the mechanisms in such scenarios. This is left for future work.

III. Description

In this section we describe ICARO, which can be divided in three stages: congestion detection, congestion notification and traffic separation.

A. Congestion Detection

Congestion is defined as a contention situation extended over the time and, in turn contention arises when two or more flows compete for the same resource. More precisely, contention arises when two or more flows compete for the same output port in the routers, since only one of these flows wins the requested output port at a given time, hence the other flows must wait for the next arbitration turn, increasing latency. Also, these flows remain in their input queues while more flits are enqueued on these input queues, forcing flow control mechanism to work in order to avoid queue overflow by halting injection at the source node and propagating this behaviour. This chain reaction is congestion.

Our proposal in order to detect congestion can be divided in two stages or parts: contention detection and congestion detection. Contention arises when two or more flows compete for the same output port. To detect this, a modified router implementation is proposed in order to keep record of how many input ports are requesting every output port at every cycle. To perform this, a counter of $ceil(\log_2(p-1))$ bits (requests counter) is added to the router, where p is the router radix of the routers in the network (including the local port). We assume a pipelined router design with four stages: IB, RT, VASA, and X. In IB the incoming flit is allocated in the input buffer. In RT the message is routed (the output port is obtained). In VASA, the flit contends for virtual channels and for the crossbar. Finally, in X stage, the flit crosses the router. In an ICARO router, in addition, at the RT stage the counter associated to the output port requested is increased by one. Analogously, when this output port forwards a flit, this output port becomes free, so its associated counter is decreased. By doing this, we keep record of the number of ports requesting every output port, so when the requests counter reaches 2 or greater value means that the port associated to this counter is requested by two or more input ports so there is contention. Once the counter of a given port reaches the value of 2, another counter (congestion counter) associated to this port starts increasing by one at every cycle. Periodically (at a regular frequency called *polling interval*; PI), an independent module in the router checks the congestion counters for every output port. When a congestion counter reaches a given threshold (*congestion time threshold*; CTT) congestion at this port is assumed so notification is triggered for this port at the current router. Notice that the notification is sent when congestion is detected but not when contention is detected. This means notifying only important events and not transient contention situations that do not necessarily lead to congestion.

B. Congestion Notification

ICARO makes use of a simple parallel network in order to notify for congestion (Congestion Notification Network, or CNN). ICARO notifies from the routers to the end-nodes, detecting the ports suffering congestion. This notification network consists of a set of N (N=num. of routers) overlapped networks or CNN lines. Each of this BNN line consist of a simple wire from a router to all NICs (see Figure 1).

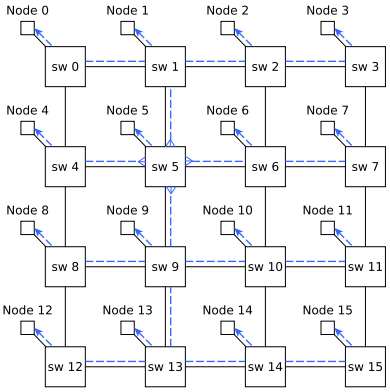


Fig. 1: CNN line for router 5

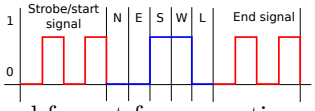


Fig. 2: Signal format for congestion notification

Every BNN line is controlled by its connected router which properly injects into the BNN line a formatted signal (explained below) in order to notify to all NICs in the network the congestion state of its ports. This signal is composed by as many bits as ports has the router. Each bit corresponds to the state of the port, corresponding the high level to *port congested* and the low level to port *non-congested*. A typical transmission of a notification from a given router would be:

1. Synchronization/start signal. A few cycles of strobe signal in order to synchronize between routers (useful for asynchronous NoCs) and to mark the beginning of a transmission.
2. p bits of data informing of the state of every port of the router.
3. A few cycles of strobe signal in order to mark the end of the transmission.

In Figure 2 an example of signal transmitted through the CNN can be seen. Transmission starts with two cycles of strobe signal (the number of cycles of strobe signal is a fixed value) that also marks the beginning of a transmission. In our example the notification informs that ports South and West are congested while the North, East and local ports are not congested. Finally, another two more cycles marks the end of the transmission.

Following this approach each router in the network will be able to inform to all NICs in the network which of its ports is congested.

Since the notification mechanism has a relative complexity it is advisable to implement a fault-tolerant mechanism to the CNN in order to deal with notification failures. When a router sends a congestion notification for a given output port, the router expects that, after a reasonable time, traffic for the congested port of such router will be received through the extra virtual network (explained latter). If too much time elapses (re-sending timeout; RST) and the congested router keeps receiving traffic to the congested port through a default VN, this router assumes that congestion notification has not been received or stored successfully by one or more nodes so the congested router will send the notification again. This process is repeated until all traffic to the congested port is received through the extra VN or congestion disappears. Note that receiving duplicated notifications has no effect for the nodes that are working correctly enqueuing traffic to the congested router port through the extra VN.

B.1 Congestion Point Storage

To decide whether a message must be reallocated into the extra virtual network or not, every end-node must know which ports of each router in the network are congested and which will be the path followed by the message to find out whether this message will be forwarded through a congested port.

We assume that most of the time only a small portion of the congested points in the network may be active simultaneously so we use a small cache memory in every end-node. This cache memory does not have a typical cache memory implementation in which providing a single value (like a memory address) a single stored value is returned. Instead, we implement a memory cache with some additional logic for storing two values in each row:

1. Router that notified congestion
2. Congested port of that router

A congestion notification is composed by the congestion state of all ports of the router. However, not all ports are reachable from a given node due to the XY routing restrictions (for avoiding dead-locks). In addition, it is expected that a router will only notify a single congested port. Therefore, congestion notifications are stored as congestion points, which is composed of a single port and the router which belongs to, each cache memory row containing a congestion point. If a router notifies more than one congested port each congested port will be stored into a different cache memory row as different congestion points.

A message will be reallocated to the extra VN depending on whether this message will cross a congestion point. The module in charge of reallocating messages when needed only knows the destination end-node of the message and only needs to know whether the message must be reallocated (or post-processed) or not, so we decided to provide the cache memory with some additional logic in order to simplify the reallocating decision process.

According to this, for the cache read operation, the destination end-node of the message is provided to the memory. The memory internally checks all rows in order to find a point that belongs to the path that will be followed by the message starting at the current router until the destination end-node. If the memory finds one or more points that belongs to that path, the memory output will be *TRUE* or *FALSE* otherwise (hit or miss).

B.2 Cache Row Allocation and Deallocation

A congestion notification is composed by the state of all ports of the router. However, not all ports are reachable so, at first, a module performs a filter to discard not-reachable ports and then the reachable congested ports are decomposed in congestion points ({router, port} pairs). If there is enough cache memory rows to store all congestion points they are stored. Otherwise, congestion points with no room

to be stored are ignored (relies in the fault-tolerant notification mechanism).

Deallocation is quite simple. When a decongestion notification arrives is decomposed in decongestion points and a matching is performed with all decongestion points and all busy rows in the cache memory and then the matching rows are deallocated.

C. Traffic Separation

In order to isolate congested traffic from the normal one ICARO makes use of virtual networks. ICARO intends one virtual network (called *extra virtual network*) for congested traffic in order to isolate it from the normal one, and the rest of virtual networks (called *default virtual networks*) are intended for normal traffic. In this way ICARO avoids the HoL-blocking produced by congested traffic. According to this, the network is provided with at least two virtual networks: one extra virtual network and one or more default virtual networks. At the end-nodes, all traffic allocated is enqueued always into a default virtual network and at every clock cycle the head of all queues are checked in parallel. If the flit at the head of a queue is a header flit (the first flit of a message), then, the destination of such message is checked to find out whether the message must be reallocated into the extra virtual network (post-processed). In order to know if a message must be post-processed the PP (Post-Processer) module (see Figure 3 checks the congestion points cache memory and if the response is true, the message is postprocessed.

C.1 Walkthrough Example

In Figure 3 we can see an overall example of ICARO taking node 8 as source node. In Figure 3a congestion notification is received from router 9 informing that the east port of such router is congested so the notification is stored in the cache memory. Next, in Figure 3b at the source node the default VN wins arbitration but, since the message destination end-node path contains a congestion point is post-processed instead of injected. Note that this transfer is performed by an independent hardware module (PP) so transferring to the extra VN and injection (if possible) of the next message in the current VN can be done in parallel. Next, in Figure 3c we can see that a notification from the east port of router 5 is received. However, due to the XY routing, that port is unreachable from node 8 so the notification is filtered out at the filter module. In addition, simultaneously the message destined to the node 13 is injected since its destination path contains no congestion point. Finally, in Figure 3d message destined to the node 14 is post-processed while the message destined to node 2 is injected through the extra VN.

IV. Evaluation

In this section we perform a preliminary evaluation of ICARO based on simulations in congestion-prone scenarios. First the simulation environment is described. Finally, results comparing ICARO-based solution against the same scenario without ICARO are presented.

A. Simulation Environment

Simulations are performed with a NoC simulator. Regarding the topology the scenario consists of 16 nodes arranged in a 4x4 2D mesh network. Since our goal is to isolate congested traffic from the normal one, 2 kinds of traffic patterns are used. In one hand a background uniform traffic is injected at a low data-rate (normal traffic). On the other hand realistic MCSL traffic pattern is used in order to simulate congestion-prone traffic. Therefore, for the simulation results both traffic latencies are separated in the graphs in order to appreciate the latency improvement in the normal traffic (synthetic traffic). More details about the characteristics of the simulation are shown in Table I.

B. ICARO vs no-ICARO Analysis

For running ICARO simulations some parameters are required in order to get it work: polling interval (PI), congestion time threshold (CTT), notification delay (ND), re-sending timeout (RST) and cache size. All paramaters but notification delay (which is a parameter given by the network hardware) are configurable parameters which determine the ICARO behaviour and may impact in the improvement achieved. These must be subject of study but, since this is a preliminary analysis we set these parameters to reasonable values (described in Table I), leaving a more deep analysis of such parameters for future work.

The left graphs of Figures 4 and 5 show the latency for both MCSL and synthetic traffic without ICARO and in the right ones we see the traffic latency for both kind of traffics with ICARO. As can be seen, with ICARO graphs show how latency is reduced drastically since the HoL-blocking caused by MCSL traffic is isolated into the extra-VN so the synthetic traffic is not harmed by congestion achieving an average latency reduction of 88%.

V. Conclusions

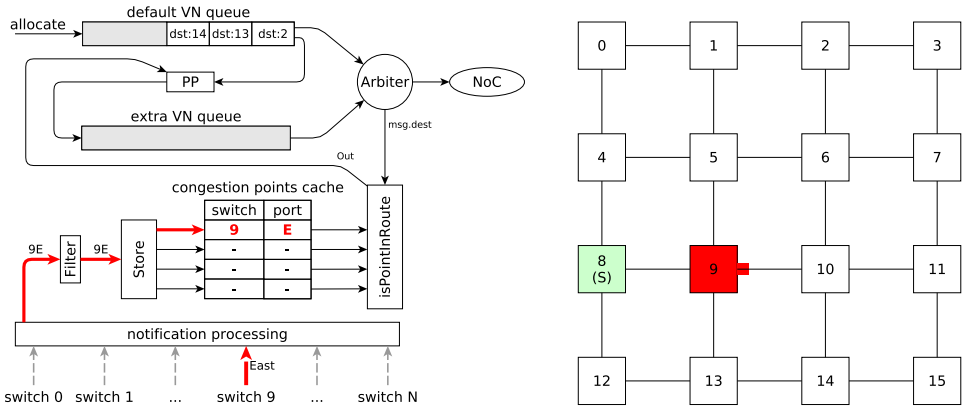
In this paper we have proposed ICARO, a mechanism for isolating congested traffic in order to avoid HoL-blocking. This mechanism has been evaluated and simulations with realistic traffic in combination with synthetic traffic shows that ICARO isolates harmful traffic successfully into a dedicated virtual network achieving a latency performance up to 88% over the normal traffic and also improving significantly the performance over the congested traffic.

VI. Future work

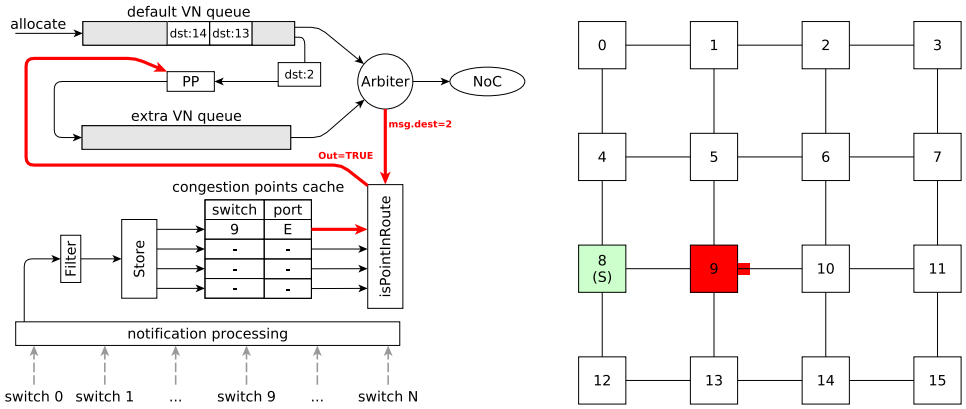
Since ICARO implies new additional hardware at the network interfaces, the routers and an additional dedicated network, we plan to perform an analysis of area and power overhead. In addition, is convenient to perform simulations in order to analize the ICARO behaviour varying its parameters (PI, CTT and RST) as well as parameters concerning the chip manufacturation (ND).

Acknowledgments

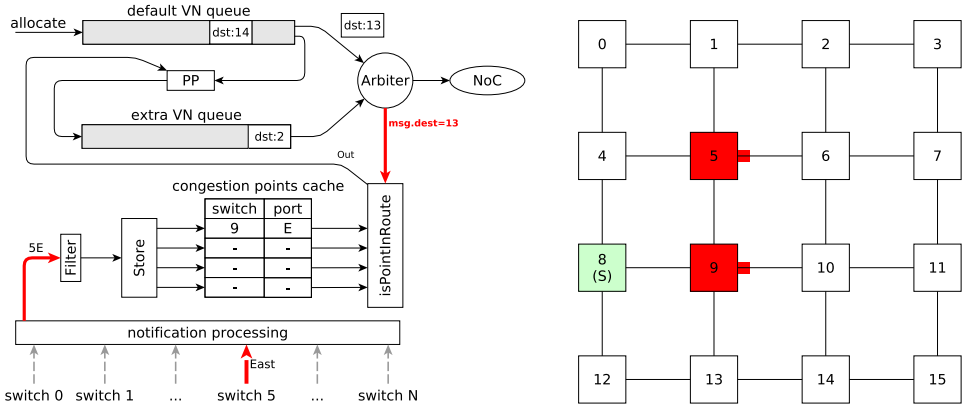
This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and Plan E funds under Grants TIN2009-M475-C04-01 and TIN2012-38341-C04-01.



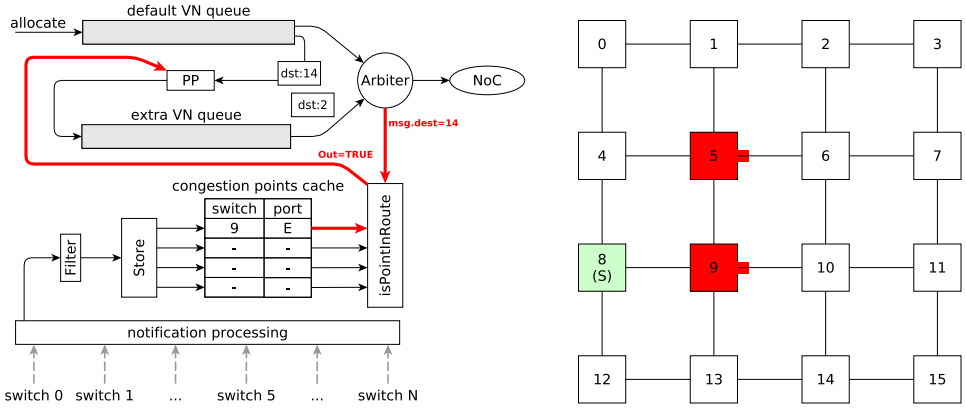
(a) Storing received congestion notification from port east of router 9.



(b) Message to node 2 is post-processed.



(c) Ignoring received congestion notification from port east of router 5 and message to node 13 is injected.



(d) Message to node 14 is post-processed and message to node 2 is injected.

Fig. 3: ICARO example

TABLE I: Scenario configuration.

Topology	4x4 2D regular mesh	
Virtual networks	no ICARO	4VNs
	ICARO	3 default-VN(s) + 1 extra-VN
Routing, switching and flow control	XY, Wormhole and Stop&go	
Flit, message and queue size	4 bytes, 10 flits and 16 flits	
Traffic pattern	h264 video enc. + uniform 0.1 flits/cycle/nic	
ICARO configuration		
PI, CTT, ND, RST and cache size	1000 cycles, 300 cycles, 300 cycles, 4 rows	

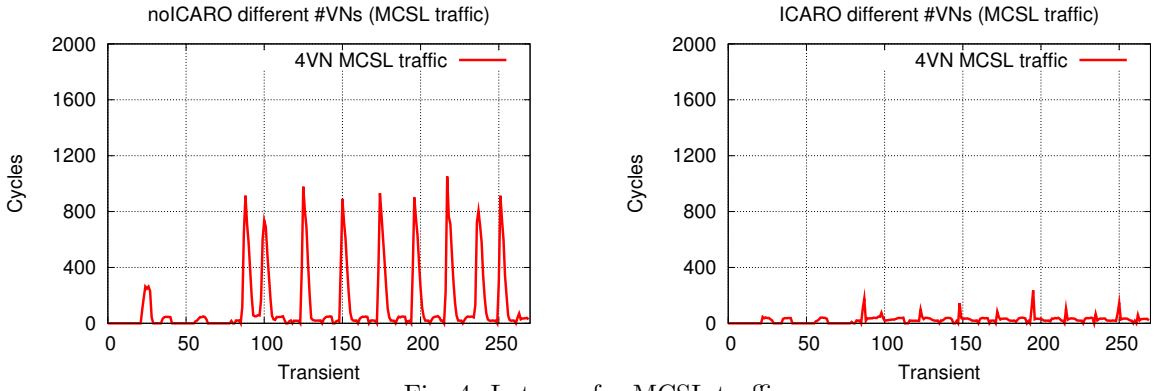


Fig. 4: Latency for MCSL traffic.

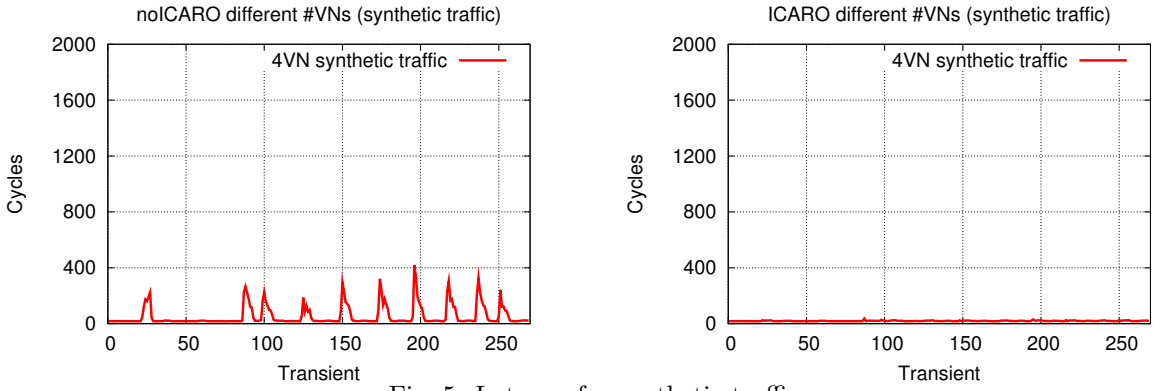


Fig. 5: Latency for synthetic traffic.

References

[1] Intel Corp., "Xeon phi," Available at `http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner?wapkw=knight+corner`.

[2] Intel Corp., "The single-chip cloud computer," Available at `http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html`.

[3] Tilera Corp., "Tilera tile multicore processors," Available at `http://www.tilera.com/products/processors/TILE-Gx_Family`.

[4] P. Gupta and Fook-Luen Heng, "Toward a systematic-variation aware timing methodology," in *Design Automation Conference, 2004. Proceedings. 41st*, 2004, pp. 321–326.

[5] Rakesh Kumar, Dean M. Tullsen, Norman P. Jouppi, and Parthasarathy Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, Nov. 2005.

[6] Seung Eun Lee, Yong Zhang, Zhen Fang, S. Srinivasan, R. Iyer, and D. Newell, "Accelerating mobile augmented reality on a handheld platform," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, 2009, pp. 419–426.

[7] Bin Li, Zhen Fang, and R. Iyer, "Template-based memory access engine for accelerators in socs," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, 2011, pp. 147–153.

[8] A.K. Mishra, N. Vijaykrishnan, and C.R. Das, "A case for heterogeneous on-chip interconnects for CMPs," in *Proceeding of the 38th annual international symposium on Computer architecture*. 2011, pp. 389–400, ACM.

[9] S.S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The alpha 21364 network architecture," in *Hot Interconnects 9, 2001.*, 2001, pp. 113 –117.

[10] Xi Chen, Zheng Xu, Hyungjun Kim, P. Gratz, Jiang Hu, M. Kishinevsky, and U. Ogras, "In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, may 2012, pp. 43 –50.

[11] Guihai Yan, Yingmin Li, Yinhe Han, Xiaowei Li, Minyi Guo, and Xiaoyao Liang, "Agileregulator: A hybrid voltage regulator scheme redeeming dark silicon for power efficiency in a multicore architecture," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, feb. 2012, pp. 1 –12.

[12] P.J. Garcia, F.J. Quiles, J. Flich, J. Duato, I. Johnson, and F. Naven, "Efficient, scalable congestion management for interconnection networks," *Micro, IEEE*, vol. 26, no. 5, pp. 52 –66, sept.-oct. 2006.

[13] Paul Gratz, Boris Grot, and Stephen W Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *HPCA*. 2008, pp. 203–214, IEEE Computer Society.

[14] Xin Chang, M. Ebrahimi, M. Daneshtalab, T. Westerlund, and J. Plosila, "Pars #x2014; an efficient congestion-aware routing method for networks-on-chip," in *Computer Architecture and Digital Systems (CADS), 2012 16th CSI International Symposium on*, 2012, pp. 166–171.

[15] U.Y. Ogras and R. Marculescu, "Prediction-based flow control for network-on-chip traffic," in *Design Automation Conference, 2006 43rd ACM/IEEE*, 2006, pp. 839–844.

Políticas para el Controlador de Memoria en Sistemas Multinúcleo de Tiempo Real

José Luis March, Salvador Petit, Julio Sahuquillo, Houcine Hassan y José Duato [1]

Resumen— **Los CMPs (chip multiprocessors) se están convirtiendo en la opción más común para implementar sistemas empotrados ya que consiguen una buena relación entre consumo y prestaciones. Debido a razones de fabricación, los CMPs suelen implementar uno o varios controladores de memoria, cada uno compartido por un conjunto de núcleos. Por tanto, las peticiones de memoria de los distintos núcleos compiten entre ellas para acceder a memoria. Esto significa que la latencia de acceso a memoria puede variar en función de las otras peticiones y de la política de planificación del controlador, produciendo un comportamiento impredecible. Este trabajo se centra en el diseño de un controlador de memoria para soportar cargas de tiempo real estricto (Hard Real-Time, o HRT) y flexible (Soft Real-Time, o SRT). Estos sistemas deben garantizar la ejecución de las aplicaciones HRT y mejorar las prestaciones de las SRT.**

En este artículo proponemos dos políticas para un controlador de memoria en un sistema empotrado multinúcleo: *HR-first* y *ATR-first*. La primera prioriza las peticiones de memoria de las tareas HRT, consiguiendo un notable ahorro energético pero con pobres prestaciones para las aplicaciones SRT. La segunda prioriza sólo aquellas peticiones de las tareas HRT que son críticas para garantizar la planificabilidad. Los resultados muestran que *ATR-first* presenta un consumo energético similar al de *HR-first*, mientras que consigue reducir el número de deadlines perdidos de las aplicaciones SRT en un 49%, de media, y consiguiendo el cumplimiento de todos los deadlines en algunos escenarios.

Palabras clave— **Multinúcleo, Consumo, Controlador de Memoria, Calidad de Servicio, Tiempo Real**

I. Introducción

En los últimos años la capacidad de cómputo y las aplicaciones en los sistemas empotrados han aumentado de forma exponencial, sin embargo, sus restricciones energéticas no han variado prácticamente. Para solucionar este desequilibrio se están implementando CMPs (chip multiprocessors) en los sistemas empotrados comerciales debido a su excelente ratio prestaciones/vatio.

El mercado actual requiere tareas de tiempo real estricto (*Hard Real-Time, o HRT*) y flexible (*Soft Real-Time, o SRT*). Como en sistemas empotrados las tareas HRT se usan para modelar aplicaciones críticas, a las tareas SRT se les asigna una prioridad más baja debido a razones de seguridad y corrección. Sin embargo, un requisito fundamental para penetrar en el mercado de sistemas como smartphones o tablets es no sólo asegurar un correcto funcionamiento sino también el mayor rendimiento para aplicaciones SRT como el video streaming.

Por otra parte, debido a restricciones de fabricación, las cuales se acentúan en sistemas empotrados, los controladores de memoria en un CMP se comparten entre un conjunto de núcleos de cómputo. En este contexto, los núcleos que comparten el mismo controlador de memoria se dice que pertencecen al mismo *dominio de memoria*. Las aplicaciones que se ejecutan en el mismo *dominio de memoria* pueden sufrir contención al acceder al controlador de memoria compartido, lo que produce un comportamiento impredecible. Por tanto, las políticas del controlador de memoria se deben diseñar especialmente para tener en cuenta este aspecto.

Con tal de evitar un comportamiento impredecible en la ejecución de peticiones de memoria de aplicaciones críticas, no son apropiadas políticas convencionales usadas en sistemas que no son de tiempo real (p.e., FIFO), ya que se introduciría variabilidad en las latencias de las peticiones de memoria, lo que podría causar pérdida de deadlines HRT. Un deadline HRT perdido podría conllevar fallos críticos en el sistema de tiempo real. Para evitar este problema, en este artículo se estudia una política de planificación simple, llamada *HR-first*, que prioriza peticiones de tareas HRT sobre el resto, lo que incrementa el número de deadlines SRT perdidos. Esto afecta negativamente a la calidad de servicio (*Quality of Service, o QoS*) de las tareas no críticas.

Para mejorar las prestaciones de las tareas no críticas, en este artículo proponemos una política para el controlador de memoria, denominada *ATR-first*, que asegura la planificabilidad EDF (*Earliest Deadline First*) de las tareas HRT sin sacrificar la QoS de las aplicaciones no críticas. La propuesta prioriza sólo aquellas peticiones de tareas HRT que son críticas para cumplir la planificabilidad de tiempo real. Para algoritmos de planificación dinámica, como EDF, este conjunto de tareas varía durante la ejecución y depende de la carga que se esté ejecutando así como de la potencia de cómputo del sistema (i.e., número de núcleos e hilos hardware).

Los resultados muestran que la propuesta presenta un consumo de energía similar al de *HR-first*. Sin embargo, reduce los deadlines SRT perdidos con respecto a *HR-first* en todos los casos estudiados. De media, esta disminución es alrededor de un 49%. Además, en algunos casos el sistema consigue cumplir todos los deadlines usando *ATR-first*.

El resto del artículo se organiza como sigue. La Sección II presenta el sistema, la carga de trabajo y el planificador EDF con control de consumo. En la Sección III se expone la nueva arquitectura para el controlador de memoria. La Sección IV discute los resultados experimentales. Finalmente, la Sección V presenta algunas breves conclusiones.

[1]Departamento de Informática de Sistemas y Computadores, e-mail: jomarcab@gap.upv.es, {spetit,jsahuqui,husein,jduato}@disca.upv.es

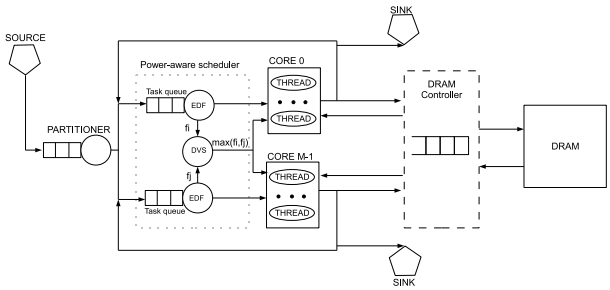


Fig. 1. Modeled system.

II. Modelo del Sistema

La Figura 1 muestra un diagrama con el sistema modelado. Cuando una tarea llega, un módulo particionador la asigna a una cola asociada a un núcleo, que contiene tareas listas para ejecutarse en ese núcleo. Estas colas son componentes de un planificador con control de consumo (ver Sección II-C) que controla un regulador DVFS global [1]. En este esquema, el planificador se encarga de ajustar la frecuencia de los núcleos para satisfacer los requisitos de la carga.

Los núcleos de procesamiento, modelados como un ARM11 MPCore [2], implementan el paradigma multihilo de grano grueso, el cual cambia el hilo en ejecución cuando ocurre un evento de larga latencia (i.e., un acceso a memoria principal). En ese caso, un nuevo hilo toma el control del procesador mientras el otro está accediendo a memoria, solapando la ejecución. De esta forma, los recursos de cómputo se asignan al hilo que ejecuta la tarea con la máxima prioridad (la cual cambia de acuerdo con el algoritmo de planificación). Si este hilo se detiene debido a un evento de memoria de larga latencia, entonces estos recursos se reasignan al hilo con la prioridad más alta que no esté esperando para acceder a memoria, hasta que el evento de memoria de larga latencia se resuelva.

A. Tareas de Tiempo Real

El sistema ejecuta tareas periódicas HRT y SRT. Una tarea está preparada para ejecutarse al principio de cada periodo activo, y debe terminar su ejecución antes del deadline. El final del periodo y el deadline de una tarea se asume que son iguales para una planificación más simple. También hay algunos periodos en los que las tareas no se ejecutan ya que no están activas (i.e., periodos inactivos). En resumen, una tarea llega al sistema, se ejecuta durante varios periodos activos consecutivos, abandona el sistema, permanece fuera del sistema durante algunos periodos inactivos, y luego vuelve a entrar en el sistema.

A parte del periodo y el deadline, una tarea tambien se caracteriza por su WCET (Worst Case Execution Time). El WCET se usa para obtener la utilización de una tarea dada como $U = WCET/Period\_Length$. La utilización de las tareas se usa tanto en el módulo particionador y como en el planificador para comprobar si un conjunto de tareas es planificable o no.

B. Particionador

Existen diversas heurísticas de particionado que se pueden usar para distribuir las tareas conforme llegan al sistema. La heurística Worst Fit (WF) se ha implementado en el módulo particionador porque está considerada como una de las mejores opciones para equilibrar la carga [3]. Esta heurística equilibra la carga al asignar cada tarea entrante al núcleo menos cargado. Si varias tareas llegan a la vez, WF las ordena de forma decreciente según su utilización y las asigna a los núcleos empezando por la tarea con más utilización.

C. Planificador con Control de Consumo

Una vez el particionador asigna una tarea a un núcleo, ésta se inserta en la cola de tareas de dicho núcleo, donde se prioriza según las siguientes reglas: i) las tareas del mismo tipo (i.e., HRT o SRT) se ordenan de acuerdo a EDF [4], que da la mayor prioridad a las tareas con el deadline más cercano, y ii) las tareas HRT tienen siempre más prioridad que las SRT, independientemente de los deadlines. Por lo tanto, si una nueva tarea HRT llega al sistema, tendrá más prioridad que cualquier tarea SRT. Finalmente, las tareas con la prioridad más alta serán las que se mapeen en los hilos hardware de cada núcleo.

El planificador calcula la velocidad requerida por cada núcleo. Con tal de minimizar el consumo, cada núcleo requiere la frecuencia mínima que cumple las restricciones temporales de su carga (i.e., los deadlines), de acuerdo a la planificabilidad EDF. El regulador DVFS global [5] selecciona la más alta de entre todas las requeridas por los núcleos. Nótese que el planificador sólo debe garantizar los deadlines de las tareas HRT. Por consiguiente, el conjunto de tareas considerado para la estimación de la frecuencia debe incluir cómo mínimo todas las tareas HRT.

En este sentido, este trabajo propone dos políticas de planificación: *H-mode* y *H+S-mode*. La primera considera sólo tareas HRT para la estimación de la velocidad de la máquina. La segunda considera tanto las tareas HRT como las SRT. Las frecuencia del sistema se recalcula sólo cuando la carga cambia, es decir, cuando una tarea llega y/o sale del sistema. En el primer caso, se puede requerir una frecuencia más alta debido a que la carga aumenta. En el último caso, una frecuencia menor podría satisfacer los deadlines de las tareas restantes.

Se han considerado diferentes valores de frecuencia/voltaje para el planificador con control de consumo, basados en los niveles de un Pentium M [6], que se muestran en la Tabla I. Este trabajo evalúa un DVFS con 8, 4 y 2 niveles. La configuración de 8 niveles (8L) permite al sistema trabajar a todos los niveles indicados en la tabla, mientras que la de 4 niveles (4L) sólo permite trabajar a 1700, 1400, 1100 y 600 MHz. La última configuración (2L) sólo soporta la frecuencia más alta y la más baja (i.e., 600 y 1700 MHz). Además, la latencia de los cambios de frecuencia se ha modelado de acuerdo a un ratio de transición de voltaje de $1mv/1\mu s$ [7].

TABLA I
FREQUENCY (F), VOLTAGE (V) AND POWER (P).

F[MHz]	1700	1500	1400	1300	1200	1100	900	600
V[Volts]	1,48	1,48	1,48	1,39	1,18	1,18	1	0,96
P[Watts]	24,5	24,5	22	22	12	12	7	6

III. CONTROLADOR DE MEMORIA

El controlador de memoria maneja peticiones de memoria de todos los hilos del sistema. El controlador de memoria modelado incluye características avanzadas como *load forwarding* y *load bypassing*. Es decir, si la dirección de una load de un hilo dado coincide con la dirección de una store previa, los datos requeridos se pasan de la store a la load. Además, si la dirección de ninguna store previa coincide con la dirección de la load, entonces la load puede adelantar a las stores. El controlador también da prioridad a las loads frente a las stores, ya que las stores no paran la actividad del procesador, así que largas latencias de stores no afectan a la planificabilidad.

Para soportar prioridades entre peticiones de diferentes hilos, como en [8], se han añadido modificaciones en el controlador de memoria. Esto era necesario ya que una política de planificación FIFO simple causaría la pérdida de deadlines HRT, ya que introduciría una variabilidad imprevista en las latencias de las loads. A parte de la propuesta, *ATR-first*, también se ha implementado una política más intuitiva, *HR-first*, con fines comparativos.

A. HRT Requests First

En esta política, el controlador de memoria se configura para priorizar las peticiones de las tareas HRT sobre las de las tareas SRT. Peticiones del mismo tipo de tareas (HRT o SRT) se ordenan de acuerdo al algoritmo EDF. Por ejemplo, si llega una nueva petición de memoria de una tarea HRT, será lanzada antes que cualquier otra petición anterior de una tarea SRT pero después de otras peticiones de tareas HRT con mayor prioridad.

Un ejemplo de esta política se muestra en la Figura 2. En esta figura el color de fondo de la petición distingue entre peticiones de tareas HRT y SRT. Las peticiones se etiquetan en la cola en función de la operación de memoria (load o store) y de su prioridad relativa (1 es la máxima prioridad). Las loads tienen más prioridad que las stores. Entre las loads, aquellas pertenecientes a tareas HRT tienen prioridad respecto a las pertenecientes a tareas SRT. Respecto a las stores, se lanzan después de todas las loads, y las stores de las tareas HRT se lanzan primero. Por tanto, la store más antigua en la cola (prioridad 6) es la última petición lanzada a memoria ya que es una store de una tarea SRT.

B. Active Task Requests First

El orden de prioridades usado en *HR-first* prioriza loads de tareas HRT. Asumiendo planificación EDF, esta condición es suficiente para asegurar la planificabilidad de las tareas HRT, pero no es necesaria. La condición necesaria es que la petición HRT con el deadline más cercano tenga la máxima prioridad. De aquí en adelante nos referiremos a la tarea con el deadline más cercano de una cola de tareas como la tarea *activa*. Como se ha explicado en la Sección II-C, esta tarea está mapeada en uno de los hilos del núcleo asociado con la cola de tareas.

Por tanto, las peticiones de las tareas SRT se pueden lanzar antes que las peticiones de tareas HRT que no son tarea *activa*, mejorando la QoS. En este sentido, *ATR-first* lanza primero peticiones HRT *activas*, luego peticiones SRT (*activas* y *no activas*), y finalmente peticiones del resto de tareas HRT (*no activas*). Al igual que en *HR-first*, las loads tienen prioridad sobre las stores, y las peticiones del mismo tipo (HRT *activas*, SRT *activas*, SRT *no activas* y HRT *no activas*) se ordenan según EDF.

En resumen, las peticiones de memoria se pueden clasificar en dos grupos en función de si la tarea que las origina es *activa* o no. Por ejemplo, las loads de tareas *activas* tienen la prioridad más alta (1 y 2) en la Figura 3, y entre ellas, la load de la tarea HRT *activa* tiene la máxima prioridad. Por contra, la prioridad de las loads de tareas no *activas* se invierte. Es decir, la load de la tarea SRT no *activa* tiene más prioridad (3) que la load de la tarea HRT no *activa* (4). Finalmente, en este ejemplo sólo hay dos stores, una de una tarea SRT no *activa* y otra de una tarea HRT no *activa*. En este caso, la store de la tarea SRT no *activa* tiene mayor prioridad (5).

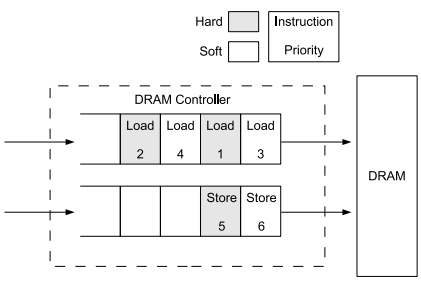


Fig. 2. HR-first request scheduling policy.

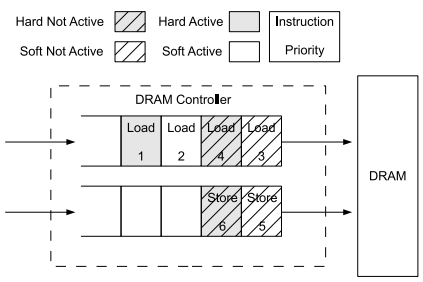
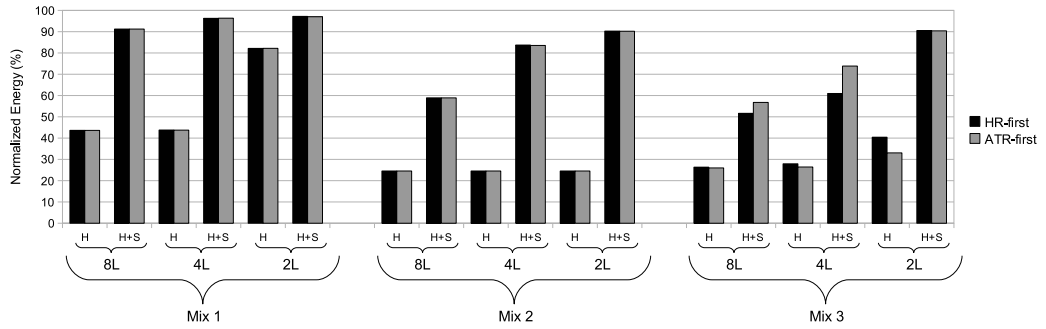
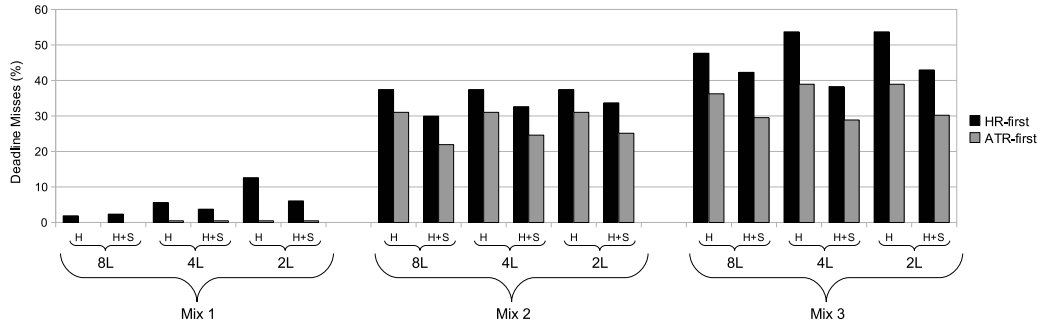


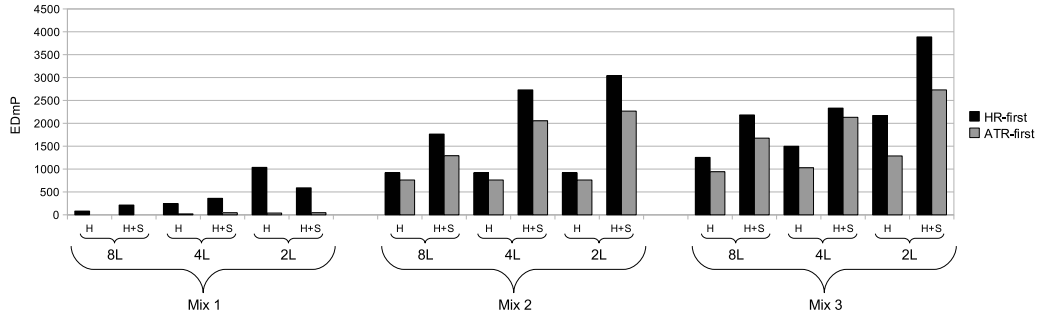
Fig. 3. ATR-first request scheduling policy.



(a) Normalized energy



(b) Deadline misses



(c) Energy per deadline misses

Fig. 4. Normalized energy and deadline misses for 2 cores.

IV. Resultados Experimentales

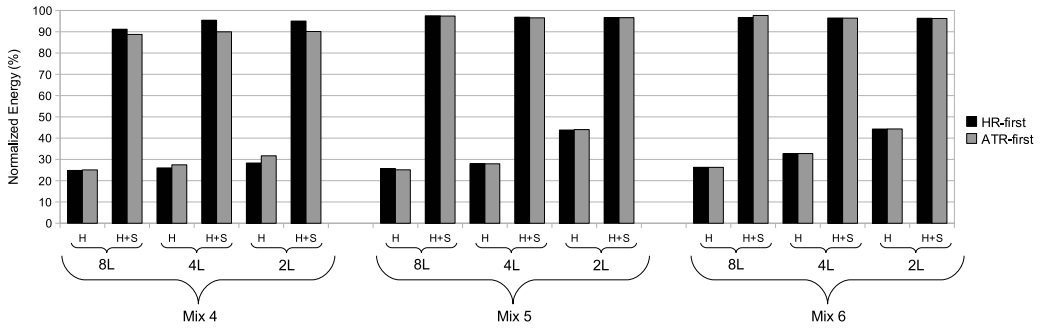
Para evaluar la propuesta se ha utilizado Multi2Sim [9], un simulador detallado dirigido por ejecución ciclo-a-ciclo para evaluar procesadores multinúcleo multihilo. Ha sido ampliamente modificado para modelar el módulo particionador, el planificador con control de consumo, el regulador DVFS y el controlador de memoria. Se ha evaluado un sistema con dos y cuatro núcleos, soportando tres hilos hardware en cada núcleo. Cada núcleo ejecuta las instrucciones en orden, con ancho de Issue 2 y una latencia de acceso a memoria de 34 ciclos.

Para preparar las mezclas de tiempo real se han usado benchmarks de [10]. El número de tareas ejecutadas en un sistema con 2 núcleos (mezclas 1, 2 y 3) varía entre 7 y 15, mientras que en un sistema con 4 núcleos (mezclas 4, 5 y 6) varía de 20 a 31. En el diseño de las mezclas se han considerado aspectos como la utilización de las tareas, su periodo, y la secuencia de periodos activos e inactivos. Para todas las mezclas, hay un subconjunto de tareas que se consideran tareas HRT. A la máxima velocidad, la utilización acumulada de este subconjunto varía de un 50% a un 95%. Como la tarea HRT con el deadline más cercano tiene la prioridad más alta, el conjunto de tareas HRT es planificable por el planificador EDF. Sin embargo, como la utilización total de la mezcla es superior al 100%, habrá tareas SRT que perderán su deadline. El diseño de mezclas nos permite comparar diferentes políticas para el controlador de memoria en término de número de deadlines perdidos.

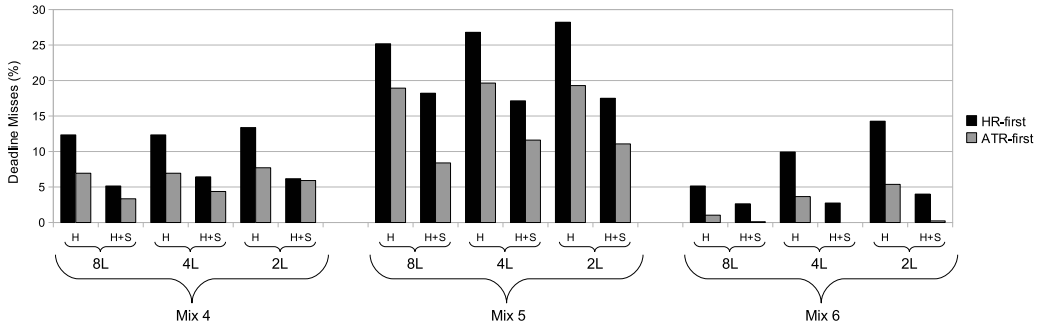
Los resultados experimentales están expresados en consumo energético normalizado y deadlines SRT perdidos. Para calcluar el consumo, se multiplica el número de ciclos trabajando a cada frecuencia por la energía requerida por cada ciclo a dicha frecuencia. Entonces, este valor se normaliza usando como base la energía consumida por el sistema trabajando siempre a la máxima velocidad.

Las Figuras 4(a) y 4(b) muestran, para diferentes mezclas, el consumo normalizado y los deadlines SRT perdidos de un sistema con dos núcleos variando el número de niveles DVFS (*2L*, *4L*, and *8L*), el conjunto de tareas considerado para el cálculo de la frecuencia (*H-mode* y *H+S-mode*), y la política del controlador de memoria (*HRT-first* y *ATR-first*).

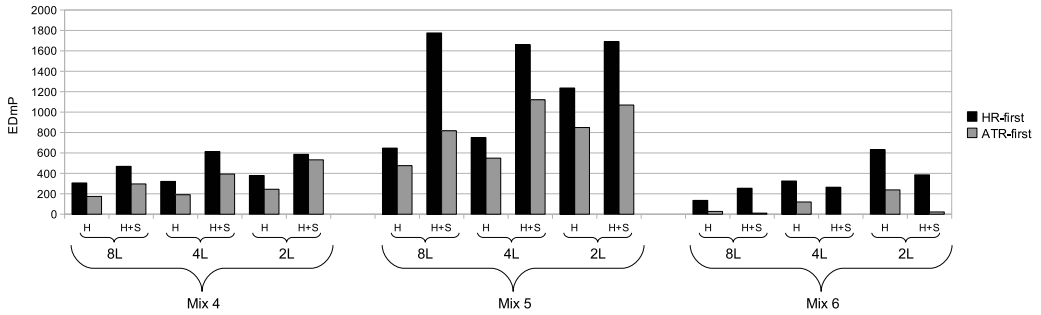
Como se observa, *ATR-first* no provoca un sobre-



(a) Normalized energy



(b) Deadline misses



(c) Energy per deadline misses

Fig. 5. Normalized energy and deadline misses for 4 cores.

coste energético importante. Es decir, la energía consumida por *ATR-first* es similar a la consumida por *HR-first*, independientemente del conjunto de tareas usado para calcular el nivel DVFS y del número de estos niveles. De hecho, la media de la diferencia entre ambas políticas es sólo un 1%. Se puede apreciar un consumo mayor en la mezcla 3, donde el consumo aumenta un 17% con 4 niveles DVFS con *H+S-mode*. No obstante, para 2 niveles DVFS con *H-mode*, *ATR-first* reduce el consumo de energía en un 22%.

Nótese que el número de niveles DVFS tiene un impacto muy fuerte en el consumo. En este sentido, los mejores resultados se obtienen cuando se trabaja con el mayor número de niveles DVFS, ya que esto permite al planificador ajustar mejor la velocidad del sistema a las necesidades de la carga. Por ejemplo, si el conjunto de tareas requiere una frecuencia de al menos 150 MHz para ser planificable, en la configuración de DVFS 3L el sistema seleccionará la frecuencia de 300 MHz, mientras que en la configuración de 5L la frecuencia elegida será de 200 MHz, con lo que este último caso ahorra más energía.

Este efecto es más acusado cuando el sistema no está ni sobrecargado (la mayor parte del tiempo trabajando a la máxima frecuencia) ni infrautilizado (la mayor parte del tiempo trabajando a la mínima frecuencia). Esto último puede pasar con *H-mode*, mientras lo primero con el *H+S-mode*. Hay que notar que *H-mode* siempre consume menos energía que *H+S-mode*, ya que se tienen en cuenta menos tareas para el cálculo de la frecuencia. Los resultados muestran que trabajando con ocho niveles de frecuencia/voltaje puede reducir el consumo en un 43%, comparado con trabajar con dos niveles.

Respecto a los deadlines perdidos, *ATR-first* consigue importantes beneficios para cualquier modo de cálculo de frecuencia y configuración DVFS. Esta propuesta reduce en un 48% la pérdida de deadlines SRT, de media, cumpliendo con todos los deadlines en algunos casos (mezcla 1 con sistema 8L). Además, a pesar que elegir *H+S-mode* consume más energía, se pierden menos deadlines que con *H-mode* para todas las mezclas evaluadas.

Un apunte interesante es que *ATR-first* puede permitir tanto la reducción de deadlines perdidos como del consumo energético, cuando se combina adecuadamente con un modo de cálculo de frecuencia y un número dado de niveles DVFS. Por ejemplo,

ATR-first pierde un 31% de deadlines y consume un 25% de la energía consumida por el sistema base para la mezcla 2 en un sistema 4L con *H-mode*, mientras que para la misma mezcla y número de frecuencias usando *HR-first* y *H+S-mode* se obtiene un 33% de deadlines perdidos y un 84% de consumo. El primer caso también consigue mejores resultados que *HR-first* en la misma mezcla en un sistema 2L con *H+S-mode*, que obtiene un 34% de deadlines perdidos y un 90% de consumo energético.

Con tal de evaluar la *eficiencia* del diseño en términos de energía y QoS proponemos la métrica *Energy Deadline misses Product (EDmP)* como el producto de la energía consumida relativa y los deadlines perdidos relativos. Esta métrica tiene sentido cuando el consumo es más o menos parecido, y proporciona indicios sobre cuál es la mejor estrategia de planificación. En otras palabras, cuando se evalúa el *EDmP* hay un límite superior en la QoS que no se debería superar. Por ejemplo, una reducción muy grande en el consumo no sería aceptable con un 90% de deadlines perdidos relativos en una batería con una carga normal. *EDmP* es similar a *Energy Delay Product (EDP)*, ampliamente usado para identificar el mejor diseño hardware.

La Figura 4(c) muestra los resultados de *EDmP* para los planificadores evaluados. Nótese que el máximo porcentaje de deadlines perdidos es alrededor de un 54%, en algunos casos cuando se aplica *HR-first*. Como se aprecia, esta métrica claramente identifica la política *ATR-first* como el mejor planificador con una notable diferencia respecto a *HR-first*.

La Figura 5(a) muestra el consumo normalizado y la Figura 5(b) los deadlines perdidos en un sistema con cuatro núcleos. Como se observa, los valores energéticos correspondientes a *HR-first* and *ATR-first* también son muy cercanos, con una diferencia media de un 0,4%. Las diferencias de energía vienen de nuevo por el modo de velocidad, según si se consideran las tareas SRT o no, y por el número de niveles DVFS. Nótese que la métrica *EDmP* en un sistema con 4 núcleos (Figure 5(c)) también resalta los beneficios obtenidos por *ATR-first* en cualquier caso.

Finalmente, *ATR-first* reduce los deadlines perdidos en un 50%, de media, consiguiendo de nuevo el cumplimiento de todos los deadlines en algunos escenarios, como en la mezcla 6 en un sistema 4L con *H+S-mode*. Por tanto, se puede afirmar que los resultados obtenidos por la propuesta siguen la misma tendencia que en un sistema con 2 núcleos.

V. Conclusiones

El diseño de sistemas empotrados multinúcleo con restricciones de tiempo real tiene tres objetivos principales: i) reducir el consumo energético, ii) asegurar el cumplimiento de los deadlines de tareas HRT, y iii) mejorar la QoS de las tareas SRT. Con ese fin, se tienen que considerar las latencias de memorias. Un componente clave para manejar estas latencias es el controlador de memoria, que maneja peticiones de acceso a memoria de las diferentes tareas.

Este artículo ha propuesto la política *ATR-first* para planificar las peticiones de memoria en el controlador de memoria. Esta política relaja las restricciones de *HR-first*, una política más intuitiva que prioriza siempre las peticiones de las tareas HRT sobre las SRT. Por contra, *ATR-first* prioriza sólo las peticiones de tareas HRT que son cruciales para conseguir la planificabilidad de las tareas. Para el resto, las peticiones SRT tienen prioridad. Esto mejora la QoS de las tareas SRT a la vez que asegura el cumplimiento de los deadlines de las tareas HRT.

Los resultados muestran que usando la propuesta *ATR-first* se consume aproximadamente la misma energía que con la política *HR-first*. Además, *ATR-first* siempre reduce el número de deadlines perdidos de tareas SRT con respecto a *HR-first*. Este descenso es de media alrededor de un 49% y, en algunos casos, permite al sistema cumplir con todos los deadlines.

Finalmente, remarcar que la política *ATR-first* es la más eficiente energéticamente en todos los experimentos, como demuestra la métrica *Energy Deadline misses Product*. A nuestro entender, esta es la primera métrica propuesta que mide la relación entre consumo energético y deadlines perdidos.

Agradecimientos

Trabajo financiado por el Ministerio de Economía y Competitividad (MINECO) y los fondos del Plan E (TIN2009-14475-C04-01 y TIN2012-38341-C04-01).

Referencias

[1] C. Hung, J. Chen, and T. Kuo, "Energy-Efficient Real-Time Task Scheduling for a DVS System with a Non-DVS Processing Element," in *Proceedings of the 27th RTSS*, Rio de Janeiro, Brazil, 5-8 December 2006, pp. 303–312.

[2] K. Hirata and J. Goodacre, "ARM MPCore; The streamlined and scalable ARM11 processor core," in *Proceedings of the ASP-DAC*, Yokohama, Japan, January 2007, pp. 747–748.

[3] T. A. AlEnawy and H. Aydin, "Energy-Aware Task Allocation for Rate Monotonic Scheduling," in *Proceedings of the 11th RTAS*, San Francisco, USA, 2005, pp. 213–223.

[4] T. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis," in *Proceedings of the 24th RTSS*, Cancun, Mexico, 3-5 December 2003, pp. 120–129.

[5] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," in *Proc. of the 33rd ISCA*, Boston, USA, 2006, pp. 78–88.

[6] INTEL Corp., Santa Clara, CA, USA, *Intel Pentium M Processor Datasheet*, 2004, [Online]. Available: `http://download.intel.com/support/processors/mobile/pm/sb/25261203.pdf`.

[7] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks, "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," in *Proceedings of the 38th IEEE/ACM MICRO*, Barcelona, Spain, 12-16 November 2005, pp. 271–282.

[8] Benny Akesson, Kees Goossens, and Markus Ringhofer, "Predator: a predictable sdram memory controller," in *Proceedings of the 5th CODES+ISSS*, Salzburg, Austria, 2007, pp. 251–256.

[9] R. Ubal, J. Sahuquillo, S. Petit, and P. López, "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," in *Proceedings of the 19th SBAC-PAD*, Gramado, RS, Brazil, 24-27 October 2007, pp. 62–68.

[10] Malardalen Real-Time Research Center, Vasteras, Sweden, *WCET Analysis Project. WCET Benchmark Programs*, 2006, [Online]. Available: `http://www.mrtc.mdh.se/projects/wcet/`.

Reserva de circuitos para tráfico reactivo en CMPs homogéneos

Marta Ortín Obón[1], Darío Suárez Gracia[1], María Villarroya Gaudó[1], Cruz Izu[2] y Víctor Viñals Yúfera[1]

Resumen— **La red de interconexión es responsable de comunicar todos los nodos del chip y tiene un impacto directo en el rendimiento global del sistema, el área y la energía consumida. Para lograr un buen diseño, se deben tener en cuenta las características de todos los elementos de manera integral. Diseñar la red al mismo tiempo que se analizan el subsistema de memoria y las cargas de trabajo permite conocer el tipo de mensajes que necesitaremos transmitir y optimizar la red para los casos más frecuentes. En este trabajo, nos centramos en multiprocesadores en chip de memoria compartida, en concreto, estudiamos mallas de 16 y 64 nodos. Observamos que la mayor parte del tráfico es de tipo petición-respuesta, lo cual nos permite conocer con antelación qué caminos se utilizarán en muchos casos y reservar los recursos antes de enviar los mensajes. Se ha optado por una implementación muy simple que permite reducir el área del router en un 8% y disminuir la energía consumida en un 27%, al mismo tiempo que se obtiene una ganancia del 2% en el rendimiento global del sistema, para un chip de 64 procesadores.**

Palabras clave— **Redes de interconexión, multiprocesadores en chip, simulación de sistema completo.**

I. Introducción

ACTUALMENTE, un solo chip puede contener múltiples procesadores y una gran cantidad de memoria. Una tendencia muy popular consiste en conectar varios nodos iguales, cada uno con un procesador y uno o varios niveles de memoria cache privada o compartida. Los nodos se comunican mediante una red de interconexión que tiene un gran impacto en el rendimiento, el consumo energético y el área del chip. En este trabajo nos centramos en multiprocesadores homogéneos con 16 y 64 procesadores conectados mediante una malla.

Para realizar un buen diseño de la red, es esencial tener una visión completa del sistema y considerar cómo interactúan todos los elementos [1], [2]. Los mensajes que viajan por la red de interconexión son consecuencia directa del funcionamiento de la jerarquía de memoria. Si comprendemos este funcionamiento, podemos adaptar el diseño de la red para que responda a las necesidades concretas del subsistema de memoria.

En concreto, partimos de un *router* en el que todos los mensajes deben calcular la ruta y reservar los canales virtuales y paso por el *crossbar* en cada salto, lo cual supone una gran latencia de comunicación. Analizando el uso que se hace de la red, detectamos que el patrón petición-respuesta se repite con frecuencia. En la Figura 1 vemos los mensajes transportados en una malla de 16 nodos, organizados en peticiones y tipos de respuestas (se puede encontrar información más detallada de los mensajes generados por el protocolo de coherencia en la tabla II). Observamos que aproximadamente la mitad de los mensajes son respuesta a otro mensaje. Por lo tanto, conocemos con antelación su origen y destino y podemos hacer por adelantado el cálculo de ruta y reserva de canales virtuales y crossbar para eliminar estas etapas del camino crítico.

El resto de este artículo está organizado del siguiente modo: en la sección II se presenta el estado del arte; en la sección III se describen la arquitectura del sistema y el método implementado; en la sección IV se explica la evaluación y la sección V concluye el artículo.

II. Estado del arte

Existen en la literatura varias propuestas que combinan la conmutación de paquetes y de circuitos para facilitar caminos más rápidos para algunos mensajes, con el objetivo de reducir la latencia de la red. Enright *et al.* construyen circuitos mediante una red de *setup* y procuran reutilizar los circuitos hasta que aparece un conflicto [3]. Palumbo *et al.* utilizan dos redes físicas: una para conmutación de circuitos y otra de paquetes, y eligen cuál utilizar según el tamaño del mensaje [4]. Abousamra *et al.* configuran circuitos en la red periódicamente, analizando información de la comunicación entre los nodos del chip [5]. Peh y Dally reservan los buffers y enlaces mediante un flit de control que viaja por una red dedicada antes de enviar los datos [6]. Duato *et al.* deciden en tiempo de compilación si será útil montar un circuito entre dos nodos según la comunicación que se espera entre ellos, y utilizan señales de control específicas [7]. Gaughan *et al.* aseguran que un mensaje dispondrá de un circuito completo mandando la cabecera con anterioridad para buscar un camino libre [8]. Lee *et al.* utilizan un sistema de *tokens* para informar a los routers vecinos de su disponibilidad y para permitir

[1]Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, e-mail: {ortin.marta, dario, mvg, victor}@unizar.es.

[2]Department of Computer Science. University of Adelaide, e-mail: cruz@cs.adelaide.edu.au.

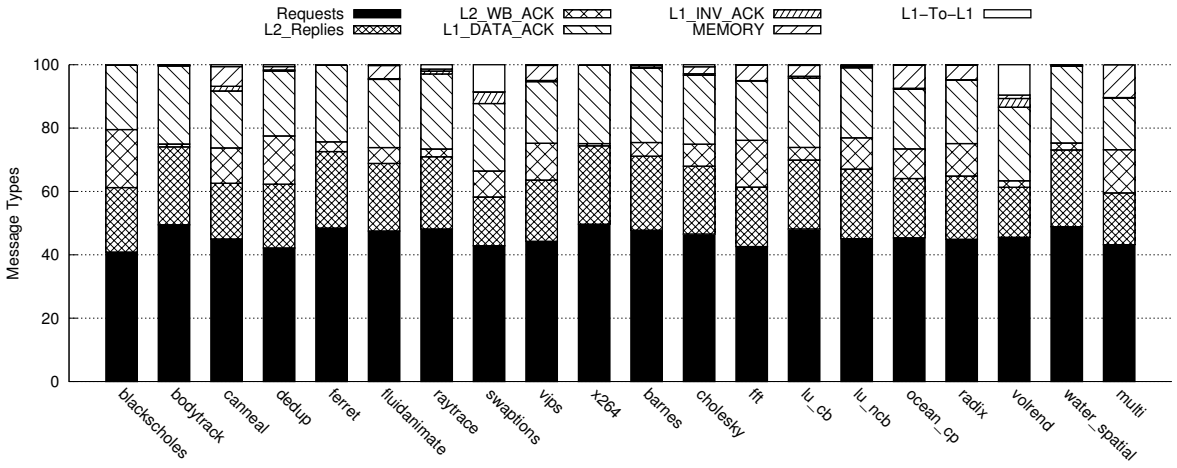


Fig. 1: Mensajes transportados por la red de interconexión en una malla de 16 nodos. Excepto la porción negra de la barra, el resto de mensajes son tráfico reactivo.

que los mensajes hagan *bypass* de los routers [9].

Otra línea de trabajo comúnmente utilizada para reducir la latencia de la red consiste en diseñar routers que incluyan especulación o caminos simples a utilizar cuando la carga es muy baja [10], [11], [12], [13]. La complejidad de estos routers es mucho mayor y puede suponer una disminución de la frecuencia o penalizaciones en energía y tiempo cuando no pueden utilizarse los atajos implementados. También hay propuestas que modifican radicalmente el router para eliminar buffers y reducir la latencia a uno o dos ciclos, pero sufren penalizaciones cuando encuentran una mínima congestión [14], [15].

Varios de los trabajos citados en esta sección centran su evaluación en tráfico sintético, impidiendo observar la interacción de los métodos propuestos con la jerarquía de memoria. Nuestra propuesta aprovecha el conocimiento del subsistema de memoria para reducir la latencia de comunicación de la red, reservando con antelación los recursos a lo largo del camino que seguirá un mensaje. No es necesario disponer de dos redes físicas, incluir mensajes adicionales que construyan los circuitos, recopilar información global del sistema ni modificar el protocolo de coherencia. Es suficiente con aplicar unas modificaciones mínimas en el modelo del router.

III. Reserva de circuitos

En esta sección presentamos las características del multiprocesador en chip y la red de interconexión de referencia. A continuación, explicamos el método seguido para reservar circuitos con antelación y utilizarlos para reducir la latencia de comunicación.

A. Arquitectura del sistema

Este trabajo se centra en un multiprocesador en chip homogéneo, en el que varios nodos se unen mediante una red de interconexión. Cada nodo contiene un procesador monohilo con una cache privada de primer nivel y una porción de la cache compartida de segundo nivel, ambos conectados directamente al router. Algunos nodos en los bordes del chip contienen también un controlador de memoria. La Tabla I resume los parámetros más importantes de la arquitectura y la Tabla II detalla los mensajes intercambiados en el protocolo de coherencia utilizado. Se trata de un protocolo MESI que permite transferencia directa de datos entre caches L1, por lo que está optimizado frente a una versión más simple que obligara a utilizar siempre la L2 como intermediario.

La red de interconexión está construida como una malla con routers sencillos de 4 etapas, encaminamiento XY y control de flujo de tipo *wormhole*. La Tabla III incluye la configuración detallada de la red utilizada como referencia en este trabajo.

B. Reserva de circuitos con antelación

Cuando una petición llega a su destino, ya sabemos que se necesitará enviar una respuesta al nodo solicitante. Con frecuencia, se trata de una petición de datos a un bloque de L2 que requerirá un mensaje de la categoría L2_Replies de la Figura 1. Por lo tanto, podríamos mandar la cabecera del paquete con antelación para reservar el camino por los routers hasta el destino. El problema de este método es que el acceso a la L2 es demasiado rápido frente al tiempo que costaría reservar el circuito (7 ciclos de acceso frente a los 10 y 24 ciclos de media que cuesta atravesar la red para chips con 16 y 64 procesadores, respectivamente).

La solución adoptada consiste en reservar el circuito mientras la petición viaja hacia la cache L2. Hay que tener en cuenta que utilizando *dimension order routing*, la petición y la respuesta no siguen

TABLA I: Principales características del multiprocesador en chip.

Procesadores	16 y 64, Ultrasparc III Plus, en orden, 1 instr/ciclo, monohilo, frecuencia 2GHz
Coherencia	Basada en directorio, MESI, directorio distribuido en los bancos de L2
Consistencia	Secuencial
Cache L1	Caches de datos e instrucciones de 32KB, 4-asociativa, 2 ciclos de acceso en acierto bloque de 64B. Privada, reemplazo pseudo-LRU
Cache L2	Distribuida, 1 banco/nodo, 1MB/banco, 16-asociativa, 7 ciclos de acceso en acierto, bloque de 64B. Compartida, inclusiva, reemplazo pseudo-LRU
Memoria	4 controladores de memoria, distribuídos en los bordes del chip (para las configuraciones de 16 y 64 nodos), latencia de 160 ciclos

TABLA II: Funcionamiento del protocolo de memoria.

Evento	**Secuencia de mensajes**
Fallo en L1	1º Petición de L1 a L2 2º L2_Replies: Datos de respuesta de L2 a L1 3º L1_DATA_ACK: ACK de recepción de datos de L1 a L2
Fallo en L1, otra L1 tiene el dato en exclusividad	1º Petición de L1 a L2 2º L2 reenvía la petición a la L1 propietaria del bloque 3º L1_To_L1: L1 propietaria envía el dato a la L1 solicitante 4º L1_DATA_ACK: ACK de recepción de datos de L1 solicitante a L2
Invalidación (por escritura o reemplazo en L2)	1º Invalidación de L2 a L1s que tienen el dato 2º L1_INV_ACK: ACK de L1s a L2
Reemplazo en L1	1º Información de reemplazo de L1 a L2 2º L2_WB_ACK: ACK de L2 a L1
Fallo en L2	1º Petición de L2 a memoria principal 2º MEMORY: Dato de memoria principal a L2
Reemplazo en L2	1º Información de reemplazo de L2 a memoria principal 2º MEMORY: ACK de memoria principal a L2

TABLA III: Principales características de la red de interconexión de referencia

General	2 redes virtuales (peticiones y respuestas), 2 canales virtuales (VC) por red virtual
Routers	4 etapas: *routing* y *buffering* de entrada, *VC allocation*, *switch allocation* y *switch traversal* VC/switch allocators en 2 fases, *Round-robin* Buffers de 5 flits por VC, suficiente para almacenar un mensaje completo
Enlaces	Flits de 16B, latencia de un ciclo

el mismo camino porque ambas viajan antes en la dirección horizontal y después en la vertical. Para resolverlo, comenzamos modificando el algoritmo de enrutamiento para utilizar prioridad XY para las peticiones e YX para las respuestas.

Las peticiones pasan por las cuatro etapas originales del router (ver Tabla III). En paralelo, cuando se pasa por *Virtual Channel allocation*, se reserva el camino para la respuesta. Al realizar la reserva, se almacena en el router la información necesaria para identificar los circuitos inequívocamente (identificador del destino y dirección del bloque de cache). Originalmente, la red virtual de respuestas cuenta con dos canales virtuales. En nuestra propuesta, comenzamos dejando un canal virtual para las respuestas que no tienen un circuito reservado y otro para las que sí lo tienen. En la red original, los canales virtuales no se utilizan de manera exhaustiva y no es frecuente que los paquetes queden bloqueados. En cambio, al mantener canales virtuales reservados durante más tiempo, detectamos que los recursos no eran suficientes para explotar el potencial de la propuesta. Por lo tanto, incluimos un canal virtual adicional para aumentar la cantidad de circuitos simultáneos, dejando un total de tres canales virtuales en la red virtual de respuestas.

De los tipos de mensajes de respuesta que observábamos en la Figura 1 y la Tabla II, nuestro método crea circuitos para las respuestas de L2 (L2_Replies) y los ACK de reemplazos (L1_WB_ACK). En media, esto supone un 52% de los mensajes de respuesta. Los ACK de invalidaciones (L1_INV_ACK),

las respuestas de memoria principal (MEMORY), y los datos intercambiados por dos caches L1 (L1_To_L1) suponen un porcentaje muy pequeño del total de respuestas. Los mensajes de tipo L1_DATA_ACK son respuestas enviadas de la L1 hacia la L2 después de la comunicación de petición-respuesta para confirmar la recepción del dato, y no siguen el mismo camino que los mensajes anteriores. Por lo tanto, no es posible aprovechar un mensaje previo para establecer el circuito.

Al intentar reservar un circuito, es posible que los recursos necesarios para garantizar que un mensaje pueda atravesar el router no estén disponibles. Ante esta situación, tenemos dos opciones: permitir que haya circuitos fragmentados intentando construir el camino desde ese punto en adelante, o permitir únicamente circuitos completos deshaciendo las reservas que se hubieran hecho ya en routers anteriores. En el primer caso, podrá haber múltiples circuitos desde el mismo puerto de entrada siempre que dispongan de canales virtuales libres en el siguiente router. En el segundo caso, habrá como máximo dos circuitos montados desde cada puerto (uno en cada canal virtual destinado a circuitos), por lo que el almacenamiento necesario para la información de reserva será menor. Además, un mensaje dispondrá del circuito completo desde el origen hasta el destino, así que nunca se quedará bloqueado en un router. Esto nos permite eliminar los buffers de los canales virtuales dedicados a circuitos, reduciendo el área respecto a la del router de referencia (incluso pasando de 2 a 3 canales virtuales para respuestas).

C. Utilización de los circuitos

Cuando un mensaje de respuesta llega a un router, verificará si hay un circuito construido para él. En ese caso, podrá pasar directamente por el crossbar usando un solo ciclo para atravesar el router en lugar de cuatro. Cuando el último flit del paquete atraviese el router, desmontará el circuito, liberando los recursos para otros paquetes. Si estamos permitiendo circuitos fragmentados, es posible que a lo largo de su recorrido un mensaje llegue a un router en el que no tenga circuito reservado. Cuando se de esta situación, el mensaje se almacenará en el *buffer* de entrada y pasará por las cuatro etapas normales del router.

Aunque haya un circuito reservado en un router, los puertos y cables podrán ser utilizados por otros paquetes si no los necesita un flit con el circuito reservado. El crossbar da preferencia a los mensajes con circuito, pero en caso de no haber conflictos, permitirá el paso a los flits almacenados en los canales virtuales.

D. Deshaciendo los circuitos

Hay varios casos en los que es necesario deshacer un circuito antes de que haya sido utilizado. En el protocolo de coherencia que estamos utilizando, una L2 puede reenviar una petición a otra L1 que tenga el dato en exclusividad, que se encargará de proveerlo directamente. Por lo tanto, el circuito establecido entre la L1 solicitante y la L2 no será utilizado nunca. También se ha considerado la posibilidad de deshacer los circuitos sin haberlos usado ante un fallo de L2, ya que el circuito estaría montado ocupando recursos durante un gran número de ciclos. Pero, tras analizar los resultados de varias simulaciones, hemos concluido que los resultados obtenidos son mejores si no los deshacemos. Por otro lado, en el caso en el que sólo permitimos circuitos completos, tendremos que deshacer el camino reservado si nos encontramos un router en el que no podemos hacer la reserva.

En todos estos casos, el circuito se deshace del mismo modo: los datos del circuito a deshacer se envían hacia el destino mediante créditos. Si coincide que tiene que mandarse un crédito para informar de un buffer libre al mismo tiempo, los datos se adjuntan en *piggybacking*; en caso contrario, se manda un crédito específicamente para deshacer el circuito.

IV. Evaluación

En esta sección presentamos la metodología de simulación y los principales resultados obtenidos.

A. Entorno de simulación

Utilizamos Simics [16], GEMS [17] y una versión extendida de Garnet [18]. Modelamos cuidadosamente todos los componentes del chip y realizamos simulación de sistema completo con procesadores monohilo sencillos y coherencia por directorio. Para obtener datos de energía, área y tiempo de ciclo usamos DSENT, una herramienta moderna de modelado de routers y enlaces. Utilizamos una frecuencia de 2GHz tanto para el procesador como para la red, y tecnología de 32nm.

B. Cargas de trabajo

Los multiprocesadores en chip pueden ejecutar aplicaciones paralelas para reducir el tiempo de ejecución, o cargas multiprogramadas (ejecución de aplicaciones independientes en cada procesador) para aumentar el *throughput*. Utilizamos aplicaciones paralelas de PARSEC [19] (`blackscholes`, `bodytrack`, `canneal`, `dedup`, `ferret`, `fluidanimate`, `raytrace`, `swaptions`, `vips` y `x264`) y SPLASH2 [20] con las entradas escaladas obtenidos de PARSEC 3.0 (`barnes`, `cholesky`, `fft`, `lu_cb`, `lu_ncb`, `ocean_cp`, `radiosity`, `radix`, `volrend`, `water_spatial`). En estas aplicaciones se simula la región paralela completa.

Para las cargas multiprogramadas, seleccionamos 16 aplicaciones con un *working set* extenso del conjunto SPEC CPU 2006 [21]. Para construir las cargas para el chip de 16 procesadores, las distribuimos

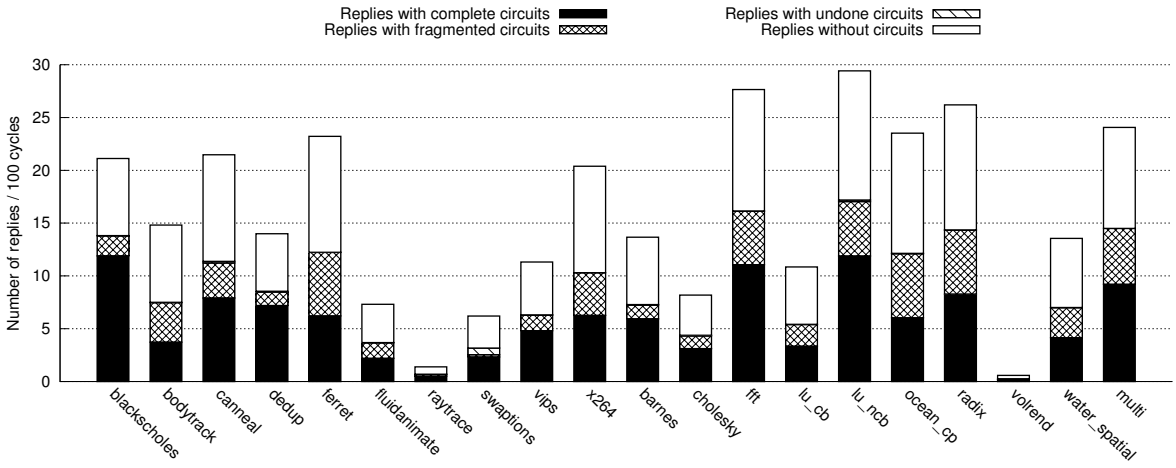


Fig. 2: Número de mensajes de respuesta que utilizan la red por cada 100 ciclos en una malla de 16 nodos, clasificados según su uso de circuitos. Estos resultados se corresponden con una configuración de la red en la que se permiten circuitos fragmentados.

aleatoriamente para formar 20 mezclas diferentes. Para el chip de 64 procesadores, repetimos cada una de las aplicaciones 4 veces y simulamos, de nuevo, 20 mezclas aleatorias diferentes. Para realizar la evaluación, se calientan las caches durante 200 millones de ciclos y después, las aplicaciones se ejecutan durante 500 millones de ciclos. Dada la poca varianza observada entre los resultados de las diferentes mezclas, se presentan únicamente los resultados de una de ellas para cada chip.

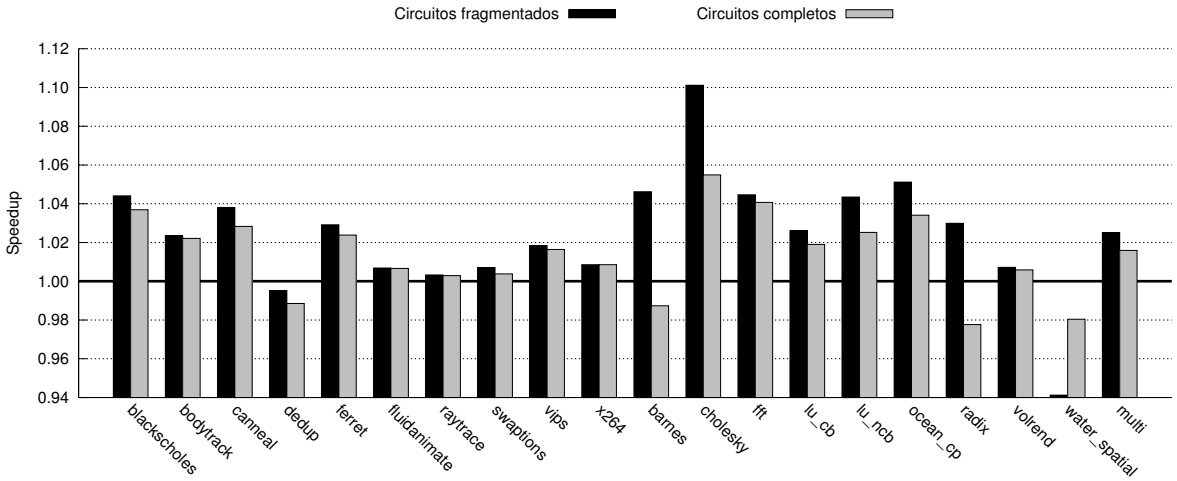
C. Resultados

La Figura 2 muestra los mensajes de respuesta que viajan por la red clasificados según su uso de circuitos: respuestas que cuentan con circuitos completos, respuestas con circuitos fragmentados (es decir, algunos de sus saltos se harán con latencia reducida y en otros deberán pasar por todas las etapas del router), respuestas con circuitos deshechos antes de ser utilizados (tal y como se explicaba en la sección III-D) y respuestas para las cuales no se intenta reservar circuitos (como se indica en la sección III-B). Observamos que gran parte de los circuitos que intentan reservarse lo logran completamente. En el caso de permitir sólo circuitos completos, la sección de circuitos fragmentados se corresponde con circuitos que deben deshacerse porque no puede reservarse desde el origen hasta el destino. Por otro lado, la fracción de circuitos completos construidos es algo mayor porque se liberan algunos recursos al deshacer los circuitos incompletos.

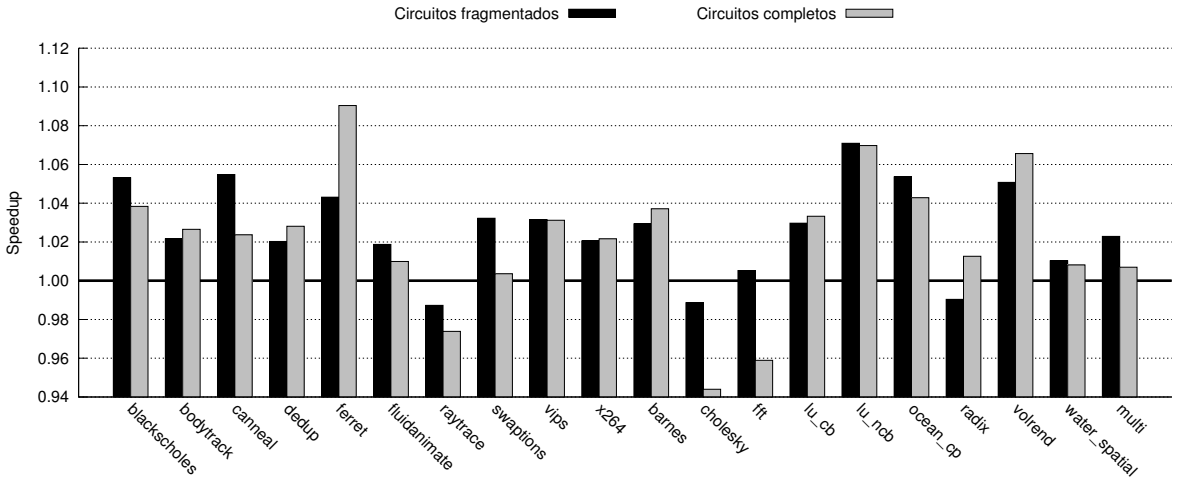
En la Figura 3 vemos la ganancia en tiempo de ejecución para cargas paralelas y número de instrucciones para cargas multiprogramadas que conseguimos con este método. Incluimos configuraciones de 16 y 64 procesadores, permitiendo la existencia de circuitos fragmentados o utilizando sólo circuitos completos. En general, la mejora de rendimiento es mayor cuando permitimos circuitos fragmentados, ya que el número de mensajes que reducen su latencia es más grande, aunque no cuenten con circuito en todos los saltos. En algunos casos, esta tendencia se invierte porque en los circuitos fragmentados hay muchos saltos en los que los mensajes no encuentran circuitos construidos, y esto se compensa con el incremento de circuitos completos que podemos reservar. También detectamos que el rendimiento disminuye en algunas aplicaciones paralelas, y hemos comprobado que en estos casos se incrementa sustancialmente el número de instrucciones ejecutadas. Esto es síntoma de un período de sincronización que está siendo más largo que antes. Por ejemplo, puede haber una barrera de sincronización esperando un mensaje bloqueado durante un tiempo en la red, lo que provoca que el resto de hilos esté en un bucle de espera activo. Esto puede venir por contención en el canal virtual único que dejamos para las respuestas que no tienen circuito.

Si comparamos los resultados para una red de 16 nodos y una de 64, la ganancia aumenta en casi todos los casos. Esto se debe a que el número de mensajes enviados por ciclo es mayor y, además, la latencia media de la red es más grande. Por lo tanto, la mejora tiene más impacto en el rendimiento del sistema. Estos resultados demuestran la escalabilidad del método y su vigencia para futuros sistemas con más nodos.

En cuanto al área necesaria para implementar la propuesta, el tamaño del router aumenta en un 23% en caso de querer usar circuitos fragmentados. En cambio, la posibilidad de eliminar espacio de almacenamiento en los canales virtuales al utilizar únicamente circuitos completos nos permite reducir



(a) 16 nodos



(b) 64 nodos

Fig. 3: Mejora lograda por la reserva previa de circuitos medida como el número de ciclos de ejecución en las aplicaciones paralelas y el número de instrucciones completadas para las cargas multiprogramadas, normalizados a los resultados de la red de referencia.

el área en un 8%. Respecto a la energía consumida, el aumento de canales virtuales provoca que, en el caso de permitir circuitos fragmentados, se incremente la energía una media del 19% y 10% para 16 y 64 nodos, respectivamente. En cambio, al usar siempre circuitos completos logramos reducir la energía consumida en todas las configuraciones y aplicaciones. En concreto, la energía disminuye una media de un 19% y 27% para 16 y 64 nodos, respectivamente.

A partir de los resultados obtenidos, pretendemos mejorar el método para incrementar la cantidad de circuitos construidos y reducir el tráfico reactivo que no utiliza circuitos, manteniendo las ganancias en área y energía.

V. Conclusiones

Los multiprocesadores en chip constan de varios nodos conectados mediante una red de interconexión, que contribuye en gran medida al área y la energía del chip. Además, la latencia de comunicación de la red tiene un gran impacto en el rendimiento del sistema. El uso de la red viene directamente determinado por la jerarquía de memoria y el protocolo de coherencia; por lo tanto, un diseño eficiente deberá tener en cuenta ambas partes. En este artículo nos centramos en multiprocesadores en chip homogéneos de memoria compartida, con 16 y 64 nodos conectados mediante una malla.

El trabajo parte de la observación de que la mayor parte del tráfico de la red es de tipo petición-respuesta, lo que nos permite conocer con antelación el camino que seguirán muchos mensajes. Hemos aprovechado esta información para reservar, al mismo tiempo que la petición recorre la red, los recursos para la respuesta. De esta manera, los mensajes que cuentan con circuito reservado no necesitan espacio de alma-

cenamiento en el router y pueden atravesarlo en 1 ciclo, en lugar de los 4 originales. Para un chip de 64 nodos, nuestra propuesta reduce la energía media consumida por el router en un 27% y el área en un 8%, mejorando el rendimiento global en un 2%.

Agradecimientos

Este trabajo ha sido financiado por los proyectos TIN2010-21291-C02-01 (Gobierno de España y Unión Europea), gaZ: T48 grupo de investigación (Gobierno de Aragón y Unión Europea) y la red de excelencia europea HiPEAC3.

Referencias

[1] Rakesh Kumar, Victor Zyuban, and Dean M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," in *Proceedings of the 32nd annual international symposium on Computer Architecture*, Washington, DC, USA, 2005, ISCA '05, pp. 408–419, IEEE Computer Society.

[2] Daniel Sanchez, George Michelogiannakis, and Christos Kozyrakis, "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 1, pp. 4:1–4:28, May 2010.

[3] Natalie D. Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti, "Circuit-switched coherence," in *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, Washington, DC, USA, 2008, NOCS '08, pp. 193–202, IEEE Computer Society.

[4] Francesca Palumbo, Danilo Pani, Andrea Congiu, and Luigi Raffo, "Concurrent hybrid switching for massively parallel systems-on-chip: the cyber architecture," in *Proceedings of the 9th conference on Computing Frontiers*, New York, NY, USA, 2012, CF '12, pp. 173–182, ACM.

[5] A. Abousamra, A.K. Jones, and R. Melhem, "Codesign of noc and cache organization for reducing access latency in chip multiprocessors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 6, pp. 1038–1046, 2012.

[6] Li-Shiuan Peh and W.J. Dally, "Flit-reservation flow control," in *High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on*, 2000, pp. 73–84.

[7] J. Duato, P. Lopez, F. Silla, and S. Yalamanchili, "A high performance router architecture for interconnection networks," in *Parallel Processing, 1996. Vol.3. Software., Proceedings of the 1996 International Conference on*, 1996, vol. 1, pp. 61–68 vol.1.

[8] P.T. Gaughan and S. Yalamanchili, "A family of fault-tolerant routing protocols for direct multiprocessor networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 6, no. 5, pp. 482–497, 1995.

[9] C.-Y. Lee, . , and N. K. Jha, "Variable-pipeline-stage router," 2012.

[10] Robert Mullins, Andrew West, and Simon Moore, "The design and implementation of a low-latency on-chip network," in *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, Piscataway, NJ, USA, 2006, ASP-DAC '06, pp. 164–169, IEEE Press.

[11] Robert Mullins, Andrew West, and Simon Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proceedings of the 31st annual international symposium on Computer architecture*, Washington, DC, USA, 2004, ISCA '04, pp. 188–, IEEE Computer Society.

[12] Li-Shiuan Peh and William J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, Washington, DC, USA, 2001, HPCA '01, pp. 255–, IEEE Computer Society.

[13] A Kumar, P. Kundu, A.P. Singhx, Li-Shiuan Peh, and N.K. Jha, "A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos," in *Computer Design, 2007. ICCD 2007. 25th International Conference on*, 2007, pp. 63–70.

[14] J. Kim, "Low-cost router microarchitecture for on-chip networks," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009, pp. 255–266.

[15] Jörg Mische and Theo Ungerer, "Low power flitwise routing in an unidirectional torus with minimal buffering," in *Proceedings of the Fifth International Workshop on Network on Chip Architectures*, New York, NY, USA, 2012, NoCArc '12, pp. 63–68, ACM.

[16] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50 –58, feb 2002.

[17] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, pp. 92–99, November 2005.

[18] N. Agarwal, T. Krishna, Li-Shiuan Peh, and N.K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, april 2009, pp. 33 –42.

[19] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, "The parsec benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, New York, NY, USA, 2008, PACT '08, pp. 72–81, ACM.

[20] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The splash-2 programs: characterization and methodological considerations," in *Proceedings of the 22nd annual international symposium on Computer architecture*, New York, NY, USA, 1995, ISCA '95, pp. 24–36, ACM.

[21] Standard Performance Evaluation Corporation (SPEC), "Spec cpu2006," 2006, `http://www.spec.org/cpu2006/` (Last access May 2013).

Diseño de mecanismos de prebúsqueda adaptativa bajo gestión eficiente de memoria para procesadores multinúcleo

Vicente Selfa [1], Paula Navarro [1], Crispín Gómez [2], María E. Gómez [1] y Julio Sahuquillo [1]

Resumen— **Los procesadores multinúcleo se encuentran presentes en la mayoría de los dispositivos electrónicos de uso cotidiano. Cada nueva generación es estos dispositivos requiere la incorporación de nuevas funcionalidades que demandan más potencia de cálculo, lo cual lleva a la necesidad de más núcleos y más potentes. El diseño de los futuros procesadores multinúcleo se ve comprometido por las altas latencias de acceso a memoria. La memoria se encuentra más lejos de los núcleos, debe cruzarse una red de interconexión cada vez más grande, y un mayor número de peticiones compiten por el acceso a memoria, lo cual hace que la memoria se convierta en el cuello de botella. Para ocultar estas latencias se han diseñado distintas técnicas como las técnicas de prebúsqueda existentes en la mayoría de los procesadores actuales. Estas técnicas intentan traer el dato en memoria antes de que el procesador lo solicite. Al tratarse de técnicas predictivas pueden traer datos no utilizados posteriormente con el consiguiente aumento de tráfico. Este trabajo se centra en el diseño de técnicas de prebúsqueda para procesadores multinúcleo donde el incremento del tráfico de la prebúsqueda puede dañar las prestaciones del sistema global debido a la mayor presión ejercida sobre el controlador de memorial.**

Palabras clave— **Multinúcleos, prebúsqueda, controlador de memoria, redes en chip.**

I. Introducción

ACTUALMENTE los dispositivos electrónicos basados en microprocesadores están presentes en las distintas facetas de nuestra vida diaria. De hecho muchas veces no los percibimos pero dependemos en gran medida de ellos. Esto ha sido posible gracias a que hace una década la industria de los microprocesadores evolucionó hacia los procesadores multinúcleo para atacar los importantes problemas de disipación de energía y calor de los procesadores del momento, así como para superar los límites inherentes del paralelismo a nivel de instrucción. Inicialmente, se comenzó con la integración únicamente de dos núcleos en un solo chip, pero pronto aparecieron procesadores comerciales con más núcleos (como el AMD Magny-Cours de 12 núcleos y el Intel Xeon Phi de 50 núcleos), y actualmente a medida que los recursos de silicio son más abundantes, chips con cientos de núcleos están en el horizonte. Sin embargo, la esperanza de incrementar las prestaciones en los futuros procesadores con tal cantidad de núcleos sólo se alcanzará si se consigue superar el reto que supone la escalabilidad en prestaciones y energía de estos procesadores.

Los avances en la arquitectura de los núcleos tienen el potencial de continuar incrementando las prestaciones de cómputo [1]. Sin embargo, otras partes del procesador no han avanzado al mismo ritmo. En concreto, la tecnología DRAM con la que se construye la memoria principal en la que se almacenan los datos que utilizan los programas para el cómputo, así como la interconexión de los núcleos con dicha memoria, se han mejorado a una velocidad mucho menor, por lo cual el acceso a memoria se ha convertido en el cuello de botella de los procesadores actuales y esta situación se agravará de forma notable en los futuros procesadores con un mayor número de núcleos.

Las altas latencias en el acceso a memoria ya supusieron el mayor cuello de botella en las prestaciones para muchas aplicaciones ejecutadas en los procesadores con un único núcleo. Como el tiempo de acceso es elevado, la productividad del procesador se ve seriamente afectada, porque el cómputo se ve detenido a la espera de que lleguen los datos a procesar. El problema se agrava todavía más en los procesadores multinúcleo por competir las peticiones enviadas desde varios núcleos en el acceso a memoria. Una aproximación para reducir el problema ha sido incorporar grandes cachés, de una capacidad de decenas de MB, en los procesadores actuales. Sin embargo, esto no resuelve el problema completamente y sólo es factible para un número de núcleos bajo.

Una aproximación clásica adoptada para ocultar las latencias de memoria en los procesadores monolíticos han sido las técnicas de prebúsqueda. Estas técnicas predicen qué dato pedirá el procesador en un futuro inmediato y buscan dicho dato en memoria antes de que el procesador lo solicite. De esta manera, se adelanta el acceso a memoria y se oculta parte o la casi totalidad de la latencia, mejorando de esta manera las prestaciones.

Las técnicas de prebúsqueda son especulativas, ya que se basan en una predicción, y ésta puede fallar. Se dice que la predicción es incorrecta si el dato buscado no se solicita posteriormente por el procesador. En dicho caso, las técnicas de prebúsqueda pueden tener efectos negativos sobre las prestaciones, ya que se incrementan el número de peticiones que se realizan a la memoria principal. A pesar de estos inconvenientes, las técnicas de prebúsqueda siguen mejorando las prestaciones en la mayoría de las aplicaciones para procesadores monolíticos.

Las técnicas actuales se han diseñado centrándose en procesadores monolíticos donde el tráfico entre el procesador y memoria lo genera un solo núcleo. Sin embargo, los inconvenientes citados se magnifican en los procesadores multinúcleo, por lo que las técnicas de prebúsqueda pueden ser contraproducentes e impactar negativamente en las prestaciones de los procesadores multinúcleo actuales. Las peticiones de

[1]Grupo de Arquitecturas Paralelas, U. Politécnica de Valencia
[2]Dpto. de Sistemas Informáticos, U. de Castilla La Mancha

los distintos núcleos, tanto por demanda como de prebúsqueda, compiten entre ellas por el acceso a memoria. Esta competencia provoca con frecuencia que el acceso a memoria se convierta en un cuello de botella todavía más acuciante en las prestaciones, ya que se producen altas latencias de espera en las colas del controlador y en la red de interconexión en las que las peticiones aguardan a que les llegue su turno para poder acceder a memoria, cosa que perjudica de manera notoria las prestaciones globales del sistema. En otras palabras, la prebúsqueda debería utilizarse solo para aquellas aplicaciones y en aquellos momentos en los que tengamos la certeza de que va a ser beneficioso.

Varias aproximaciones se han propuesto para atacar el problema de la congestión en el controlador de memoria. Una de estas aproximaciones consiste en implementar varios bancos conectados a un canal para ofrecer acceso paralelo tanto a nivel de banco como de canal. No obstante, la latencia de acceso a memoria todavía es considerable, por lo que es necesario diseñar políticas para una gestión eficiente del controlador que ayuden a mejorar las prestaciones globales del sistema.

De lo expuesto se deduce que las prestaciones del procesador dependen en gran medida de las prestaciones de la lógica de prebúsqueda y del controlador de memoria. Más aún, depende de como ambas partes interactúen entre ellas. Por ello en el presente trabajo perseguimos:

1. Implementar los mecanismos de prebúsqueda de los procesadores actuales en el simulador.
2. Implementar un controlador de memoria.
3. Diseño, evaluación y propuesta de mecanismos de prebúsqueda adaptativos para futuros procesadores multinúcleo que mejoren las prestaciones de los mecanismos actuales.

Este artículo se divide de la siguiente forma: La Sección II muestra el trabajo relacionado realizado hasta la fecha, la Sección III muestra el sistema base sobre el que se trabajará, la Sección IV presenta nuestra propuesta, que se evalúa en la Sección V, y finalmente se presentan unas breves conclusiones.

II. Trabajo relacionado

Hasta la fecha se ha invertido un gran esfuerzo en el diseño de técnicas de prebúsqueda para procesadores mononúcleo [2] [3] [4]. En [3] se propone por primera vez el uso de técnicas basadas en la detección de patrones regulares o strides, en las que se basa nuestra propuesta. [4] realiza un resumen completo de las técnicas de prebúsqueda más utilizadas en procesadores mononúcleo, ya que su propuesta es utilizar un buen número de técnicas distintas y seleccionar de forma adaptativa la que mejores prestaciones alcanza en cada momento de la ejecución.

Con la aparición de los procesadores multinúcleo, la distancia desde el núcleo a la memoria crece y los distintos núcleos compiten en el acceso a memoria. Sin embargo la mayoría de los estudios publicados han utilizado técnicas diseñadas para procesadores mononúcleo sin tener en cuenta que en este nuevo entorno es más importante hacer un buen uso de la prebúsqueda e incluso desactivar según la aplicación y el momento.

Este trabajo va en la línea de activar la prebúsqueda sólo en los intervalos de tiempo y para las aplicaciones que vayan a tener un claro beneficio. Hasta ahora, que tengamos conocimiento nosotros, solo hay dos trabajos publicados que persiguen esta línea pero con claras diferencias con respecto a nuestra propuesta. El primero de ellos [5] se centra en gestionar de forma distinta las peticiones de prebúsqueda y las peticiones bajo demanda en el controlador de memoria según el núcleo del que proceda. En concreto, esta propuesta trata las peticiones de prebúsqueda con la misma prioridad que las de bajo demanda cuando se estima que van a ser útiles, pero se tratan con menos prioridad que las de bajo demanda cuando se estima que van a ser prebúsquedas inútiles. Nuestra propuesta en lugar de mantener activa la prebúsqueda y bajar la prioridad en el controlador de memoria para aquellos núcleos en que es poco útil, no genera esas peticiones que van a consumir recursos en la red y en el controlador de memoria y que además probablemente no se utilicen. En el segundo trabajo los dos tipos de peticiones se tratan de forma distinta en la red, utilizando dos canales virtuales distintos (es decir, colas diferentes en la red) y se da prioridad a las peticiones por demanda frente a las de prebúsqueda. El principal punto débil de esta propuesta es que el controlador de memoria no se modela y no se hace ninguna distinción entre ambos tipos en éste. Centrarse sólo en la red reduce el problema pero no lo ataca desde la raíz porque la latencia de ésta es usualmente mucho menor que la sufrida en el controlador de memoria.

III. Sistema base

El presente trabajo se centra en el diseño de mecanismos de prebúsqueda y controladores de memoria eficientes para procesadores multinúcleo para atacar los problemas de latencia que sufren los procesadores actuales en el acceso a la memoria principal. A continuación se describe el sistema de prebúsqueda básico y las propuestas de mejora del presente trabajo.

A. *Sistema de prebúsqueda*

El objetivo de las técnicas de prebúsqueda es traer los datos a la caché antes de que el procesador los solicite. De esta manera se puede reducir o eliminar la latencia que conlleva el acceso a memoria. Hay dos aspectos de diseño claves. Por una parte, deben predecirse los accesos futuros que el procesador realizará. Por otra, debe decidirse donde se ubicarán estos datos.

A.1 Detección de patrones de acceso

Las técnicas actuales se basan en la detección de patrones regulares de acceso a la memoria para predecir los accesos futuros. Para ello, se usan tablas auxiliares hardware que guardan un número limitado de los últimos accesos a la memoria. En este trabajo empleamos la técnica descrita en [6] e implementada en muchos procesadores modernos. Esta técnica consiste en dividir el espacio de direcciones en parti-

ciones, y utilizar los bits superiores de la dirección como etiqueta para identificar la partición. Cuando el procesador ejecuta una instrucción de acceso a memoria, ésta accede primero a la caché. Cada vez que se produce dicho acceso, la lógica de detección de patrones calcula la distancia como la diferencia entre la dirección de memoria solicitada por este acceso y la dirección solicitada por el último acceso que se realizó a la misma partición. El valor de la distancia se guarda en la tabla auxiliar. Cuando se produce un nuevo acceso, se calcula la nueva distancia, y si la distancia calculada coincide con la guardada en la tabla, entonces se dice que ha detectado un patrón regular o "stride" y se puede iniciarse la prebúsqueda de bloques asociados a dicho stride.

A.2 Stream buffers

Una cuestión de diseño clave es si los bloques prebuscados se almacenan en la misma caché o en hardware auxiliar dedicado. En esta sección se analizan ambas opciones. Uno de los inconvenientes de las técnicas de prebúsqueda es la polución que pueden causar en las memorias caché si la predicción es incorrecta y los bloques de memoria prebuscados resultan inútiles al final. Estos bloques pueden reemplazar bloques útiles en las cachés, por lo que tendrán que traerse de nuevo cuando vuelvan a ser accedidos. En estos casos, las técnicas de prebúsqueda pueden aumentar la latencia de acceso a memoria en lugar de reducirla. Para evitar este efecto nocivo se ha propuesto el uso de los llamados stream buffers [3]. Estos buffers actúan como una memoria adicional adyacente a la caché y que solo contiene bloques prebuscados que todavía no han sido solicitados por el procesador. Cuando llega una petición de referencia a memoria, se busca en la caché en paralelo con los stream buffers. Si el dato accedido se encuentra en los stream buffer, deja de ser especulativo y se copia el bloque contenedor desde el stream buffer a la caché, evitando de esta manera los efectos nocivos de la polución. Cada stream buffer es una estructura FIFO que solo puede contener un stream, por lo que es necesario usar tantos como patrones deseen detectarse. Los últimos procesadores de Intel dan soporte a 32.

B. *Controlador y organización de la memoria principal*

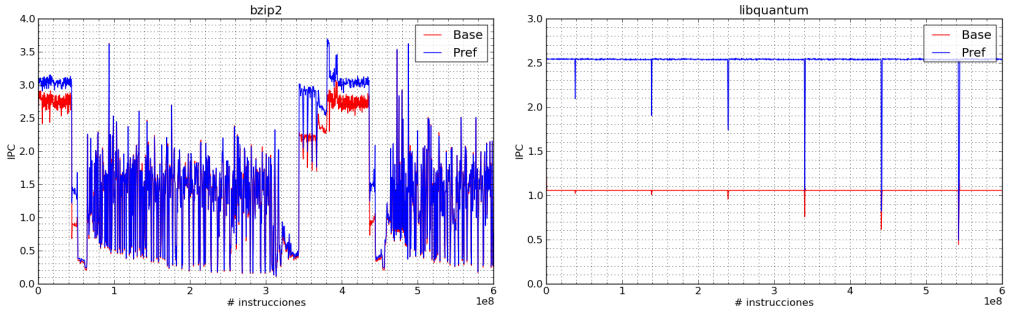
Son muchos los estudios que no han considerado el impacto del controlador de memoria en las prestaciones, a pesar de que este tiene un gran impacto sobre los beneficios obtenidos. Las peticiones de prebúsqueda y por demanda disparadas por el procesador cuando llegan al controlador esperan en una cola a poder acceder. Las memorias se implementan en bancos en forma de matrices organizadas en filas y columnas. La lógica del controlador puede lanzar un acceso a memoria si el banco destino y el canal de acceso se encuentran libres. En el momento del lanzamiento, el controlador determina la fila y la columna que contienen el bloque y el acceso se realiza en dos tiempos. Primero, se accede a la fila y su contenido se almacena en un buffer (row buffer) donde permanece de manera estable. A continuación, se accede a la posición que ocupa el bloque dentro de la fila (es decir, la columna) y el bloque se transfiere desde el row buffer hasta el controlador. En caso de que el row buffer contenga ya el bloque accedido se omite el primer paso y se accede directamente al buffer con el consiguiente ahorro de tiempo (sobre el 66 % del tiempo total). En el modelo básico se ha asumido que el controlador modela contención de banco, contención de canal y tiempos distintos según el bloque solicitado se encuentre o no en el row buffer. También se ha asumido que todas peticiones que llegan son atendidas atendiendo a una política que prioriza las peticiones que aciertan en el row buffer de su banco.

IV. Propuesta

A. *Técnicas de prebúsqueda adaptativa*

El efecto de la prebúsqueda sobre las prestaciones varía dinámicamente a lo largo del tiempo de ejecución del programa. Es más, en muchos de los benchmarks estudiados se detectan fases de ejecución en las que la prebúsqueda no produce ningún beneficio o incluso perjudica las prestaciones. Este hecho se puede observar en la Figura 1 que muestra el IPC por intervalos de 500K instrucciones de los benchmarks bzip2 y libquantum a lo largo de su ejecución con y sin prebúsqueda. Como se puede apreciar, en bzip2 la prebúsqueda se puede desactivar durante la mayor parte de su ejecución ya que no contribuye a mejorar las prestaciones. Sin embargo, en libquantum las prestaciones de la prebúsqueda son siempre mucho mejores, por lo que desactivarlo sería contraproducente. En ciertas situaciones, que cada vez se darán más en los procesadores multinúcleo, las prebúsquedas pueden perjudicar las prestaciones. Este efecto nocivo se debe a que las prebúsquedas inútiles consumen ancho de banda de memoria e incrementan la congestión, tanto en la red como en el controlador de memoria, que precisamente son los puntos débiles de cara a la escalabilidad de los procesadores multinúcleo. Para atacar el problema, en este trabajo se proponen técnicas de prebúsqueda adaptativa que activan la prebúsqueda cuando ésta contribuye a mejorar las prestaciones y la desactivan en aquellos intervalos en que su contribución es mínima o nula. De esta manera, se reduce el tráfico de peticiones de prebúsqueda de los benchmarks individuales con una penalización mínima en las prestaciones, lo que lleva a aumentar el ancho de banda disponible y minimizar la contención con el consecuente incremento de escalabilidad.

En este trabajo se presenta un ejemplo de política básica adaptativa *Activate on misses* y se propone la política Activate on ROB-stalls. Ambas, activan y desactivan la prebúsqueda en tiempo de ejecución. Para ello, se usan estadísticas que el procesador ya calcula para tomar la decisión de desactivar la prebúsqueda cuando ésta no aporta una mejora en las prestaciones y activarlo cuando se estima que incrementará las prestaciones. En ambas políticas, el mecanismo de desactivación calcula los fallos de caché que la prebúsqueda está ahorrando y la desactiva cuando el ratio entre los fallos con la prebúsqueda activa y los que habría si estuviese desactivada



(a) bzip2 (b) libquantum

Fig. 1
Evolución del IPC en tiempo de ejecución. Un valor más alto indica mejores prestaciones.

se sitúa por encima de cierto umbral (umbral de desactivación). Cuando la prebúsqueda está activa, se aproximan los fallos de cache sin prebúsqueda como los fallos con prebúsqueda más los aciertos en los streams.

La diferencia entre ambas políticas estriba en el mecanismo de activación. La activación resulta más complicada ya que cuando la prebúsqueda está activa se dispone de más información para poder saber si está funcionando bien o mal, pero no es así cuando está desactivada. En líneas generales, una política de activación debe estimar si las prestaciones no son las adecuadas y si se podrían mejorar activando la prebúsqueda. A continuación se describen los mecanismos de activación de ambas políticas:

- *Activate on misses*. Esta aproximación se basa en que las prestaciones de un procesador se encuentran muy ligadas a los fallos de caché y activa la prebúsqueda de nuevo cuando detecta un incremento en los fallos de caché que sobrepasa cierto umbral durante el intervalo actual con respecto al último intervalo de prebúsqueda activada. Esta política presenta ciertos inconvenientes de prestaciones y se ha refinado para elaborar la propuesta, por este motivo no se presentan los resultados.
- *Activate on ROB-stalls*. Este mecanismo no solo tiene en cuenta los fallos de caché, también analiza distintos eventos acontecidos en la microarquitectura del procesador. El mecanismo detecta caídas de prestaciones y analiza si se podrían evitar activando la prebúsqueda. Primero, detecta si la caída de prestaciones se debe a instrucciones de acceso a memoria. Para ello considera: el porcentaje de ciclos que el procesador no puede continuar ejecutando instrucciones (ROB - reorder buffer - bloqueado por culpa de una instrucción de memoria). Después analiza, si activando la prebúsqueda se mejorarían las prestaciones. Para ello, contabiliza el número de patrones que se han detectado (pero no se han lanzado prebúsqueda por estar desactivada).

V. Evaluación

A. Entorno de simulación

Se ha modelado el sistema mediante el simulador Multi2Sim [7] de uso generalizado entre la comunidad científica y desarrollado inicialmente por miembros de nuestro grupo de investigación. El simulador modela con detalle el funcionamiento del núcleo del procesador, y se ha ampliado de manera notable el código para modelar el sistema propuesto. La Tabla I muestra los parámetros de configuración del núcleo y la Tabla II los de la red y el subsistema de memoria principal. Para lanzar los experimentos se han utilizado las cargas SPEC2006 de uso generalizado por la comunidad científica. Cada benchmark se ha ejecutado durante 600 millones de ciclos después de calentar el sistema durante 500 millones de ciclos.

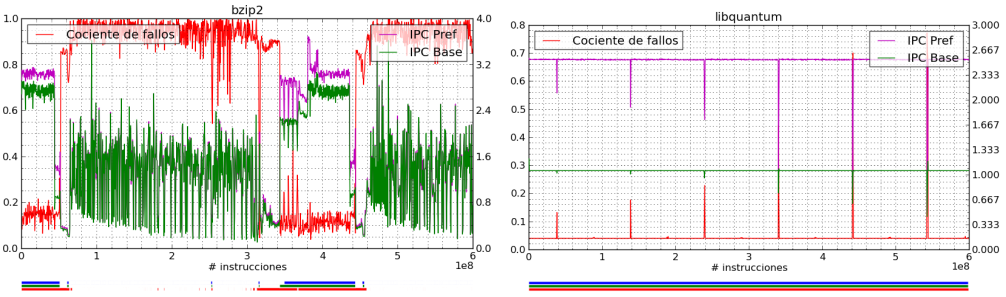
VI. Resultados

A. Prebúsqueda para procesadores con un sólo núcleo

En este apartado se compara nuestra propuesta en un sistema con un único núcleo con el sistema de pre-

TABLA I
Configuración base del núcleo.

Núcleo de ejecución	
Política de lanzamiento	Fuera de orden
Predictor de Saltos	Híbrido gshare/bimodal: gshare historia global 14-bit + 16K 2-bit contadores, bimodal 4K 2-bit contadores, y selección 4K 2-bit contadores
Ancho de lanzamiento	4 instrucciones/ciclo
Tamaño del ROB	256 entradas
Cola de loads/stores	64/48 entradas
Jerarquía de cache	
L1	32KB, 8 vías, 64B-line, 2 ciclos, 1 puerto
L2	256KB, 16 vías, 64B-line, 2 cycles, 1 puerto
Lógica de prebúsqueda	
Prebúsqueda de stream	32 streams de 4 entradas



(a) Cociente de fallos en L2.

(b) Accesos a memoria.

Fig. 2
Cociente de fallos en bzip2 y en libquantum.

búsqueda básico en términos de prestaciones y reducción de peticiones al controlador. Si la prebúsqueda tiene mucha precisión hay poco tráfico extra por prebúsqueda, ya que los accesos de prebúsqueda ahorran muchos accesos regulares. Sin embargo, si la precisión de la prebúsqueda no es buena esto es conveniente aplicar técnicas de reducción de tráfico. La Figura 3 muestra el número total de accesos a memoria principal con prebúsqueda básica normalizado para un sistema sin prebúsqueda. Como se puede apreciar, 9 de 16 aplicaciones realizan entre 9 % y 45 % más accesos a sistema en un sistema con prebúsqueda. A pesar de ello, las prestaciones no sufren en estos casos porque el controlador es capaz de atender satisfactoriamente el escaso tráfico de un solo núcleo.

A continuación, se analizan los beneficios que aporta la prebúsqueda adaptativa propuesta. La Figura 2 muestra como puede usarse el cociente de fallos de L2 entre prebúsqueda y sin prebúsqueda para desactivar la prebúsqueda. Las barras horizontales situadas en la parte inferior de la figura ilustran los instantes de activación (cuando la barra desaparece indica no activación). La barra superior representa los instantes de tiempo (en intervalos de 500K instrucciones) en los que la prebúsqueda está activada. La segunda representa cuando el cociente entre fallos de L2 se encuentra por debajo de cierto umbral (0.8 en los experimentos). Finalmente, la barra inferior representa los intervalos en los que la prebúsqueda gana al menos un 3 % en prestaciones. La diferencia entre la primera barra y la segunda estriba en que mientras

TABLA II
Configuración base de la red y la memoria.

Red de Interconexión	
Topología	Malla
Encaminamiento	X-Y
Buffer de entrada	128 entradas
Ancho del enlace	64B
Memoria principal	
Controlador de Mem.	Row Buffer Hit. 1 cola por banco
Bus de memoria	8B/ciclo
Tiempo de acceso	180 ciclos row buffer conflict. 60 row buffer hit

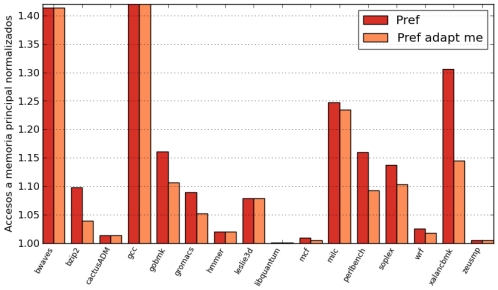


Fig. 3
Accesos a memoria con prebúsqueda y con prebúsqueda adaptativa normalizados respecto a los accesos sin prebúsqueda.

la prebúsqueda está desactivada no se puede calcular el cociente de fallos y hay que recurrir a las políticas de activación de las que hablábamos en la introducción. En bzip2 durante una parte muy importante del tiempo la prebúsqueda se mantiene desactivada, ahorrando un número significativo de accesos. En libquantum, la prebúsqueda mejora muy significativamente las prestaciones por lo que se mantiene activada durante toda la ejecución. La Figura 3 muestra el número total de accesos a memoria de un sistema con memoria con prebúsqueda básica y prebúsqueda adaptativa normalizados respecto al número de accesos a memoria sin prebúsqueda. Como se puede observar, la prebúsqueda adaptativa reduce en muchas aplicaciones el incremento de tráfico en un 25 % y un 50 % respecto a un sistema con prebúsqueda básica. Además los beneficios anteriores se consiguen con pérdidas despreciables del IPC. En la mayoría de los casos es inferior al 1 % y en ningún caso superior al 3 %.

B. Prebúsqueda en sistemas con varios núcleos

La contención por los recursos aumenta con el número de núcleos, lo que cambia el escenario ya que puede actuar como un catalizador magnificando los inconvenientes de la prebúsqueda. En este apartado se estudia este escenario. Para ello se han diseñado mezclas compuestas por varios benchmarks, tantos como núcleos, y cada benchmark se ejecuta en un núcleo distinto durante el experimento. La Tabla III muestra las 5 mezclas que se han diseñado para llevar

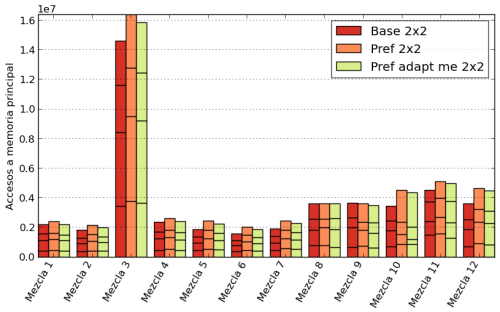


Fig. 4
ACCESOS A MEMORIA PRINCIPAL.

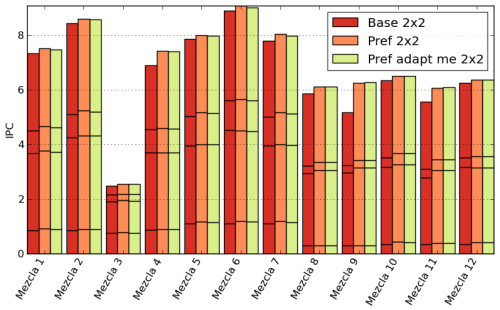


Fig. 5
IPC NORMALIZADO RESPECTO A LA EJECUCIÓN EN SOLITARIO.

a cabo los experimentos. Las siguientes figuras muestran los resultados de accesos a memoria (Figura 4) y prestaciones (Figura 5) para una malla de 2x2 con 4 núcleos. Se observa una disminución muy notable de los accesos a memoria y las prestaciones no se ven afectadas. Aunque con 4 núcleos los incrementos de prestaciones de la prebúsqueda adaptativa frente a la básica son pequeños, el hecho de que se reduzca mucho el número de accesos a memoria indica que esta diferencia se incrementará al aumentar el número de núcleos.

VII. CONCLUSIONES

Los procesadores multinúcleo se encuentran en la mayoría de los productos electrónicos de uso cotidiano demandando cada vez más funcionalidades y por lo tanto un mayor número de núcleos. El problema es que aunque se incremente el número de núcleos, la escalabilidad depende de otros elementos del sistema. En este trabajo, se ha mostrado como el subsistema de memoria en general y el controlador de memoria en particular ejercen de cuello de botella en las prestaciones. Esto es especialmente delicado en sistemas multinúcloe ya que las peticiones de todos los núcleos confluyen en este punto. Esto significa que la latencia aumenta por lo que es necesario aplicar técnicas para reducirla.

Las técnicas de prebúsqueda se han utilizado con éxito en la mayoría de los procesadores comerciales mononúcleo. Sin embargo, estas técnicas aplicadas a sistemas multinúcleo pueden tener efectos nocivos ya que aumentan la presión sobre el controlador de memoria, ya congestionado, por lo que sus beneficios están en discusión al aumentar el número de núcleos. Por ello en este trabajo se ha propuesto una técnica de prebúsqueda adaptativa que activa y desactiva la prebúsqueda para utilizarla solo en aquellos intervalos de tiempo en los que se sabe que va a ser beneficiosa y de esta manera reducir el tráfico. Los resultados muestran que esta técnica de prebúsqueda reduce el tráfico entre un 15 % y un 50 % en una gran cantidad de aplicaciones con una penalización de prestaciones despreciable en la mayoría de las aplicaciones. Además se ha demostrado que los beneficios respecto a la prebúsqueda básica aumentan con el número de núcleos.

TABLA III
COMPOSICIÓN DE LAS MEZCLAS.

Mezcla	Benchmarks
m1	bzip2, gobmk, lbm, mcf
m2	bzip2, gamess, bzip2, gamess
m3	bzip2, GemsFDTD, bzip2, GemsFDTD
m4	bzip2, h264ref, bzip2, h264ref
m5	gobmk, gamess, gobmk, gamess
m6	gobmk,GemsFDTD, gobmk,GemsFDTD
m7	gobmk, h264ref, gobmk, h264ref
m8	lbm, GemsFDTD, lbm, GemsFDTD
m9	lbm, h264ref, lbm, h264ref
m10	mcf, gamess, mcf, gamess
m11	mcf, GemsFDTD, mcf, GemsFDTD
m12	mcf, h264ref, mcf, h264ref

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por el Programa de Apoyo a la Investigación y Desarrollo (PAID-05-12) de la Universitat Politècnica de València bajo la clave SP20120748, y por el MINECO bajo las claves TIN2012-38341-C04.

REFERENCIAS

[1] Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid, *Web search using mobile cores: quantifying and mitigating the price of efficiency*, in Proceedings of the 37th Annual International Symposium on Computer Architecture, 2010.

[2] Jean-Loup Baer and Tien-Fu Chen, *An effective on-chip preloading scheme to reduce data access penalty*, in Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, 1991.

[3] N.P. Jouppi, *Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers*, in Proceedings of the 17th Annual International Symposium on Computer Architecture, 1990.

[4] Kyle J. Nesbit, Ashutosh S. Dhodapkar, and James E. Smith, *AC/DC: An Adaptive Data Cache Prefetcher*, in Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, 2004.

[5] Chang Joo Lee, Onur Mutlu, Veynu Narasiman, and Yale N. Patt, *Prefetch-Aware DRAM Controllers*, in Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture, 2008.

[6] S. Palacharla and R. E. Kessler, *Evaluating stream buffers as a secondary cache replacement*, in Proceedings of the 21st Annual International Symposium on Computer Architecture, 1994.

[7] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, *Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors*, in Proceedings of 19th International Symposium on Computer Architecture and High Performance Computing. SBAC-PAD, 2007.

Explotando el Compromiso Rendimiento-Justicia en Procesadores Multicore Asimétricos mediante Planificación

Juan Carlos Sáez[1], Fernando Castro[1], Daniel Chaver[1] y Manuel Prieto[1]

Resumen— **Los procesadores multicore asimétricos con repertorio común de instrucciones –AMPs (*Asymmetric Multicore Processors*)– han sido propuestos como alternativa de bajo consumo a los procesadores multicores simétricos convencionales. Los AMPs combinan *cores rápidos* y complejos, de elevado consumo, con *cores lentos* de consumo reducido. En este artículo proponemos Prop-SP, un nuevo algoritmo de planificación a nivel de sistema operativo (SO) cuyo objetivo es mejorar el compromiso entre el rendimiento y la justicia en un sistema AMP. Nuestra evaluación, llevada a cabo sobre hardware real y empleando implementaciones en un SO de propósito general, demuestra que nuestra propuesta obtiene un mejor compromiso rendimiento-justicia que otros planificadores propuestos previamente para un amplio espectro de cargas de trabajo.**

Palabras clave— **AMP, procesadores multicore asimétricos, planificación, sistema operativo.**

I. Introducción

LOS procesadores multicore asimétricos (AMPs) integran en un mismo chip diversos tipos de cores con distintas características (frecuencia, microarquitectura o consumo), pero con un mismo repertorio de instrucciones. Investigaciones previas han demostrado que los AMPs ofrecen un mayor rendimiento por vatio y unidad de área que los CMPs convencionales [1], [2]. Además se ha demostrado que la integración de sólo dos tipos de core ("rápidos" y "lentos") en el chip permite una utilización muy eficiente del área del procesador y simplifica considerablemente el diseño [1].

A pesar de las ventajas de los AMPs, éstos plantean retos importantes al sistema operativo. Uno de los más significativos es cómo distribuir eficientemente los cores rápidos entre las aplicaciones que comparten el sistema. La mayor parte de propuestas al respecto persiguen maximizar el rendimiento global o productividad [1], [3], [4], [5]. Para ello, el planificador debe mapear a los cores rápidos aquellas aplicaciones que derivan un mayor beneficio al ejecutarse en cores rápidos frente a cores lentos [3], [5], [4]. En este artículo nos referiremos a este beneficio como *speedup* de la aplicación en el AMP. Además, pueden obtenerse mejoras adicionales en el rendimiento empleando los cores rápidos como aceleradores de las fases secuenciales de aplicaciones paralelas [2], [6].

Otros objetivos relevantes, como proporcionar justicia en el reparto de los cores rápidos o dar soporte a prioridades, han recibido menor atención. Las propuestas existentes que persiguen este objetivo buscan mejorar la justicia en entornos AMP repartiendo equitativamente los cores rápidos entre las distintas aplicaciones. Sin embargo, dado que una carga multiprogramada puede estar constituida por aplicaciones con distinto speedup, un reparto equitativo de los cores rápidos entre aplicaciones de la misma prioridad no garantiza que todas experimenten la misma degradación en rendimiento (*slowdown*) con respecto a cuando ejecutan solas. Esto resulta inapropiado si se pretende proporcionar justicia y soporte a prioridades [7]. Además, el hecho de no tener en cuenta la diversidad de *speedups* entre aplicaciones puede degradar significativamente el rendimiento global [8], [4].

Para mitigar estas deficiencias, proponemos el algoritmo de planificación Prop-SP, que da soporte a prioridades y trata de equilibrar el *slowdown* experimentado por aplicaciones la misma prioridad. La comparativa de nuestro algoritmo con planificadores propuestos recientemente, como A-DWRR [9] o CAMP [4], revela que Prop-SP proporciona un mejor compromiso rendimiento-justicia para un amplio espectro de cargas multiaplicación, sin necesidad de ningún tipo de soporte HW ni requerir cambios en las aplicaciones.

El resto del artículo se organiza del siguiente modo. La Sección 2 motiva nuestro trabajo. La Sección 3 ilustra el diseño del planificador Prop-SP. En la Sección 4 mostramos y discutimos los resultados experimentales obtenidos. Finalmente, la Sección 5 presenta el trabajo relacionado y la Sección 6 expone las conclusiones derivadas de nuestro trabajo.

II. Motivación

En esta sección analizaremos la productividad y la justicia obtenidas por diversos planificadores de procesos/hilos en un sistema AMP. Como veremos, ningún algoritmo consigue maximizar ambas métricas simultáneamente.

Para cuantificar la productividad optamos por emplear el Speedup Agregado (*aggregate speedup*) que se define del siguiente modo:

$$Speedup\ Agregado = \sum_{i=1}^{n} \left(\frac{CT_{\text{slow,i}}}{CT_{\text{sched,i}}} - 1 \right) \qquad (1)$$

donde n es el número de aplicaciones que contiene la carga de trabajo, $CT_{slow,i}$ es el tiempo de ejecución (o de retorno) de la aplicación i cuando se ejecuta

[1]Grupo ArTeCS, Dpto. de Arquitectura de Computadores y Atomática, Universidad Complutense de Madrid, e-mail: {jcsaezal,fcastror,dani02,mpmatias}@ucm.es.

sola en el sistema y sólo usa cores lentos, y $CT_{sched,i}$ es el tiempo de ejecución de la aplicación i bajo un planificador determinado.

Por otra parte, consideramos que un sistema es justo, si el incremento en el tiempo de ejecución (*slowdown*) que experimenta una aplicación al compartir el sistema con otras respecto a la situación en la que la aplicación se ejecuta sola en el sistema, es el mismo para todas las aplicaciones de la misma prioridad [10], [11], [7]. Para evaluar el comportamiento de una estrategia de planificación en cuanto a justicia emplearemos la Injusticia (*unfairness*), métrica propuesta en [11], [7] y definida como sigue:

$$Injusticia = \frac{MAX(\text{Slowdown}_1, ..., \text{Slowdown}_n)}{MIN(\text{Slowdown}_1, ..., \text{Slowdown}_n)} \quad (2)$$

donde $Slowdown_i = CT_{sched,i}/CT_{fast,i}$, y $CT_{fast,i}$ es el tiempo de ejecución de la aplicación i cuando se ejecuta sola en el sistema (todos los cores rápidos están disponibles para ella en todo momento).

La figura 1 muestra el speedup agregado y la injusticia para cuatro cargas multiprogramadas en un AMP que consta de un core rápido y uno lento. Cada carga de trabajo está constituida por dos aplicaciones secuenciales pertenecientes a SPEC CPU2006. El número que figura entre paréntesis junto al nombre de cada aplicación (eje X) indica el speedup de la aplicación. Para aplicaciones secuenciales el speedup coincide con el *speedup factor* (SF) de su único hilo de ejecución, que se define como $\frac{IPS_{fast}}{IPS_{slow}}$, donde IPS_{fast} y IPS_{slow} representan el número de instrucciones retiradas por segundo al ejecutar el hilo en un core rápido y en uno lento, respectivamente.

La figura muestra los resultados obtenidos por tres algoritmos de planificación. El primero, denotado como HSP (High SPeedup) [1][5][4], destina los cores rápidos a ejecutar las aplicaciones de mayor speedup, mientras que las aplicaciones restantes se asignan a cores lentos. El segundo planificador es un algoritmo Round Robin (RR) asimétrico que garantiza un uso equitativo de los cores rápidos entre todas las aplicaciones [8], [12]. Finalmente, el tercero constituye nuestra propuesta, Prop-SP, cuyo funcionamiento se explicará en detalle en la siguiente sección. Para aplicaciones de la misma prioridad, Prop-SP asigna una fracción de tiempo de core rápido proporcional al speedup de la aplicación. Por ejemplo, para dos aplicaciones (n=2) de la misma prioridad, ejecutándose en el sistema AMP mencionado previamente, cada aplicación con speedup SP_i recibe una fracción de core rápido de $(\frac{(SP_i-1)}{\sum_{j=1}^{n}(SP_j-1)})$. Nótese que la distribución se realiza considerando el *speedup neto* $(SP-1)$.

Como se ve en la figura, HSP consigue optimizar el speedup agregado a costa de degradar la justicia significativamente. Por su parte, RR consigue un reparto mucho más justo, pero degrada la productividad (speedup agregado). Finalmente, Prop-SP logra una mayor productividad que RR, y a la vez consigue los mejores resultados en lo que respecta a justicia.

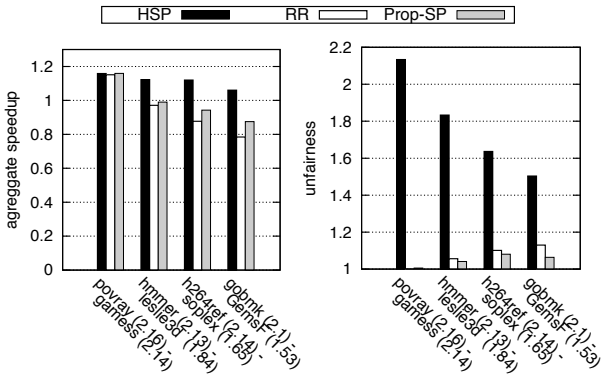


Fig. 1: speedup agregado e injusticia para los algoritmos de planificación RR, HSP y Prop-SP.

III. Diseño e Implementación

El planificador Prop-SP asigna hilos a los cores rápidos y lentos garantizando un equilibrio en la carga del sistema AMP. Por otra parte, migra hilos entre ambos tipos de cores para asegurar que éstos se ejecutan en los cores rápidos durante un tiempo determinado. Para llevar a cabo la asignación de hilos a cores, Prop-SP explota dos mecanismos: *distribución de créditos de core rápido* e *intercambio de hilos*.

La distribución de créditos de core rápido es un mecanismo empleado para controlar la fracción de tiempo de core rápido asignada a cada hilo en un AMP. Este mecanismo está inspirado en la técnica que emplea el *Credit Scheduler* de Xen [13] para realizar un reparto equitativo de las CPUs virtuales en entornos SMP. A grandes rasgos, la distribución de créditos de core rápido funciona de la siguiente forma. Cada hilo tiene asociado un contador de créditos de core rápido. Al ejecutarse en un core rápido, un hilo consume créditos. El planificador asigna de forma preferente a los cores rápidos aquellos hilos con contadores de créditos positivos. Cada cierto tiempo, el SO ejecuta un algoritmo de asignación de créditos que adjudica créditos de core rápido a las aplicaciones con hilos activos (listos para ejecutar). El periodo de tiempo comprendido entre dos asignaciones de créditos consecutivas, denominado *periodo de ejecución*, lo fija dinámicamente el planificador.

El algoritmo 1 ilustra el mecanismo empleado por Prop-SP para adjudicar créditos de core rápido. Cada aplicación recibe créditos en base a su peso dinámico (*dynamic weight*) asociado, que se define como el producto de su speedup neto y su peso estático (*static weight*). En este contexto, el speedup indica el beneficio relativo que la aplicación experimentaría si todos los cores rápidos del AMP se destinasen a ejecutar los hilos activos de esa aplicación con respecto a ejecutar todos los hilos en cores lentos. Prop-SP estima el speedup en tiempo de ejecución sin la intervención del usuario (ver sección III-A). Por otro lado, el peso estático se deriva directamente de la prioridad de la aplicación, fijada por el usuario. Como se muestra en el algoritmo 1, cada aplicacion recibe un número de créditos proporcional a su peso dinámico. Los créditos asignados a la aplicación son a su vez distribuidos entre sus hilos

ALGORITMO 1: Asignación de Créditos

{ • R *es el conjunto de aplicaciones con hilos activos.*
• N_{FC} *es el número de cores rápidos del AMP.*
• CRED_1FC_REF *es la cantidad de créditos consumidos en cada core rápido durante un periodo de ejecución de referencia.*
• credits_per_fc_next_period *es la cantidad de créditos consumidos en cada core rápido durante el siguiente periodo de ejecución.* }

S:= [ ]; total_weight:=0;
total_credits:=CRED_1FC_REF $* N_{FC}$;
{ *Calcular* total_weight *y el peso dinámico de todas las aplicaciones* }
foreach *app in R* **do**
  speedup$_{app}$:= estimar el speedup para *app*;
  dyn_weight$_{app}$:= (speedup$_{app}$ − 1) $*$ static_weight$_{app}$;
  total_weight := total_weight + dyn_weight$_{app}$;
  Insertar *app* en S para mantener S ordenada en orden descendente por dyn_weight$_{app}$;
end
{ *Distribución de créditos entre aplicaciones* }
foreach *app in* S **do**
  $\text{credit}_{app} := \frac{\text{total\_credits} * \text{dyn\_weight}_{app}}{\text{total\_weight}}$;
  { *Garantizar que cada aplicación recibe como mucho* credit_peak$_{app}$ *créditos* }
  credit_peak$_{app}$:=CRED_1FC_REF $*$ nr_run_threads$_{app}$;
  if credit$_{app}$ > credit_peak$_{app}$ **then**
    credit$_{app}$:=credit_peak$_{app}$;
    $\text{corr\_weight} := \frac{\text{credit\_peak}_{app} * (\text{total\_weight} - \text{dyn\_weight}_{app})}{\text{total\_credits} - \text{credit\_peak}_{app}}$;
    total_weight − :=dyn_weight$_{app}$ + corr_weight ;
    dyn_weight$_{app}$:=corr_weight;
  end
end
{ *Determinar la longitud del siguiente periodo de ejecución y distribuir créditos entre los hilos* }
Calcular credits_per_fc_next_period;
scale_factor:=credits_per_fc_next_period/CRED_1FC_REF;
foreach *app in* S **do**
  credit$_{app}$:=credit$_{app}$ $*$ scale_factor;
  Distribuir credit$_{app}$ créditos entre los hilos de *app*
end

activos. Para programas monohilo, esto supone incrementar el contador de créditos del único hilo en la cantidad de créditos adjudicados. En el caso de las aplicaciones multihilo, la distribución se lleva a cabo empleando una de las dos estrategias de distribución de créditos entre hilos que ofrece Prop-SP: Even y BusyFCs. *Even* los distribuye uniformemente entre los hilos activos de la aplicación. *BusyFCs* recorre secuencialmente la lista de hilos activos de la aplicación y asigna a cada uno de ellos la máxima cantidad de créditos que puede consumir en el siguiente intervalo de ejecución (`credits_per_fc_next_period`) hasta que no haya más créditos que repartir.

El intercambio de hilos es el mecanismo empleado por Prop-SP para garantizar que los hilos con créditos de core rápido tengan la oportunidad de agotar todos sus créditos sin romper el equilibrio en la carga del sistema. Para ilustrar el funcionamiento de este mecanismo, consideremos un AMP que consta de un core rápido y uno lento. Supongamos que hay dos hilos con créditos de core rápido ejecutándose en el sistema, cada uno de ellos asignado a un core diferente para preservar el equilibrio de carga. En un determinado instante, el hilo que se ejecuta en el core rápido agota sus créditos. En este punto, el planificador *intercambia* los hilos entre cores para brindar la oportunidad de consumir sus créditos al hilo que estaba ejecutándose en el core lento. Notese que este intercambio no altera el equilibrio en la carga del AMP. En general, Prop-SP efectuará un intercambio de hilos cuando un hilo ejecutándose en un core rápido agote sus créditos y, al mismo tiempo, haya hilos con créditos ejecutándose en cores lentos. Para controlar la frecuencia de intercambio de hilos durante el periodo de ejecución, el planificador ajusta adecuadamente la longitud de este periodo de ejecución al concluir el proceso de asignación de créditos. En nuestra configuración experimental, Prop-SP realiza un migración por segundo en promedio, lo que permite reducir de manera significativa la sobrecarga.

Los dos mecanismos descritos anteriormente no garantizan que los hilos mapeados a cores del mismo tipo reciban una fracción de tiempo de CPU *justa* ni tampoco garantizan el equilibrio de carga en el AMP. Para conseguir esto, Prop-SP se basa en mecanismos ya existentes en el SO. En concreto, en nuestro entorno experimental recurrimos a los *CPU sets* [14] de Solaris para crear dos conjuntos de CPUs con cores de distinto tipo: el conjunto "rápido" y el "lento". El planificador Prop-SP se ha implementado sobre la infraestructura de planificación de Solaris, que garantiza equilibrio de carga y justicia entre los cores *dentro de cada conjunto de CPUs*. Del equilibrio de carga *entre conjuntos de CPUs*, sin embargo, se encarga Prop-SP, para lo cual mantiene las colas de hilos por core (*run queues*) cargadas uniformemente y, al mismo tiempo, evita que los cores lentos reciban una carga mayor que los cores rápidos. Además, Prop-SP migra hilos de cores lentos a rápidos cuando éstos están inactivos, lo que permite maximizar la utilización de los cores rápidos.

A. Estimación del Speedup

Para poder calcular el peso dinámico de la aplicación (necesario para el algoritmo de distribución de créditos), Prop-SP ha de estimar su speedup en tiempo de ejecución. Determinar el speedup de una aplicación secuencial se reduce a obtener el SF (speedup factor) de su único hilo. Para ello, Prop-SP monitoriza valores de distintas métricas de rendimiento (como el número de instrucciones por ciclo o la tasa de fallos de cache) durante la ejecución usando los contadores hardware del procesador. Los valores obtenidos se usan como entrada a un modelo de estimación específico de plataforma, que puede construirse aplicando la técnica propuesta en [15].

Para estimar el speedup de una aplicación multihilo se deben tener en cuenta otros factores además del SF de cada hilo [16], [15], como su grado de paralelismo a nivel de hilo (TLP) o cómo se distribuyan los créditos de core rápido entre sus hilos activos. Para ello, Prop-SP emplea fórmulas que se han derivado analíticamente.

IV. Experimentos

Para el análisis del planificador Prop-SP hemos empleado un servidor multicore tipo NUMA con dos

TABLA I: Cargas de trabajo multiaplicación constituidas por aplicaciones paralelas y secuenciales

Categorías	Benchmarks
3STH-1HPH	hmmer, gobmk, h264ref, fma3d_m(9)
3STH-1HPL	povray, gamess, gobmk, swim_m(9)
2STH-1PSH-1HPM	gamess, bzip2, BLAST(4), wupwise_m(6)
1STH-1STM-1STL-1PSH	gamess, astar, soplex, blackscholes(9)
1PSH-1PSL	semphy(6), FFTW3D(6)
2PSH-1HPM	BLAST(4), semphy(4), wupwise_m(4)
1PSH-1HPL	semphy(6), equake_m(6)
1HPH-1HPL	fma3d_m(6), equake_m(6)
1PSH-1HPH	blackscholes(6), fma3d_m(6)

TABLA II: Cargas de trabajo multiaplicación constituidas por aplicaciones secuenciales

Categorías	Benchmarks
4H	povray, gobmk, bzip2, sjeng
3H-1M	povray, h264ref, perlbench, astar
3H-1L_A	hmmer, namd, perlbench, soplex
3H-1L_B	hmmer, h264ref, gobmk, milc
2H-2M_A	povray, bzip2, leslie3d, sphinx3
2H-2M_B	gamess, gobmk, xalancbmk, astar
2H-2L_A	hmmer, gobmk, lbm, soplex
2H-2L_B	povray, h264ref, lbm, omnetpp
1H-1M-2L	sjeng, leslie3d, lbm, soplex
2M-2L	sphinx3, leslie3d, soplex, mcf

procesadores hex-*core* AMD Opteron 2435 "Istanbul" –doce cores en total–. Cada procesador tiene una cache L3 de 6MB compartida por los seis cores de cada chip. Cada core incluye dos niveles de cache privados. La asimetría en este sistema se ha introducido mediante la reducción de la frecuencia de trabajo de algunos cores de la plataforma (cores "lentos"). Más concretamente, las configuraciones asimétricas incluyen cores "rápidos" que operan a 2.6Ghz y cores "lentos" con una frecuencia de 800MHz.

En la evaluación hemos empleado dos configuraciones AMP, (1) 2FC-2SC – dos cores rápidos y dos lentos distribuidos en dos chips de tal forma que cada chip incluye un core rápido y uno lento compartiendo un último nivel de cache – y (2) 2FC-10SC – dos cores rápidos y diez lentos en dos chips, donde cada chip contiene un core rápido y cinco lentos –.

En esta sección comparamos la productividad y justicia de Prop-SP con la de otros cuatro algoritmos de planificación propuestos previamente: HSP [5], [8], CAMP [4], A-DWRR [9] y RR [12]. Las implementaciones existentes de HSP y RR se diseñaron para cargas de trabajo constituidas por aplicaciones secuenciales únicamente. Para llevar a cabo nuestra evaluación hemos ampliado estos algoritmos para que soporten aplicaciones paralelas.

Como se indicó en la sección III-A, Prop-SP estima dinámicamente el SF de los hilos empleando modelos de estimación. Nos referiremos a esta implementación básica con el nombre de *Prop-SP (dynamic)*. Para evaluar el impacto de los fallos de predicción del modelo de estimación, comparamos la efectividad de Prop-SP con una versión estática del algoritmo –Prop-SP (static)–, en la cual el SF se mide antes de la ejecución (ratio de los tiempos de ejecución de la aplicación en el core lento y en el rápido).

Para realizar un estudio exhaustivo de las distintas políticas de planificación, construimos diversas cargas de trabajo multiprogramadas constituidas por aplicaciones secuenciales y paralelas de distintas suites de benchmarks – SPEC CPU2006, SPEC OMP2001, NAS Parallel, Minebench y PARSEC– así como BLAST –aplicación utilizada en bioinformática– y FFTW –un benchmark científico que realiza la transformada rápida de Fourier–. Las cargas de trabajo estudiadas se dividen en dos grupos. El primer grupo está constituido por cargas de trabajo con aplicaciones paralelas y secuenciales (tabla I). El segundo incluye únicamente conjuntos de aplicaciones secuenciales (tabla II).

En el proceso de construcción de cargas de trabajo representativas, clasificamos las distintas aplicaciones teniendo en cuenta su grado de paralelismo - ST (*single-threaded*), PS (*partially sequential*[1]) o HP (*highly parallel*) – así como en función de sus factores de ganancia – H (*high*), M (*medium*) o L (*low*)-. La primera columna de las tablas I y II especifica la composición de la carga de trabajo, indicando la clase de cada aplicación en el mismo orden en el que figura en la tabla. El número entre paréntesis que aparece junto al nombre de cada aplicación multihilo indica el numero de hilos con el que se ejecuta. Para medir CT_{fast} para una aplicacion específica –lo cual es necesario para calcular la injusticia– medimos su tiempo de ejecucion cuando se ejecuta sola en el AMP con la estrategia de distribución de créditos a nivel de hilo que da mejores resultados para esa aplicación. Esta estrategia de distribución se usa también al ejecutar la aplicación en la carga de trabajo multiprogramada con Prop-SP y HSP.

La figura 2 muestra los resultados para las cargas de trabajo de las tablas I y II ejecutándose en 2FC-10SC y 2FC-2SC, respectivamente. Los resultados muestran una mayor variación de speedup agregado para el primer grupo de cargas de trabajo que para el segundo. Esto se debe a la combinación de aplicaciones paralelas, que no experimentan una gran aceleración al usar cores rápidos (p.ej., aplicaciones HPL), con otras aplicaciones que sí extraen un beneficio significativo al ejecutarse en este tipo de cores (p.ej., aplicaciones secuenciales). Las cargas de trabajo que no incluyen aplicaciones secuenciales extraen un menor speedup agregado del AMP.

En primer lugar analizaremos los resultados correspondientes a los algoritmos HSP y CAMP, cuyo objetivo es maximizar el speedup agregado que se extrae de la plataforma. A simple vista puede apreciarse que el planificador HSP obtiene los speedups agregados más altos, pero a costa de conseguir los peores resultados (los más altos) en cuanto a injusticia para todas las cargas de trabajo. CAMP exhibe resultados similares a HSP en cuanto al speedup agregado pero consigue reducir la injusticia para cargas de trabajo como 3STH-1HPH or 3STH-1HPL. La

[1]Esta categoría incluye aplicaciones paralelas con fases de ejecución secuencial que constituyen al menos un 20 % de su tiempo de ejecución total.

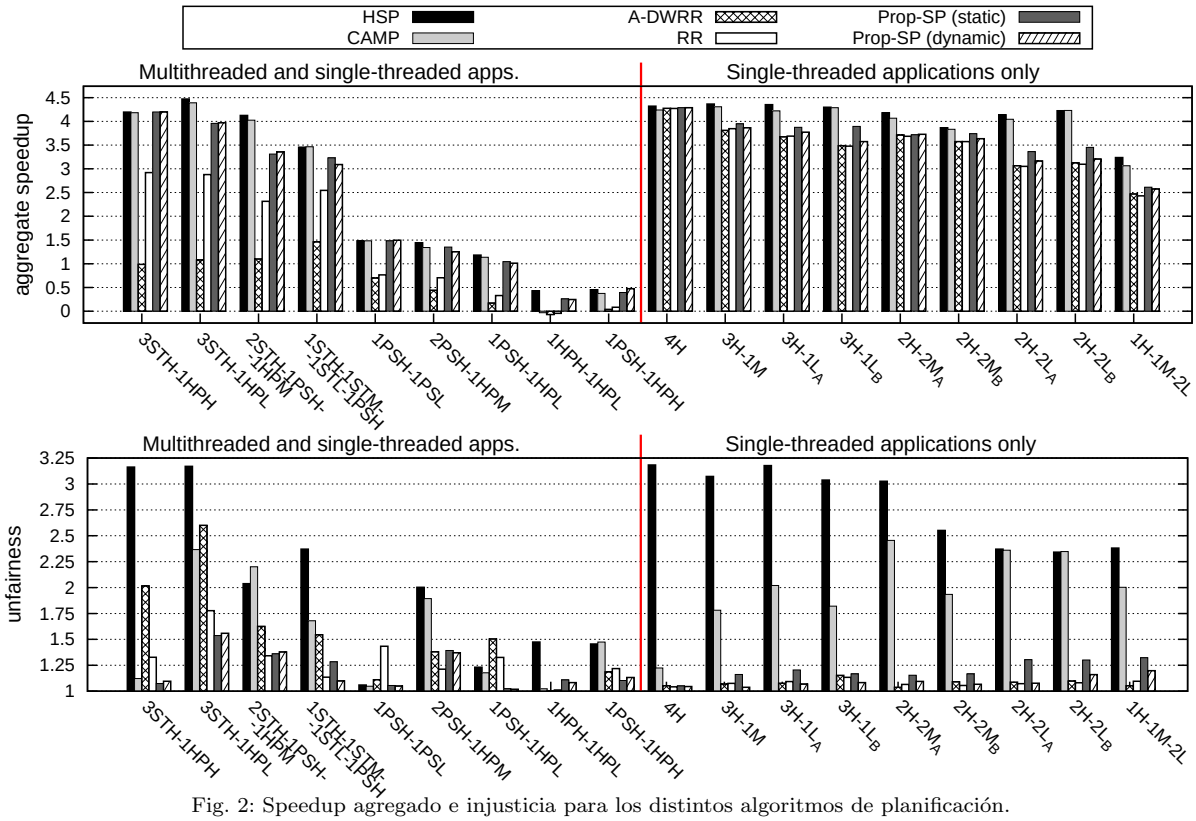


Fig. 2: Speedup agregado e injusticia para los distintos algoritmos de planificación.

razón de esta mejora se debe a que CAMP reparte de forma equitativa los cores rápidos entre todos los hilos que atraviesan una fase de ejecución con speedup elevado[2]. A pesar de este reparto equitativo, los números de injusticia de CAMP son aún mucho más elevados que los de Prop-SP para la mayor parte de cargas de trabajo.

Por otra parte, los resultados revelan que los algoritmos RR y A-DWRR no son capaces de obtener speedups agregados tan elevados como en el resto de algoritmos, especialmente para cargas de trabajo que incluyen aplicaciones paralelas. A diferencia de HSP o CAMP, RR y A-DWRR intentan reducir la injusticia en el uso del AMP realizando un reparto equitativo[3] de los cores rápidos y lentos entre las distintas tareas de la carga de trabajo en este escenario. Mientras que A-DWRR realiza este reparto *entre todos los hilos* de la carga de trabajo, RR lo lleva a cabo *entre aplicaciones*. De este modo, A-DWRR otorga más tiempo de core rápido a las aplicaciones que posean un mayor número de hilos. Como se muestra en trabajos previos [2], [4], conceder una mayor fracción de tiempo de core rápido a aplicaciones muy paralelas degrada el rendimiento global del sistema. Por lo tanto, el reparto que realiza RR (por aplicación) permite obtener un mayor speedup agregado y una menor injusticia en el AMP que el de A-DWRR.

Por último, podemos observar que, para cargas de trabajo como 3STH-1HPH, 1PSH-1PSL, 1PSH-1HPH o 4H, Prop-SP es capaz de obtener speedups agregados similares a los de HSP y CAMP. Para el resto de cargas de trabajo, el planificador Prop-SP logra speedups agregados más elevados que RR/A-DWRR. Además, en términos generales reduce notablemente la injusticia en comparación con HSP y CAMP. Cabe también destacar que para las cargas de trabajo que incluyen tanto aplicaciones paralelas como secuenciales, las dos variantes de Prop-SP exhiben un comportamiento similar. Como se apunta en [6], para este tipo de cargas de trabajo la cantidad de paralelismo a nivel de hilo (TLP) de una aplicación –que Prop-SP aproxima mediante la cuenta del número de hilos activos visible al sistema operativo– es el factor dominante en el speedup que la aplicación extrae al usar los cores rápidos de la plataforma. Habitualmente, cuanto mayor es el número de hilos de una aplicación, menor es el speedup que ésta deriva de usar los escasos cores rápidos del AMP. En consecuencia, los fallos de predicción leves que puedan producirse al estimar el SF de cada hilo no afectan significativamente a la calidad de la estimación del speedup de la aplicación. Por el contrario, estos fallos de estimación tienen un mayor impacto para cargas de trabajo que incluyen programas secuenciales únicamente. Como ilustran los resultados, pueden observarse diferencias entre el comportamiento de Prop-SP (static) y Prop-SP (dynamic) para este tipo de cargas de trabajo.

V. Trabajo Relacionado

La mayoría de los planificadores existentes para AMPs persiguen maximizar el rendimiento global de la plataforma. Aquellos algoritmos diseñados para

[2]Estas fases abarcan principalmente fases de ejecución secuencial con SF alto tanto en aplicaciones monohilo como multihilo.

[3]En escenarios donde las aplicaciones tengan distinta prioridad, RR y A-DWRR realizan un reparto ponderado en base a la prioridad de cada hilo.

cargas de trabajo que incluyen sólo aplicaciones secuenciales [1], [8], [5], [3], [12], [17] pretenden optimizar el rendimiento ejecutando en los cores rápidos aquellos programas con un mayor SF. En [12] se ha demostrado que los planificadores que se basan en modelos de estimación de SF ([12], [5]) obtienen SFs más precisos e introducen menor sobrecarga que las estrategias que requieren el muestreo del IPS en ambos tipos de cores ([1], [8]). Por ello, Prop-SP está equipado con modelos de estimación de SFs.

Para maximizar el rendimiento en cargas de trabajo que incluyen programas multihilo, algunos planificadores hacen uso de los cores rápidos del AMP como aceleradores de las fases secuenciales de las aplicaciones paralelas [2], [6], [4]. El planificador a nivel de aplicación propuesto en [2] es efectivo cuando la aplicación en la que dicho planificador está implementado es la única ejecutándose en el sistema, pero no así para cargas de trabajo multiprogramadas. Los algoritmos de planificación PA [6] y CAMP [4] superan esta limitación proporcionando aceleración automática de las fases secuenciales de las aplicaciones desde el sistema operativo. CAMP toma en consideración tanto la cantidad de TLP de la aplicación como el SF de sus threads al realizar la planificación (al igual que Prop-SP). Esto permite a CAMP conseguir un rendimiento superior al de PA [4].

El algoritmo de planificación A-DWRR [9] es el único planificador existente orientado a proporcionar tanto justicia como soporte a prioridades en AMPs con repertorio común de instrucciones. A diferencia de Prop-SP, A-DWRR no tiene en cuenta la diversidad entre los speedups de las aplicaciones y el hecho que éstos pueden variar durante la ejecución. Además, A-DWRR realiza la distribución de tiempo de CPU entre hilos y no entre aplicaciones. Como revela nuestra evaluación experimental, estos factores llevan a A-DWRR a degradar la productividad de forma significativa y evitan que este planificador garantice *slowdowns* uniformes para aplicaciones de la misma prioridad, especialmente cuando la carga de trabajo incluye aplicaciones multihilo.

VI. Conclusiones

En este artículo proponemos Prop-SP, un planificador que garantiza un uso justo de los cores en un AMP a la vez que extrae un rendimiento aceptable del sistema asimétrico. Para ello, Prop-SP equilibra el incremento en el tiempo de ejecución que sufren las aplicaciones de una carga de trabajo multiprogramada, explotando el hecho de que cada aplicación puede derivar un beneficio distinto (speedup) del uso de los cores rápidos de la plataforma. En nuestra evaluación, hemos analizado la efectividad de la implementación de Prop-SP en el kernel de Solaris y lo hemos comparado con varios planificadores existentes para AMPs (HSP, CAMP, RR y A-DWRR). Los resultados revelan que los planificadores HSP y CAMP, que destinan los cores rápidos a ejecutar las aplicaciones con mayor speedup, obtienen los mejores resultados en cuanto a productividad pero fallan a la hora de proporcionar justicia. Por otra parte, A-DWRR y RR se comportan bien en términos de justicia y rendimiento para cargas de trabajo constituidas por aplicaciones secuenciales únicamente. Sin embargo, ambos degradan el rendimiento significativamente en escenarios en los que la carga de trabajo incluye aplicaciones multihilo. Por el contrario, Prop-SP es capaz de hacer un uso eficiente del AMP y mejorar el compromiso rendimiento-justicia para un amplio rango de cargas de trabajo. Los beneficios de Prop-SP son especialmente pronunciados cuando las cargas de trabajo incluyen programas multihilo.

Agradecimientos

El presente trabajo ha sido financiado por los proyectos TIN2012-32180 e Ingenio 2010 Consolider ESP00C-07-20811.

Referencias

[1] Rakesh et al. Kumar, "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," in *Proc. of ISCA '04*, 2004.

[2] Murali Annavaram, Ed Grochowski, and John Shen, "Mitigating Amdahl's Law through EPI Throttling," in *Proc. of ISCA'05*, 2005, pp. 298–309.

[3] Daniel Shelepov et al., "HASS: a Scheduler for Heterogeneous Multicore Systems," *ACM SIGOPS Operating System Review*, vol. 43, no. 2, 2009.

[4] Juan Carlos Saez et al., "A Comprehensive Scheduler for Asymmetric Multicore Systems," in *Proc. of ACM Eurosys '10*, 2010.

[5] David Koufaty, Dheeraj Reddy, and Scott Hahn, "Bias Scheduling in Heterogeneous Multi-core Architectures," in *Proc. of Eurosys '10*, 2010.

[6] Juan Carlos Saez et al., "Operating System Support for Mitigating Software Scalability Bottlenecks on AMPs," in *Proc. of CF '10*, 2010.

[7] Eiman Ebrahimi et al., "Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems," 2010, ASPLOS '10.

[8] Michela Becchi and Patrick Crowley, "Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures," in *Proc. of CF '06*, 2006, pp. 29–40.

[9] Tong Li et al., "Operating system support for overlapping-isa heterogeneous multi-core architectures," in *HPCA'10*, 2010, pp. 1–12.

[10] Ron Gabor, Shlomo Weiss, and Avi Mendelson, "Fairness and throughput in switch on event multithreading," in *Proc. of MICRO '06*, Washington, DC, USA, 2006, IEEE Computer Society.

[11] Onur Mutlu and Thomas Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *Proc. of MICRO '07*, 2007.

[12] Juan Carlos Saez, Daniel Shelepov, Alexandra Fedorova, and Manuel Prieto, "Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems," *J. Parallel Distrib. Comput.*, vol. 71, pp. 114–131, January 2011.

[13] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat, "Comparison of the three CPU schedulers in Xen," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, Sept. 2007.

[14] Jim Mauro and Richard McDougall, *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture, Second Edition*, Prentice Hall, Mountain View, CA, USA, 2006.

[15] Juan Carlos Saez et al., "Leveraging core specialization via os scheduling to improve performance on asymmetric multicore systems," *ACM Trans. Comput. Syst.*, vol. 30, no. 2, pp. 6:1–6:38, Apr. 2012.

[16] M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *IEEE Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[17] Kenzo Van Craeynest et al., "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," 2012, ISCA '12, pp. 213–224.

Arquitecturas del Subsistema de Memoria y Almacenamiento Secundario

Prestaciones y consumo de caches híbridas variando la proporción de bancos SRAM

Alejandro Valero, Julio Sahuquillo, Salvador Petit, Pedro López y José Duato [1]

Resumen— **Las memorias cache se han implementado normalmente con tecnología SRAM. Esta tecnología presenta un tiempo de acceso rápido pero consumo de energía elevado y baja densidad. Al contrario, la reciente tecnología eDRAM permite que las caches tengan menor consumo de energía y área, aunque un tiempo de acceso más lento. La tecnología eDRAM ofrece una reducción significativa de leakage y área, especialmente en LLCs grandes que ocupan casi la mitad del área de silicio en algunos microprocesadores recientes.**

Este artículo propone una LLC híbrida que combina bancos SRAM y eDRAM para abordar el compromiso existente entre prestaciones y energía. Con este fin, se explora la proporción óptima de bancos SRAM y eDRAM que consigue el mejor compromiso. Se consideran mecanismos arquitectónicos para mantener los bloques MRU en bancos SRAM rápidos así como para evitar lecturas destructivas innecesarias.

Los resultados experimentales muestran que, comparado con una LCC SRAM convencional, la degradación de prestaciones no sobrepasa 2.9% en la media (incluso con un 12.5% de bancos SRAM), mientras que la reducción de área puede llegar hasta un 46% para una LLC de 1MB-16vías. Para una tecnología de 45nm, la métrica ED2P confirma que una cache híbrida resulta en un diseño mejor que una cache SRAM convencional independientemente del número de bancos eDRAM, y también mejor que una cache eDRAM convencional cuando el número de bancos SRAM es un cuarto o un octavo de los bancos de cache.

Palabras clave— **Bancos eDRAM, bancos SRAM, caches híbridas.**

I. Introducción

LAS jerarquías de cache multinivel se han implementado normalmente utilizando tecnología *Static Random-Access Memory* (SRAM) porque es la tecnología de memoria electrónica más rápida. Hoy en día se están explorando y utilizando tecnologías alternativas ya que SRAM presenta inconvenientes importantes como baja densidad y consumo de energía estática o *leakage* elevado, el cual es proporcional al número de transistores. Estos inconvenientes se han convertido en retos de diseño importantes, de forma que no se contempla que en el futuro la jerarquía de memoria esté sólo construida con tecnología SRAM, especialmente en el contexto de multiprocesadores.

Nuevos avances tecnológicos han permitido construir caches con otras tecnologías como *embedded Dynamic* RAM (eDRAM) o *Magnetic* RAM (MRAM). La tabla I resume algunas propiedades de estas tecnologías. La primera de ellas ofrece mayor densidad y leakage mínimo y se ha utilizado para implementar caches de último nivel (*Last-Level Caches*, -LLCs) de procesadores comerciales como el IBM POWER7 [1]. Esta memoria basada en condensador integra celdas DRAM en tecnología de circuito lógico [2], lo cual reduce significativamente el área respecto a celdas de 6 transistores utilizadas en tecnología SRAM. En otras palabras, comparada con SRAM, la tecnología eDRAM incrementa la capacidad de almacenamiento en un factor *3x* para una área de silicio dada, ofreciendo así una reducción importante de energía, sobretodo en LLCs grandes.

Un aspecto importante de diseño es que eDRAM puede ser fabricada por tecnologías lógicas con cambios mínimos en el proceso de fabricación. Esto significa que ambas tecnologías eDRAM y SRAM se pueden combinar en el *die* como ya se ha hecho en algunas compañías [3]. Aunque otras tecnologías como MRAM presentan menos leakage que eDRAM, las limitaciones que existen en el proceso de fabricación previenen que sean mezcladas en chips 2D convencionales. Además, la baja velocidad de MRAM, en particular para operaciones de escritura, sugiere el uso de esta tecnología para memoria principal en lugar de caches.

Debido a que la tecnología eDRAM es más lenta que la tecnología SRAM, las organizaciones basadas en SRAM se diseñan normalmente como caches L1 ya que este nivel es especialmente importante para las prestaciones; mientras que las caches eDRAM se diseñan para obtener grandes capacidades y ahorro de energía (principalmente leakage) trabajando como LLCs.

Como cada tecnología presenta sus ventajas e inconvenientes, algunos trabajos anteriores se han centrado en diseños de cache híbridos que tienen como objetivo aprovechar las ventajas de cada tecnología en diferentes componentes del procesador como caches L1 [4] [5] y NUCAs [6] [7]. En [4] se demuestra que en caches L1 híbridas eDRAM/SRAM, debido a la alta localidad que exhiben los datos en este nivel de la jerarquía de cache, las prestaciones se mantienen implementando una sola vía con tecnología SRAM. Esta configuración no funciona bien

TABLA I
Características de SRAM, eDRAM y MRAM

Tecnol.	Característica		
	Velocidad	Densidad	Leak.
SRAM	rápida	baja	alto
eDRAM	lenta	alta	bajo
MRAM	lenta (lect.) muy lenta (escr.)	alta	muy bajo

[1]Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mails: alvabre@gap.upv.es, {jsahuqui, spetit, plopez, jduato}@disca.upv.es.

en LLCs con un gran número de vías, ya que la localidad de los datos es mucho menos predecible.

En este artículo se propone un diseño de LLC híbrida y se explora el mejor compromiso entre prestaciones y energía. Con este objetivo, el vector de datos de la cache se implementa mediante bancos con distintas tecnologías, cada una de ellas almacenando bloques de cache específicos. El diseño persigue tres objetivos principales con respecto a una cache SRAM convencional con la misma capacidad: i) minimizar la pérdida de prestaciones, ii) reducir el área de ocupación y iii) aumentar el ahorro de consumo energético. El primer objetivo se consigue almacenando los bloques más recientemente utilizados (MRU) en bancos SRAM, mientras que los dos restantes se consiguen implementando un porcentaje significativo de bancos con tecnología eDRAM. Los resultados experimentales muestran que, comparado con una cache SRAM convencional con la misma capacidad de almacenamiento, la pérdida de prestaciones nunca excede, en la media, 2.9%, mientras que el área se reduce en un 46%. Para una tecnología de 45nm, la métrica *Energy-Delay squared Product* (ED2P) muestra que la cache híbrida es un diseño mejor que la cache SRAM convencional independientemente de la proporción de bancos eDRAM, y también mejor que una cache eDRAM convencional cuando la cuarta u octava parte de los bancos se implementa con SRAM.

El resto del artículo se organiza como sigue. La Sección II muestra la distribución de aciertos de cache en las posiciones de la pila *Least Recently Used* (LRU) para estimar la proporción de bancos SRAM y eDRAM. La Sección III presenta el diseño de la LLC híbrida propuesta. La Sección IV analiza las prestaciones, ED2P y área que ofrece la propuesta. La Sección V sintetiza los trabajos relacionados y finalmente, las conclusiones se muestran en la Sección VI.

II. Motivación

Las caches L1 de datos concentran la mayoría de sus aciertos (p.e. más de un 90%) en el bloque MRU [8]. Por tanto, en caches L1 híbridas eDRAM/SRAM es suficiente implementar una sola vía de cache con tecnología SRAM y forzar que esta vía mantenga el bloque MRU [4]. Sin embargo, la localidad de los datos en caches L2 es mucho menor que en caches L1, por lo que esta implementación conllevaría prestaciones inaceptables en caches L2.

Variando la proporción de bancos eDRAM y SRAM se puede obtener un amplio abanico de configuraciones de cache. Los dos extremos definen caches implementadas con una sola tecnología. Estas caches serán referidas como caches puras SRAM y eDRAM. Estos extremos ofrecen las mejores prestaciones pero mayor consumo de leakage y área, y las peores prestaciones pero menor consumo y área, respectivamente. Entre estos extremos podemos variar el porcentaje de bancos SRAM y eDRAM para beneficiar a las prestaciones, consumo o área.

En un diseño de cache híbrida se debe encontrar el punto óptimo en el espacio de diseño, es decir, el diseño debe proporcionar el mejor compromiso entre prestaciones, consumo energético y área.

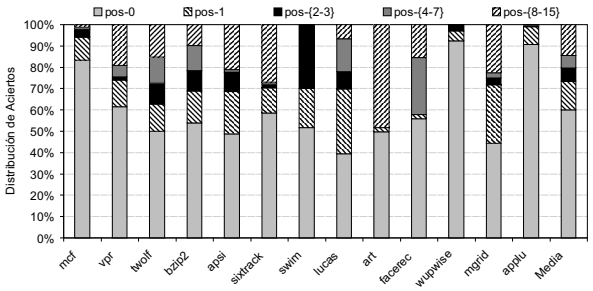


Fig. 1. Porcentaje de aciertos en las posiciones de la pila LRU.

En este artículo se asume una LLC (cache L2) asociativa por conjuntos de 16 vías y una capacidad de 1MB. Con el objetivo de servir como guía para estimar cuantas vías se deben implementar con bancos SRAM, esta sección analiza la distribución de aciertos.

Esta distribución se ha obtenido para la pila LRU con el objetivo de analizar si los aciertos se concentran en unos pocos bloques en la cabeza de la pila. En ese caso, estos bloques se deben almacenar en los bancos SRAM rápidos mediante operaciones de intercambio o *swap* para mover bloques entre bancos de distintas tecnologías. La Figura 1 muestra los resultados para las aplicaciones SPEC2000 [9][1]. La etiqueta *pos-0* se refiere a la posición que almacena el bloque MRU, mientras que *pos-15* es la localidad que alberga el bloque LRU. La etiqueta *pos-*{*x-y*} denota aciertos entre las posiciones x e y en la pila, ambas inclusive.

Como puede observarse, al contrario que en caches L1 donde más de un 90% de aciertos se concentran en el bloque MRU, los aciertos se distribuyen entre las diferentes posiciones de la pila LRU en caches L2. Aunque esta distribución está claramente acumulada hacia las posiciones en la cima de la pila, la cache requeriría más de la mitad de su capacidad (8 vías) para conseguir un 90% de aciertos de cache. Nótese que en 8 de 12 aplicaciones la vía MRU captura *sólo* un 50% de ciertos de cache. Por tanto, implementar sólo esta vía con tecnología SRAM conllevaría prestaciones inaceptables.

III. Diseño LLC Híbrida

Esta sección discute los mecanismos arquitectónicos que se utilizan para evitar lecturas destructivas innecesarias en bancos eDRAM y para mantener los datos MRU en bancos SRAM. Además, se presenta un mecanismo de refresco distribuido para evitar descargas de condensador.

Este artículo asume que cada banco implementa dos vías, lo que resulta en una LLC de 16 vías con 8 bancos. Este número de bancos es aceptable, puesto que se pueden encontrar otros diseños en la literatura

[1] Aquellas aplicaciones que exhiben una tasa de acierto L2 mayor que un 95% no se han considerado en este estudio. Los parámetros arquitectónicos se muestran en la Sección IV.

TABLA II
CACHES CONVENCIONALES E HÍBRIDAS CON EL NÚMERO CORRESPONDIENTE DE VÍAS, BANCOS Y PORCENTAJE DE BANCOS SRAM

Configuración	vías SRAM	vías eDRAM	bancos SRAM	bancos eDRAM	porcentaje SRAM
16S	16	0	8	0	100
8S-8D	8	8	4	4	50
4S-12D	4	12	2	6	25
2S-14D	2	14	1	7	12.5
16D	0	16	0	8	0

con más bancos [10]. Como cada banco del vector de datos se puede implementar con tecnología SRAM o eDRAM, se pueden estudiar diversas configuraciones de cache. La Tabla II resume las configuraciones consideradas, especificando el número de vías y bancos SRAM y eDRAM de cada configuración de cache y la proporción de bancos SRAM.

Las caches convencionales en ambos extremos de la tabla son las SRAM pura (16S) y eDRAM pura (16D), que tienen todos sus bancos implementados con tecnología SRAM y eDRAM, respectivamente. El vector de etiquetas se construye con celdas SRAM independientemente de la configuración de cache, puesto que las etiquetas son mucho más pequeñas que el vector de datos, es decir, los beneficios en consumo y área son mucho menores en esta estructura, e implementarla con eDRAM afectaría negativamente al tiempo de acceso.

A. *Acceso a la Cache Híbrida*

Para reducir el tiempo de acceso, las caches convencionales normalmente solapan el acceso al vector de etiquetas con el acceso a todas las vías del vector de datos, seleccionando finalmente la vía objetivo. Sin embargo, esto hace que el consumo de energía aumente en caches híbridas porque las lecturas en eDRAM son destructivas y estas operaciones requieren re-escribir los datos. Por tanto, el acceso a caches híbridas se separa en dos etapas como se muestra en la Figura 2. En la primera etapa, se accede a todas las etiquetas (implementadas con SRAM) y a todos los bancos SRAM (vector de datos SRAM) en paralelo. Si los datos solicitados están en una vía SRAM, el tiempo de acceso a la cache híbrida es tan rápido como un acierto en una cache SRAM convencional y la segunda etapa no se lleva a cabo, es decir, no se accede a ninguna vía eDRAM. En caso de un fallo de cache en el vector de datos SRAM pero acierto en una etiqueta asociada a una vía eDRAM, la segunda etapa se lleva a cabo y sólo se accede a la vía eDRAM requerida. En este caso, el tiempo de acceso incluye la comparación de etiquetas más el acceso a los datos eDRAM. Ante un fallo de cache, no se accede a ninguna vía eDRAM y los datos requeridos se obtienen de memoria principal.

B. *Cómo Mantener los Bloques en la Cima de la Pila LRU en Bancos SRAM*

Para mantener los datos MRU en bancos SRAM rápidos se debe modificar el controlador de cache para manejar operaciones de swap entre bancos SRAM y eDRAM, similarmente a como se hace en [7]. Para seleccionar adecuadamente los bloques a transferir, también se ha de mantener una pila LRU separada en cada vector de datos SRAM y eDRAM. El mecanismo funciona como sigue. En un acierto en eDRAM, el bloque solicitado se transfiere desde su banco eDRAM al banco SRAM que contiene el bloque LRU del vector de datos SRAM, el cual a su vez se transfiere al banco eDRAM. Después de esta operación de swap, los dos bloques involucrados serán los MRU de cada vector de datos. Puesto que el vector de datos SRAM tiene su propia pila LRU, un bloque no deja este vector y se transfiere al vector eDRAM hasta que pasa a ser el LRU y se selecciona para llevar a cabo la operación de swap. Sin embargo, nótese que este bloque reside en el mismo banco SRAM desde su llegada hasta que se reemplaza del vector SRAM y no hay movimientos de bloques mientras estos se encuentran en un vector dado (SRAM o eDRAM).

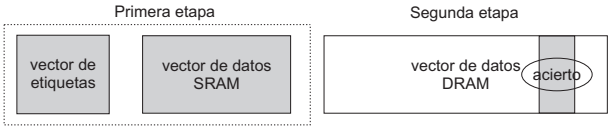


Fig. 2. Diagrama de acceso a la cache híbrida. Las cajas grises representan las partes accedidas de la cache.

En un fallo de cache, el bloque LRU del vector eDRAM se selecciona como reemplazo. Debido a que el bloque entrante que se copia desde memoria principal se aloja en un banco SRAM, el controlador dispara una transferencia de datos unidireccional desde el bloque LRU del vector SRAM hacia el bloque eDRAM que contiene el bloque víctima. Por supuesto, los bits de estado se deben actualizar en consecuencia. Nótese también que dividiendo la pila LRU en dos se reduce el número de bits de estado.

C. *Refresco Distribuido*

Aunque las operaciones de swap evitan el refresco de los datos eDRAM accedidos, los contenidos en eDRAM que no se referencian durante un periodo de tiempo largo se pueden perder porque los condensadores pierden su estado con el tiempo. Esta pérdida de datos afecta a las prestaciones puesto que estos datos, si se requieren, deben obtenerse de memoria principal. Para evitar estas situaciones, se deben llevar a cabo operaciones de refresco en todos los bloques eDRAM en caches híbridas y eDRAM puras antes de que los condensadores pierdan el valor almacenado, es decir, antes de que expire su tiempo de retención.

El tiempo de retención depende de la capacitan-

TABLA III
PARÁMETROS DE LA MÁQUINA

Núcleo del microprocesador	
Política issue	Fuera de orden
Predictor de saltos	Hybrid gShare/Bimodal: gShare: 14-bits de historia global y 16K contadores de 2-bits. Bimodal: 4K contadores de 2-bits. Selector de predictor con 4K contadores de 2-bits
Penalización predictor	10 ciclos
Ancho fetch, issue y commit	4 instrucciones/ciclo
Tamaño ROB (entradas)	128
# Int/FP ALUs	4/4
Jerarquía de memoria	
Cache L1 de datos	16KB-2vías, 64B-línea
Tiempo de acceso L1	2 ciclos
Cache L2 unificada	1MB-16vías, 64B-línea
Tiempo de acceso L2	Etiquetas: 2 ciclos Bancos SRAM: 6 ciclos Bancos eDRAM: 9 ciclos
Memoria principal	100 ciclos

cia. En este trabajo se asume que las celdas eDRAM se implementan con condensadores de *trinchera* [11] con una capacitancia de 10fF, la cual corresponde a un tiempo de retención de 190K ciclos de procesador para una frecuencia de 3GHz [12]. Para mitigar la penalización de refresco, asumimos un mecanismo de refresco distribuido entre los bancos, donde cada bloque eDRAM se refresca regularmente. El periodo entre dos operaciones de refresco consecutivas se establece como el tiempo de retención dividido por el número de bloques eDRAM. Esto garantiza que todos los bloques eDRAM se refrescan antes de que el tiempo de retención expire.

IV. Evaluación Experimental

Esta sección presenta el entorno de simulación utilizado para evaluar prestaciones, energía y área de las caches estudiadas. Las caches híbridas han sido modeladas con el simulador SimpleScalar [13]. Los resultados obtenidos incluyen el tiempo de ejecución de las aplicaciones y los eventos de memoria (aciertos de cache, fallos, writebacks, swaps[2] y refrescos) requeridos para estimar el consumo de leakage y dinámico, respectivamente. También se ha modelado contención de bancos debido a todos estos eventos de memoria en caches híbridas y eDRAM puras. En otras palabras, un acceso a un banco dado debe esperar hasta que el acceso previo al mismo banco finaliza y el controlador de cache lo libera. Accesos a bancos diferentes pueden hacerse concurrentemente. La energía dinámica por acceso, leakage y área fueron estimados con CACTI 5.3 [14] para una tecnología de 45nm. El consumo total fue calculado combinando los resultados de los dos simuladores.

Los resultados experimentales se obtuvieron configurando SimpleScalar para el juego de instrucciones Alpha y lanzando las aplicaciones SPEC2000, que se evaluaron utilizando las entradas *ref*, ejecutando 1000M de instrucciones antes de obtener estadísticas y simulando posteriormente 500M de instrucciones con detalle. La Tabla III resume los parámetros arquitectónicos utilizados en los experimentos. Los tiempos de acceso fueron obtenidos con la herramienta CACTI para una frecuencia de 3GHz.

[2]El consumo dinámico de las operaciones de swap se ha computado como la suma de energía de un acceso de lectura a los bancos SRAM, un acceso de lectura a un banco eDRAM, un acceso de escritura a un banco eDRAM y un acceso de escritura a un banco SRAM.

A. Evaluación de las Prestaciones

Para entender mejor los resultados de prestaciones, cuantificamos en primer lugar la tasa de aciertos en los tipos de banco de cache puesto que trabajan a velocidades diferentes. Recuérdese que el diseño no permite pérdida de información debido a descarga de condensadores. Por tanto, la tasa de acierto total es la misma que la obtenida en caches puras. La Figura 3 muestra los resultados.

Como se esperaba, la tasa de acierto en bancos eDRAM incrementa con el número de vías eDRAM. No obstante, esta tendencia no se cumple en unas pocas aplicaciones. Por ejemplo, la tasa de acierto eDRAM en *art* es casi la misma independientemente de la configuración de cache. Este comportamiento se puede explicar observando la Figura 1. En este *benchmark*, la posición MRU y la siguiente capturan un 50% de los aciertos, mientras que las localidades desde 8 hasta 15 en la pila LRU capturan casi todo el resto de aciertos. La tasa de aciertos eDRAM es, en la media, un 8%, 13% y 17% para las caches híbridas 8S-8D, 4S-12D y 2S-14D, respectivamente.

Para mejorar las prestaciones es importante que la tasa de aciertos eDRAM se mantenga tan baja como sea posible ya que un acceso a una vía eDRAM necesita más ciclos que un acceso a una vía SRAM. La degradación de prestaciones o contención de banco también viene determinada por las operaciones periódicas de refresco. Estas pérdidas no son constantes puesto que el tiempo que pasa entre dos refrescos periódicos consecutivos se hace más corto conforme el número de bloques eDRAM aumenta. Además, en las caches puras 16D, las lecturas requieren refrescar los datos porque estas operaciones son destructivas, lo que también induce contención de banco. En contraste, la contención de banco en un acierto eDRAM en caches híbridas viene inducida por la operación de swap entre bancos eDRAM y SRAM.

La Figura 4 muestra el *slowdown* de las caches analizadas respecto a la cache SRAM pura (mejor cuanto más bajo). Como esperábamos, el slowdown

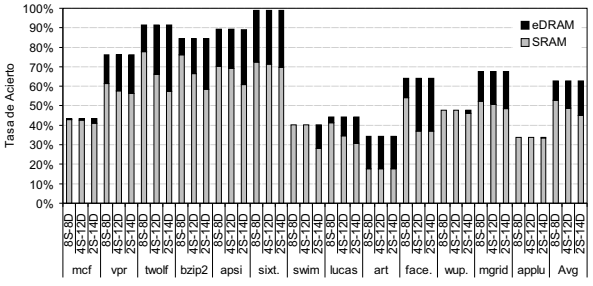


Fig. 3. Tasa de acierto separada en bancos SRAM y eDRAM.

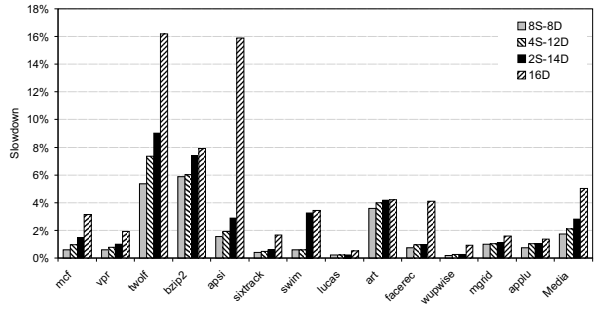


Fig. 4. Slowdown (%) de las caches analizadas respecto a la cache SRAM convencional.

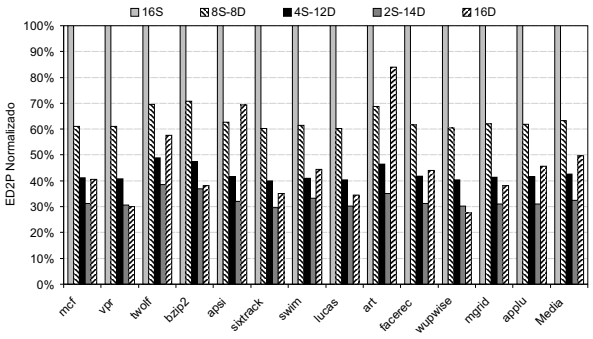


Fig. 5. ED2P normalizado (%) respecto a la cache SRAM convencional.

aumenta con el número de vías eDRAM. En general, la pérdida de prestaciones es mayor en aquellas aplicaciones con una tasa de acierto alta en bancos eDRAM. Por ejemplo, en *twolf*, la tasa de acierto eDRAM es tan alta como un 34% en la configuración 2S-14D, lo que se traduce en un 9% de slowdown. La arquitectura 16D se ve fuertemente afectada tanto por el acceso más lento a eDRAM como por contención de banco. Por ejemplo, en *twolf* y *apsi*, el slowdown es un 16.2% y 15.9%, respectivamente, lo que resulta en prestaciones inaceptables. La degradación de prestaciones es, en la media, un 1.8%, 2.2% y 2.9% en las caches híbridas 8S-8D, 4S-12D y 2S-14D, respectivamente. En la configuración 16D, el slowdown llega hasta un 5.2%. Finalmente, nótese que un número de 8 bancos es suficiente para obtener un slowdown razonable para LLCs híbridas.

B. ED2P y Área

Esta sección evalúa en primer lugar el compromiso entre prestaciones y energía utilizando la métrica ED2P puesto que refleja si el diseño híbrido es efectivo para las tecnologías futuras. Después se determinan los beneficios en área.

Para computar el ED2P, el consumo de energía se ha obtenido como la suma de leakage y energía dinámica después de lanzar 500M de instrucciones. Los resultados incluyen el consumo tanto del vector de etiquetas como del vector de datos. Nótese que el impacto de las operaciones de swap se ha tenido en cuenta, y representa entre un 11% y 13% del total de energía dinámica dependiendo de la configuración híbrida. La Figura 5 muestra el ED2P normalizado respecto a la cache 16S (mejor cuanto más bajo).

Comparado con la cache SRAM convencional, todas las configuraciones estudiadas reducen el ED2P a pesar de la degradación de prestaciones obtenida. Esto es principalmente por el hecho de que las caches híbridas y eDRAM puras reducen significativamente el leakage por diseño. Esto muestra la importancia de diseños de LLC basados en eDRAM, y especialmente diseños híbridos, ya que las caches 4S-12D y 2S-14D presentan, en la media, una mayor reducción de ED2P respecto a la cache 16D. En particular, la propuesta 4S-12D reduce el ED2P en 6 de 12 benchmarks respecto a 16D, mientras que 2S-14D obtiene mejores resultados en 10 aplicaciones. Esto es principalmente porque, respecto a las caches híbridas, la configuración 16D, a pesar de ser la cache que consume menos leakage, aumenta el tiempo de ejecución (ver Figura 4) y la energía dinámica. Esto se debe a que la cache 16D accede a todas las vías en paralelo, sus lecturas son destructivas y refresca más bloques eDRAM. Nótese que la cache eDRAM pura presenta mejor ED2P comparada con la configuración 8S-8D, puesto que la cache híbrida, al estar la mitad basada en SRAM, consume una cantidad de leakage considerable.

La reducción de ED2P es, en la media, un 57% y 66% para 4S-12D y 2S-14D, respectivamente. Este porcentaje es un 50% para 16D. Por tanto, una cache híbrida con un 12.5% de sus bancos implementados con tecnología SRAM (2S-14D) o 25% (4S-12D) es un diseño de cache mejor que la propuesta eDRAM pura.

Las caches híbridas y eDRAM puras requieren menos área que la cache SRAM puesto que las celdas eDRAM tienen mayor densidad que las SRAM. Teniendo en cuenta que, de acuerdo con CACTI, los valores de área de una celda SRAM y eDRAM son $0.296\mu m^2$ y $0.062\mu m^2$, respectivamente, para una tecnología de 45nm, la configuración 16D es la que más reduce el área del vector de datos (un 56%), seguida de cerca por la propuesta 2S-14D (49%). Como el vector de etiquetas se construye con celdas SRAM, no se pueden obtener beneficios de esta *pequeña* estructura de cache. Por tanto, la reducción de área total es un 53% y 46% para la cache eDRAM pura y 2S-14D, respectivamente.

V. Trabajos Relacionados

Para obtener las ventajas que cada tecnología de memoria ofrece, algunos trabajos anteriores se han centrado en arquitecturas de cache híbridas en diferentes estructuras del microprocesador como caches L1 y NUCAs.

Valero *et al.* [4] [5] proponen una celda híbrida de n bits, referida como *macrocelda*, que consiste en una celda SRAM, *n-1* celdas eDRAM y *n-1* transistores *puente* que permiten movimientos internos entre celdas SRAM y eDRAM. La macrocelda se utiliza para implementar caches L1 de datos asociativas por conjuntos de n vías, resultando en una cache con una vía SRAM y *n-1* vías eDRAM. Debido a la alta localidad de los datos en caches L1, se fuerza a que la vía SRAM mantenga los datos MRU. Desafortunadamente, la localidad de los datos en las LLC es

muy diferente, por lo que un número significante de accesos se harían en las celdas eDRAM lentas.

En [6], Wu *et al.* proponen dos diseños híbridos: LHCA y RHCA. En el primer diseño se implementa la cache L3 con eDRAM, MRAM o *Phase-change* RAM (PRAM), mientras que las caches L1 y L2 se implementan con tecnología SRAM. En el segundo, las caches L2 y L3 se juntan en un solo nivel para formar un par de regiones. Una región está formada por celdas SRAM y la otra por celdas eDRAM, MRAM o PRAM, mientras que la cache L1 es SRAM. Este diseño requiere más complejidad hardware que nuestra propuesta para manejar operaciones de swap entre regiones, puesto que el diseño no sólo requiere la pila LRU para todos los bloques de un conjunto, sino también un bit adicional por bloque SRAM y un contador en saturación de 2 bits por bloque eDRAM. Al contrario que en este trabajo, no se varía el tamaño de la región SRAM, la cual se fija a 256KB en todos los experimentos.

Lira *et al.* [7] proponen dos arquitecturas diferentes (homogénea y heterogénea) para NUCAs híbridas eDRAM/SRAM. En la organización homogénea, los bancos SRAM rápidos se encuentran cerca de los núcleos y almacenan los bloques más frecuentemente utilizados, mientras que los bancos eDRAM se alojan en el centro de la NUCA. Sin embargo, los datos compartidos afectan a esta propuesta porque estos datos suelen encontrarse en los bancos eDRAM lentos. Por otra parte, la arquitectura heterogénea balancea el número de bancos SRAM y eDRAM con la posición de los mismos (cerca de los núcleos o en el centro de la NUCA). Los resultados muestran que el mismo número de bancos SRAM y eDRAM ofrecen el mejor compromiso entre prestaciones, consumo y área en esta organización.

VI. Conclusiones

Las memorias cache han sido implementadas normalmente con tecnología SRAM para conseguir velocidad alta de acceso. Sin embargo, esta tecnología presenta inconvenientes importantes como consumo de leakage y área. Por otra parte, los avances tecnológicos han permitido que las caches se implementen con tecnología eDRAM, que presenta un consumo menor de leakage y área a expensas de un acceso no tan rápido como SRAM. Como ambas tecnologías son compatibles CMOS, se pueden combinar en el mismo die durante el proceso de fabricación. La tecnología eDRAM ha sido utilizada en LLCs, donde la energía es un aspecto de diseño importante.

En este artículo, ambas tecnologías SRAM y eDRAM se han combinado en la LLC, resultando en un diseño de cache híbrida novel consistente en bancos SRAM y eDRAM. La proporción óptima de bancos SRAM se ha explorado para obtener el mejor compromiso entre prestaciones, energía y área. Se han considerado mecanismos arquitectónicos para mantener los datos más recientemente accedidos en los bancos SRAM y para evitar lecturas destructivas innecesarias en los bancos eDRAM.

Los resultados experimentales mostraron que, comparado con una LLC SRAM convencional con la misma capacidad, la pérdida de prestaciones nunca excede, en la media, 2.9%, mientras que la reducción de área es un 46% para una cache híbrida de 1MB-16vías. Para una tecnología de 45nm, la métrica ED2P ha confirmado que, en la media, la cache híbrida es un diseño mejor que la cache SRAM independientemente del número de bancos eDRAM. Más aún, al contrario de lo esperado, los resultados mostraron que el diseño híbrido es mejor que una cache eDRAM cuando el porcentaje de bancos SRAM es 12.5% y 25%.

Finalmente, como trabajo futuro pretendemos evaluar el potencial de la arquitectura híbrida en sistemas *Chip Multi-Processor* (CMP), donde diferentes hilos comparten la LLC y la localidad de los datos cambia.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO) y por los fondos de Plan E, mediante subvenciones a los proyectos TIN2009-14475-C04-01 y TIN2012-38341-C04-01. Adicionalmente, también ha sido financiado por el Programa de Apoyo a la Investigación y Desarrollo (PAID-05-12) de la Universitat Politècnica de València (SP20120748).

Referencias

[1] B. Sinharoy et al., "IBM POWER7 multicore server processor," *IBM J. Research and Development*, vol. 55, no. 3, 2011.

[2] R. E. Matick and S. E. Schuster, "Logic-Based eDRAM: Origins and Rationale for Use," *IBM J. Research and Development*, vol. 49, no. 1, pp. 145–165, 2005.

[3] *http://www.uniramtech.com/embedded_dram.php.*

[4] A. Valero et al., "An Hybrid eDRAM/SRAM Macrocell to Implement First-Level Data Caches," in *Proc. 42th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2009, pp. 213–221.

[5] A. Valero et al., "Design, Performance, and Energy Consumption of eDRAM/SRAM Macrocells for L1 Data Caches," *IEEE Trans. Computers*, vol. 61, no. 9, pp. 1231–1242, 2012.

[6] X. Wu et al., "Hybrid Cache Architecture with Disparate Memory Technologies," in *Proc. 36th Ann. Int'l Symp. Computer Architecture*, 2009, pp. 34–45.

[7] J. Lira et al., "Implementing a hybrid SRAM / eDRAM NUCA architecture," in *Proc. 18th Int'l Conf. High Performance Computing*, 2011, pp. 1–10.

[8] S. Petit et al., "Exploiting Temporal Locality in Drowsy Cache Policies," in *Proc. 2nd Conf. Computing Frontiers*, 2005, pp. 371–377.

[9] *Standard Performance Evaluation Corporation, available online at http://www.spec.org/cpu2000.*

[10] T. Kirihata et al., "An 800-MHz Embedded DRAM with a Concurrent Refresh Mode," *IEEE J. Solid-State Circuits*, vol. 40, no. 6, pp. 1377–1387, 2005.

[11] B. Keeth et al., *DRAM Circuit Design. Fundamental and High-Speed Topics*, John Wiley and Sons, Inc., Hoboken, New Jersey, 2008.

[12] A. Valero et al., "Impact on Performance and Energy of the Retention Time and Processor Frequency in L1 Macrocell-Based Data Caches," *IEEE Trans. Very Large Scale Integration Systems*, vol. 20, no. 6, pp. 1108–1117, 2012.

[13] D. C. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *Computer Architecture News*, vol. 25, no. 3, pp. 13–25, 1997.

[14] S. Thoziyoor et al., "CACTI 5.1," *Hewlett-Packard Laboratories, Palo Alto, Technical Report*, 2008.

Planificación Considerando Degradación de Prestaciones por Contención

Josué Feliu [1], Julio Sahuquillo [1], Salvador Petit [1] y José Duato [1]

Resumen— **Algunos trabajos recientes se han centrado en la planificación de tareas con el fin de reducir la contención en el ancho de banda disponible. Los planificadores propuestos típicamente calculan el ancho de banda medio requerido por la carga y tratan de mantener este consumo a lo largo de toda la ejecución, lo que provoca que la degradación de cada benchmark sea bastante constante independientemente de la carga.**

En este artículo se analiza, en primer lugar, la degradación de prestaciones que sufren los benchmarks al ejecutarse en situaciones con ancho de banda consumido alto y constante, simulando la situación descrita anteriormente. A partir de este análisis, se propone una estrategia de planificación que favorece la ejecución de los benchmarks con mayor degradación en situaciones de menor contención.

La propuesta de planificación se ha evaluado en un sistema real mejorando las prestaciones ofrecidas tanto por Linux como por planificadores actuales que consideran la contención en el ancho de banda. La aceleración media obtenida del tiempo de ejecución de las cargas es del 6.64 % respecto a Linux, con una aceleración máxima superior al 9.5 %.

Palabras clave— **Contención de memoria, planificación, jerarquía de cache, degradación de prestaciones**

I. Introducción

LOS procesadores multinúcleo se han convertido en el diseño común para los microprocesadores de altas prestaciones. Estos multiprocesadores en un chip (CMP) incorporan varios núcleos en un mismo chip y tienen potencial para ofrecer unas prestaciones superiores a las de los procesadores mononúcleo reduciendo, además, los problemas de consumo, encapsulado y refrigeración que éstos presentaban.

El principal cuello de botella de los CMPs se encuentra en la interconexión entre el propio chip y la memoria principal. En los primeros CMPs, la componente más importante de este cuello de botella era la latencia de acceso a memoria. Sin embargo, a medida que las capacidades multinúcleo y multihilo de los procesadores han ido creciendo, la contención en el ancho de banda disponible para acceder a memoria se ha convertido en la principal preocupación, pues además de reducir las prestaciones de los CMPs complica su escalabilidad.

Para tratar de reducir el número de accesos a memoria principal, los procesadores actuales implementan jerarquías de memoria con tres niveles de cache y grandes caches de último nivel (LLC) que ya alcanzan varias decenas de MBs. Además, para mejorar la utilización de estas caches tan grandes, muchas veces se implementan como caches compartidas entre un subconjunto de núcleos. Así, procesadores como el Intel Dunnington o el Power 5 utilizan caches compartidas en dos niveles de la jerarquía de cache. La presencia de núcleos multihilo también provoca un comportamiento similar en caches privadas como la L1, pues son varios hilos de ejecución los que comparten el ancho de banda disponible para acceder a esta cache.

Las estrategias de planificación también pueden ayudar a reducir la contención en el acceso a memoria principal. Por ello, recientemente se han propuesto estrategias que tienen en cuenta el ancho de banda requerido por los procesos en ejecución para planificarlos de manera que no se exceda la capacidad del bus. De este modo, se consigue evitar que una excesiva demanda de acceso a memoria principal cree contención, provocando pérdida de prestaciones. Además, si el número de peticiones es muy alto y no se distribuye correctamente puede provocar la saturación del bus, lo que implicaría una caída de prestaciones mucho más drástica.

El artículo presenta dos aportaciones principales. En primer lugar se analiza la degradación que sufren los benchmarks cuando se ejecutan con planificadores que consideran la contención en el acceso a memoria principal. El análisis muestra que esta degradación está relacionada con el ancho de banda de memoria requerido por los benchmarks y permite diferenciar entre benchmarks *muy sensibles* a esta degradación y benchmarks *poco sensibles*. A partir de este análisis, y tomando como base un planificador que considera la contención a lo largo de toda la jerarquía de memoria, se propone una estrategia de planificación que favorece la ejecución de los procesos más sensibles en situación de menor contención de memoria para minimizar su degradación de prestaciones.

Los resultado experimentales muestran que el planificador propuesto mejora las prestaciones de planificadores que consideran la contención actuales [1] [2]. Comparado con Linux, ofrece una aceleración media del 6.64 % para las cargas evaluadas, con una mejora de prestaciones máxima por encima del 9.5 %. Además, cabe destacar que la evaluación se ha realizado con un procesador de cuatro núcleos con lo que es de esperar que la contención, y por tanto, el incremento de prestaciones sea mayor en futuros procesador con un mayor número de núcleos.

El resto del artículo se organiza como sigue. La Sección II presenta el estado del arte. La Sección III describe la plataforma experimental. La Sección IV analiza la degradación de prestaciones provocada por contención al ejecutar procesos con un planificador que considera la contención en el acceso a memoria

[1]Departamento de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, e-mail: jofepre@fiv.upv.es, {jsahuqui, spetit, jduato}@disca.upv.es

principal. La Sección V explica el algoritmo de planificación propuesto. La Sección VI describe la metodología empleada y evalúa las prestaciones de la propuesta. Finalmente, la Sección VII presenta las conclusiones del trabajo.

II. Estado del arte

Un gran número de trabajos de investigación recientes se han centrado en los recursos compartidos por núcleos o threads en los actuales CMPs. Dado que el ancho de banda es uno de los recursos que presenta un mayor impacto en las prestaciones de los sistemas, muchos trabajos han propuesto estrategias de planificación y ubicación de procesos tratando de evitar los efectos negativos provocados por la contención.

Alguno trabajos preliminares [3], [1] en el tema se centraron en la contención en el acceso a memoria principal. Antonopoulos et al. [3] propuso varias políticas de planificación tratando de que el ancho de banda total consumido en cada quantum se aproximara al ancho de banda pico del sistema. Sin embargo, Xu et al. [1] demostró que la contención podía aparecer incluso cunado el ancho de banda total consumido está por debajo del pico debido a los patrones de acceso irregulares que pueden presentar los procesos. Por ello, propuso distribuir el total de accesos a memoria de un carga a lo largo de su tiempo de ejecución con el fin de minimizar la contención mediante una política de planificación.

En relación a la contención en la LLC, Tang et al. [4] analizó el impacto de la compartición de los recursos de memoria en aplicaciones de centros de datos y observó que la compartición inapropiada de los recursos de la LLC puede provocar una degradación importante. Para abordarla, los autores propusieron dos aproximaciones la asignación de los procesos a los núcleos de forma que se mejora la compartición de recursos en la LLC. En [5], Zhuralev et al. propusieron un algoritmo de planificación que consideraba, entre otros, la contención por el espacio en la LLC. Knauerhase et al. [6] presentaron un planificador que observaba las propiedades de los procesos utilizando contadores de prestaciones para seleccionar los procesos a ejecutar reduciendo la interferencia en las caches. Fedorova et al. [7] propusieron un algoritmo de planificación para favorecer los procesos cuyas prestaciones se ven más afectadas por una compartición injusta de la cache.

Por último, trabajos más recientes consideran diversos niveles de la jerarquía de cache. Feliu et al. [2] abordó la contención en el ancho de banda a lo largo de la jerarquía de memoria para minimizar la contención equilibrando los accesos a cada una de las caches del sistema, mientras que Eyerman y Eeckhout [8] utilizan los fallos en los diferentes niveles de cache para estimar la simbiosis entre procesos de una manera probabilística.

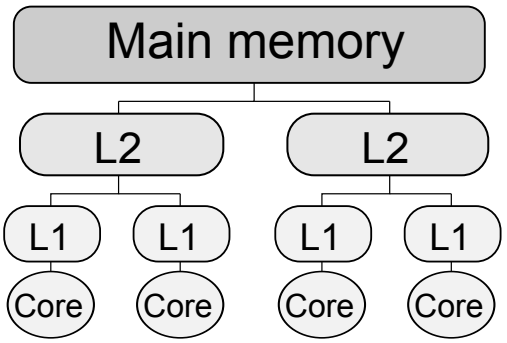


Fig. 1. Jerarquía de memoria del Intel Xeon X3320

III. Plataforma experimental

La plataforma experimental sobre la que se han realizado las evaluaciones dispone de un procesador Intel Xeon X3320 [9] con cuatro núcleos que funciona a una frecuencia de reloj de 2.5 GHz y cuenta con una memoria principal de 4 GB DDR2. La jerarquía de cache del procesador dispone de dos niveles de cache. La cache L1 es privada para cada núcleo y dispone de 32 KB para datos y 32 KB para instrucciones, mientras que el nivel L2 está formado por dos caches de 3 MB, cada una compartidas por dos núcleos. La Figura 1 presenta la jerarquía de memoria del procesador, donde el ancho de banda para acceder a L2 y memoria principal es compartido por varios núcleos, convirtiéndose en un posible cuello de botella.

El sistema utiliza la distribución Fedora Core 10 Linux con el núcleo 2.6.29. La librería *libpfm* y el programa *perfmon2* se utilizan para acceder a los contadores de prestaciones [10]. Para medir las prestaciones de los procesos se mide el número de instrucciones y ciclos que ejecuta cada uno, mientras que las peticiones a caches y memoria principal se utilizan para obtener su ancho de banda consumido en cada nivel de la jerarquía.

IV. Inconvenientes de la planificación considerando la contención

El comportamiento típico de los planificadores que consideran la contención en el ancho de banda de acceso a memoria principal consiste en planificar los procesos de forma que el ancho de banda total consumido en cada quantum sea relativamente alto y constante.

El experimento que proponemos para analizar la degradación de prestaciones en esta situación consiste en ejecutar cada benchmark junto con tres co-runners de forma que el ancho de banda total consumido sea de 30 transacciones por microsegundo, el cual es el valor medio consumido por las cargas evaluadas en la Sección VI. Como co-runner se utiliza un microbenchmark [2], configurado en función del benchmark a analizar para obtener la tasa global de accesos a memoria deseada.

La Figura 2 presenta los resultados de este experimento, ordenando los benchmarks de mayor a menor degradación. Las barras muestran, para cada benchmark, la degradación sufrida en el experimento des-

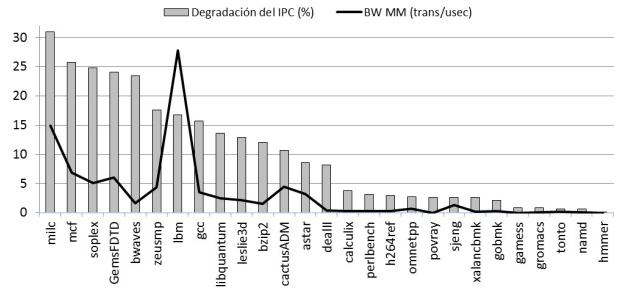


Fig. 2. Degradación del IPC al ejecutarse con tres micro-benchmarks y un BW_{L2} global de 30 trans/usec

crito, mientras que la línea indica el ancho de banda de acceso a memoria principal consumido por cada benchmark (BW_{MM}) cuando se ejecuta en solitario.

Se puede observar que la degradación y el ancho consumido por los benchmarks están muy relacionados. Todos los benchmarks con un ancho de banda superior a 2 trans/usec sufren una degradación de prestaciones superior al 10 % (excepto *astar*), mientras que por el contrario, todos los benchmarks con un ancho de banda inferior no llegan a sufrir una degradación del 5 % (excepto *dealII*). Además, sin considerar *lbm* que con un ancho de banda de 27 trans/usec sufre una degradación alrededor del 17 %, se cumple bastante bien que la tendencia decreciente del ancho de banda consumido implica la reducción de la degradación de prestaciones.

A partir de la pérdida de prestaciones observada, podemos clasificar los benchmarks en dos grupos. Por un lado, los benchmarks cuya degradación es inferior al 5 % se clasifican como *poco sensibles*, pues no se ven muy afectado por la contención en el acceso a memoria principal. Por otro lado, los benchmarks con una degradación superior al 5 % se agrupan como benchmarks *muy sensibles*, pues sufren una pérdida de prestaciones provocada por contención en el acceso a memoria muy superior. Las dos categorías están bien diferenciadas, ya que solo dos benchmarks presentan una degradación entre el 5 % y el 10 %.

La diferencia en la degradación observada en este experimento motiva el diseño del planificador propuesto en la Sección V-B, que utilizará los resultados de este experimento para favorecer la ejecución de los procesos con mayor degradación en situaciones de menor contención con el fin de mejorar las prestaciones globales de las cargas.

V. Planificación considerando el ancho de banda

A. *Planificación considerando contención en la jerarquía de memoria*

Como planificador base se utiliza un planificador que considera el ancho de banda a lo largo de toda la jerarquía de memoria [2], abordando el ancho de banda en cada punto de contención y planificando los procesos para minimizar la contención desde la memoria principal hasta el nivel más alto en la jerarquía con varias caches compartidas.

Para abordar la contención en el ancho de banda de acceso a memoria principal, se calcula el ancho de banda medio requerido por la carga ($AVG\_BW\_MM_{Wk}$) en una etapa de inicialización. El $AVG\_BW\_MM_{Wk}$ se calcula como la suma total de accesos a memoria de los benchmarks que forman la carga dividido por su tiempo de ejecución. Por tanto, seleccionando los procesos de forma que el ancho de banda consumido en cada quantum se acerque al $AVG\_BW\_MM_{Wk}$, el planificador consigue distribuir los accesos a memoria de manera equilibrada a lo largo de la ejecución, reduciendo la contención.

A continuación se aborda la contención a lo largo de la jerarquía de cache. Para cada nivel con al menos dos caches compartidas, se ubican los procesos seleccionados anteriormente de manera que el número de accesos sobre cada estructura de cache quede equilibrado. De este modo se consigue minimizar la contención en todos los niveles y por tanto maximizar las prestaciones de los procesos.

B. *Planificación considerando degradación de prestaciones por contención*

Con el fin de minimizar la pérdida de prestaciones provocada por contención, la degradación observada en el análisis de la Sección IV se utiliza como información útil para el planificador a la hora de seleccionar los procesos a ejecutar. La mejora que se propone consiste en favorecer la ejecución de los benchmarks clasificados como *muy sensibles*, en detrimento de los benchmarks *poco sensibles*.

Para incluir la mejora en el algoritmo de planificación, se utiliza un coeficiente de penalización que se aplicará tanto al cálculo del $AVG\_BW\_MM_{Wk}$ como a la función utilizada para seleccionar los procesos a ejecutar (función fitness). El coeficiente de penalización se define como una parte proporcional de la degradación del IPC sufrida por cada benchmark (Sección IV). Para obtener las máximas prestaciones se evaluaron diferentes coeficientes, obteniendo los mejores resultados cuando el coeficiente de penalización se define como una quinta parte de la degradación sufrida para los benchmarks *muy sensibles*. Para los benchmarks *poco sensibles* el coeficiente de penalización utilizado es cero.

El Algoritmo 1 presenta el pseudocódigo de la política de planificación propuesta, extendiendo la propuesta de planificación descrita en la Sección V-A. Su funcionamiento completo puede verse como un paso de inicialización y tres fases. En la inicialización se calcula el ancho de banda medio ponderado ($P\_AVG\_BW\_MM_{Wk}$), incluyendo el coeficiente de penalización de cada benchmarks que no se utilizaba en el planificador base. Después, las tres fases se repiten hasta que finaliza la ejecución de la carga.

En la primera fase (líneas 2 a 8) el planificador bloquea los procesos en ejecución, actualiza el ancho de banda en cada nivel de cache para cada proceso e inserta los procesos en la cola de procesos pendientes. El ancho de banda se actualiza con la utilización del último quantum en ejecución mediante los va-

lores obtenidos de los contadores de prestaciones y se utiliza como el ancho de banda previsto para el quantum siguiente.

En la segunda fase (líneas 9 a 17) el planificador selecciona los procesos que se ejecutarán durante el siguiente quantum a partir de su ancho de banda de memoria principal requerido. El primer proceso de la cola siempre se elige para evitar la inanición de algún proceso por sus requisitos, y el resto de procesos, hasta completar el número de núcleos, se elige con la función fitness, que ha sido modificada respecto al algoritmo base para considerar los coeficientes de penalización en la selección de procesos.

La inclusión del coeficiente de penalización en el P_AVG_BW_MM$_{Wk}$ provoca que este sea mayor que el AVG_BW_MM$_{Wk}$ calculado anteriormente. No obstante, cuando un proceso *muy sensible* se selecciona para su ejecución, tanto su ancho de banda requerido como su coeficiente de penalización se sustraen del ancho de banda restante. Con ello se consigue que cuando se seleccionan procesos *muy sensibles*, el ancho de banda total se aproxime a un valor inferior al AVG_BW_MM$_{Wk}$, favoreciendo su ejecución en situaciones con menor contención.

Finalmente, la tercera fase (líneas 18 a 28) se mantiene del algoritmo base. En esta fase los procesos se van asignando a subconjuntos de núcleos de forma que las peticiones a cada una de las caches compartidas queden equilibradas para minimizar la contención a lo largo de la jerarquía de cache.

En resumen, con la mejora propuesta se consigue ejecutar los procesos *muy sensibles* en periodos de ejecución y con co-runners donde el ancho de banda total requerido de memoria principal es menor, minimizando su degradación de prestaciones. Por el contrario, los procesos *poco sensibles* se ejecutan en situaciones donde el ancho de banda medio requerido es ligeramente superior, pero no sufren una degradación significativamente mayor, con lo que se consigue una mejora global de las prestaciones.

Algorithm 1 Planificación considerando contención en la jerarquía de memoria y degradación de prestaciones por contención

Require: Conocer el tiempo de ejecución y BW$_{MM}$ de los benchmarks en solitario.

1: $AVG\_BW\_MM = \frac{\sum_{p=0}^{P}(BW_{MM}^{p}+Coef.Penal.^{p})*T^{p}}{\frac{\sum_{p=0}^{P}T^{p}}{\#nucleos}}$

2: **while** haya procesos sin finalizar **do**
3: Detiene los procesos en ejecución y los inserta al final de la cola
4: **for** cada proceso P ejecutado el último quantum **do**
5: **for** cada nivel L de la jerarquía con caches compartidas **do**
6: Actualiza el BW de P en el nivel de cache L
7: **end for**
8: **end for**
9: BW$_{Restante}$ = AVG_BW_MM
10: Selecciona el primer proceso de la cola P$_{head}$
11: BW$_{Restante}$ − = $BW_{MM}^{P\_head} + Coef.Penal.^{P\_head}$
12: CPU$_{Restante}$ = #nucleos−1
13: **while** CPU$_{Restante}$ > 0 **do**
14: selecciona el proceso P que maximice
15: FITNESS(p) = $\frac{1}{\left|\frac{BW_{Restante}}{CPU_{Restantes}} - (BW_{requerido}^{P} + Coef.Penal.^{p})\right|}$
16: BW$_{Restante}$ − = (BW$_{LLC}^{P}$+Coef.Penal.^{P}), CPU$_{Restante}$ − −
17: **end while**
18: **for** cada nivel i en la jerarquía de cache con caches compartidas empezando desde la LLC **do**
19: AVG$_{BW}$(L$_{i-1}$)=$\frac{\sum BW_{L(i)}}{\#Caches\ \ at\ \ Li}$
20: **for** cada cache en el nivel L$_i$ **do**
21: BW$_{Restante}$ = AVG$_{BW}$(L$_i$)
CPU$_{Restante}$ = # núcleos compartiendo la cache
22: **while** CPU$_{Restante}$ > 0 **do**
23: A partir de los procesos restantes seleccionados para compartir la cache en el nivel inmediatamente inferior, selecciona el proceso P que maximice
24: FITNESS(p) = $\frac{1}{\left|\frac{BW_{Restante}}{CPU_{Restante}} - BW_{requerido}^{P}\right|}$
25: BW$_{Restante}$ − = BW$_{requerido}^{P}$
CPU$_{Restante}$ − −
26: **end while**
27: **end for**
28: **end for**
29: Desbloque los procesos y los asigna a los núcleos seleccionados
30: Duerme durante el quantum
31: **end while**

VI. Evaluación de la propuesta

A. Metodología de evaluación y diseño de las cargas

Para evaluar las prestaciones del planificador se ha diseñado un conjunto de diez cargas. Las cargas 1 a 7 están formadas por ocho procesos (el doble que núcleos en la plataforma experimental), mientras que las cargas 8 a 10 cuentan con doce procesos.

Para comparar las prestaciones del planificador propuesto se han implementado los siguientes algoritmos: i) planificador de Linux ii) planificador considerando la contención en el acceso a memoria principal (MaS) iii) planificador considerando la contención a lo largo de la jerarquía de memoria (CaS) iv) planificador considerando la degradación de los procesos por contención (DaS).

Los planificadores se han implementado como planificadores a nivel de usuario y comparten la mayor parte del código, lo que favorece una comparación más justa. La diferencia entre ellos radica en la selección y ubicación de los procesos. Para simular el comportamiento de Linux, todos los procesos se seleccionan cada quantum, de forma que se deja a Linux la decisión de que procesos ejecutar y los núcleos en los que hacerlo. El planificador MaS selecciona los procesos tratando de aproximar el ancho de banda consumido en cada quantum al AVG_BW_MM$_{Wk}$ [1]. El planificador CaS además de seleccionar los procesos como MaS los ubica en los núcleos apropiados para equilibrar los accesos a cada una de las caches compartidas (Sección V-A). Y por último, el planificador DaS sigue el algoritmo presentado en la Sección V-B.

B. Prestaciones del planificador

La Figura 3 presenta la aceleración del tiempo de ejecución de las cargas obtenida por los planificadores MaS, CaS y DaS respecto al tiempo obtenido con

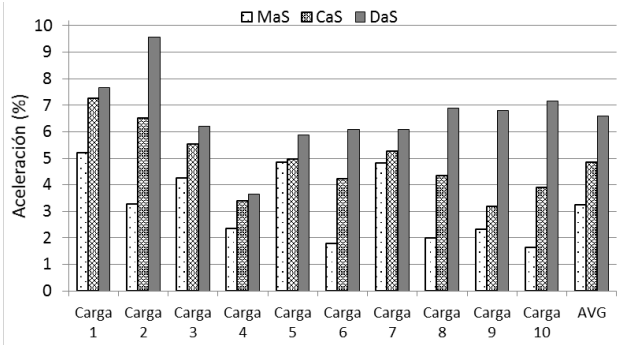


Fig. 3. Aceleración del tiempo de ejecución de las cargas con los planificadores MaS, CaS y DaS respecto a Linux

el planificador de Linux. Se puede observar que, para todas las cargas evaluadas, los tres planificadores mejoran el tiempo de ejecución obtenido por Linux. Las aceleraciones obtenidas por el planificador MaS, que únicamente considera la contención en la memoria principal, varían entre 1.63 % y 5.22 %, con un valor medio del 3.61 %. Al extender esta propuesta para que se consideren los anchos de banda a lo largo de toda la jerarquía de memoria, las prestaciones se incrementan. De este modo, el planificador CaS obtiene aceleraciones entre 3.38 % y 7.26 % con un valor medio de 5.38 %.

Las prestaciones de ambos planificadores se mejoran utilizando el planificador propuesto. El algoritmo de planificación DaS mejora al algoritmo CaS para considerar también la degradación que los benchmarks sufren al ejecutarse con un consumo de ancho de banda de memoria constante. Las aceleraciones que obtiene varían entre 3.66 % y 9.56 %, con un valor medio del 6.64 %. Además de mejorar la aceleración media, DaS está cerca de doblar la aceleración obtenida por MaS en todas las cargas y la triplica en la mitad de ellas (cargas 2, 6, 8, 9 y 10).

Para comparar los beneficios de la planificación DaS respecto a la planificación CaS, calculamos el porcentaje de benchmarks en cada carga que reducen su tiempo de ejecución (aceleración) y el de los que alargan su ejecución (deceleración). La Figura 4 muestra estos resultados. Se puede observar que nueve de las diez cargas son beneficiadas por una planificación DaS. Además, seis de las cargas presentan aceleración en un 60 % de los benchmarks. Esto se

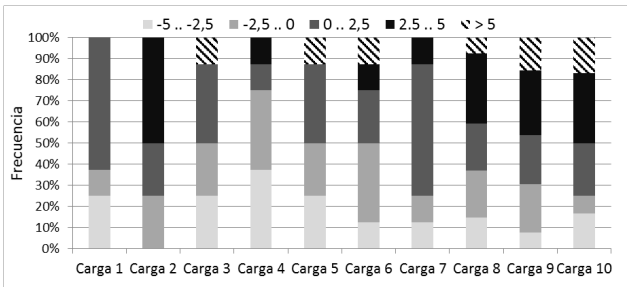


Fig. 4. Aceleración del tiempo de ejecución los benchmarks de cada carga con DaS respecto a CaS

produce gracias al coeficiente de penalización incluido, que favorece la ejecución de los benchmarks *muy sensibles* en situaciones con menor contención, obteniendo por tanto, una aceleración de sus prestaciones. Por el contrario, los benchmarks *poco sensibles* se deberán ejecutar en escenarios con mayor contención, decelerando su ejecución, aunque con un menor impacto en las prestaciones.

Únicamente en la carga 4 el porcentaje de benchmarks con deceleración es superior al de aceleración, y aún así se consigue mejorar el tiempo de ejecución de la carga. Esto se debe a que últimos procesos de la carga se ejecutan con menor contención cuando quedan menos procesos que núcleos y si bien pueden conseguir mejorar sus prestaciones individuales, también pueden alargar el tiempo de ejecución de la carga completa.

No obstante, resulta más llamativo comprobar la evolución del ancho de banda a lo largo de una ejecución completa para apreciar mejor el incremento de prestaciones. Recordemos que la mejora que proporciona el planificador CaS respecto a MaS se basa en balancear las transacciones que acceden a cada una de las caches compartidas. En nuestro caso, la jerarquía de cache únicamente cuenta con un nivel L2 con caches compartidas, por lo que la mejora proviene de minimizar la diferencia entre los accesos a cada una de ellas. Con esto, se consigue reducir la contención en las L2 y por tanto, minimizar la degradación de prestaciones de los procesos en ejecución.

Para mostrar como se consigue mejorar las prestaciones de las cargas balanceando los accesos a las caches L2, nos centramos en la carga 2, donde la diferencia entre cada uno de los planificadores es más significativa. La Figura 5 muestra la evolución tem-

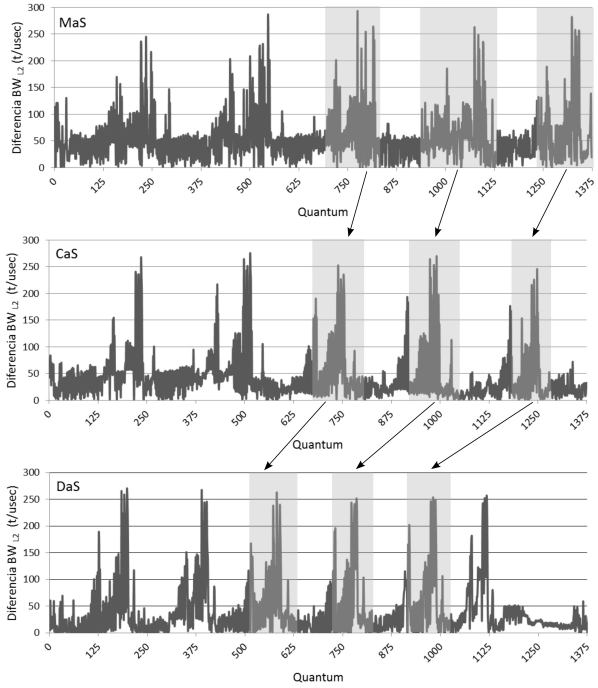


Fig. 5. Evolución con el tiempo de la diferencia del BW_{L2}

poral de la diferencia entre el ancho de banda que accede a cada una de las dos caches L2 durante los primeros 1375 quantums de ejecución. Si comparamos las gráficas para MaS y CaS, podemos observar que al ejecutar la carga con CaS, los picos de diferencia se adelantan, mostrando un avance en la ejecución de los benchmarks. Además de este avance temporal, también se consigue reducir la diferencia entre los accesos a cada una de las dos caches L2 tanto en su duración como en magnitud. Especialmente notoria es la diferencia al observar la marca de graduación de 50 trans/usec, que queda completamente oculta por la diferencia en la planificación MaS, pero aparece visible durante algunos periodos significativos en la planificación CaS, por ejemplo, entre los quantums 525 y 625 o 1025 y 1150.

Si además observamos la gráfica para el planificador propuesto DaS, se observa una clara mejoría respecto a CaS. A pesar de que la modificación propuesta sobre el planificador no se dirigía directamente a reducir las diferencias entre los accesos a cada una de las dos caches, sino a ejecutar en situaciones con menor contención los benchmarks que sufrían una mayor degradación, la gráfica muestra claramente que la mejora propuesta en la selección de los procesos también repercute en reducir las diferencias, dejando la marca de 50 trans/usec visible durante un mayor número de quantums. Además, se observa un gran avance en la ejecución de los benchmarks con mayores anchos de banda sin que ello implique sufrir una mayor degradación.

VII. Conclusiones

Este trabajo se ha centrado en la mejora de un planificador que considera la contención en el ancho de banda a lo largo de la jerarquía de cache, para considerar también la degradación de prestaciones implícita que sufren los benchmarks al ejecutarse con planificadores que consideran la contención en el acceso a memoria principal.

Para ello, en primer lugar se ha analizado la degradación de prestaciones que sufren los benchmarks al ejecutarse en una situación con un ancho de banda consumido relativamente alto y estable, tratando de simular la situación en la que posteriormente se ejecutarán los benchmarks usando planificadores que consideran la contención. Los resultados obtenidos permiten observar que existe bastante relación entre el ancho de banda consumido por un benchmark y su degradación en esta situación. En función de esta degradación, los benchmarks se clasifican en dos grupos. El primer grupo incluye los benchmarks *muy sensibles* a este tipo de contención, y el segundo grupo incluye los benchmarks *poco sensibles*.

Para aprovechas esta observación, se ha diseñado un algoritmo de planificación que mejora la propuesta presentada en [2]. El algoritmo que proponemos utiliza la clasificación realizada en el análisis de la degradación para definir un coeficiente de penalización. Con este coeficiente se favorece la ejecución de los benchmarks que sufren mayor degradación en situaciones donde los procesos que se ejecutan concurrentemente presentan una menor tasa de acceso a memoria. De este modo se consigue reducir su degradación de prestaciones. Aunque esto implica que los benchmarks con menor degradación se ejecuten en situaciones de mayor contención, su degradación de prestaciones es inferior a la mejora de los procesos *muy sensibles*, obteniendo por tanto, una mejora en el tiempo de ejecución de las cargas.

Los resultados experimentales muestran que la mejora propuesta reduce el tiempo de ejecución de las carga evaluadas en comparación con Linux y los planificadores MaS y CaS. La aceleración media obtenida con el planificador propuesto con respecto a Linux es del 6.64 %, con picos que superan el 9.5 %. Además, cabe considerar que a medida que el número de núcleos y la contención en el acceso a memoria crezca, las prestaciones obtenidas con este tipo de planificación deberían verse incrementadas.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competividad (MINECO) y por los fondos FEDER mediante la subvención TIN2012-38341-C04-01, así como por el Programa de Apoyo a la Investigación y Desarrollo (PAID-05-12) de la Universitat Politècnica de València mediante la subvención SP20120748.

Referencias

[1] D. Xu, C. Wu, and P.-C. Yew, "On mitigating memory bandwidth contention through bandwidth-aware scheduling," in *International Conference on Parallel Architectures and Compilation Techniques, PACT*, 2010, pp. 237–248.

[2] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, "Understanding Cache Hierarchy Contention in CMPs to Improve Job Scheduling," in *International Parallel Distributed Processing Symposium, IPDPS*, 2012, pp. 508 –519.

[3] C. D. Antonopoulos, D. S. Nikolopoulos, and T. S. Papatheodorou, "Realistic workload scheduling policies for taming the memory bandwidth bottleneck of smps," in *International Conference on High Performance Computing, HiPC*, 2004, pp. 286–296.

[4] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M.-L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *International Symposium on Computer Architecture, ISCA*, 2011, pp. 283–294.

[5] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*, 2010, pp. 129–142.

[6] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn, "Using os observations to improve performance in multicore systems," *IEEE Micro*, vol. 28, no. 3, pp. 54–66, may 2008.

[7] A. Fedorova, S. Blagodurov, and S. Zhuravlev, "Managing contention for shared resources on multicore processors," *Communications of the ACM*, vol. 53, no. 2, pp. 49–57, feb 2010.

[8] S. Eyerman and L. Eeckhout, "Probabilistic job symbiosis modeling for smt processor scheduling," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS*, 2010, pp. 91–102.

[9] G. Varghese, J. Sanjeev, T. Chao, S. Ken, et al., "Penryn: 45-nm next generation intel core 2 processor," in *Solid-State Circuits Conference, 2007. ASSCC '07. IEEE Asian*, 2007, pp. 14–17.

[10] S. Eranian, "What can performance counters do for memory subsystem analysis?," in *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, 2008, pp. 26–30.

Object-based Data Sharing for High Performance Virtualized Systems

Pablo Llopis
Dept. de Informática
Univ. Carlos III de Madrid
pllopis@arcos.inf.uc3m.es

Javier García Blas
Dept. de Informática
Univ. Carlos III de Madrid
fjblas@arcos.inf.uc3m.es

Florin Isaila
Dept. de Informática
Univ. Carlos III de Madrid
florin@arcos.inf.uc3m.es

Jesús Carretero
Dept. de Informática
Univ. Carlos III de Madrid
jcarrete@arcos.inf.uc3m.es

Resumen— **With scientific computing in the cloud gaining popularity and using every time larger data sets, high performance storage I/O in virtualized environments is substantially increasing in importance. However, exploiting the performance potential of the storage I/O on today's virtualized architectures is complex, due to the limitations of POSIX standard for storage I/O and the lack of integration of related mechanisms such as data sharing, storage I/O coordination, relaxing the consistency semantics, and data locality awareness.**

In this paper we propose VIDAS (Virtualized DAta Sharing), an object-based virtualized data store that targets to integrate the above mechanisms through a simple and powerful interface. VIDAS can be used to efficiently and consistently share access to externally stored data in virtualized environments based on a shared pool of storage objects. We show how VIDAS can be used for straightforwardly implementing I/O coordination and data sharing for two common high-performance patterns: inter-domain write-reader and inter-domain collective I/O. We present the implementation and evaluation of VIDAS for the Xen virtualization solution. In addition, we present a novel mechanism for efficiently sharing memory among an arbitrary number of virtual machines.

Palabras clave— **HPC, cloud computing, storage I/O virtualization, data sharing**

I. Introduction

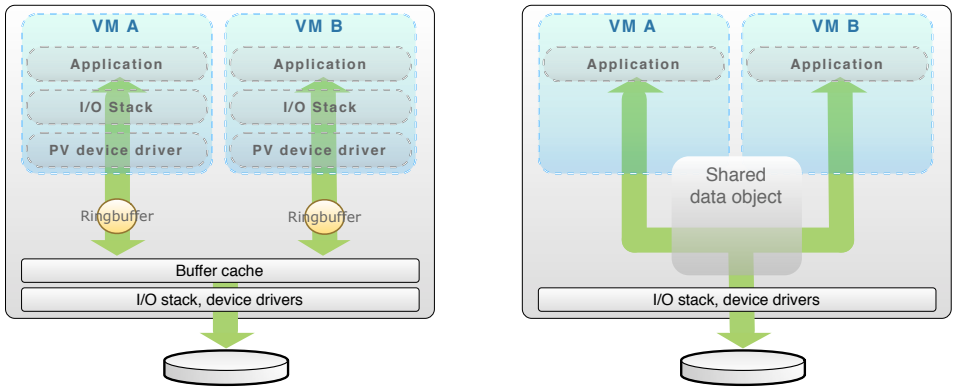
IN the last years the clouds based on virtualized architectures have become increasingly attractive for running scientific codes. On one hand clouds offer cost- and resource-efficient solutions due to on-demand scalability and pay-per use model. On the other hand virtualization technologies provide several advantages including flexibility to run customized versions of the operating system, performance isolation of workloads, improved security and reliability, migration for load balancing and fault tolerance.

One of the most important current challenges for broadening the adoption of virtualized cloud solutions by the HPC community is providing efficient access to data-intensive scientific applications. Scientific applications require both high-performance storage I/O and efficient data sharing solutions. Various options exists for data sharing in a virtualized cloud environment, including distributed and parallel file systems, object-based storage systems, and databases [4]. However, there are still several challenges, which have to be faced for exploiting the full performance and scalability potential of the cloud resources [15]. One of the main current challenges is how to address the limitation of the still popular POSIX file system interface. There is a wide agreement that POSIX consistency model is not suitable for high-performance and scalable applications. Another critique of POSIX is that it does not expose data locality, while researchers agree that locality-awareness is a key factor for building high performance scalable systems [13]. These issues are magnified in virtualized environments, one of the reasons being the trade off between protection and performance. Protection across domains is enforced at the cost of memory copy operations, which degrade the performance. Cooperating applications (e.g. on-line visualization of a scientific application) or processes of a single application (e.g. workflow application) running in several virtualized domains are currently offered few possibilities to coordinate the storage I/O across domains and to trade off performance and protection.

Our main contribution in this paper is to address this gap by proposing a set of abstractions and mechanisms which enable building efficient virtualized storage systems for data-sharing applications, while trading off protection and performance. More precisely, we propose abstractions and mechanisms, which allow to: coordinate storage I/O across domains, create shared access spaces across node-local domains, relax the POSIX consistency, allow for flexible data write and data update policies, and expose data locality. VIDAS is built on top of a new mechanism for collectively sharing memory among multiple virtual machines.

II. Related work

The most efficient virtualized storage solutions employ paravirtualized device drivers in order to avoid virtualizing a device, and move data more quickly from the virtualized environment to the host. Paravirtualized device drivers are available for most virtualization solutions, such as KVM virtio drivers [14],



(a) Data flow in solutions with paravirtualized device drivers.

(b) Data flow in VIDAS, our proposed solution.

Fig. 1. Comparison of how data flows in common virtualization solutions versus our proposal.

Xen split drivers [12], and VMWare's guest disk driver.

A *paravirtualized file system* file system solution can be simply based on running a unmodified file system on a paravirtualized device driver as shown in B2. However, more complex paravirtualized file system solutions (B3) [10] allow a stonger inter-domain coordination in order to optimize away redundant functionality. VIDAS, as a paravirtualized object-level storage pool, can not be straightforwardly categorized in this classification. We see VIDAS as an intermediary layer between disk-level virtualization and file system-level virtualization. On one hand, VIDAS can be used to consistently share access to a non-shared disk. On the other hand, VIDAS can serve as an intermediary layer for building a shared paravirtualized file system by allowing a straightforward implementation of a shared name space or a shared buffer cache on top of storage objects. In general, VIDAS can be used for offering guests high-performance data sharing and locality awareness based on a shared pool of storage objects. The shared pool consists of objects that can be mapped to every virtual machine running on the same host. Our memory sharing mechanism differs from existing mechanisms for communicating and data sharing between virtual machines such as XenSocket [19], XWAY [5], and others [7], which typically share a page between two domains. Our solution supports the capability of collectively sharing data among an arbitrary number of domains, as detailed in Section III. Figure 1 compares our proposed solution with paravirtualized solutions.

III. Abstractions and mechanisms for virtualized data sharing

This section introduces the novel abstractions and mechanisms for data sharing in virtualized environments. Our virtualized data sharing solution is based on two abstractions: containers and storage objects

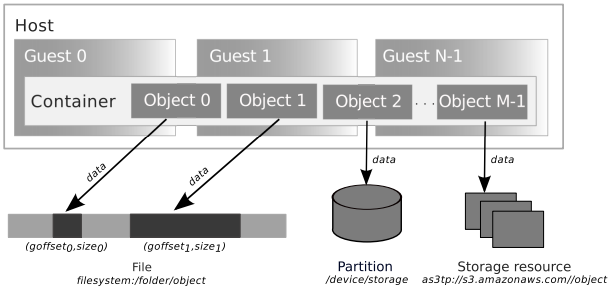


Fig. 2. Several guests sharing objects through a container. Each object is mapped on an external storage resource.

(as shown in Figure 2). A *container* (described in Section III-A) is an abstraction which allows data sharing between virtual domains running on the same physical node. The data sharing can be done at the granularity of a data *storage object* (described in Section III-B). All domains with access to a common container can use it to share data objects among each other. Storage I/O coordination, data sharing, asynchronous I/O can be done at object-level.

A. Containers

A container in VIDAS is an abstraction which facilitates storage object sharing across virtual domains. A container has a unique name, which has to be known by the domains who want to share it. VIDAS provides a restricted set of container operations. A container is created by an initiator domain by specifying a unique name and an array of potential domains that are allowed to share it. A container can be destroyed by the initiator domain only if there is no other domain sharing it. After sharing a container, domains are able to start to manipulate and access storage objects shared by any one of them as described in the next section.

B. Storage objects

A storage object in VIDAS is an abstraction for data sharing across domains. Each storage object is uniquely associated to an external storage resource through its name. The external storage resource can be a file from a file system, an object from a storage system, a disk partition, or any other storage resource that can be uniquely identified through a name and offers a linear address space. For instance, the external storage resource can be a file from a locally mounted NFS or a URL of a remote object stored in Amazon S3. In this paper we will assume that a simple get/put interface is available for accessing these external storage resources and we will concentrate on the node-level data sharing in a virtualized environment.

The storage objects are different from traditional POSIX files in several aspects: each object is associated with a user-extensible list of name-value attributes, strong consistency is not enforced, but optional, data writes to external storage resources can be guided by a configurable policy such as write-through or write-back, applications can learn if object data is cached in memory, providing locality awareness, and operations on objects are stateless.

IV. Implementation

We have implemented VIDAS interface based on Xen virtualization solution [12]. This section describes Xen implementation details. In order to simplify the reading, we start by presenting the Xen inter-domain mechanisms leveraged by our implementation. Subsequently, we present the implementation of the multi-domain data sharing mechanism. Finally, we discuss container and object implementations.

A. Xen inter-domain mechanisms in VIDAS

VIDAS implementation leverages shared memory for sharing object data, attributes, and other opaque metadata across used domains and ring buffers for communication between the user domains and the host. Figure 3 depicts an overview of our implementation, which shows how ring buffers and shared memory are used in VIDAS.

The memory is shared between two domains based on the procedure described in the previous subsection. For sharing a page among n domains, in our solution the domain initiating the sharing inserts in its grant table $n-1$ entries, all of which are associated with the same page. Subsequently, each of the other $n-1$ domains receives a different grant reference, representing the same physical page. The page is then mapped as in a two page case[1].

[1]We provide this new Xen feature at https://github.com/pllopis/linux/commit/fb6dca

The ring buffers are used for implementing a lightweight Remote Procedure Call (RPC) based on a front-end, running on the calling domain, and a back-end running on the host. The back-end waits on the ring buffer to receive call messages from the front-end, performs the call, and returns the result in the ring buffer. After performing the call, the front-end waits for the result from the back-end. In this work both the back-end and front-end use polling, while an interrupt based approach was left for the future work. Our initial choice was based on the conclusion of a study, which showed that the use of polling in optimizing the storage virtualization can substantially reduce the overhead of interrupt handling [1].

B. Container implementation

The container management is performed at the host. All container operations are implemented as lightweight RPCs, as described in the previous subsection. When creating a container, the guest forwards the call to the host, which stores the domain IDs sharing access. A subsequent container attach operation is successful only if the calling domain belongs to the list specified by the creating domain. Destroying a container and leaving from a container are simple RPCs that remove or update the container metadata from the host.

C. Object implementation

Object management is also performed at the host. However, most operations on object data and attributes do not involve the host, as described below. Seven operations from VIDAS interface rely on RPCs to the host. In order to create an object (`object_create`), a domain follows the steps described above: allocates memory for object data, object attributes and other object metadata, is assigned a grant reference, passes the grant reference to all memory sharing domains, and maps the page to its virtual memory. Finally, it contacts the host with an RPC and informs about the newly created object. At this point each sharing remote domain is entitled to share the object by calling `object_attach`, which maps the object memory to the remote domain virtual memory and informs the host through an RPC. The operations `object_destroy` and `object_leave` undo the create and attach operations, respectively, and inform the host to remove the object from the index. The function `object_get_locality` is implemented as an RPC which returns from the host all the local objects (created or attached) associated to the external storage resource. The other two operations, whose implementation leverages RPCs to the host are `object_flush` and `object_update`, which simply ask the host to flush/update the object to/-from the remote storage resource.

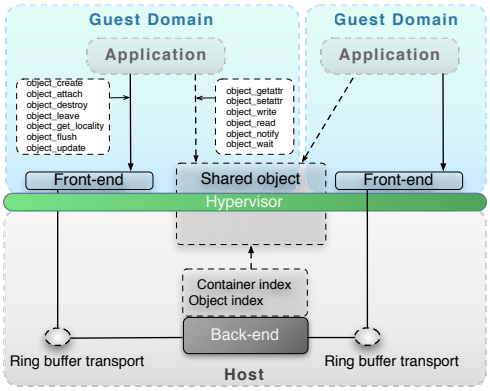


Fig. 3. VIDAS implementation. Seven operations rely on RPCs to the host, while the other six VIDAS operations rely on shared memory and do not directly involve the host.

All the other six VIDAS operations rely on shared memory and do not directly involve the host. The data access operations `object_write` and `object_read` access the shared memory directly and are atomic only if the *synchronized* object attribute is set to "*true*". In VIDAS, accesses to an object are serialized only if the accessed domains overlap. For addressing this issue in our current implementation, the object metadata includes an array of mutexes which provide mutual exclusion to overlapping access domains. A further extension to this implementation for providing multiple-reader one-writer access is straightforward.

While there are numerous ways to design cooperative data sharing, we made an effort to minimize host intervention where it was not strictly needed. This is an important design decision because context switches result in hypervisor *exit* operations, which are known to be the main cause of performance overheads for virtualized I/O-intensive workloads [2]. Therefore, operations which manipulate object data and metadata are carried out with minimal context switches. For other operations such as object creation and removal (which are still very fast, as demonstrated in the evaluation), we chose to sacrifice a context switch for consistency and simplicity, by having a single copy of container and object indexes maintained by the host.

V. Use cases

In this section we present two use cases based on VIDAS: inter-domain write-read sharing of a file (Subsection V-A) and inter-domain collective I/O (Subsection V-B). These two use cases are part of the evaluation in Section VI.

A. Inter-domain write and read sharing

A significant class of scientific applications is based on workflows, in which the output of one process is the input of the next [17]. While processes communicating through files within the same host can take advantage of data locality through the buffer cache, processes running within the same host but on different domains require efficient inter-domain data sharing solutions. In this section we show how the building block of a write-read pattern of a workflow can be simply and efficiently implemented based on VIDAS.

Instead of communicating through files, VIDAS allows faster shared-memory communication through shared data objects which can be stored persistently on a disk, or other storage media. By creating a shared container, two domains can create data objects which can be read and written to by more than one domain directly. In VIDAS, it is possible to have a domain write in a synchronized manner (atomic write access) to an object, while another domain (or N domains) read from it concurrently.

The way the data modifications propagate from the shared object to the external storage resource can be controlled through attributes.

The write/read operations presented above can be straightforwardly used as a building block for a producer-consumer implementation or for a data streaming implementation.

B. Inter-domain collective I/O

I/O intensive applications often face the problem of accessing non-contiguous portions of data. In scientific applications it is often the case that while different processes access non-contiguous portions of data, requests of a group of processes may together span a contiguous portion [16]. The optimizations merging different requests from cooperating processes into a single large I/O operation are referred to as collective I/O [16].

In a purely virtualized environment, efficient collective I/O is difficult to achieve because domains are isolated from each other and data has to be shared through a network file system or network communication protocols. However, VIDAS abstractions and mechanisms allow for efficiently sharing data and coordinating accesses. For instance, if one domain creates a shared object with other domains, the domains need only to write the data to the shared object (which are simple memory copy operations), and notify the object creator that the write is complete. When all domains are finished, i.e. the first domain has received all notifications), modifications can be flushed to storage. Collective read operations can be implemented in a similar fashion.

VI. Evaluation

We evaluated the VIDAS prototype on a 12-core Intel(R) Xeon(R) CPU E5-2620 @ 2.0Ghz with 64GB of DDR3 synchronous memory clocked at

1333Mhz. The hard disk is a Toshiba MK1002TS with a capacity of 1TB, a speed of 7200 rpm, a 64MB cache, and is partitioned with LVM and ext4. Xen version 4.2 runs Linux 3.5.7 as Dom0, modified to support sharing memory pages across an arbitrary number of domains. Preceding each experiment, we cleared caches, directory entries and inodes from memory using the Linux drop_caches interface.

A. Object operations

In this section we present an evaluation of VIDAS object and container operations. The container operations take between 64μs and 70μs, which correspond to the time of the RPC between the guest and host domains. The creation of an object of one 4096 byte page takes 207μs, representing the time to allocate memory, send the grant reference to the remote domain, map the page, and perform an RPC for registering the domain. Joining an object takes 98μs, corresponding to the time to map the page and register at the host through a RPC. Leaving an object involves an unmap operation and an RPC and takes 82μs. The other operations involve shared memory and take 2μs (setting and getting an attribute of 1 byte) and 6μs (notify/wait a message of 64 bytes).

B. Inter-domain communication

Due to the fact that the goal of this work is to improve data sharing in virtualized environments, we first evaluate data communication between virtual machines without performing I/O. We evaluate broadcasting 128MB data objects to 2, 4, 8 and 16 virtual machines using existing inter-domain memory sharing solutions such as Xen ringbuffers, the OSU broadcast benchmark, MPI broadcast, and our solution. Figure 4 shows the effectiveness of our multi-domain memory sharing mechanism. Because the other inter-domain communicators are restricted by Xen's current memory sharing mechanism, which limits the amount of domains sharing a page to 2, they are required to do additional memory copies. Our solution requires a single memory copy and scales better. For fairness, we introduced an additional memory copy per virtual machine in order to obtain different copies of the data object, as opposed to just one shared copy. This is also the reason why performance drops slightly when scaling the number of virtual machines.

C. Write and read sharing

In this subsection we compare the writer-reader implemented in VIDAS (described in V-A) with a writer-reader based on PVFS2 and NFS using 512 MB files. VIDAS semantics enables us to perform an asynchronous write to disk while readers read object data directly from memory, thus obtaining memory read speeds, while NFS and PVFS2 rely on

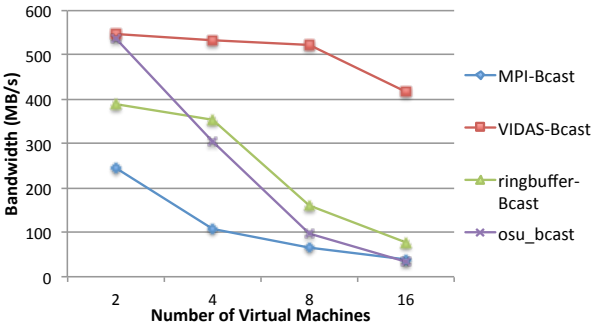


Fig. 4. Evaluation of broadcast communication in VIDAS, MPI, OSU benchmark, and Xen ringbuffers.

the disk write buffers for performing reads. We obtained a sustained throughput of over 500MB/s for VIDAS, while NFS and PVFS2 performed at close to 140MB/s.

D. Independent shared file read

We also evaluated shared file read, scaling the number of domains which perform overlapping reads of a 512MB file concurrently. VIDAS uses the broadcasting mechanism evaluated in Section VI-B. Figure 5 shows that VIDAS outperforms PVFS2 and NFS. VIDAS reads data into a shared object and does not require additional memory copies to transfer the data to every other domain. However, for fairness we performed an additional copy in order for each domain to have a different copy. Without this additional copy, the throughput corresponding to a disk file read would have been sustained for any number of virtual machines.

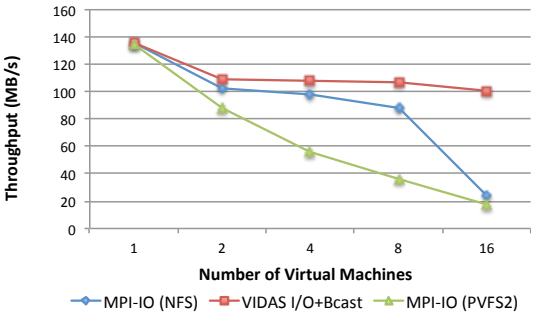


Fig. 5. Evaluation of multiple reader in VIDAS, NFS, and PVFS2.

E. Collective I/O

In this section we compare the VIDAS inter-domain collective I/O operations (described in Subsection V-B) with a standard collective I/O implementation from ROMIO, the most popular MPI-IO distribution. The collective I/O implementation of ROMIO is based on two-phase I/O [16], an optimization which merges non-contiguous I/O requests

into contiguous ones at aggregator processes before sending them to file system (we have employed one aggregator). We have used ROMIO included in the MPICH2 1.4.1 MPI distribution. Figure 6 depicts the results for collectively writing and reading data to/-from an object/file of 512 MB. The domains are accessing non-overlappingly interleaved strided vectors of 2MB blocks. Figure 6 shows the results for VIDAS collective I/O and ROMIO collective I/O implementations for reading and writing, respectively. We note that VIDAS collectives outperform collective I/O ROMIO operations. The main explanation for the better performance of VIDAS is the shared collective buffer, which helps avoid copy operations. On the other hand, ROMIO collective operations copy the data into collective buffers before sending them to disks, which makes performance drop dramatically when increasing the number of virtual machines.

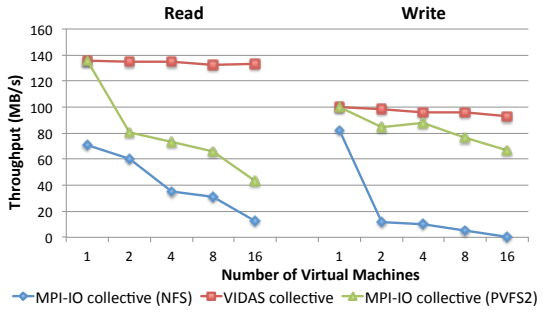


Fig. 6. Comparison of VIDAS collective I/O and ROMIO collective I/O

VII. Conclusions

In this paper we provide the design, implementation, and evaluation of VIDAS, an object-based virtualized data store that can be used to efficiently and consistently share access to externally stored data in virtualized environments based on a shared pool of storage objects. VIDAS provides integrated abstractions and mechanisms that allow to coordinate storage I/O across domains, create shared access spaces across node-local domains, relax the POSIX consistency, control the write and update policies, and control data locality. In order to efficiently implement VIDAS virtualized data sharing abstractions, we have proposed and evaluated a new data sharing mechanism which extends Xen to provide multi-domain memory sharing to user-space applications. The mechanisms and abstractions shown in this work would be best complemented with a higher-level layer which controls placement of VM machines, computing jobs, and data distribution. This would enable cloud-based HPC workloads to share data much more effectively by using VIDAS. In the future we plan to integrate VIDAS with our solution for I/O forwarding for cloud environments in order to combine node-local data sharing capabilities with high performance inter-node I/O delegation [8].

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science under the grant IPT-430000-2010-14.

References

[1] M. Ben-Yehuda, M. Factor, E. Rom, A. Traeger, E. Borovik, and B.-A. Yassour. Adding advanced storage controller functionality via low-overhead virtualization. FAST'12, pages 15–15, 2012.

[2] A. Gordon, N. Amit, N. HarÉl, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir. ELI: bare-metal performance for I/O virtualization. In *ASPLOS '12*, pages 411–422, Mar. 2012.

[3] W. Huang, M. J. Koop, Q. Gao, and D. K. Panda. Virtual machine aware communication libraries for high performance computing. In *SC '07*, page 9, Nov. 2007.

[4] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Data Sharing Options for Scientific Workflows on Amazon EC2. pages 1–9, 2010.

[5] K. Kim, C. Kim, S.-I. Jung, H.-S. Shin, and J.-S. Kim. Inter-domain socket communications supporting high performance and full binary compatibility on Xen. In *VEE '08*, pages 11–20. ACM, 2008.

[6] D. Le, H. Huang, and H. Wang. Understanding performance implications of nested file systems in a virtualized environment. pages 8–8, 2012.

[7] D. Li, H. Jin, Y. Shao, and X. Liao. A high-efficient inter-domain data transferring system for virtual machines. In *ICUIMC '09*, pages 385–390, 2009.

[8] P. Llopis, G. Martín, B. Bergua, and J. Carretero. Virtual I/O forwarding for cloud-based HPC applications. In *ISPA '12*, pages 1–2, Apr. 2012.

[9] D. T. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M. J. Feeley, N. C. Hutchinson, and A. Warfield. Parallax: virtual disks for virtual machines. In *Eurosys '08*, pages 41–54. ACM, 2008.

[10] B. Pfaff, T. Garfinkel, and M. Rosenblum. Virtualization aware file systems: getting beyond the limitations of virtual disks. In *NSDI '06*, 2006.

[11] B. Pfaff, T. Garfinkel, and M. Rosenblum. Virtualization aware file systems: getting beyond the limitations of virtual disks. pages 26–26, 2006.

[12] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, and A. Mallick. Xen 3.0 and the Art of Virtualization. In *Proceedings of the 2005 Ottawa Linux Symposium*, July 2005.

[13] I. Raicu, I. T. Foster, and P. Beckman. Making a case for distributed file systems at Exascale. In *HPDC '11*, pages 11–18. ACM, 2011.

[14] R. Russell. Virtio: towards a de-facto standard for virtual I/O devices. *ACM SIGOPS Operating Systems Review*, 42(5):95–103, 2008.

[15] J. Shafer. I/O virtualization bottlenecks in cloud computing today. WIOV'10, pages 5–5, 2010.

[16] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. FRONTIERS '99, pages 182–. IEEE Computer Society, 1999.

[17] E. Vairavanathan, S. Al-Kiswany, L. Costa, Z. Zhang, D. Katz, M. Wilde, and M. Ripeanu. A Workflow-Aware Storage System: An Opportunity Study. *CCGRID '12*, pages 326–334, 2012.

[18] C. Waldspurger and M. Rosenblum. I/O virtualization. *Communications of the ACM*, 55(1), Jan. 2012.

[19] X. Zhang, S. McIntosh, P. Rohatgi, and J. Griffin. Xensocket: A high-throughput interdomain transport for virtual machines. pages 184–203. Springer, 2007.

Design and implementation of a hierarchical parallel storage system

Francisco J. Rodrigo[1] , Javier García Blas[2] y Jesús Carretero[3]

Abstract— **This paper presents the design and implementation of a storage system for high performance systems based on a multiple level I/O caching architecture. The solution relies on Memcached as a parallel storage system, preserving its powerful capacities such as transparency, quick deployment, and scalability. The designed parallel storage system targets to reduce the I/O latency in data-intensive high performance applications. The proposed solution consists of a user-level library and extended Memcached servers. The solution aims to be hierarchical by deploying Memcached-based I/O servers across all the infrastructure data path. Our experiments demonstrate that our solution is up to 40 % faster than PVFS2.**

Keywords— **Memcached, parallel storage system, distributed cache**

I. Introduction

NOWADAYS , storage systems are one of the main bottlenecks in high performance systems and this challenge is expected to continue in next generation *exascale* systems [1]. It is greatly accepted that large scale storage systems will be necessarily hierarchical [2]. This can be done by organizing the memory spaces in a complex hierarchy, moving data to local cache as fast as possible and throwing it to slower devices in an asynchronous way [3]. This hierarchical structure should be constructed with decoupling in mind, which consists on splitting and isolating compute nodes, storage nodes, and services (network, admin, etc.) as much as possible. Current research works depict that emerging high-speed networks outperform physical disk performance, reducing the relevance of disk locality [4].

One of the problems addressed in cloud computing environments is the data management. Currently, one of the techniques to optimize I/O systems in cloud environments is to decouple the virtual instances from the storage resources. Move information between different domains has never been a simple task. First, it is costly to deploy virtual machines that need to process a huge amount of data. Second, the data access between geographically dispersed infrastructure is significantly increasing the latency perceived by users. Currently, there are research works that propose the use of cloud environments for high performance applications [5]. Such applications process large amount of data and therefore storage systems are a critical resource.

One of the most popular solutions for providing an efficient and scalable distributed cache infrastructure is Memcached [6], [7]. Memcached aims to provide a caching distributed system for web-based database services, and is used in some of the most important web solutions such as YouTube, Facebook, and Twitter. The generic design of Memcached allows to store raw data inside its distributed cache as key-value entries. Furthermore, Memcached can be used to provide a distributed memory solution for different application fields.

The main motivation of this work is to present a hierarchical storage solution for large scale parallel systems. Our proposed solution aims to fill the latency gap between compute nodes and the final storage sub-systems, regarding the continuous increase of data access latencies. We present a Memcached based storage system, namely MemcacheFS (MFS). Our solution could be used to deploy a storage system on all the levels of the data path hierarchy. Applications benefit of this solution by improving data locality, reducing storage latencies, and overlapping computation and I/O operations.

The contributions of this work are the following. First, we present a hierarchical storage system that address to reduce the data access latency of current large scale infrastructures. Second, the proposed solution could be easily deployed on heterogeneous computational systems. Third, thanks to the hash algorithms included in Memcached, the client side is completely decoupled from the I/O servers, resulting in an increase of scalability.

The remainder of this paper is organized as follows. Section II reviews related work. Section III introduces the Memcached software architecture. Section IV presents the design details of MemcachedFS. The experimental results are presented in Section V. Finally, we conclude in Section VI.

II. Related work

Google File System (GFS) is a well-known file system for large scale computation architectures since its introduction in 2003 [8]. The most popular implementation of GFS is Hadoop and is used as a *de-facto*

[1]Dpto. de Informática, Univ. Carlos III de Madrid, e-mail: frodrigo@arcos.inf.uc3m.es
[2]Dpto. de Informática, Univ. Carlos III de Madrid, e-mail: fjblas@arcos.inf.uc3m.es
[3]Dpto. de Informática, Univ. Carlos III de Madrid, e-mail: jcarrete@arcos.inf.uc3m.es

standard on Map Reduce based applications [9]. Its main characteristics are the massive data management and fault tolerance. Main disadvantage of GFS is the centralized metadata server, leaving an unique point of failure and a possible bottleneck for the whole system. MemcachedFS fully distributes both data and metadata blocks across the infrastructure. Another problem of GFS is the co-designing requirement of every application for its deployment. Our solution aims to be as generic as possible by providing POSIX-like and MPI-IO interfaces.

There are some similarities between GFS and MemcachedFS. First, their high availability philosophy. Every launch of an I/O node is done the same way, without discriminating between a fresh launch or restart of a crashed node. Second, GFS has diagnostic and debugging tools, a very important help for developers using distributed file systems. By default, Memcached servers offer a verbose mode, in which every operation is registered by server nodes, and complex statistics about their usage.

Another distributed file system currently growing in popularity is Ceph [10]. In order to avoid metadata access bottlenecks, Ceph takes advantage of a distributed metadata cluster architecture based on Dynamic Subtree Partitioning [11]. Our approach deals with metadata bottleneck by distributing data and metadata accross every possible Memcached node. Another similarity between Ceph and MemcachedFS is the mechanism for obtaining and addressing data objects. It is not necessary to store the addresses of the blocks or objects used by each file on the metadata as done in traditional file systems. Ceph eliminates allocation lists entirely. Instead, file data are striped onto predictably named objects, while a special-purpose data distribution function assigns objects to storage nodes or storage devices. This allows any party to calculate (rather than look up) the name and location of objects comprising a file's contents. This solution eliminates the need to maintain and distribute object lists, simplifying the design of the system, and reducing the metadata cluster workload. Ceph, GFS, and MFS agree on the proposed solution of requesting all the necessary information about the file on metadata, and then, access directly to the I/O nodes. Ceph and GFS follow a similar replication scheme, in which data object replicas are distributed through the cluster. Currently, MemcachedFS does not support any kind of replication but we plan to implement this feature in the future.

Relative to high performance I/O and non persistent storage, there are some interesting solutions. Nahanni Memcached [12] is a solution totally applicable to our proposed storage system in a multi-tenant environment. Nahanni Memacached uses mechanisms of shared memory between virtual machines running on the same host to avoid communications over sockets, achieving much better performance. CloudScale[13] and Scarlett[14] have in common some ideas directly applicable to our solution. In order to prevent bursty load pikes, both take advantage of replicas. Replicating the most requested objects, queries can be distributed to various nodes as opposed to overload one specific node. Memcached offers data access statistics, which can be used to design smart replication solutions for MemcachedFS based on the mechanisms proposed by CloudScale and Scarlett.

Finally, it is important to remark that there are previous works using Memcached in a distributed file system. Wang et al. benefit from Memcached by promoting read throughput in a massive file system, in which small files are predominant [15]. In contrast, our proposed solution supports any file size given that a multiple-block file system is implemented on top of Memcached.

III. Memcached background

Memcached is a free and open source, high performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load [16]. Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering. The simple design of Memcached allows a quick and easy deployment and solves many problems facing large and distributed data caches.

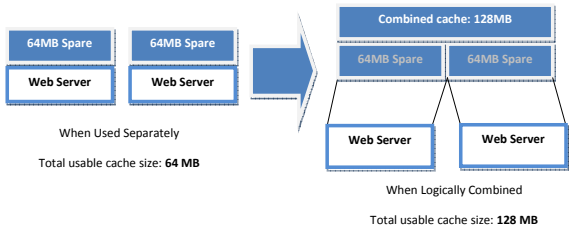


Fig. 1. Example of use of Memcached. Memcached shows all the physically distributed caches as an unique pool of data to the user.

Its main functionality is to show to the clients any number of individual caches in any number of nodes as a unique caching space as shown in Figure 1. Client have a list of servers in order to store raw data on this unique caching space as key-value pairs. These key-value pairs can store raw data of up to 1 MBytes associated with keys up to 250 Bytes. Clients store data in a transparent way, without knowing other clients. Servers do not need any information about clients or about other servers. This is possible thanks to the two-phase hash algorithm. With the list of servers, any client knows where any key should be stored or retrieved and queries this specific server.

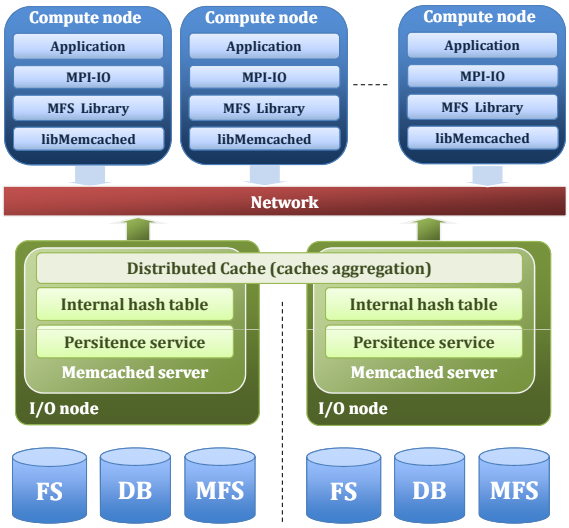


Fig. 2. MemcachedFS architecture. Parallel applications access data using MPI-IO. The MemcachedFS (MFS) library maps key-value based blocks to Memcached-based I/O nodes. The I/O nodes provide a distributed cache system which aggregates items in a transparent way. Finally the persistence service forwards items to the next level of the hierarchy.

Servers receive queries and store data inside its hash table, divided in memory slabs for different values sizes.

IV. Memcached storage system

In this section we present the detailed design of MemcachedFS. Our solution relies on two main components: client library and Memcached servers (as shown in Figure 2). First, clients access data across a user-level library, denoted as *MFS library* in the figure. This library has been designed with two main goals: portability and flexibility. Portability has been achieved by providing a POSIX-like interface and flexibility by designing the solution on top of Memcached in a generic implementation based on two layers. The first layer contains the external view of the library and the logic of the file system. The second layer translates generic *get*/*set* calls to specific libMemcached calls. This generic approach allows to port the library to other distributed memory solutions like Redis [17]. This port only requires to modify the communication layer, not the logic of the file system or the interface offered to the users.

The library provides a logical view of the Memcached servers as a block device. All Memcached servers conform an unique shared memory space while offering transparent data mapping. The key-value pairs symbolize blocks in the device. The key field is used as the block address and the value stores its raw data content. When a new block is needed, the key is generated based on both a file unique identifier and the file offset. Following accesses are done calculating again this block identifier. The benefit of this method is that avoids the necessity of storing this information as metadata and only requires one access to search the block (most file systems use a tree with various levels). File identifier is calculated only in the creation of the file and is stored on the metadata. Data and metadata are treated equally as key-value items, so are fully distributed between all the Memcached nodes. This solution avoids possible bottlenecks produced by heavy metadata accesses.

MemcachedFS library takes advantage of the hashing mechanisms included in Memcached. Every client maps blocks with a known list of servers. This map is done at the client side, so no information exchange is needed between clients and servers.

Focusing on the potential use of the proposed solution for parallel applications based on MPI, we have implemented an ADIO interface for MemcachedFS over ROMIO. As shown in Figure 3 the ADIO interface is built on top of the MemcachedFS library and can be easily used by any MPI application, like other default interfaces such as PVFS2, NFS, etc.

Fig. 3. Scheme of the ADIO architecture. The implemented ADIO interfeace of MemcachedFS (MFS) grant access to MemcachedFS to any MPI-IO application.

The second component is the Memcached servers. Memcached has been selected as our storage server for two main reasons. First, its popularity: Memcached is largely used in production environments, guaranteeing the reliability and optimization of the code (multi-thread, light protocol, memory use optimization, etc.) Second, the simplicity of deployment of Memcached servers. However, the Memcached server was initially designed without any kind of persistence in mind: items dropped by LRU become unrecoverable. In order to provide a persistent storage, we have modified the Memcached server regular operation. MemcachedFS relies on a persistence management service, which stores dropeed key-value entries into a mounted file system (FS), a Berkeley DB (DB), or another instance of MemcachedFS, resulting in a hierarchical architecture. We have extended the native cache mechanisms by adding a dirty flag in the Memcached's items, achieving better performance storing into the next level of the hierarchy only dirty items. The recovery of items from persistence storage is done using the Memcached functions, making it completely transparent to the user. Retrieved items are stored again in cache and their life cycle is exactly

as the followed by new items stored by user.

V. Evaluation

The evaluation of our prototype implementation for MemcachedFS was performed on a beowulf cluster with the following characteristics: 24 compute nodes, each one Intel Xeon CPU E5405, 4 GBytes DDR3. A Gigabit Ethernet network connects all compute nodes. MPICH2 1.4.1p1 [18]. In order to evaluate our solution, we have used PVFS2, also known as OrangeFS, as reference parallel file system [19]. PVFS2 version 2.8.4 has been configured with 64 KBytes of stripping size and 4 I/O nodes. MemcachedFS was configured with 128 MBytes cache in each node and a block size of 512 KBytes and its persistence service is configured to save items discarded by LRU on a Berkeley DB. We have evaluated our solution against PVFS2 using IOR benchmark and NAS BTIO.

A. IOR benchmark

IOR [20] was developed to set up performance targets for the acquisition of the ASC Purple supercomputer by the LLNL. The benchmark focus its objectives on the measurement of the performance in sequential read/writes operations with different file sizes, I/O transactions sizes, and concurrency. The benchmark can use shared files or one for each processor. IOR supports POSIX and other advanced parallel I/O interfaces such as MPI-IO, HDF5 and parallelNetCDF. In this case, IOR accesses data through the ADIO interface implemented for MemcachedFS.

IOR benchmark was executed writing/reading 4 GBytes of data, independently of the number of clients and always using 512 KBytes blocks. The results exposed in the graphs are the best values obtained for each case in five iterations. As shown in Figure 4 (top), the throughput obtained by MemcachedFS increases with the number of clients making write operations, while in the case of PVFS2 it is penalized. This trend is explained because MemcachedFS overlaps disk operations (Berkeley DB based *put* operations) with data transfers across the network. In the best case, MemcachedFS outperforms PVFS2 by 40 %, reaching almost the limits of the Gigabit interconnection network.

In Figure 4 (bottom), MemcachedFS results are more on par with the results obtained from PVFS2. However, MemcachedFS outperforms PVFS2 in almost every case (only with one client MFS obtains worse results). Again, MemcachedFS reaches the limits of the Gigabit network, standing near 125 MBytes/s from 2 to 32 clients. In both cases, a larger number of I/O nodes used by MemcachedFS improves the performance, but not in a extremely significant way.

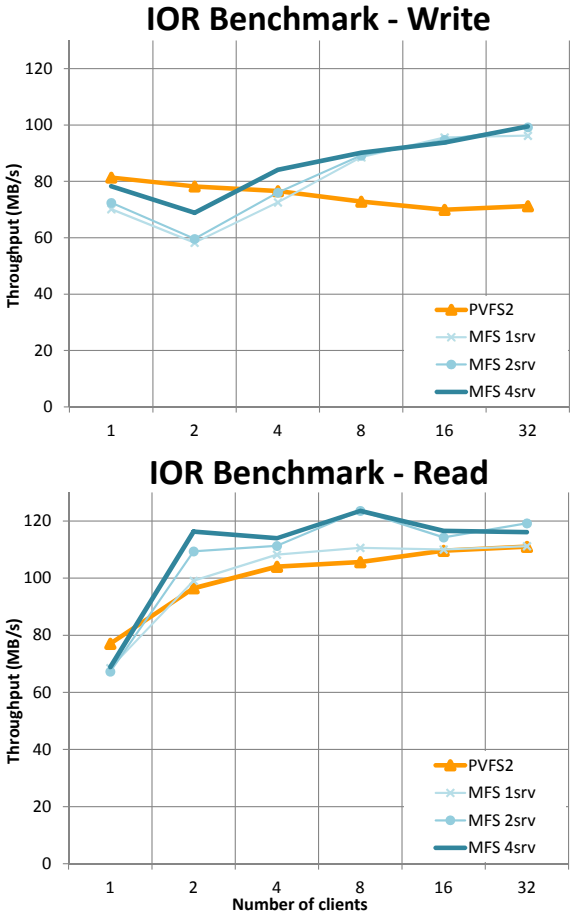


Fig. 4. Throughput performance on 4 GBytes operations of MemcachedFS against PVFS2 on IOR benchmark. Best result out of five iterations. Top figure shows performance on write operations. Bottom figure shows performance on read operations.

B. NAS BTIO benchmark

NASA's BTIO benchmark [21] solves the Block-Tridiagonal (BT) problem, which employs a complex domain decomposition across a square number of compute nodes. Each compute node is responsible for multiple Cartesian subsets of the entire data set. The execution alternates computation and I/O phases. Initially, all compute nodes collectively open a file and declare views on the relevant file regions. After each five computing steps the compute nodes write the solution to a file through a collective operation. At the end, the resulted file is collectively read and the solution verified for correctness. We use 1 to 36 processes and class A of data set size (400 MBytes). The benchmark reports the total time including the time spent to write the solution to the file. However, the verification phase time containing the reading of data from files is not included in the reported total time. For ROMIO-based collective operations, BTIO explicitly sets the size of the collective buffer to 1 MByte and assigns all compute nodes for two-phase

aggregators or view-based aggregators.

Figure 5 (top) plots the results obtained for write operations, best case of three repetitions. In this test, all the results are far from the network maximum throughput. PVFS2 obtains better results than MFS, but the impact on performance when the number of clients increases implies worst scalability than the proposed solution.

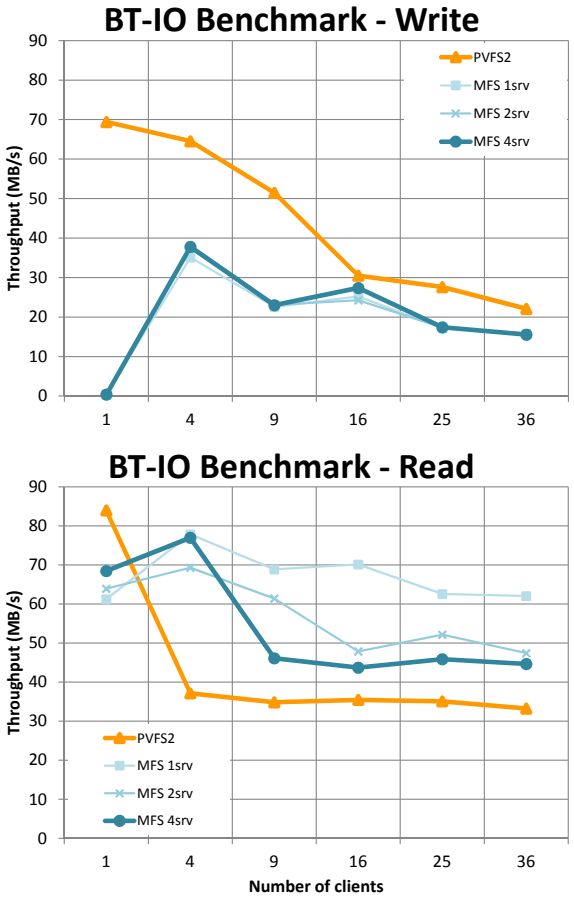


Fig. 5. Throughput performance on 400 MBytes operations (class A) of MemcachedFS against PVFS2 on BTIO benchmark. Best result out of three iterations. Top figure shows performance on write operations. Bottom figure shows performance on read operations.

The last experiment case is BT-IO benchmark read operations. Results, as can be seen in Figure 5 (bottom), are out of the norm. Performance is better with only one I/O node than with four of them. MFS is better in almost all cases and achieves better scalability. MemcachedFS best case scenario is 80 % better than PVFS2, while the improvement is around 25 % with four I/O nodes.

VI. Conclusions

We have presented a highly portable and flexible storage system for hierarchical large computational systems. Our solution is based on Memcached, the most used distributed cache memory software architecture. The solution takes in advantage the main features offered by Memcached such as scalability, flexibility, and performance. Our solution outperforms PVFS2 by 40 % in the best case, while does not suffer any special performance hits. As we demonstrate in the evaluation section, our solution scales with the number of clients. Best performance cases are around the limits of the network used. Our ongoing work targets the persistence mechanisms included in MemcachedFS. These mechanisms consist in optimizing the storing process with a more asynchronous implementation and improving the persistence mechanisms over other Memcached servers, allowing the construction of complex hierarchical storage architectures.

VII. Acknowledgments

This work was supported in part by Spanish Ministry of Science and Innovation under the project “Input/Output Scalable Techniques for distributed and high-performance computing environments” - TIN2010-16497.

References

[1] G. Bell, J. Gray, and A. Szalay, “Petascale computational systems,” *Computer*, vol. 39, no. 1, pp. 110 – 112, jan. 2006.

[2] Jack Dongarra, Pete Beckman, Terry Moore, and Aerts, “The International Exascale Software Project roadmap,” *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 1, pp. 3–60, Feb. 2011.

[3] Florin Isaila, Javier Garcia Blas, Jesus Carretero, Robert Latham, and Robert Ross, “Design and evaluation of multiple-level data staging for blue gene systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 946–959, 2011.

[4] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, “Disk-locality in datacenter computing considered irrelevant,” in *HotOS’13*, Berkeley, CA, USA, 2011, pp. 12–12, USENIX Association.

[5] Christian Vecchiola, Suraj Pandey, and Rajkumar Buyya, “High-performance cloud computing: A view of scientific applications,” in *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, Washington, DC, USA, 2009, ISPAN ’09, pp. 4–16, IEEE Computer Society.

[6] Brad Fitzpatrick, “Distributed caching with memcached,” *Linux J.*, vol. 2004, no. 124, pp. 5–, aug 2004.

[7] Alan Harris and Alan Harris, “Distributed caching via memcached,” in *Pro ASP.NET 4 CMS*, pp. 165–196. Apress, 2010.

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, “The google file system,” *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, Oct. 2003.

[9] Tom White, *Hadoop: The Definitive Guide*, O’Reilly Media, original edition, June 2009.

[10] Sage A. Weil, Scott A. Brandt, and more, “Ceph: A Scalable, High-Performance Distributed File System,” 2006.

[11] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, and Ethan L. Miller, “Dynamic metadata management for petabyte-scale file systems,” in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2004, SC ’04, pp. 4–, IEEE Computer Society.

[12] Adam Wolfe Gordon and Paul Lu, “Low-Latency Caching for Cloud-Based Web Applications,” 2011.

[13] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes, “Cloudscale: elastic resource scaling for multi-tenant cloud systems,” in *Proceedings of the 2nd*

ACM Symposium on Cloud Computing, New York, NY, USA, 2011, SOCC '11, pp. 5:1–5:14, ACM.

[14] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," in *Proceedings of the sixth conference on Computer systems*, New York, NY, USA, 2011, EuroSys '11, pp. 287–300, ACM.

[15] Chien-Ming Wang, Chi-Chang Huang, and Huan-Ming Liang, "Asdf: An autonomous and scalable distributed file system," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Washington, DC, USA, 2011, CCGRID '11, pp. 485–493, IEEE Computer Society.

[16] *Memcached - A distributed memory object caching system*, http://www.memcached.org.

[17] Tiago Macedo and Fred Oliveira, *Redis Cookbook*, O'Reilly Media, Sebastopol, 2011.

[18] *MPICH - A portable implementation of MPI*, http://www.mpich.org.

[19] W.B. Ligon and R.B. Ross, "An Overview of the Parallel Virtual File System," in *Proceedings of the Extreme Linux Workshop*, June 1999.

[20] Hongzhang Shan, Katie Antypas, and John Shalf, "Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, 2008, SC '08, pp. 42:1–42:12, IEEE Press.

[21] P. Wong and R. der Wijngaart, "Nas parallel benchmarks I/O version 2.4," 2003.

Redes y Comunicaciones

V2V Real-time Video Flooding on Highways

Alvaro Torres[1], Pablo Piñol[2], Carlos T. Calafate[1], Juan-Carlos Cano[1], Pietro Manzoni[1]

Abstract— **Vehicular ad-hoc networks are one of the most promising network technologies, as they will provide vehicles with real-time information about accidents or congestion on the road. In these cases, obtaining accurate information could be a valuable resource for traffic authorities and drivers. One of the most accurate sources of information would be a real-time video showing the state of a critical event, so a proper diffusion of the video is critical to maximize the user-perceived quality. This task is really difficult due to the typical VANET problems such as signal attenuation, packet losses, high relative speeds, etc.**

In this paper we address these problems by comparing solutions that combine different video codecs and different flooding algorithms to assess the effectiveness of long-distance real-time video flooding. In particular, we will compare the most effective video coding standard available (H.264) with the upcoming H.265 codec in terms of both frame loss and PSNR.

Keywords—**Video streaming, V2V, VANET, flooding, h.264/AVC, h.265/HEVC**

I. Introduction

VEHICULAR ad-hoc networks (VANETs) are receiving a lot of attention from the vehicle industry since they promote the design of smarter, cleaner and safer vehicles. In a near future it is expected that VANETs will be as extended as mobile phones are nowadays. Vehicles will be equipped with different sensors which can provide useful information to other drivers or traffic authorities. One of the most useful data flows a vehicle can provide is the live video of an accident situation, not only to allow the emergency services to know in advance the exact status of an accident, but also for other vehicles to decide whether to change their current route.

VANETs provide one of the most difficult environments to achieve a good quality in the transmission process since this type of networks involve high relative speeds, which cause short connection times and transmission problems such as the Doppler effect. Bandwidth is typically very limited, thus becoming one of the worst case scenarios for real-time video transmission.

The recent approval of the new H.265 video compression standard [1], which intends to replace the widely used and well-known H.264 standard [2], provides a new opportunity for real-time video transmission in critical contexts. The new standard outperforms the old one, achieving the same video quality with 50% of the bit-rate [3].

The rest of the paper is structured as follows. In section II, we review the state of the art in terms of both flooding in wireless networking and video transmission over VANETs. In section III details are provided about the scenario characteristics and the intended video transmission mechanism. Afterward, section IV presents the tools adopted to simulate the proposed scenario. Section V presents the obtained results and finally, in section VI, we summarize the conclusions obtained and present some future work.

[1]Department of Computer Engineering. Universitat Politècnica de València. Valencia, Spain atcortes@batousay.com, {calafate,jucano,pmanzoni}@disca.upv.es

[2]Dpto. de Física y Arquitectura de Computadores. Miguel Hernández University. Elche, Spain pablop@umh.es

II. Related Work

In the existing literature several flooding mechanisms have been presented, although the majority of them are intended for MANETs.

Due to the nature of the IEEE 802.11 protocol, which provides a contention-based broadcast mechanism, flooding protocols are focused on avoiding the broadcast storm problem. Yu-Chee Tseng et al. [4] presented the most basic algorithms to solve this problem. The same author published [5] an improved version of the same algorithms by adding adaptive conditions to further reduce the broadcast storm problem. More specifically, the authors present the basis of flooding in MANET networks: in the first article they present the basic versions of some flooding mechanisms such as the Counter-Based Scheme, the Distance-Based Scheme or the Location-Based Scheme. In the second article they slightly improve these flooding schemes adding information of the environment to decide whether to rebroadcast a packet.

Martínez et al. [6] presented another flooding mechanism that takes into account the particularities of VANETs, following the guidelines of the Distance-Based mechanism; in particular, they tweak it to provide a fast dissemination of accident alerts in urban scenarios by using information such as the town layout to achieve a smarter flooding. For highway scenarios the algorithm does not significantly differ from the Distance-Based approach.

The field of video transmission over VANETs has been studied by several authors, although the main purpose of most articles on video transmission is entertainment, so the video is streamed from Road Side Units (RSU) into the vehicular network.

Meng Guo et al. [7] presented several scenarios where the live video streaming between vehicles, and from vehicles to RSU, is both feasible and desirable.

F. Soldo et al. [8] presented the SUV protocol, a distributed solution to disseminate video streams in VANETs, although a special MAC layer is required to support TDMA scheduling, which avoids its implementation on actual IEEE 802.11p devices.

Overall, although the aforementioned works provide a good number of statistics, none of them presents actual video quality results such as PSNR, being unaware of decoding problems such as the interdependence between frames.

A first approach towards a proper simulation environment capable of representing real time video transmission was proposed in [9], where authors present a simulation platform and PSNR results, following the guidelines presented in [10], thereby providing a useful approach on how to simulate a correct video transmission.

In this paper we present an evaluation of different real time video flooding methods, evaluating the error resilience in terms of frame loss and PSNR for the standard H.264 codec and for the new H.265 standard.

III. Overview of the video delivery strategy

Based on the current state of the art, we have implemented two video flooding solutions focusing on some desirable characteristics:

1. Achieve a high percentage of delivered packets.
2. Promote a fast message propagation.
3. Make the implementation feasible in a real environment.

To meet these goals, our implementation of the flooding mechanisms is located at the application layer, using UDP as the transport mechanism. This selection avoids the possibility of aborting the transmission of a message, and also impedes us from obtaining MAC-layer statistics, such as the state of the wireless channel or the number of collisions, while processing the incoming messages. However, the implementation does not depend of any tweaked MAC layer, and it can be deployed on real existing devices.

The two flooding strategies implemented are the Counter-Based strategy and the Distance-Based strategy. Below we provide a brief overview of both of them.

Counter-Based strategy

The parameters employed by this algorithm are two: the first one is the number of copies (C) that a node should hear to stop rebroadcasting a message, and the second one is the maximum time ($MaxTime$) to rebroadcast.

The exact behavior of our implementation is the following:

1. When a new message arrives, initialize the *seen* counter to 0. If the received message was heard previously, increment the local *seen* counter.
2. Wait a random time between 0 and $MaxTime$.
3. If $seen < C$, rebroadcast the message, go back to point number 2 and wait for $MaxTime$. In the other case ($seen >= C$), discard the message.

Distance-Based strategy

Implemented based on the Counter-Based strategy, our implementation adds a new parameter, the minimum distance ($MinDistance$) to rebroadcast a message without having to wait for a period of $MaxTime$.

For this implementation it is assumed that every vehicle is equipped with GPS, that the messages are marked with the position of the original sender and, if applicable, with the position of the rebroadcasting node.

The exact behavior of our implementation is the following:

1. When a new message arrives initialize the *seen* counter to 0. If the received message was heard previously increment the local *seen* counter.
2. Obtain the distances to the original sending node ($OriginalDistance$) and the rebroadcaster node ($RebroadcasterDistance$).
3. Wait for a time between 0 and $MaxTime$, where the time is inversely proportional to the minimum of the previously obtained distances.
4. If $seen < C$, rebroadcast the message, go back to point number 2 and wait for $MaxTime$. In the other case ($seen >= C$), discard the message.

IV. Simulation Environment

The simulation environment consists of three main components: SUMO [11], OMNeT++ [12] and INET [13].

To achieve a realistic vehicle mobility we employ the well known open-source vehicular traffic simulator SUMO (Simulation of Urban MObility), which runs coupled with the event-driven OMNeT++ simulator. To simulate the wireless environment we use the INET package that provides an implementation of the IEEE 802.11p standard, as well as higher application levels such as TCP and UDP.

The transmission range in the INET framework is not defined as a fixed distance. Instead, it requires tuning different parameters such as the frequency or the level of attenuation with distance. To achieve a proper level of similarity with reality, we employed the parameters proposed by Báguena et al. [14].

Since highways are a very specific scenario due to the high speeds involved, we have reduced the default SUMO step time from 1s to 0.1s. This provides a greater resemblance with real-life behavior since vehicles only move for a distance of 3 meters on each mobility simulation step, while in the default case they would move by 30 meters. Another issue handled by SUMO is the speed of the different nodes; in this case we have created four types of vehicles with different probabilities and speeds in order to achieve realistic vehicle behaviors in terms of overtaking, avoiding large vehicle queues on highways.

Concerning the scenario itself, the highway consists of a two-lane one-way 10km long straight line. A set of 11 Road Side Units (RSU) have been placed along the road every kilometer. The distance to the road is 5 meters. Scenario details are provided in figure 1.

In our experiments, RSUs do not resend any messages, merely acting as traffic sinks. Our goal is to determine the performance of video delivery at RSUs at different distances from the source vehicle.

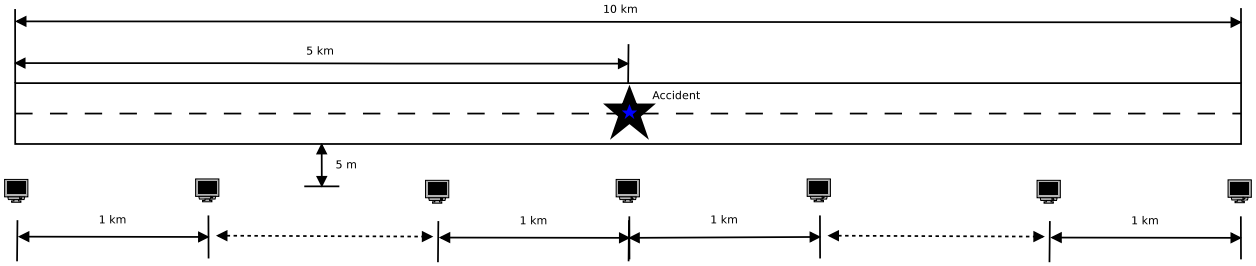


Fig. 1. Overview of the simulated scenario.

The vehicle acting as a video source is stopped when it arrives at middle of the scenario (fifth kilometer) and starts transmitting a video sequence simulating an accident which involves an emergency video stream.

The actual transmitted video is the CIF version of the "Highway" video sequence, which contains 2000 frames [15]. The video has been compressed with the H.264 and H.265 reference encoders, both set to a quality level of about 37.9 dB. (37.95 dB and 37.87 dB respectively). This quality level corresponds to data rates of 347 kbit/s and 283 kbit/s for H.264 and H.265, respectively.

Additionally, to add real-time constraints, we model a video buffer of $1s$, meaning that each vehicle will start the video playback $1s$ after the arrival of the first packet, and will discard every packet received after this time limit.

V. Results

This section is structured in the following way: first we make an analysis of the different parameters for both flooding algorithms - distance and counter-based - at different vehicle densities. Afterward, and focusing on the video codecs, we will compare the two codecs analyzed in terms of PSNR and frame loss using the best transmission algorithm according to the previous analysis.

A. Tuning the flooding algorithms

The main objective of a good flooding algorithm is to reduce the number of transmissions to avoid the broadcast storm problem, while achieving a good packet delivery ratio.

Concerning the first algorithm (Counter-Based) we tested three levels for each parameter, achieving a total of nine configurations. In this case, the C value denotes the additional number of times that a host should hear the message to stop rebroadcasting it, and $MaxTime$ stands for the maximum time (in ms) that a node waits when attempting to rebroadcast a message. The second algorithm (Distance-Based) adds another parameter, $MinDistance$. As explained before, if the receiver node is closer than $MinDistance$ to the sender node, it will wait for $MaxTime$ to rebroadcast the message. If the distance is higher, the node will rebroadcast earlier to maximize the number of useful transmissions.

The actual set of values employed for these simulations are the following:

- $C = \{1, 2, 3\}$
- $MaxTime = \{50ms, 200ms, 333ms\}$
- $MinDistance = \{50m, 250m, 400m\}$

Figure 2 shows the performance of both algorithms when the vehicle inter-arrival time is an exponential with a mean of $1s$ (high-density scenario).

The best configuration for the Counter-based solution is, without any doubt, the one with $C = 1$ and $MaxTime = 333ms$ one. This allows achieving a delivery rate of 72.23% at the five kilometers point, and a 82.63% at the third kilometer. This configuration clearly outperforms the other ones due to the lowering in the number of transmissions. Increasing the number of copies a node should receive to stop rebroadcasting significantly reduces the effectiveness of the flooding process since it just adds more interference to the medium. The same thing happens when we reduce the $MaxTime$ parameter, as nodes have more probability to collide when gaining access to the medium.

Looking at the right part of figure 2, we can see the results for the Distance-based algorithm. Since the number of different configurations for this algorithm is too high to be shown in a graph, we selected the nine best performing configurations. The best arrival rate at five kilometers is obtained by the configuration: $C = 1$, $MinDistance = 50$ and $MaxTime = 200ms$, achieving a packet delivery ratio of 78.5% at five kilometers and an 86.45% at the third kilometer.

If we compare both solutions, the Distance-Based algorithm is able to deliver more packets since it provides a better diffusion of the information, prioritizing the furthest nodes and making each transmission more profitable.

Figure 3 provides the data for vehicle inter-arrival times of $2s$ (low-density scenario), and shows that the best Counter-Based configuration is the same than in a high-density scenario but, due to the lower number of vehicles, the arrival rate is clearly reduced, achieving only a 50.34% of delivered packets at a distance of five kilometers.

Switching to the Distance-Based algorithms, the graph shows that the previous configuration is not the best one for the low-density scenario. The best configuration for this scenario is $C = 2$, $MinDistance = 50$ and $MaxTime = 333ms$, achieving a delivery rate at the fifth kilometer of 66.53%. This change in the best configuration is

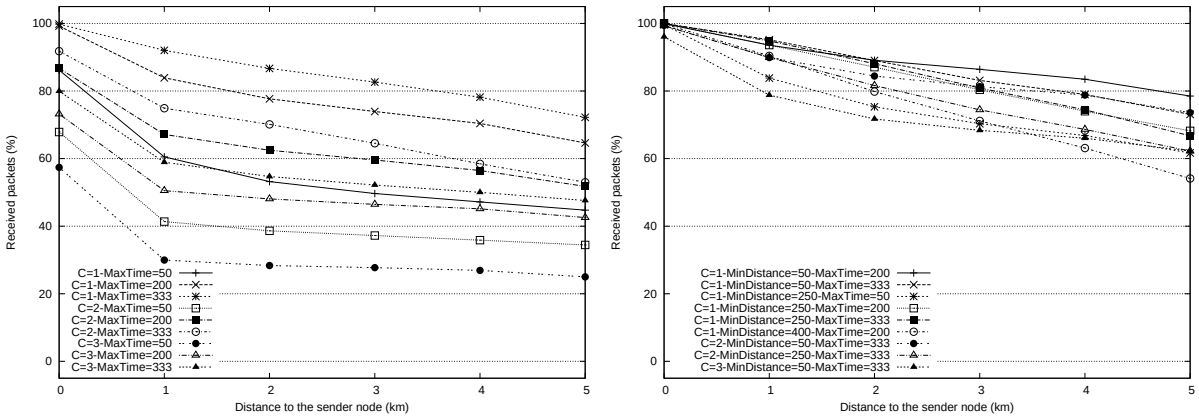


Fig. 2. Counter-Based (left) vs. Distance-Based (right). Vehicle arrival rate of 1s.

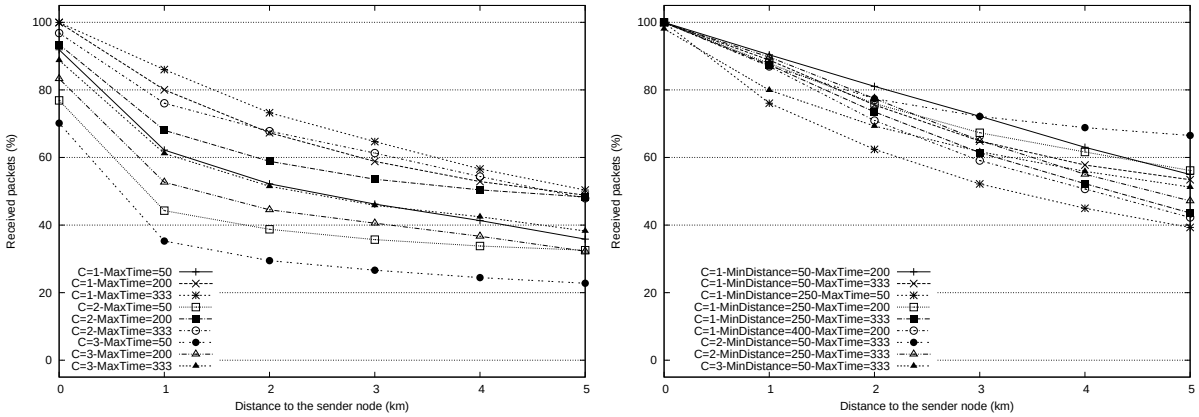


Fig. 3. Counter-Based (left) vs. Distance-Based (right). Vehicle arrival rate of 2s.

the result of collisions near the source vehicle. On the right hand side of the figure 3, and focusing on the delivery rates from $0km$ to $3km$, the best configuration is $C = 1$, $MinDistance = 50$ and $MaxTime = 200ms$, while from the third to the fifth kilometer one the best configuration is the one referred above. The main difference between these two configurations is the C parameter, as it stands for the number of times a message should be heard before stopping its rebroadcasting.

B. *Evaluating the best codec - flooding algorithm combination*

With the purpose of evaluating the actual received video quality, we have select the three best flooding configurations analyzed in the previous section.

Figure 4 shows the ratio of packets received for different vehicle arrival rates depending on the codec employed. The three graphs clearly show the differences between flooding backwards or forwards. When flooding backwards (from kilometer 0 to 5) the delivery ratio increases as the vehicles start creating a traffic jam, obtaining a higher density of vehicles which permits the use of more vehicles as relays. On the other hand, forward flooding is a harder task since, when vehicles surpass the accident position, they continue driving at a high speed, therefore creating several disjoint groups.

Focusing on the differences between codecs, when employing the same flooding configuration, the H.264 compressed video has a lower packet arrival rate than the H.265 compressed video due to the higher bitrate that H.264 injects into the network.

The best flooding configuration is different for each codec since they have different traffic demands. The best configuration for the H.264 codec is the Distance-Based algorithm with a configuration of $C = 1$, $MinDistance = 50m$, and $MaxTime = 200ms$. The H.265 codec has a lower requirement in terms of bandwidth and allows increasing the C value to add more redundancy without increasing the number of collisions. Figure 4 shows that a Distance-Based flooding algorithm with a configuration of $C = 2$, $MinDistance = 50m$, and $MaxTime = 333ms$ is able to achieve the best packet arrival rate in the low-density scenario, while maintaining a good performance in the high-density scenario.

C. *Error resilience and impact on the user-perceived quality*

As in previous section, we have selected the optimum flooding configuration for each codec to obtain the frame loss and the PSNR results.

Figure 5 shows the frame loss when employing the best configuration for each codec. While the H.265 codec is able to maintain frame loss at low levels, the H.264 codec suffers an extremely high frame loss when packet losses occur. In fact, the H.264 decoder

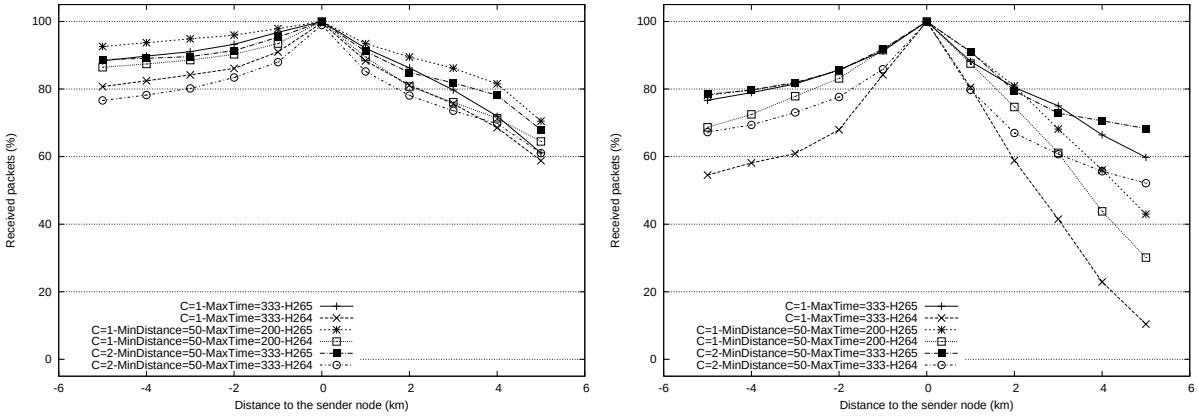


Fig. 4. Packet delivery ratio. Vehicle arrival rate of 1s (left), 2s (right).

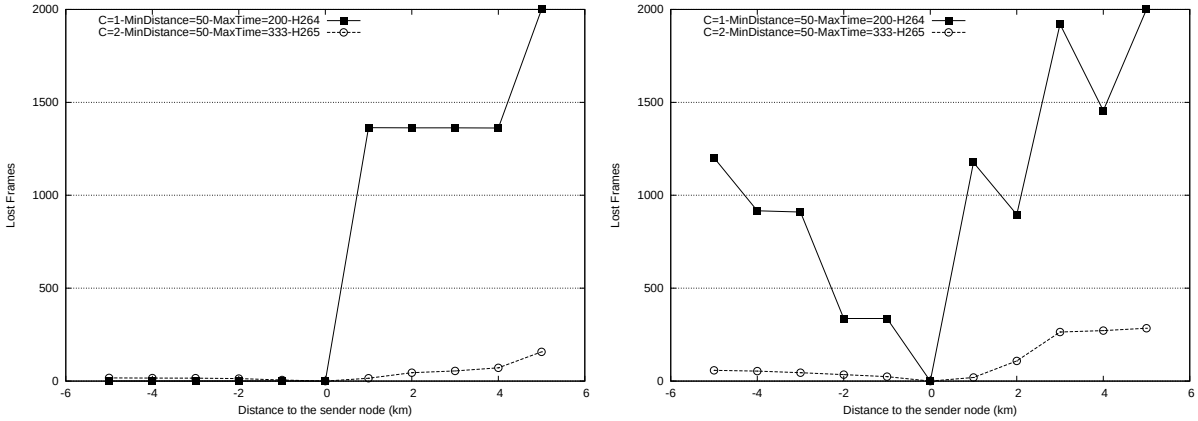


Fig. 5. Frame loss. Vehicle arrival rate of 1s (left), 2s (right).

is often unable to decode several consecutive frames. Although the packet loss difference for both codecs is not very high, H.264 suffers from packet losses in almost every frame, being the decoder unable to properly reconstruct the original video despite using error recovery techniques like frame-freezing. On the other hand, the H.265 decoder is able to compensate for packet losses with the frame-freezing technique, being able to provide a high number of decoded frames, even though they may have some glitches that affect both the PSNR and the visual quality.

When the vehicle arrival rate is 1s (high-density scenario) the H.264 codec is able to decode all the frames of the video when flooding backwards, while in the same scenario, when flooding forward (from kilometer 5 to 10), significant frame losses occur even for similar packet loss values. To gain further insight into this phenomenon, we have analyzed the packets loss patterns, detecting differences in terms of the frame types affected. In particular, more I-frame losses occur in the forward transmission, being the I-frames type (reference images) crucial to achieve a proper decoding of the video, while the P-frames and B-frames (frames that encode differences between the current frame and the previous I-frames) are not so critical.

Figure 6 shows the PSNR for the decoded frames. Although in some cases H.264 performs better than H.265, we have to take into account the differences in the number of decoded frames, since H.264 is only able to decode frames with a reduced packet loss, while H.265 is usually able to decode the entire video sequence, which explains why the H.264 decoded frames have better quality than the H.265 decoded ones, on average.

When both decoders have similar frame loss ratios, H.265 performs slightly better than H.264 in terms of PSNR.

These graphs highlight that H.265 outperforms H.264 in terms of both frame loss and PSNR quality. From the user perspective, the video flooded using the H.265 codec is able to provide a much smoother experience than H.264 encoded videos.

For informative purposes, some samples of the decoded videos are made available for download[1].

VI. Conclusions and future work

In this paper we evaluated the user-perceived quality for real-time V2V video flooding in highway scenarios providing frame loss and PSNR data for the widely used H.264 codec and for the new H.265 video codec.

In our experiments we compared two flooding algorithms and selected the best configurations for each of the codecs in terms of frame loss. Afterward, a detailed analysis in terms of frame loss and PSNR for

[1] http://www.grc.upv.es/Jornadas-2013/videos.html

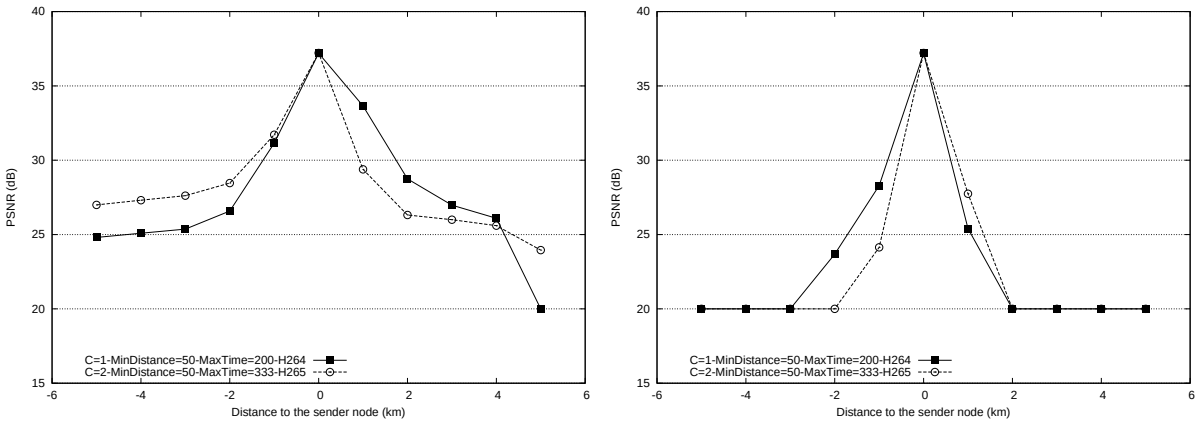


Fig. 6. PSNR. Vehicle arrival rate of 1s (left), 2s (right).

both of the codecs and for both forward and backward highway directions is provided.

H.265 has shown to perform way better than the H.264 codec, mainly due to the excessive frame loss provoked by the latter. Additionally, H.265 is able to maintain a controlled frame loss in all the situations, while H.264 is unable to provide an adequate frame rate to achieve a good user experience. In terms of PSNR for the decoded frames, both compression algorithms are unable to sustain the PSNR over $30dB$ for distances greater than 1km, although the H.265 codec is able to provide a better visual quality than H.264 for a comparable frame loss rate.

Reducing the packet loss in general, especially losses associated with key-frames, to avoid massive frame loss is a key factor to achieve a good user experience. So, as a future work, we plan to design a new flooding algorithm that takes all these considerations into account.

Acknowledgments

This work was partially supported by the *Ministerio de Economía y Competitividad*, Spain, under Grant TIN2011-27543-C03-01, and by the *Ministerio de Educación*, Spain, under the FPU program, AP2009-2415.

References

[1] "H.265 standard," http://www.itu.int/rec/T-REC-H.265.

[2] "H.264 standard," http://www.itu.int/rec/T-REC-H.264.

[3] H. Koumaras, M. Kourtis, and Drakoulis Martakos, "Benchmarking the encoding efficiency of h.265/hevc and h.264/avc," in *Future Network Mobile Summit (FutureNetw), 2012*, 2012, pp. 1–7.

[4] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, New York, NY, USA, 1999, MobiCom '99, pp. 151–162, ACM.

[5] Yu-Chee Tseng, Sze-Yao Ni, and En-Yu Shih, "Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network," *Computers, IEEE Transactions on*, vol. 52, no. 5, pp. 545 – 557, may 2003.

[6] FranciscoJ. Martinez, Manuel Fogue, Manuel Coll, Juan-Carlos Cano, CarlosT. Calafate, and Pietro Manzoni, "Evaluating the impact of a novel warning message dissemination scheme for vanets using real city maps," in *NETWORKING 2010*, Mark Crovella, LauraMarie Feeney, Dan Rubenstein, and S.V. Raghavan, Eds., vol. 6091 of *Lecture Notes in Computer Science*, pp. 265–276. Springer Berlin Heidelberg, 2010.

[7] Meng Guo, M.H. Ammar, and E.W. Zegura, "V3: a vehicle-to-vehicle live video streaming architecture," in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, 2005, pp. 171–180.

[8] F. Soldo, C. Casetti, C. Chiasserini, and P.A. Chaparro, "Video streaming distribution in vanets," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 7, pp. 1085–1091, 2011.

[9] Pablo Piñol, Otoniel López, Miguel Martínez, José Oliver, and Manuel P. Malumbres, "Modeling video streaming over vanets," in *Proceedings of the 7th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, New York, NY, USA, 2012, PM2HW2N '12, pp. 7–14, ACM.

[10] Patrick Seeling and Martin Reisslein, "Video transport evaluation with h.264 video traces," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 4, pp. 1142–1165, 2012.

[11] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, Barcelona, Spain, October 2011, pp. 63–68.

[12] "OMNeT++ simulator," http://www.omnetpp.org/, Acessed: April 4, 2013.

[13] "INET framework," http://inet.omnetpp.org/, Acessed: April 4, 2013.

[14] M. Baguena, C.T. Calafate, J. Cano, and P. Manzoni, "Towards realistic vehicular network simulation models," in *Wireless Days (WD), 2012 IFIP*, 2012, pp. 1–3.

[15] "Video Traces," http://trace.eas.asu.edu, Acessed: January 15, 2013.

Comparing Different DTN Protocols Under Realistic Urban Environments

Sergio Martínez Tornell[1], Carlos T. Calafate[2], Juan-Carlos Cano[3] y Pietro Manzoni[4]

Abstract— **The mobility of people and goods consumes a significant amount of resources. To minimize this consumption, new traffic management systems based on the Smart Cities paradigm have been proposed. These systems require collecting and distributing as much information about road conditions as possible. Traditionally, this information collection was carried out by means of passive elements, typically by coils or road side cameras. In previous works, we have proposed to use a Vehicular Delay Tolerant Network (VDTN) to collect and distribute this information. However, achieving a fair comparison between different DTN solutions in an urban environment is not easy. In this paper we present a generic DTN model that we use to compare various representative DTN solutions in an urban scenario. We highlight the weak and strong points of each evaluated proposal.**

keywords— **Delay tolerant networks, Vehicular networks, VANET, Modeling.**

I. Introduction

INTELLIGENT Transportation Systems (ITS) require collecting and distributing as much relevant information as possible to provide their services. Recently, the number of sensors available in our vehicles increased notably. Nowadays, vehicles carry sensors not only related to engine status or speed, but also related to weather (rain and temperature sensor) or the state of the road (ESP sensors). If the authorities in charge of traffic management and safety were able to harvest this information, the required investments in infrastructure and road deployed sensors could be significantly reduced. Smart City service providers could also take advantage of this huge amount of data.

The distribution of this information is carried out through various vehicular networking strategies, the most flexible of all being Delay Tolerant Networking (DTN) . DTN protocols can cope with the problems derived from high mobility and the possibility of high node sparsity. Nevertheless, achieving a fair comparison of DTN solutions in an urban environment is a hard task.

When comparing and evaluating the performance of different Delay Tolerant Network (DTN) protocols, the criteria used to select the next forwarding node, also called the "routing metric", is not the only element that can make the difference. Several low-level sending mechanisms for one-hop communications such as: (a) the use of ACK packets, (b) the existence of flow control mechanisms, or (c) whether the protocol uses unicast or broadcast messages, heavily affect its performance.

The classical DTN protocols, e.g., [1], [2], [3] were not specifically designed for vehicular contexts. More recently, DTN *geographic protocols* have been presented for this specific purpose. The most simple geographic protocol is the *Greedy* protocol, which is a DTN variation of another non-DTN geographic protocol, called GPSR [4]. In [5] the authors present *GeOpps*, one of the most advanced DTN protocols for Vehicular Ad-Hoc Networks (VANETs), and compare it against Greedy and MoVe [6], the latter originally designed for Mobile Ad-hoc NETworks (MANETs) and not considering geographic information. In the comparison, authors does not specify which propagation model was used, leading to biased results, as stated in [7]. In [8] GeoDTN+Nav is presented and compared to GPCR [9], which is a non-DTN protocol. From our point of view, any DTN protocol performs better than a non-DTN protocol in a sparse network, so it would have been more valuable to compare GeoDTN+Nav against others DTN protocols such as GeOpps or GeoSpray [10].

In this paper we present a generic DTN model called Generic One-Copy DTN Model (GOD) with the objective of fairly comparing various DTN solutions in a metropolitan scenario. Through this model, and using simulations, we highlight the weak and strong points of the various proposals studied. In our study, we compare a protocol we developed, called Map-based Sensor-data Delivery Protocol (MSDP)[11], against the GeOpps and the Greedy protocols using accurate mobility and propagation models.

This paper is organized as follows: in Section II we present our Generic One-Copy DTN Model (GOD). The compared protocols are introduced in Section III. The simulation environment and settings are presented in Section IV. Section V presents our results and findings. Finally, section VI concludes this paper and provides details about future work.

II. The Generic One-Copy DTN Model

A DTN protocol can be subdivided into various components, the most important one being the forwarding criteria. The forwarding criteria, also known as the routing metric, represents the criteria the protocol uses to chose the next forwarding node. Other minor and more generic components or mechanisms, such as the use of ACK packets or the adoption of flow control mechanisms, can anyway heavily influ-

[1]Dpto. de Informática, Univ. Pol. de València, e-mail: sermarto@upv.es
[2]Dpto. de Informática, Univ. Pol. de València, e-mail: calafate@disca.upv.es
[3]Dpto. de Informática, Univ. Pol. de València, e-mail: jucano@disca.upv.es
[4]Dpto. de Informática, Univ. Pol. de València, e-mail: pmanzoni@disca.upv.es

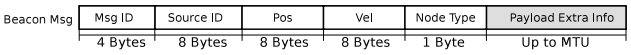


Fig. 1: Beacon messages format.

ence the performance of the DTN protocol as well.

In order to achieve a fair comparison between different DTN protocols we have developed a Generic One-Copy DTN Model (GOD), which can be considered as a super-class of every DTN protocol. This approach not only simplifies the comparison of different DTN solutions, but could also speed up the implementation of new protocols.

Our model integrates within the TCP/IP protocols architecture as a new layer between the application and the transport layers, allowing our DTN protocol to work independently of the IP routing protocol.

Our GOD model implements the following configurable generic mechanisms:

- ACK Messages: ACK messages are used to confirm every one-hop transmission, to ensure that no fragment is lost during a one-hop transmission;
- Unconfirmed messages: maximum number of unconfirmed data messages. This is specially useful when the communications channel is unstable.
- Redundancy: redundancy fragments can be added to reduce the impact of fragment losses.
- Location: an interface to the Navigation System (NS) to allow the use of geographic data, such as location, direction, programmed route, etc.

The GOD structure is based around three different modules which work coupled: the beacon module, the fragment generator module, and the core module.

A. The beacon module

The beacon module implements the announcements mechanism. It periodically broadcasts the node information, and it also manages the collected information in order to construct a list of the current neighbors. Every beacon packet contains the source address as well as its location, its velocity, its direction, and the node type obtained from the Navigation System. Moreover, extra information may be included within the beacon packet payload if required by a specific DTN protocol; Figure 1 shows the beacons message format.

A *New Neighbor Event* is notified to the **core module** every time a new neighbor is detected. When a certain number of consecutive beacons from the same neighbor are lost, a *Neighbor Disconnected Event* is notified to the **core module**. This module also notifies the **core module** when the information about a neighbor has been updated based on a new received beacon.

Inside this module, the inter-beacon time can be defined to meet protocol requirements. Currently, we only support static inter-beacon times, but in the future we plan to implement dynamic inter-beacon times that may depend, for example, on mobility, or on network density parameters.

B. The fragments generator module

This module is directly connected to the application layer and it is in charge, if required, of dividing large messages into fragments smaller than the MTU. Since the GOD uses UDP packets for one-hop communication, large messages must be split-up to avoid IP fragmentation, which would interfere with routing decisions. Figure 2 shows the relationship between the information messages and their fragments. Then, the fragments are sent to the core module. The fragments generator module can also create **redundancy** fragments to increase reliability. When redundancy is enabled, a percentage of extra fragments are created using Foward Error Correction (FEC) techniques. That is, if a message is divided into N fragments, $N * \alpha$ total fragments will be generated, where redundancy factor α is greater than 1 and depends on the configuration. The basic hypothesis is that the original message can be reassembled with whatever subset of size N of the sent fragments. This redundancy allows reducing the impact of possible fragment losses.

C. The core module

This module is connected to the beacon module, and it is in charge of managing transmission opportunities between nodes. The core module has a buffer where fragments are stored. Fragments are enqueued and dequeued based on their timestamp: older fragments receive higher priority. The events notified by the beacon module are used to keep a sorted list of neighbors according to the chosen routing metric. This module is activated by five different types of event:

- *Data Messages.* A Data message can arrive from the fragments generator module or from the network, i.e., from a neighbor. When a Data message arrives, the fragments contained in it are enqueued in the local buffer and, in case it is enabled, its reception is confirmed through an ACK message. This ACK allows the previous source node to remove the confirmed fragments from its buffer.
- *ACK messages.* An ACK message confirms the reception of a Data message, and the fragments contained inside the confirmed Data message can now be definitely deleted from the buffer.
- *ACK Time Out.* When an ACK Time Out occurs, the data fragments contained in the unconfirmed data message are re-enqueued in the

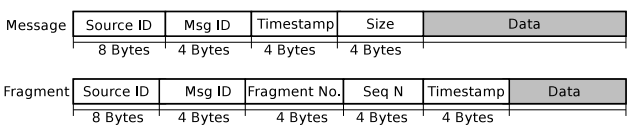


Fig. 2: Information message format and fragment format.

sending buffer.

- *New Neighbor and Neighbor Changed.* Whenever one of these events is notified, the value of the transmission window is restarted. The neighbor is evaluated using the metric defined by the specific DTN protocol, and, in case the neighbor is evaluated as a "better" node than the current carrier, a new transmission process is started. Obviously, if the neighbor is detected as an Road Side Unit (RSU), it is always evaluated as the best possible neighbor and a new transmission process is immediately started.

The transmission process tries to transmit as much fragments as possible to the best neighbor; its behavior is as follows:

1. The module checks if there are fragments in the buffer to be transmitted; if there are no fragments, the process is over.
2. It obtains the best next node from the node list according to the protocol-specific metric.
3. It checks if the best node's location estimation is inside the defined transmission range.
4. It checks if the transmission window of the best node is equal to 0; in this case, the process is over.
5. It sends a data packet containing as many fragments as possible. This action dequeues the fragments from the buffer.
6. If the ACK mechanism is enabled, it schedules an ACK Time Out for the previously sent data message. If it is not running, it removes the sent fragments.
7. It waits for the time required to send a data packet and its corresponding ACK packets.
8. It starts a new transmission process.

It is worth noticing that, since all the fragments have the same destination (any of the RSUs), the next forwarding node selection is based only on the state of the current carrier and its neighbors. Moreover, we consider that all the RSUs behave as *sinks*. They simply send beacons to announce their presence to vehicles. When vehicles send Data messages to the RSUs, they may confirm their reception through an ACK message if required, and the fragments encapsulated in it are forwarded to the control center through the backbone network.

With GOD, when the ACK mechanism is active, it is ensured that no fragment is removed from the buffer until a neighbor have been confirmed as the new carrier. ACK messages are also used to limit the number of data messages pending confirmation. The use of ACK, as well as the communication type, *i.e.* broadcast or unicast, can be configured in order to model different DTN protocol variants.

III. Overview of the evaluated DTN Protocols

We used our *Generic DTN One-Copy protocol* to implement and compare three different DTN protocols, namely: *Greedy*, *GeOpps*, and *MSDP*. A common assumption of all these protocols is the presence of a Navigation System (NS) installed in the vehicles. This way, each vehicle is aware of its geographical location and its route. This information is used to increase the packet delivery ratio in DTNs.

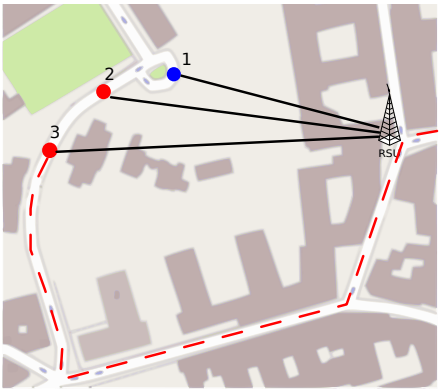


Fig. 3: Example of a local minimum: messages get blocked at node 1.

A. Greedy

The *Greedy* protocol, or a slightly modified version of it, has been widely used in order to evaluate other proposals, e.g., [5], [8]. It is a DTN variation of existing *location-based* greedy algorithms [12], [9], where the fragments are forwarded to the neighbor that is the closest one to the destination (if closer than the current carrier). Therefore, it uses the distance to the destination as its routing metric. This metric may lead to message losses or high delays when a geographic local minimum exists. Figure 3 shows an example of a local minimum where messages get blocked for a long time on the position of node number 1, since it is the closest node to the RSU.

B. GeOpps

The *GeOpps* protocol was presented in [5]. The next forwarding node selection is based on the Estimated Time of Arrival (ETA) to the destination. GeOpps assumes that each node is aware of the programed route in their Navigation System (NS). Using this information, nodes calculate the closest point to the destination along their route. Then, the closest point is used together with the map information to calculate the ETA from the current position to this closest point. The routing metric, which is called Minimum Estimated Time of Delivery (METD), is the sum of the ETA from the current position to the closest point plus the ETA from the closest point to the destination. Figure 4 shows an example of calculation of the closest point where the solid lines indicate the programmed routes while the dashed lines indicate the estimated route from the closest point to the destination. Finally, the value of the METD is attached to beacon messages, which allows nodes to be aware of the METD of their neighbors, and chose the minimum METD neighbor as the next forwarding node.

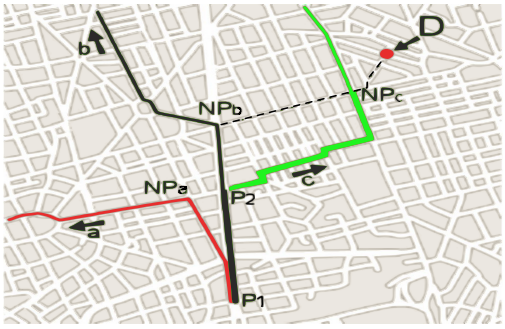


Fig. 4: Example of calculation of the closest point [5].

C. MSDP

In this article we used a slightly modified version of our MSDP protocol presented in [11]. In order to make routing decisions, MSDP uses the value of a function, called UtilityIndex, to determine which is the best neighbor to forward fragments to. In this work we add a second term to the UtilityIndex, $1/D$, which relates to the distance between the node and the destination of the message. The UtilityIndex, which depends on four parameters, is calculated locally and attached to beacons. The higher the UtilityIndex is, the better the candidate, see Equation 1.

$$UtilityIndex = \frac{P^2}{T} * Q + 1/D \qquad (1)$$

Parameters P, T, Q, and D refer to *Trustworthy factor*, *Time to reach an RSU*, *Transmission availability*, and *Distance to an RSU*, respectively. In this paper we will only explain in detail this new term, and refer to our previous publication for the rest of the parameters. The previous version of MSDP suffered of inactivity when the carrier node and all its neighbors experienced a value of 0 for the first term of the equation 1. Under this circumstance, no messages were forwarded. By the parameter D we ensure that, at least, messages will be forwarded to the closest neighbor, which increases the probability of eventually finding a node whose first term of the equation is bigger than 0.

IV. Simulation Environment

In this Section we use our *Generic On-Copy Dtn Model* to compare our MSDP against the *Greedy* and *GeOpps* protocols through simulations.

We consider as the reference scenario the one selected in [11]. In that work vehicles used a DTN protocol to collect information from a vehicular sensor network and to deliver it to a remote control center. The information is obtained from in-vehicle sensors, which retrieve it using an On Board Diagnostics 2 (OBD-II) unit [13]. The information messages are then fragmented and routed. A fragment is considered to be delivered when any of the RSUs correctly receive it. Once an RSU receives a fragment, the fragment is sent over the backbone network to the control center. The control center will then reassemble the fragments into the original message and process the content. RSUs are supposed to be placed in strategical places by entities interested in collecting the information, like city councils, or road administrators. The locations of the RSUs are available to vehicles' NSs through a dynamic updating service.

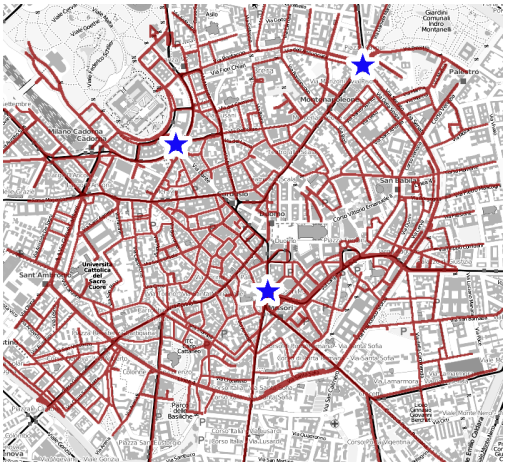


Fig. 5: Map used in our simulations, stars indicate the location of the RSUs (2.6 x 2.6 km).

We implemented all the models using the Inet framework for the Omnet++ event-driven simulator [14]. The Inet framework includes detailed implementations of the 802.11 physical and MAC layers.

One of the most important issues in DTN simulations is the mobility of nodes. In our simulations we have generated the node mobility by using VaCaMobil [15]. VACaMobil introduces a configured number of vehicles in the network and keeps this number inside the user defined upper and lower bounds. When a node finishes its trip, it is removed from the network. When using VACaMobil, the network simulator can influence the mobility of the nodes, which is simulated using Simulation of Urban MObility (SUMO) [16]. For the layout we used a 6.7 km^2-area map of the city center of Milan, which has a typical European old city structure. Figure 5 shows the portion of the map used, and the locations of the RSUs are indicated by stars.

We consider that the use of a very simplistic propagation model is one of the main drawbacks of previous studies in this topic. To accurately model real world conditions, we used the propagation model presented in [17], which combines the Nakagami fading model [18] with a visibility model which deals with power losses due to the effect of obstacles.

Every node in our network scenario generates a 2 kBytes message every 10 seconds. The size of the fragments is 450 Bytes. The simulation lasts 3600 seconds, and nodes will generate traffic throughout the entire simulation. Concerning the communication interfaces, each node has two 5.9 GHz 802.11p interfaces tuned at different channels, where the first one is used for beacon broadcasting, while the second one is used for one-hop transmissions. This double-interface model mimics the multi-channel communications scheme of the Dedicated short-range communications (DSRC) [19]. The transmission power of both interfaces is configured to 63 mW, while the

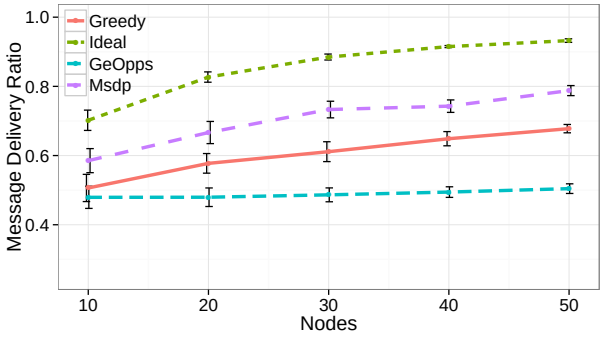


Fig. 6: Delivery Ratio

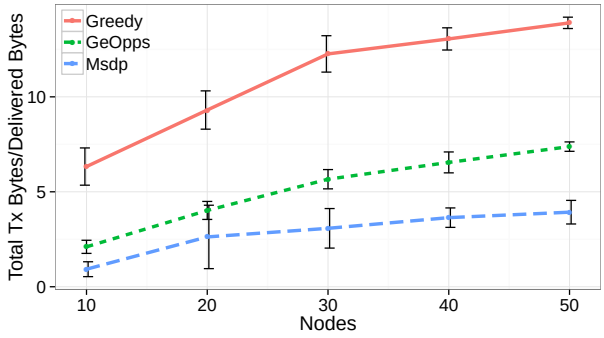


Fig. 7: Total Tx Bytes per Delivered Bytes

gain of the antenna is 5 dBi.

To achieve reasonably conclusive results, we have simulated every scenario 10 times, varying the seed of the simulation. Measures are represented with a 95% confidence interval.

V. Results

In this section we compare the three evaluated protocols according to: the delivery ratio, and the consumed resources (overhead). To evaluate the impact of node density we vary the number of nodes in the network from 10 to 50, thus varying the node density from 1.47 to 7.4 nodes/km^2. For this evaluation we configure the GOD model in order to use unicast communications an ACK messages.

A. Delivery Ratio

The delivery ratio is the ratio between the effectively delivered and the sent messages. Due to mobility-related effects, it is typically impossible to achieve a perfect delivery ratio. To determine a reference upper bound for this metric we have implemented an *Ideal Protocol*. The *Ideal Protocol* is an implementation of the Epidemic protocol [1] that runs on the top of a collision-free MAC layer which only considers propagation and transmission delays, and defines a limited transmission range. Since a copy of every fragment is sent to all neighbors every time a contact occurs, the first copy of a fragment arriving to any of the sinks must have traversed the best possible path in terms of delay.

Figure 6 shows the delivery ratio of the different protocols; it is worth noticing that not even the Ideal protocol reaches a delivery ratio of 1. When there are very few nodes in the network, the delivery ratio depends a lot on the mobility patterns of nodes, which explains the high confidence intervals obtained. On the other side, as we increase the number of nodes in the network, the delivery ratio also increases, because more transmission opportunities become available. As can be seen, MSDP performs better than any other of the compared protocols, and GeOpps performs worse than Greedy.

B. Overhead

Last but not the least, we evaluate the overhead introduced by each protocol. To obtain the overhead ratio of the evaluated protocols, we divide the total number of transmitted bytes by the number of delivered bytes. This measurement is closely related with the number of hops a fragment must traverse before it arrives to an RSU. Since it does not take into account the amount of data interchanged between nodes and RSU, its value may be smaller than one when most of the delivered fragments are directly transmitted to an RSU.

Figure 7 shows the results we obtained for the different protocols; The first thing that we appreciate is that our MSDP is the one experiencing the smallest overhead ratio. Both GeOpps and MSDP perform better than the Greedy protocol, which consumes significantly more resources. This difference is explained by the presence of loops; when two nodes move in opposite directions, one node moving away from the RSU; and another one moving close to it, meet, the fragments are sent firstly to the closest one and then, after they cross, are sent back to the original carrier, which is moving closer to the RSU, the same situation occurs when a vehicle overtakes another vehicle. The UtilityIndex of MSDP, as well as the METD of GeOpps, are more stable and, therefore, most loops are avoided.

VI. Conclusions

In this paper we tackled the problems that arise when comparing different DTN protocols. To solve some of them, we have presented the Generic One-Copy DTN Model (GOD). This model allows us to fairly compare different DTN protocols, making it easy to define and apply common low-level sending mechanisms in one-hop communications.

We used GOD to fairly compare our previously presented proposal, called MSDP, against the Greedy and GeOpps protocols. The results obtained show that MSDP outperforms both Greedy and GeOpps protocols in terms of both delivery ratio and introduced overhead.

In this paper we also presented an Ideal DTN model that represents the best possible case where all the forwarding decisions are optimal. The inclusion of this model pointed out that MSDP still has room for improvements. In future works, we will analyze the optimal routes used by the Ideal DTN model in order to mimic them and make new improvements to our MSDP scheme.

Acknowledgments

This work was partially supported by the *Ministerio de Economía y Competitividad*, Spain, under Grants TIN2011-27543-C03-01 and BES-2012-052673.

References

[1] Amin Vahdat and David Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks," in *Technical Report CS-200006*. Apr. 2000, vol. CS-200006, pp. CS–2000–06, Duke University.

[2] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra, "Spray and wait," in *Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking - WDTN '05*, New York, New York, USA, 2005, WDTN '05, pp. 252–259, ACM Press.

[3] Ting-Kai Huang, Chia-Keng Lee, and Ling-Jyh Chen, "PRoPHET+: An Adaptive PRoPHET-Based Routing Protocol for Opportunistic Network," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. Apr. 2010, pp. 112–119, Ieee.

[4] Brad Karp and H T Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, New York, NY, USA, 2000, ACM, number MobiCom in MobiCom '00, pp. 243–254, ACM.

[5] Ilias Leontiadis and Cecilia Mascolo, "GeOpps: Geographical Opportunistic Routing for Vehicular Networks," in *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*. June 2007, pp. 1–6, IEEE.

[6] J LeBrun, Chen-Nee Chuah, D Ghosal, and M Zhang, "Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks," in *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, 2005, vol. 4, pp. 2289–2293 Vol. 4.

[7] Marc Torrent-Moreno, Steven Corroy, Felix Schmidt-Eisenlohr, and Hannes Hartenstein, "IEEE 802.11-based one-hop broadcast communications: understanding transmission success and failure under different radio propagation environments," in *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems - MSWiM '06*, New York, New York, USA, 2006, MSWiM '06, pp. 68–77, ACM Press.

[8] Pei-Chun Cheng, Kevin C. Lee, Mario Gerla, and Jérôme Härri, "GeoDTN+Nav: Geographic DTN Routing with Navigator Prediction for Urban Vehicular Environments," *Mobile Networks and Applications*, vol. 15, no. 1, pp. 61–82, June 2009.

[9] Christian Lochert, Martin Mauve, Holger Füß ler, and Hannes Hartenstein, "Geographic routing in city scenarios," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 1, pp. 69, 2005.

[10] Vasco N G J Soares, Joel J P C Rodrigues, and Farid Farahmand, "GeoSpray: A Geographic Routing Protocol for Vehicular Delay-Tolerant Networks," *Information Fusion*, Nov. 2011.

[11] Sergio Martinez Tornell, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni, "A Map-based Sensor data Delivery Protocol for vehicular networks," in *2012 The 11th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, Ayia Napa, Cyprus, June 2012, pp. 1–8, IEEE.

[12] C Lochert, H Hartenstein, J Tian, H Fussler, D Hermann, and M Mauve, "A routing strategy for vehicular ad hoc networks in city environments," in *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*. ACM, 2003, vol. 2000, pp. 156–161, Ieee.

[13] International Organization for Standardization, "ISO 15765: Road vehicles, Diagnostics on Controller Area Networks (CAN)," 2004.

[14] "Omnet++," http://www.omnetpp.org/, last visit march 2013.

[15] Miguel Báguena, Sergio M Tornell, Alvaro Torres, Carlos T Calafate, Juan-Carlos Cano, and Pietro Manzoni, "VACaMobil: VANET Car Mobility Manager for OMNeT++," in *3rd IEEE International Workshop on Smart Communication Protocols and Algorithms (SCPA 2013) (ICC'13 - IEEE ICC'13 - Workshop SCPA)*, Budapest, Hungary, June 2013.

[16] M Behrisch, L Bieker, J Erdmann, and D Krajzewicz, "SUMO - Simulation of Urban MObility: An Overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, Barcelona, Spain, Oct. 2011, pp. 63–68.

[17] Miguel Baguena, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni, "Towards realistic vehicular network simulation models," in *2012 IFIP Wireless Days*. Nov. 2012, pp. 1–3, IEEE.

[18] TS Rappaport, *Wireless Communications-principles and practice 2002, edition second*, Prentice Hall PTR, 2 edition, Jan. 2002.

[19] Daniel Jiang and Luca Delgrossi, "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments," *VTC Spring 2008 IEEE Vehicular Technology Conference*, pp. 2036–2040, 2008.

A modeling tool offering steady-state mobility conditions in vehicular environments

Miguel Báguena, Sergio M. Tornell, Álvaro Torres, Carlos T. Calafate, Juan-Carlos Cano, Pietro Manzoni[1]

Abstract— **Designing representative vehicular network simulations is a particularly challenging task because since there are several new parameters which researchers must take care about. In this paper we evaluate VACaMobil, a mobility manager for the OMNeT++ simulator which guarantees a constant vehicle number throughout an entire network simulation. It can also be configured to create minor fluctuations around the average value, within user defined bounds, to get a realistic vehicle behavior. We examine VACaMobil's performance in a typical urban scenario and we compare its behavior against current tools provided for the SUMO simulator. Results show that VACaMobil is the only tool able to achieve and maintain the target average vehicle number in all scenarios.**

Keywords— **Vehicular Networks, Mobility patterns, Simulation Tool, SUMO, TraCI.**

I. Introduction

THE reproducibility of experiments is a major issue when evaluating smart communication protocols and algorithms, especially over Vehicular Ad-hoc NETworks (VANETs). In [1] the authors provide a complete review of the minimum set of parameters that should be identified in order to allow other researchers to reproduce simulation experiments. They pointed out several key parameters, such as the simulated hardware, the network simulator, the scenario, and the road traffic simulator. However, regarding node mobility, there are other parameters that have been mostly ignored by the research community: the density of traffic and the traffic demand.

As other authors pointed out in previous studies, mobility models [2] and the chosen scenario [3], as well as the node density, heavily influence the final network performance. However, since mobility generators and road traffic simulators are often tough to configure, the simulated node density and distribution may depend on complex data that are usually not included in the published academic results, thereby compromising reproducibility.

In this paper we present VACaMobil (VANET Car Mobility manager), a mobility manager module for the OMNeT++ simulator which is the first, to the best of our knowledge, able to generate SUMO [4] driven nodes in a vehicular network while ensuring certain user-defined parameters, such as the average, maximum, and minimum number of vehicles. This goal is useful for mid-length simulations, typically one hour, allowing researchers to assume that the vehicle density is stable. At the same time, since our solution is tightly coupled with SUMO through the TraCI interface, it is able to mimic real vehicle behavior. By running in parallel with SUMO, VACaMobil executes the following tasks: (i) it manages when a new vehicle must be introduced in the network, (ii) it assigns a random route from a predefined set to each vehicle, and (iii) it determines which type of vehicle should be added. When using VACaMobil, and given a specific road map, researchers will be able to completely define the network mobility merely by defining the desired average number of vehicles and its standard deviation value (upper and lower bounds).

Going a step further, our tool also aids researchers at selecting among the different types of vehicles previously defined in SUMO, such as "cars", "buses", or "trucks". This allows researchers to easily define road traffic simulations with heterogeneous vehicles.

The rest of this paper is organized as follows: In section II, we shortly introduce the different methods for generating VANET mobility patterns that the research community commonly uses. In section III, VACaMobil is fully described. In section IV, we compare our proposal with the duarouter and duaIterate.py tools, both included in SUMO. Finally, in section V, we present our conclusions and some future plans to improve VACaMobil.

II. A review of existing mobility generators for VANETs

Before presenting the details of our proposal, we analyze some of the methods commonly used to obtain suitable mobility patterns in urban vehicular scenarios. We have analyzed several papers published during the last few years, most of them published in conferences and journals related to Intelligent Transportation Systems. Early approaches relied on too simple mobility models based merely on random mobility. Since these simple models do not represent vehicle mobility properly, other mobility models have been recently developed based on real world traces, and also on artificial mobility models from the fields of transportation and traffic science. In this section, we briefly describe the most relevant works.

A. Random Vehicle Movement

At the beginning of the previous decade, the "Random Way-Point" was extensively used in Mobility Ad-Hoc NETwork (MANET) research. However, in 2003, the authors in [5] demonstrated how harmful the Random Way-Point mobility model really

[1]Department of Computer Engineering. Universitat Politècnica de València. Valencia, Spain mibaal@upvnet.upv.es, sermarto@upv.es, atcortes@batousay.com, {calafate,jucano,pmanzoni}@disca.upv.es

is. Moreover, the effects described in this work are even worse when simulating VANETs. Later on, some other authors have extended the "Random Way-Point" mobility model by restricting the mobility of nodes to a map layout, as in [6]. However, this improvement does not solve the majority of the "Random Way-Point" model problems stated previously.

In our research group we developed a tool called "CityMob" [7]. CityMob allows users to create random mobility patters restricted to a grid. It also adds support for *downtown* definition, where a *downtown* is a region inside the simulated map which concentrates the majority of the selected routes along the simulation. Although CityMob presents a big improvement compared to non restricted mobility models, as well as random mobility models, it also presents some problems; the most important one is that vehicular mobility is not influenced by other vehicles, *i.e.* two different vehicles can occupy the same location and no minimal distance between vehicles is required. Moreover, vehicles do not change their speed during a trip. However, in the real world, vehicles continuously change their speed according to traffic conditions and road characteristics. Last but not least, vehicles keep moving throughout the whole simulation, which especially influences the performance of protocols that keep data stored in buffers. The research community quickly realized the problems derived from inaccurate simulation patterns, and started to work on other methods to obtain suitable mobility traces.

B. Real Mobility Traces

Compared to the use of random mobility, real traces present a clear improvement. Such traces are usually obtained from a certain set of nodes, e.g. from taxis in the city of Shangai [8]. Mobility traces can be obtained by tracking the mobility of nodes using On-Board units, as in [8], or by using road-side equipment, as in [9]. Although real traces represent the most realistic mobility patterns, we can not obviate the fact that the mobility of the tracked nodes is highly influenced by the movement of other non tracked vehicles, *e.g.* taxis' mobility is influenced by other users on the road whose movement is not reflected in the collected traces. Moreover, real traces lack the flexibility to allow for an exhaustive evaluation of VANET protocols, *e.g.* changing the vehicle density without modifying their speed is clearly unreal.

C. Assisted Traffic Simulation

The restrictions of real traces can be overcome, with almost no loss of realism, by using mobility models taken from the field of transportation and traffic science. Several road traffic simulators are widely used among the VANET research community. One of the most widely used mobility generators is SUMO [4]. When simulating traffic mobility for VANETs not only the vehicles' behavior is important, but also the traffic demand. SUMO allows defining traffic demand in two different ways: trips and flows. The former defines only a vehicle, its origin and its destination, while the latter defines a set of vehicles which execute the same trip. SUMO currently provides several tools to generate traffic demand:

- randomTrips.py: A random trip generator. This tool generates a trip every second having a random origin and destination. It does not check if the origin and destination are connected, or whether the trip is possible.
- duarouter: A Dijkstra router. Given a file with trips and flows, this tool generates the actual traffic demand, expressed in vehicles with an assigned route. Routes are calculated using the Dijkstra algorithm, and every unconnected trip is discarded.
- duaIterate.py: This python script will produce a set of optimal routes from a trip file, *i.e.* all the nodes will follow that route which minimizes the total trip-time for all nodes. This tool repeats a routing-simulation loop until optimal routes are found.

Authors have used these tools in order to generate traffic demands for SUMO. The most simplistic one is to define different flows inside the network. Although drivers usually move from certain districts to others, following patterns associated with their working and living places, defining the traffic only by creating fixed flows lacks any realism, as we can see in [10] where only a few flows are defined by the user. Another common approach is to generate random trips using the *randomTrips.py* tool. This approach presents the problem that only one vehicle is introduced every second, which leads to long transitory periods until the network reaches a stable state. A more sophisticated traffic demand generation strategy is presented in [11], where a predefined number of vehicles following random routes are randomly placed at the beginning of the simulation. Following this trend, in previous works we used C4R [12], which is a software developed by our group to automate the task of generating random vehicles with random routes at random places. The work presented in [13] is the only one that we could find which uses the *duaIterate.py* script to generate a "stable and optimal distribution of flows". This type of traffic definition presents a problem: the trip duration can not be predicted before running the simulations, and, as a consequence, there is no way to ensure, or even determine, if the road traffic simulation will last until the end of the network simulation. As some works have stated before, this lack of realism and generality in mobility patterns can lead to biased results [2].

D. Bidirectionally coupled network and traffic simulations

In [14] its authors go a step further and present a new simulation framework called Veins, which in-

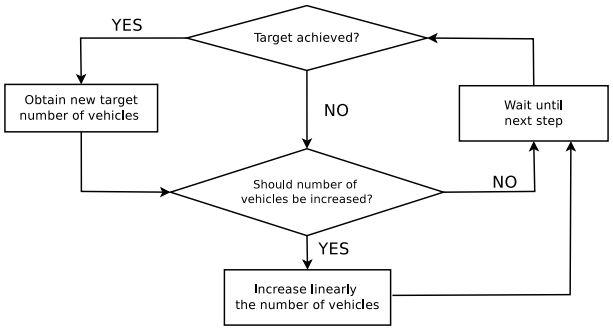


Fig. 1. Main loop of the VACaMobil tool.

cludes the TraCI interface to allow the network simulator to interact with the traffic simulator running in parallel. Although, it presents much novelty and opens a lot of possibilities for VANET simulation, authors do not address the traffic demand generation problem. This framework demonstrated its new characteristics in [15], and it is one of the main elements of our VACaMobil module, allowing us to interact with SUMO during the network simulation and create new vehicles.

III. VACaMobil Mobility Manager

In this section we present our VANET mobility generator providing the main characteristics and its implementation details.

A. Characteristics

The main characteristic of VACaMobil is to offer realistic mobility scenarios. To that end it can guarantee an average number of vehicles while keeping the current number of vehicles within the given upper and lower bounds, the latter being associated to the defined standard deviation value. These features allowed creating a tool that ensures repeatability of network simulations under the same road traffic conditions just by defining the average number of vehicles and the standard deviation value.

Another characteristic it offers is the possibility of introducing types of vehicles in simulations, such as “car”, “bus” and “truck”, each one with its own characteristics. This is an important feature because high level decisions may be based on the type of the vehicle.

VACaMobil also provides a realistic vehicle distribution based on a list of different predefined routes.

Finally it works on-line with the SUMO traffic simulator, obtaining all the needed information about routes and type of vehicles through the TraCI communication interface, thereby avoiding the duplicity of configuration files.

B. Implementation details

This tool extends the module collection available in the Veins framework [14] with new capabilities. We explain, for each of the characteristics described before, the different implementation decisions taken.

B.1 Average number of vehicles

Figure 1 shows the VACaMobil iterative control loop. At every step of the mobility simulation, VACaMobil compares the current number of vehicles in the simulation with the target number of vehicles. Depending on whether it is greater or lower than the target value, VACaMobil waits until the number of vehicles decreases towards the target value, or, in the second case, starts inserting several new vehicles in every step of the mobility simulation in order to increase the current value until the desired value is achieved. To avoid having an average number of vehicles higher than the one defined by the user, the time during which new vehicles are inserted is as long as the last period where the number of vehicles has decreased. By following this approach we also avoid high frequency fluctuations in the total number of vehicles throughout the simulation time.

Values for the *target number of vehicles* variable are obtained from a normal distribution whose mean is the desired average number of vehicles, and whose standard deviation is defined by the user. Its value is limited to the upper and lower bounds, which are defined as the $mean \pm 3 * standard\,deviation$: this value avoids extremely high or extremely low values for the current number of vehicles.

B.2 Different types of vehicles

One of the parameters we can obtain via TraCI indicates the types of vehicles that are available in the traffic simulation. The user can set different probabilities associated to each vehicle type. In this case, every time a new car is generated, we obtain a uniform random value and select the correspondent vehicle type. If no probability is defined for a certain type of vehicles, we assume it is equal to 0. However, if no probability value is assigned to any of the defined types of vehicle, only vehicles of the first defined type will be generated.

B.3 Routing set and vehicle distribution

Since SUMO itself loads all the different routes at startup, we can also retrieve them through TraCI, and, as in the previous item, we select one of them with an uniform probability every time we generate a new vehicle.

VACaMobil does not compute routes at simulation time; instead it relies on the goodness of the different routes made available by SUMO. To guarantee that vehicles are distributed realistically, we also developed a tool based on *duaIterate.py* and *randomTrips.py* which creates a SUMO route file with several random routes.

Finally, to ensure that a new vehicle is correctly added, the default behavior is to attempt to insert the vehicle at any of the lanes available on the first edge of the route. If the edge is full, the module selects a new route sequentially from a list, repeating the operation until it finds a free place to insert the vehicle, or until the first selected route is selected

TABLE I
VEHICLE STATISTICS SUMMARY

	mean	std. dev.
Target	225	8
duarouter	319.165	79.4342
duaIterate.py	219.610	34.2232
VACaMobil	223.718	8.32201

again, which means that there is no room on the road map for the new vehicle.

IV. EVALUATION

In this section we evaluate VACaMobil to verify whether the objectives and characteristics described in section III-A have been accomplished. In order to do that, we have compared VACaMobil against the tools currently included in SUMO, *i.e.* duarouter and duaIterate.py, that were described in section II. We have selected a urban real map which we extracted from the OpenStreetMap database. It is a scenario from the suburbs of Moscow characterized by long road segments and high capacity roads (Figure2).

In both scenarios, the set of random routes provided by VACaMobil is extracted from the traffic demand generated by duaIterate.py. In the following subsection, we compare the vehicle density and its evolution along the simulation time for the aforementioned tools and scenario.

A. Vehicle distribution study

We first evaluate one of the most important issues in vehicular mobility: how vehicles are distributed over the simulated road map.

Figure 2 shows performance results for the urban scenario. In this case, duarouter is unable to spread the vehicles properly. Since some roads are faster than others, all the vehicles are routed through them, even when these streets are congested. Therefore, an undesired traffic congestion is created in the fastest inner roads. However, this is an unrealistic scenario because drivers tend to avoid traffic jams whenever possible. When using either *duaIterate.py* or VACaMobil, vehicles are routed through alternative streets, avoiding traffic jams. This strategy has a higher degree of similitude compared to real road traffic, since drivers prefer faster roads but often change their route to avoid traffic jams.

B. Vehicle density study

To make simulations more easily comparable, a similar traffic density is desirable in all simulated city layouts. Current tools can not correctly handle this problem. In order to compare the behavior of the three selected methods previously presented, we have measured the average number of vehicles, its standard deviation, and its evolution along simulation time.

Table I shows the differences in terms of number of vehicles for the two target scenarios for each traffic

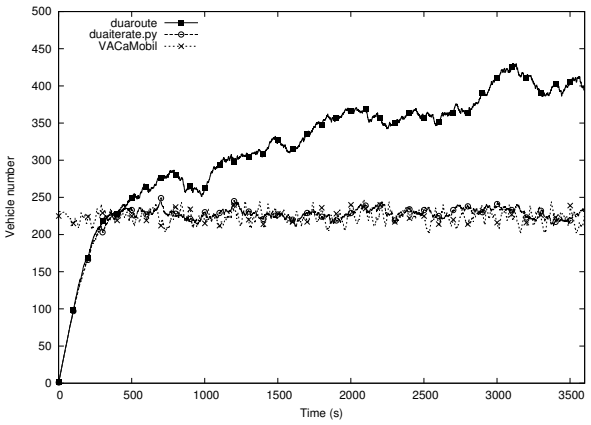


Fig. 3. Vehicle number evolution for the selected scenario.

generation tool. In the scenario, neither duarouter nor duaIterate allow to *a priori* configure the average vehicle value. On the contrary, VACaMobil is not only able to populate the network with the desired number of vehicles, but also allows defining a maximum and a minimum number of vehicles by using the standard deviation feature, which will bound the number of vehicles. In complex maps like the urban scenario, VACaMobil is the only tool able to maintain the standard deviation value within the predefined bounds.

To better understand the aforementioned values, figure 3 show the number of vehicles in the scenario along time for each tool. Since *duarouter* and *duaIterate.py* are only able to add one vehicle per second, the user cannot predict when vehicles will arrive to their destination and disappear from the network. Therefore, the number of vehicles when the simulation reaches the steady-state is not known *a priori*, converting protocol analysis based on the number of vehicles in a mere act of faith. Moreover, in urban maps, where traffic jams are very common, it takes more time for vehicles to reach their destination and leave the network, which leads to a constantly increasing number of vehicles in the network when not using VACaMobil. Comparing the data in table I, we can conclude that both the target number of vehicles and the standard deviation goal are clearly achieved with our VACaMobil approach.

V. CONCLUSIONS AND FUTURE WORK

In this paper we presented VACaMobil[1], a new mobility manager for the OMNeT++ simulator which promotes the full repeatability of VANET simulations. In particular, we added critical features to previously existing tools, such as ensuring a constant number of vehicles during the entire simulation period, disseminating vehicles throughout the whole route-map, and offering the possibility of defining different vehicle types with different probabilities.

Contrarily to other existing tools, which are not able to control the mean number of vehicles nor its standard deviation, VACaMobil is able to maintain

[1]VACaMobil is freely available at *www.grc.upv.es/software*.

Fig. 2. Heat map for the urban scenario when using duarouter, *duaIterate.py*, and VACaMobil (from left to right).

the mean number of vehicles and the standard deviation value within user-defined bounds. To the best of our knowledge, this is currently the only tool that allows studying a vehicular network in a steady situation without losing the realistic vehicle behavior provided by SUMO.

As future work we plan to improve VACaMobil offering downtown definition and automatic placement of Road Side Units (RSU).

Acknowledgements

This work was partially supported by the *Ministerio de Economía y Competitividad*, Spain, under Grants TIN2011-27543-C03-01 and BES-2012-052673, and by the *Ministerio de Educación*, Spain, under the FPU program, AP2010-4397, AP2009-2415.

References

[1] S. Joerer, C. Sommer, and F. Dressler, "Toward Reproducibility and Comparability of IVC Simulation Studies: A Literature Survey," *IEEE Communications Magazine*, vol. 50, no. 10, pp. 82–88, October 2012.

[2] C. Sommer and F. Dressler, "Progressing toward realistic mobility models in vanet simulations," *Communications Magazine, IEEE*, vol. 46, no. 11, pp. 132 –137, november 2008.

[3] M. Fogue, P. Garrido, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni, "An adaptive system based on roadmap profiling to enhance warning message dissemination in vanets," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, p. 1, 2012.

[4] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, Barcelona, Spain, October 2011, pp. 63–68.

[5] J. Yoon, M. Liu, and B. Noble, "Random waypoint considered harmful," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2, march-3 april 2003, pp. 1312 – 1321 vol.2.

[6] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, ser. WDTN '05. New York, NY, USA: ACM, 2005, pp. 252–259.

[7] F. Martinez, J.-C. Cano, C. Calafate, and P. Manzoni, "Citymob: A mobility model pattern generator for vanets," in *Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference on*, may 2008, pp. 370 –374.

[8] X. Li, W. Shu, M. Li, H. Huang, and M.-Y. Wu, "DTN Routing in Vehicular Sensor Networks," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*. IEEE, Nov. 2008, pp. 1–5.

[9] M. Gramaglia, M. Calderon, and C. Bernardos, "Trebol: Tree-based routing and address autoconfiguration for vehicle-to-internet communications," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, may 2011, pp. 1 –5.

[10] L.-J. Chen, Y.-Y. Chen, K. chan Lan, and C.-M. Chou, "Localized data dissemination in vehicular sensing networks," in *Vehicular Networking Conference (VNC), 2009 IEEE*, oct. 2009, pp. 1 –6.

[11] C.-C. Lo, J.-W. Lee, C.-H. Lin, M.-F. Horng, and Y.-H. Kuo, "A cooperative destination discovery scheme to support adaptive routing in vanets," in *Vehicular Networking Conference (VNC), 2010 IEEE*, dec. 2010, pp. 202 –208.

[12] M. Fogue, P. Garrido, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni, "Using roadmap profiling to enhance the warning message dissemination in vehicular environments," in *36th IEEE Conference on Local Computer Networks (LCN 2011)*, Bonn, Germany, October 2011.

[13] C. Sommer, O. Tonguz, and F. Dressler, "Traffic information systems: efficient message dissemination via adaptive beaconing," *Communications Magazine, IEEE*, vol. 49, no. 5, pp. 173 –179, may 2011.

[14] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved ivc analysis," *Mobile Computing, IEEE Transactions on*, vol. 10, no. 1, pp. 3 –15, jan. 2011.

[15] C. Sommer, O. Tonguz, and F. Dressler, "Adaptive beaconing for delay-sensitive and congestion-aware traffic information systems," in *Vehicular Networking Conference (VNC), 2010 IEEE*, dec. 2010, pp. 1 –8.

A Novel 802.11 Contention Window Control Scheme for Vehicular Environments

Ali Balador[1], Carlos T. Calafate[2], Juan-Carlos Cano[3] y Pietro Manzoni[4]

Abstract— **Intelligent Transportation Systems (ITS) have attractive potential in order to decrease the ordinary traffic jams and avoid transportation disasters. Also, they are able to provide various infotainment services like browsing, reading e-mail or using social networks that makes a trip more interesting. In order to make it more efficient in real vehicular environments, achieving a well-designed Medium Access Control (MAC) protocol is a challenging issue due to the dynamic nature of VANETs, scalability issues, and the variety of application requirements. Different standardization organizations have selected IEEE 802.11 as the first choice for VANET environments considering its availability, maturity, and cost. The research results for IEEE 802.11 MAC protocol show the importance of contention window adjustment on the communications performance. The impact of adjusting the contention window has been studied in MANETs, but the vehicular communication community has not yet addressed this issue thoroughly.**

This paper proposes e-HBCWC, a new contention window control scheme for VANET environments based on estimating the network condition. Analysis and simulation results using OMNeT++ in urban scenarios show that e-HBCWC clearly outperforms 802.11 DCF, even in very high network density, by increasing the packet delivery rate while decreasing the number of collisions and the end-to-end delay for unicast applications.

Keywords— **IEEE 802.11 DCF, Contention Window, Density Estimation, Vehicular Ad Hoc Networks**

I. Introduction

ADVANCES in wireless communication technologies, along with the increasing demand for a wide variety of applications, make Vehicular Ad Hoc Networks (VANETs) an attractive research area in both academia and industry. VANETs are formed by vehicles equipped with wireless devices in order to communicate with other vehicles in infrastructureless wireless networks (vehicle-to-vehicle, V2V) or with roadside units (vehicle-to-infrastructure, V2I). Vehicular environments with inter-vehicle communications can be assumed as a special form of Mobile Ad-Hoc Networks (MANETs) in which nodes are road-constrained [1].

The main differences between VANETs and MANETs have to do with rapid topology changes and diversity of network scenarios. For example, in highway scenarios, vehicles have mostly one straightforward direction, but high relative speeds up to 100 km/h that can lead to frequent network disconnections. On the other hand, in urban scenarios, high-density networks appear during rush hours. Also, network disconnection can occur in the late night hours or idle daytime hours. Due to these issues, the design of protocols that can satisfy different types of scenarios become necessary.

Medium Access Control protocols play an important role since critical communications must rely on them. Unfortunately, research results [2] highlight that the topic of MAC support in VANETs has received less attention than other research fields. Also, most of these relatively few research works are dedicated to V2I communications; therefore, MAC support for V2V communication needs more attention. MAC layer design challenges in VANET environments can be summarized as follows [3]: (a) Achieving an effective channel access coordination in the presence of changing vehicle locations and variable channel characteristics; (b) supporting scalability in the presence of various traffic densities; and (c) supporting a diverse set of application requirements.

Applications for VANETs can be categorized into three groups: safety-related, traffic efficiency, and infotainment; each of these applications has its own QoS requirements, which can be in contract with others. For example, on one hand, safety applications require low delays and high reliability, while messages are relatively short. On the other hand, infotainment applications need longer messages, but they are not time-critical. A lot of research has been done by the research community with the idea of supporting broadcast transmissions in mind (e.g,[4], [5], [6], [7]). In contrast to the most common research trend, this paper targets unicast applications including infotainment, P2P or VoIP.

IEEE 802.11 has been selected by a wide range of research works for vehicular environments as the MAC layer standard because of its availability, maturity, and cost. However, it shows poor performance in VANETs compared to MANETs as a result of the differences between these two networks. A well-known problem in IEEE 802.11 is scalability, which becomes more challenging in VANETs in the presence of high and variable network densities. A lot of works have been proposed and carried out for MANET environments to either solve or reduce this problem. Most of these schemes, such as [8], [9], [10], require complex computations that lead to a high overhead in terms of time, power, etc.

Authors in [11] proposed the HBCWC protocol, a new scheme to adapt the contention window (CW) for IEEE 802.11 based on the channel condition history. This algorithm shows a significant improve-

[1]Dpto. de Informática, Univ. Pol. de València, e-mail: alba6@upv.es
[2]Dpto. de Informática, Univ. Pol. de València, e-mail: calafate@disca.upv.es
[3]Dpto. de Informática, Univ. Pol. de València, e-mail: jucano@disca.upv.es
[4]Dpto. de Informática, Univ. Pol. de València, e-mail: pmanzoni@disca.upv.es

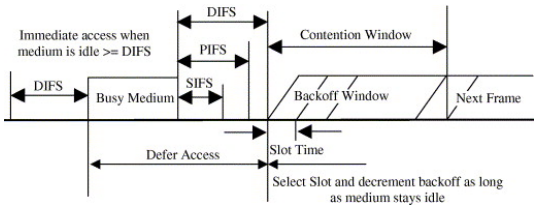


Fig. 1. The IEEE 802.11 DCF mechanism.

ment in the presence of high network densities while keeping the implementation simple and similar to the original IEEE 802.11 MAC. In this paper we propose a new contention window control scheme, called e-HBCWC (enhanced-HBCWC), based on the HBCWC protocol, and evaluate our approach in Vehicle-to-Vehicle communications on a dense urban scenario.

The rest of this paper organized as follows. First, we review the IEEE 802.11 DCF mechanisms. Then, in section III, we describe e-HBCWC, the new proposed contention window control scheme, in detail. Performance evaluation of e-HBCWC, including simulation results in an urban scenario, is presented in section IV. Finally, section V concludes this paper.

II. IEEE 802.11 DCF

Distributed Coordination Function (DCF) [12] is the default MAC scheme of IEEE 802.11 in infrastructure-less environments that uses the carrier sense multiple access with collision avoidance (CSMA/CA) mechanism. IEEE 802.11 DCF uses two main mechanisms in order to avoid common problems in the MAC layer, such as collisions and hidden node problems. First, the Request-to-Send (RTS)/Clear-to-Send (CTS) mechanism is used in order to reserve the medium by sending small packets before transmitting large data packets. Second, each node must wait a random time (called backoff) before sending the packet so as to avoid packet collisions. Also, IEEE 802.11 proposes three different Inter Frame Space (IFS) time intervals in order to provide priority access to the wireless channel that can be seen in Figure 1.

The IEEE 802.11 protocol uses the RTS-CTS-DATA-ACK sequence for data transmission. When a new data packet is provided for transmission, each node must wait for a time equal to DIFS, plus a backoff time before it is allowed to transmit an RTS packet. The node should sense the channel, and if it finds the channel idle for a time interval longer than DIFS, then it needs to select a random backoff time. The Binary Exponential Backoff (BEB) algorithm uniformly selects the backoff time from the interval (0,CW). The IEEE 802.11 DCF initializes the contention window to the predefined value, CW_{min}, and doubles it upon transmission failures up to the predefined value, CW_{max}. The CW is reset to the CW_{min} by a successful transmission. The node senses the channel and, if it finds the channel idle, decreases the backoff timer by one; otherwise it pauses the timer. The backoff timer is resumed when the channel again remains idle for a period longer than DIFS.

The transmission is started by sending an RTS packet when the backoff timer reaches zero. After RTS packet transmission, the node waits in order to receive a CTS packet from the receiver. Receiving a CTS packet shows that the receiver is ready to receive a data packet. Therefore, the transmitter begins to send the data packet after waiting for a SIFS time interval.

Each node has one antenna and so, when it sends data packets, it cannot listen to the channel to ensure that the packet is received correctly. For this reason, 802.11 DCF uses positive acknowledgement (ACK) to inform the transmitter. After receiving a correct data packet, the receiver waits for a SIFS time and sends an ACK to the transmitter. Receiving a correct ACK packet is considered as a successful transmission, while the lack of an ACK reception is considered as a collision; as a consequence, the packet must be retransmitted up to a predefined number of times.

The research results in this field have shown that resetting the CW value to CW_{min} upon a successful transmission causes a high collision rate under high network densities, which leads to a significant reduction of the network performance. Thus, significant efforts have been done in order to propose new strategies for decreasing the CW value.

III. The Proposed Algorithm

As mentioned above, in the IEEE 802.11 DCF, a significant reduction of the network performance, especially in dense traffic networks, stems from an aggressive CW reduction. To tackle these issues, we propose a novel CW control mechanism, called e-HBCWC, in which the history of packet transmission is taken into account for the optimization of the CW size. In e-HBCWC, the channel condition is checked upon each packet reception, and the result is stored into a Channel State (CS) vector. A significant part of this scheme relies on how the channel condition is captured by the CS vector, and how this vector is used to update the CW value in order to improve throughput. These two issues will be further explained in the following sections.

A. The Channel State Vector

In legacy 802.11 DCF, after each packet transmission, each node sets its timer and waits for an acknowledgement (a CTS or ACK packet). In e-HBCWC, if the transmitter receives an error-free packet from the receiver, a value of 1 is inserted into the channel state vector. Otherwise, if a collided/faulty packet is received, or if the transmitter waiting timer expires before receiving the acknowledgement, a value of 0 is inserted into the channel vector.

Upon each channel state change notification, the oldest value in the CS vector is removed and the remaining stored states are shifted to the left. Then,

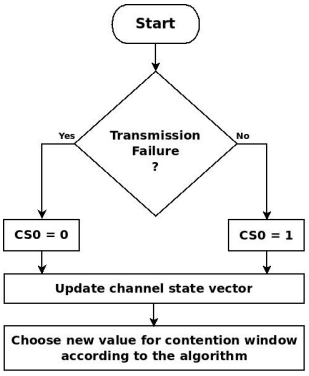


Fig. 2. CS vector updating in e-HBCWC.

the new channel state is stored in the first element of the vector. Based extensive simulation results, we chose a three-element array in e-HBCWC in order to achieve a trade-off between overhead and performance. If we choose a smaller array, it will not be able to assess the real network condition, while larger array values do not lead to a significant performance improvement. Figure 2 shows the basic of e-HBCWC operation.

B. Changing the Contention Window Size

In e-HBCWC, we apply two parameters, A and B, in order to precisely control the CW size based on the channel state vector. Upon each packet loss, timer expiration or collision, the CW size is multiplied by 2, except for the case in which CS array contains two consecutive ones before the new state, in that case the CW is multiplied by A. The CW size is set to the minimum CW, CW_{min}, upon each acknowledgement reception, except for the case in which the CS array contains two consecutive zeros before the new state; in that case CW is multiplied by a parameter that is assumed to be B. Table I details how the CW size is chosen for each CS array state. The CS array is initialized with the value 111. Also, Parameter 'A' can be selected in the range of 1 to 2 while, 'B' can be less than 1. In this paper, these parameters are chosen equal to 1.7 and 0.8 based on the simulation experiments.

Table I
CONTENTION WINDOW SIZE UPDATING

Status	CW range
000	
010	CW_{old} * 2
100	
110	CW_{old} * A
001	CW_{old} * B
011	
101	CW_{min}
111	

IV. SIMULATION

In this section we study the performance of e-HBCWC in comparison with IEEE 802.11 DCF in vehicular environments by using the OMNeT++ (version 4.2.2) network simulator [13]. OMNeT++ provides a powerful basis for network simulation but it has some shortages in the modelling of wireless networks. Therefore, we chose this simulator coupled with INET framework [14] and SUMO [15] in order to provide a realistic vehicular scenario. The INET framework provides detailed models for simulating wireless networks in OMNeT++ such as wireless channels, connectivity, mobility, and MAC layer protocols. Also, SUMO is used to generate real vehicular traffic in road networks.

In order to study the efficiency of e-HBCWC, we evaluate its performance in an urban scenario. The general simulation parameters are as follows: each vehicle a generates constant bit rate traffic. The size of the data payload is 512 bytes, and each node generates data packets at the rate of 4 packets per second. Each vehicle starts a new connection after joining the network and sends its packets to the randomly selected destination among the current vehicles in the network. Also, each vehicle immediately changes its destination when the destination leaves the network. The chosen routing protocol is AODV, and the radio range for each node is 250 m. We used the Nakagami radio propagation model that is commonly used by VANET community. Moreover, each point in the figures that follow represents average of 10 independent simulation experiments. Table II summarizes the simulation parameters.

The metrics used to evaluate the performance of the proposed scheme are the following: (a) Packet Delivery Ratio (PDR), which represents the ratio of the total number of packets received by the final destination and the packets originated by the source; (b) average end-to-end delay, which represents the average time required for a packet to travel from source to destination; and (c) average MAC collisions, which shows the average number of collisions experienced per source.

Table II
THE SIMULATION PARAMETERS

Simulation Parameter	Value
Traffic type	CBR
CBR packet size	512 byte
CBR data rate	4 packet/s
Transport protocol	UDP
Routing protocol	AODV
Max. and Min. of CW	7, 1023
Max. number of retransmissions	7
Max. queue size	14
RTS/CTS threshold	2346 byte
slot time	13us
Max. transmission range	250 m
Propagation model	Nakagami
Nakagami-m	0.7

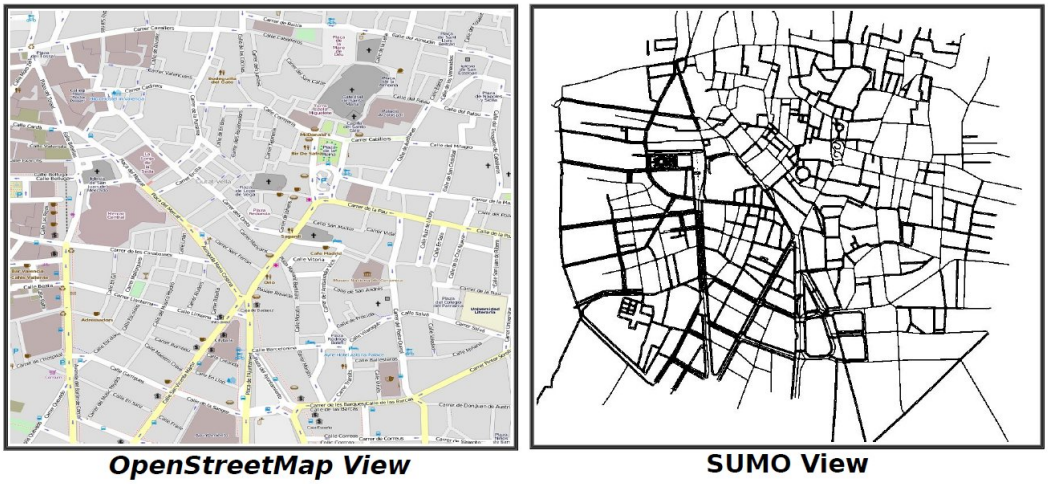


Fig. 3. Valencia real urban scenario.

A. Scenario Description

We used SUMO and connected it to OMNeT in order to generate realistic urban mobility traces. Our scenario represents an area of $1,500 \times 1,500\ m^2$ that is selected from the downtown area of Valencia (Spain) with real obstacles by using digital maps freely available in OpenStreetMap [16]. Figure 3 shows two map views: the OpenStreetMap view and the SUMO view. The vehicle generation and mobility is handled by the VACaMobil [17] tool. In this work, we vary the number of vehicles from 50 to 200 throughout the simulation time (300 seconds).

B. Result and Analysis

Figures 4-6 represent the PDR, the number of collisions and the average end-to-end delay, respectively, for the legacy 802.11 and e-HBCWC with different numbers of nodes. Figure 4 shows a clear improvement in packet delivery ratio for different number of nodes. In low density networks (less than 75 nodes), the improvement ratio is not as high as in high density networks because 802.11 DCF rapidly decreases the CW value, while it takes longer for e-HBCWC to do that, thereby decreasing the number of dropped packets and causing PDR to increase. Collisions occur due to network partitioning or reaching the retransmission limit. Concerning the average number of collisions, there are clear differences between 802.11 DCF and e-HBCWC, as shown in Figure 5.

Even though e-HBCWC does not reset the CW to CW_{min}, gradually decreasing the CW instead, simulation results show improvements in terms of end-to-end delay for e-HBCWC as a result of decreasing the number of collisions. As Figure 6 shows, e-HBCWC has the lowest improvement in comparison to other areas in low density networks because its delay causes more packets to be dropped, but it still works slightly better than DCF. As we expected, in high density situations, e-HBCWC shows a high improvement ratio by avoiding to reset the CW to the minimum value upon a successful transmission. Overall, we obtain 40% improvement in PDR, 37% improvement in MAC collisions, and 26% improvement in end-to-end delay compared with IEEE 802.11 DCF.

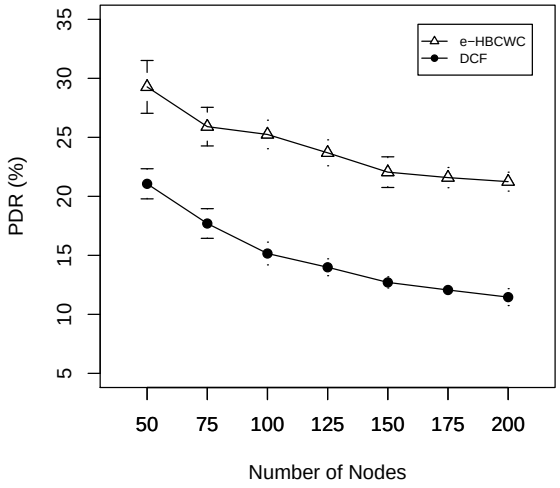


Fig. 4. PDR for the urban scenario.

V. Conclusion

The impact of controlling the contention window on the performance of the IEEE 802.11 MAC is an issue that has been studied by the MANET research community, remaining mostly untackled in VANET environments. In this paper we propose a new contention window scheme based on HBCWC that was proposed for IEEE 802.11 based MANETs. Each node in e-HBCWC stores its transmission trials history in the network as an array which is used in order to determine the optimal contention window value.

In this paper we focused on urban environments to demonstrate the performance benefits of e-HBCWC. Simulation results prove that our scheme outperforms 802.11 DCF in terms of PDR, end-to-end delay and collisions in these scenarios.

As a future work, we will study the effect of the

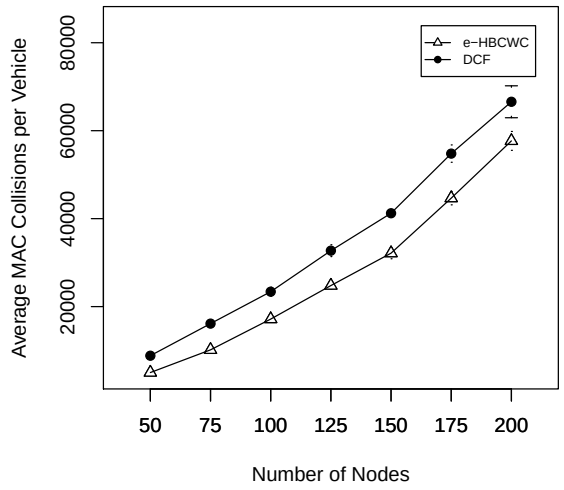


Fig. 5. Average number of collisions for the urban scenario.

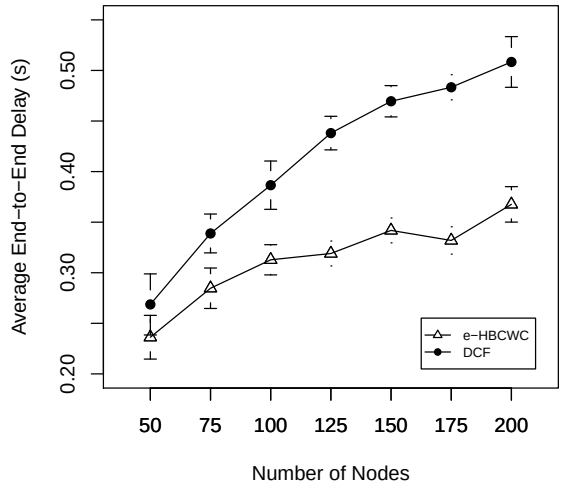


Fig. 6. Average end-to-end delay for the urban scenario.

existence of RSUs on the performance of our scheme. Also, the performance of e-HBCWC in highway scenarios will be evaluated. Moreover, we will dynamically adapt the algorithm's parameters based on each specific network scenario.

Acknowledgment

This work was partially supported by the *Ministerio de Ciencia e Innovación*, Spain, under Grant TIN2011-27543-C03-01.

References

[1] Hartenstein, H.; Laberteaux, K.P., "A tutorial survey on vehicular ad hoc networks," Communications Magazine, IEEE , vol.46, no.6, pp.164,171, June 2008.

[2] Booysen, M.J.; Zeadally, S.; van Rooyen, G.-J., "Survey of media access control protocols for vehicular Ad hoc networks," Communications, IET , vol.5, no.11, pp.1619,1631, July 22 2011.

[3] Kenney, J.; "Standards and regulations," in Hartenstein, H., Laberteaux, K.P. (Eds.): "VANET: vehicular applications and inter-networking technologies" (Wiley, 2010), Ch. 10, pp. 365? 428.

[4] Asadallahi, S.; Refai, H.H., "Modified R-ALOHA: Broadcast MAC protocol for Vehicular Ad hoc Networks," Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International , vol., no., pp.734,738, 27-31 Aug. 2012.

[5] Omar, H.; Zhuang, W.; Li, L., "VeMAC: A TDMA-based MAC Protocol for Reliable Broadcast in VANETs," Mobile Computing, IEEE Transactions on , vol.PP, no.99, pp.1,1, 0 doi: 10.1109/TMC.2012.142.

[6] Sheng-Shih Wang; Hung-Chang Chen; Jia-Kang Chang, "A distributed adaptive MAC protocol for efficient broadcasting in vehicular ad hoc networks," Wireless Communications and Networking Conference (WCNC), 2012 IEEE , vol., no., pp.1555,1560, 1-4 April 2012.

[7] Stanica, R.; Chaput, E.; Beylot, A.-L., "Enhancements of IEEE 802.11p Protocol for Access Control on a VANET Control Channel," Communications (ICC), 2011 IEEE International Conference on , vol., no., pp.1,5, 5-9 June 2011.

[8] Bononi, L.; Conti, M.; Gregori, Enrico, "Runtime optimization of IEEE 802.11 wireless LANs performance," Parallel and Distributed Systems, IEEE Transactions on , vol.15, no.1, pp.66,80, Jan. 2004.

[9] Bianchi, G.; Tinnirello, I., "Kalman filter estimation of the number of competing terminals in an IEEE 802.11 network," INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies , vol.2, no., pp.844,852 vol.2, March 30 2003-April 3 2003.

[10] Cali, F.; Conti, M.; Gregori, Enrico, "Dynamic tuning of the IEEE 802.11 protocol to achieve a theoretical throughput limit," Networking, IEEE/ACM Transactions on , vol.8, no.6, pp.785,799, Dec 2000.

[11] Balador, A.; Movaghar, A.; Jabbehdari, S., "History based contention window control in ieee 802.11 mac protocol in error prone channel," Journal of Computer Science, vol.6, no.2, pp.205,209, 2010.

[12] "Supplement to ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications: High-speed physical layer in the 5 ghz band," IEEE Std 802.11a-1999, p. i, 1999.

[13] Website: http://www.omnetpp.org/

[14] Website: http://inet.omnetpp.org/

[15] Behrisch, M.; Bieker, L.; Erdmann, J.; Krajzewicz, D., "SUMO - Simulation of Urban MObility: An Overview In: SIMUL 2011," The Third International Conference on Advances in System Simulation, 2011.

[16] Website: http://www.openstreetmap.org/

[17] Website: http://www.grc.upv.es/software/vacamobil.html

Un Sencillo Esquema de Colas para Reducir el *HoL-Blocking* en Redes Híbridas de Altas Prestaciones

Pedro Yébenes, Jesús Escudero-Sahuquillo, Crispín Gómez, Pedro J. García, Francisco J. Quiles [1]

Resumen— **El *Head-of-Line (HoL) blocking* es un fenómeno que degrada terriblemente las prestaciones de las redes de interconexión de alto rendimiento. Para solucionar este problema, se han propuesto numerosas técnicas, la mayoría de ellas basadas en separar los flujos de tráfico en diferentes colas en los puertos de los conmutadores. Sin embargo, la eficiencia de estas propuestas puede variar dependiendo de la topología de red o del algoritmo de encaminamiento, ya que muchas de ellas no tienen en cuenta ninguna configuración de red específica. Por el contrario, otros esquemas se ajustan a topologías específicas como los *fat-trees*, logrando una mejor eficiencia que los esquemas independientes de la topología. En este artículo proponemos un esquema de colas simple para una topología híbrida muy eficiente propuesta recientemente: KNS. Nuestro planteamiento mejora significativamente el rendimiento de la red con respecto a otros esquemas de colas utilizando los mismos o menos recursos.**

Palabras clave— **Redes de Interconexión, Topologías Híbridas, Head-of-Line Blocking, InfiniBand.**

I. Motivación

LAS redes de interconexión de altas prestaciones son elementos esenciales en los sistemas utilizados actualmente para la computación paralela: procesadores masivamente paralelos, *clusters* de PCs y *workstations* o redes en chip. En estos entornos, el rendimiento que alcanza el sistema en su conjunto depende de la red de interconexión, especialmente si el número de nodos finales es elevado, pudiendo ser el cuello de botella del sistema si no es capaz de cumplir con las necesidades de comunicación de las aplicaciones. Por tanto, alcanzar un buen rendimiento en la red es fundamental.

El efecto *HoL-blocking* puede reducir drásticamente el rendimiento de las redes de interconexión de altas prestaciones actuales. Este puede limitar la productividad de los conmutadores basados en colas en los puertos *buffers* de los puertos de entrada/salida, al 58 % de su rendimiento máximo [1]. Cabe destacar que los conmutadores comerciales de las actuales tecnologías de redes de alta velocidad (InfiniBand [2], Myrinet [3], etc.) soportan algunos tipos de esquemas de colas basados en *buffer*, por lo que pueden sufrir *HoL-blocking*. Básicamente, este efecto ocurre cuando un paquete situado en la cabeza de una cola se bloquea, por no poder avanzar a su destino, impidiendo la transmisión del resto de los paquetes de esta cola, incluso si estos están solicitando puertos disponibles. Como consecuencia, la latencia media de paquete se incrementa y la productividad de la red, decae. Este efecto está estrechamente relacionado con cargas de tráfico altas y situaciones de congestión [4], aunque puede ocurrir en otros escenarios de tráfico. Por otro lado, especialmente en situaciones de congestión, el *HoL-blocking* tiende a propagarse, ya que las colas bloqueadas se llenan y como consecuencia, bloquean otras colas en otros conmutadores debido al control de flujo. Este último fenómeno es conocido como *HoL-blocking* de alto nivel [5], en contraste con el *HoL-blocking* «original», también conocido *HoL-blocking* de bajo nivel.

Para disminuir la probabilidad de *HoL-blocking* se han propuesto muchas técnicas que limitan la carga de tráfico de la red o evitan o eliminan situaciones de congestión (véase la sección II-B). La mayoría de ellas basadas en dividir el espacio del *buffer* de los puertos del conmutador en diferentes colas y asignar los paquetes a distintas colas de modo que, en la medida de los posible, distintos flujos de paquetes no compartan colas. Aunque muchos de estos esquemas de colas previenen el *HoL-blocking* completamente [6], [7], [8], requieren una gran cantidad de recursos adicionales que no ofrecen los actuales conmutadores comerciales. Por este motivo, otros esquemas son mucho más populares, por ser factibles aunque sólo reducen el *HoL-blocking* parcialmente [9], [10], [11]. Sin embargo, hemos descubierto que los esquemas de colas pueden reducir más eficientemente el *HoL-blocking* si son diseñados para topologías y algoritmos de encaminamiento específicos, ya que se optimiza el uso de las colas. En este sentido, en [12] y [13] se propusieron esquemas de colas que aprovechan las propiedades de los *fat-trees* y de las del algoritmo de encaminamiento DESTRO [14] para reducir el *HoL-blocking* más eficientemente que los esquemas genéricos que requieren un número de colas por puerto mayor (o similar).

Siguiendo esta filosofía, en este artículo presentamos un esquema de colas que aprovecha las propiedades de una topología híbrida recientemente propuesta: KNS [15] para reducir el *HoL-blocking* de una manera sencilla. Aunque esta topología híbrida ha sido presentada como más eficiente que otras (tanto topologías directas como indirectas, véase la sección II-A), su rendimiento también decae cuando en la red hay cargas de tráfico altas o en situaciones de congestión en las que aparece *HoL-blocking* masivamente. Por el contrario, si se utiliza el esquema de

[1]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha. E-mail:{pedro.yebenes, jesus.escudero, crispin.gomez, pedrojavier.garcia, francisco.quiles}@uclm.es

colas que proponemos, el *HoL-blocking* se reduce de manera significativa, como mostramos en la sección IV. Específicamente, nuestra propuesta se basa en una asignación simple pero inteligente del destino de los paquetes a las colas, de modo que todas las colas en cada puerto de un conmutador en la red, se utilizan para separa los flujos de tráfico que pueden cruzar ese puerto. Hemos llamado a este esquema *Band-Based Queuing* (BBQ).

El resto del artículo se organiza así: la sección II ofrece una visión general de la topología híbrida propuesta y de las técnicas existentes que previenen o reducen el *HoL-blocking*. El esquema de colas propuesto se describe en la sección III. Además evaluamos nuestra propuesta en la sección IV, donde se muestran resultados de simulación comparando su eficacia con otros esquemas. Para finalizar, en la sección V se indican algunas conclusiones.

II. Trabajo Relacionado

A. Redes Híbridas

Tradicionalmente, las topologías de red se clasifican en directas o indirectas. Las topologías indirectas suelen ofrecer un mejor rendimiento para un mismo número de nodos que las topologías directas, a costa de utilizar más conmutadores y enlaces (es decir, mayor coste), y de incrementar la complejidad del cableado, que crece con el tamaño de la red.

Recientemente, se han propuesto topologías híbridas y jerárquicas para obtener los beneficios tanto de las topologías directas como de las indirectas, para ofrecer un alto rendimiento como las indirectas, pero con un coste en componentes similar al de las directas. Con este objetivo en mente, los autores de [16] propusieron la *Flattened-Butterfly*, una topología derivada del *fat-tree* donde todos los conmutadores de una columna del *fat-tree* están concentrados en un único conmutador, convirtiéndose en un hipercubo generalizado. Extendiendo esta topología para aprovechar conmutadores de alto grado, en [17] se propone la topología *Dragonfly*. En esta topología, los conmutadores de la misma fila del cubo se conectan totalmente para formar un grupo. Los grupos se conectan entre ellos mediante un número de enlaces parametrizable. El problema de las topologías jerárquicas es que requieren el mismo número de conmutadores que una red directa para el mismo número de nodos, pero los conmutadores son considerablemente más complejos. Por tanto, la complejidad global y el coste de la red es bastante más alto que en las redes directas, a pesar de ser menor que en las indirectas.

Con el fin de superar este inconveniente, se ha propuesto la topología KNS: k-ary n-direct s-indirect [15]. En esta topología, los nodos se organizan en n dimensiones, como en una topología directa, pero los nodos de una dimensión dada están conectados por medio de una red indirecta. Esta red indirecta puede ser un simple conmutador, como en la figura 1. Si el número de nodos por dimensión supera el número de puertos del conmutador se utiliza una MIN. Por lo tanto, esta familia de topologías propuesta se define

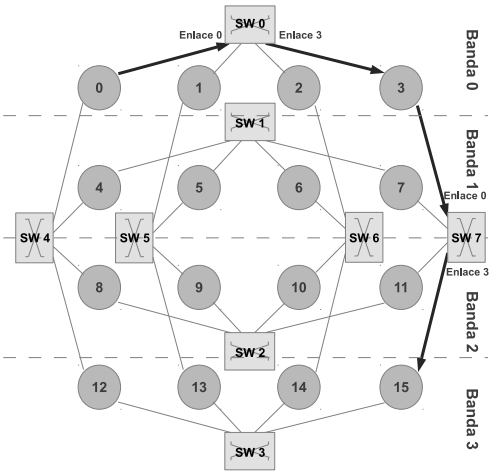


Fig. 1. Topología 4-ary 2-direct 1-indirect. Se resalta la ruta desde el nodo 0 al 15 utilizando *Hybrid-DOR*.

por tres parámetros: el número de dimensiones n, el número de nodos por dimensión k yel número de etapas de cada subred indirecta s. El número de nodos de procesamiento que puede conectar es $N = k^n$. En este artículo, nos vamos a centrar en las topologías k-ary n-direct 1-indirect ya que al utilizar una MIN para conectar los nodos de una dimensión dispara el número de conmutadores de la red.

En [15], los autores proponen un algoritmo de encaminamiento determinista para estas topologías llamado *Hybrid-DOR*, en el cual las dimensiones de la red se cruzan en un orden establecido para evitar los interbloqueos (primero la X y luego la Y). Sin embargo, los paquetes no realizan varios saltos en cada dimensión, ya que los nodos envían los paquetes por el único enlace por el que se conectan a cada dimensión, y luego el paquete se recibe en un conmutador donde se envía a través del enlace indicado por la componente del destino correspondiente a la siguiente dimensión. Es decir, el paquete requiere únicamente cruzar tantos conmutadores como dimensiones existen, para alcanzar el nodo destino. Por ejemplo, en la figura 1 se resalta la ruta del nodo 0 (0,0) al nodo 15 (3,3). Como se puede observar, el paquete abandona el nodo 0 a través del único enlace de la dimensión X, alcanzando un conmutador (sw0) donde se envía a través del enlace 3, para alcanzar el nodo 3 (0,3). Después se repite el proceso en la siguiente dimensión. Obsérvese que no importa la distancia que el paquete tiene que viajar en una dimensión, esta puede ser atravesada únicamente en 2 saltos. En [15] se muestra que la topología híbrida KNS con *Hybrid-DOR* obtiene un mejor rendimiento que otras configuraciones de red, como toros, *fat-trees* o *flattened butterflies* (con tamaños similares).

B. Soluciones al problema del HoL-blocking

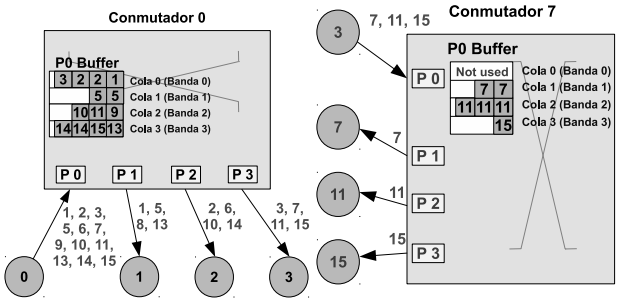
Como se ha mencionado anteriormente, existen varios tipos de técnicas para solucionar el problema del *HoL-blocking* en redes de interconexión de altas prestaciones, pero en esta sección nos centramos únicamente en aquellas diseñadas ex profeso para ello.

Entre estas soluciones, probablemente las más eficientes se basan en la asignación dinámica de colas (o canales virtuales [10]) para aislar los flujos de paquetes que contribuyen a la congestión (referidos como «flujos calientes»), ya que se previene el *HoL-blocking* que estos flujos puedan provocar en otros («flujos fríos»). Estas técnicas requieren mecanismos para detectar la congestión e identificar (y luego separar) los flujos calientes. Soluciones como [18] y [19] identifican los flujos calientes basándose en su destino final, mientras que otras soluciones como [7] y [8], los detectan mediante el camino hacia su destino final. Aunque, estas técnicas son bastante efectivas, pueden tener problemas cuando el número de *hot-spots* en la red es mayor que el número de colas (o VLs) disponibles para aislar flujos calientes. Y más importante aún es que requieren memorias de control en los puertos de los conmutadores para llevar la cuenta de los *hot-spots*, y pueden requerir mensajes de control, que en general, no están soportados por los componentes de red comerciales actuales.

Otras soluciones para tratar con el *HoL-blocking* asignan flujos de paquetes a diferentes colas independientemente de las condiciones de tráfico, separando «estáticamente» los flujos de paquetes entre las colas disponibles. Este enfoque da como resultado distintos esquemas de colas «estáticos» basados en un criterio de asignación simple, que en general no requiere de recursos adicionales, aparte del conjunto de colas de cada puerto. Sin embargo, esto no son siempre asequibles como *Virtual Output Queues* a nivel de red (VOQnet) [6] que utiliza en cada puerto del conmutador tantas colas como destinos en la red, de modo que cualquier paquete es asignado a la cola correspondiente a su destino, compartiendo la cola únicamente con otros paquetes dirigidos al mismo destino. Este esquema previene totalmente el *HoL-blocking* de alto y bajo nivel, pero es inviable porque necesita muchas colas por puerto, requiriendo cada una, un mínimo espacio de memoria.

Otros esquemas de colas estáticos son más viables porque requieren un número reducido de colas por puerto. Entre ellos, *Virtual Output Queues* a nivel de conmutador (VOQsw) [11] utiliza en cada puerto tantas colas como puertos de salida existen en el conmutador, por lo que a cada paquete entrante se le asigna la cola correspondiente a su siguiente puerto de salida. De este modo, VOQsw evita totalmente el *HoL-blocking* de bajo nivel, pero no el de alto. Otro esquema de colas «estático» y factible es DBBM [20], que selecciona la cola en la que se va a almacenar un paquete de acuerdo con la fórmula $Destino\ MOD\ \#Colas$, donde $\#Colas$ es el número de colas por puerto (es decir, DBBM asigna paquetes a colas consultando los $log_2(\#Colas)$ bits menos significativos del identificador de destino de cada paquete). También podemos encontrar esquemas con la misma filosofía que reducen el *HoL-blocking* parcialmente como *Dynamically Allocated Multi-Queues* (DAMQs)[9] y *Dynamic Switch Buffer Management* (DSBM) [21].

En general, estos esquemas estáticos no tienen en cuenta ni el algoritmo de encaminamiento, ni la topología de red puedo ser ineficientes en determinadas topologías cuando aparece el *HoL-blocking*. En contraste, otros esquemas de colas como *Output-Based Queue Assignment* (OBQA) [12], [13] y *vFTree* [22] tienen en cuenta la topología de red (específicamente, están diseñadas para *fat-trees*) y el algoritmo de encaminamiento (respectivamente, los propuestos en [14] y [23]) y aprovechan sus características para reducir el *HoL-blocking* de una manera tan efectiva (o mejor) que esquemas como VOQsw, ya que requieren la mitad, o incluso una cuarta parte, del número de colas por puerto que requiere este último esquema. Esta idea básica de un esquema de colas que se ajuste a una topología específica para reducir el *HoL-blocking* nos lleva a proponer BBQ, como se describe en la siguiente sección.



(a) Eje X en el conmutador #0.(b) Eje Y en el conmutador #7.

Fig. 2. Distribución de los destinos de los paquetes para BBQ en una red *2*-ary *4*-direct *1*-indirect.

III. Descripción de BBQ

En esta sección describimos nuestra sencilla propuesta para reducir el *HoL-blocking* en redes KNS que utilizan el algoritmo de encaminamiento *Hybrid-DOR*. Básicamente nuestra propuesta consiste en un sencillo esquema de cola «estático», que reduce significativamente el *HoL-blocking*. Concretamente, la política de asignación propuesta selecciona la cola para cada paquete de acuerdo la siguiente fórmula:

$$ColaSeleccionada = \frac{Destino \times \#Colas}{\#NodosFinales} \quad (1)$$

Donde *Destino* es el destino del paquete, $\#Colas$ es el número de colas que configura BBQ en cada puerto del conmutador (es decir, el número de colas en el que se divide cada *buffer* de cada puerto), y $\#NodosFinales$ es el número de nodos finales en la red. Esta política de selección de colas divide «virtualmente» la red en un número horizontal de zonas o bandas, y cada una de ellas consiste en una o más filas consecutivas de destinos (nodos finales) que obtienen la misma «ColaSeleccionada» cuando se introducen como «Destino» en la fórmula. Esta es la razón por la que llamamos a esta propuesta *Band-Based Queuing* (BBQ). BBQ divide la red en un número de bandas igual al número de colas configuradas en cada puerto. Por ejemplo, en la figura 1, se asumen cuatro colas por puerto, así que la red

se divide en cuatro bandas, cada una consiste en una fila de nodos finales, ya que hay cuatro filas en la red.

Como consecuencia de esta política de asignación, a un paquete se le asigna, en cualquier puerto de la red, la misma cola que a cualquier otro paquete cuyo destino este en la misma banda que el destino del primero. Es decir, los paquetes dirigidos a la misma banda se almacenan en la misma cola, y esto ocurre tanto en los puertos de los conmutadores de la dimensión X ó Y. Por otro lado, si se utiliza *Hybrid-DOR*, los paquetes que se dirigen a cualquier banda pueden encontrase en cualquier puerto de los conmutadores tanto de la dimensión X como de la Y, ya que todo conmutador en la dimensión Y, está conectado directamente con los nodos finales que pertenecen a cualquier banda de la red. Así pues, todas las colas definidas en cualquier puerto de los conmutadores pueden recibir paquetes, de modo que se utilizan de manera eficiente para separar distintos flujos de paquetes, es decir, para reducir las probabilidades de *HoL-blocking*. Nótese que los paquetes dirigidos a una banda concreta nunca comparten colas con otros paquetes dirigidos a otra banda, por lo que si aparece un *hot-spot* en una de las bandas, solo los paquetes dirigidos a esa banda pueden sufrir *HoL-blocking* causado por los flujos calientes que contribuyen a ese *hot-spot*.

La figura 2 muestra un ejemplo del comportamiento interno de BBQ tanto en el conmutador #0 de la figura 1 (el conmutador está en la dimensión X, véase la figura 2a) como en el conmutador #7 de la figura 1 (el conmutador está en la dimensión Y, véase la figura 2b). Cada enlace se representa con una flecha etiquetada con los posibles destinos de paquetes que pueden cruzar a través de él en el sentido indicado por la flecha. Los paquetes se representan como cuadrados dentro de sus colas correspondientes, con su destino en su interior. Por ejemplo, paquetes con destino 1,2 y 3 (todos en la Banda 0) han sido almacenados en la cola 0 en el puerto 0 del conmutador 0. Nótese que, la única cola que no ha sido utilizada en el conmutador #7 es la cola 0, pero esto es debido a que solo se han recibido tres destinos en el puerto P0. Sin embargo, la cola 0 podría ser utilizada si la banda 0 consistiese en más de una columna, para almacenar los paquetes que vayan a cambiar de fila dentro de la misma banda.

A diferencia de BBQ, otro esquema de colas «estático» y factible como DBBM [20] podría no aprovechar todas las colas disponibles en ambas dimensiones, si se utiliza en redes KNS. Su fórmula, descrita en la sección II-B divide la red en zonas verticales compuestas por columnas no consecutivas de nodos finales, de modo que los paquetes dirigidos a la misma zona vertical son asignados a la misma cola. Como consecuencia, si DBBM se aplica en estas redes con *Hybrid-DOR*, los conmutadores en la dimensión Y podrían no utilizar todas las colas de sus puertos para reducir el *HoL-blocking*, produciendo una disminución del rendimiento de la red (como puede comprobarse en los resultados de la sección IV). Por ejemplo, si DBBM se utiliza (configurado con cuatro colas por puerto) en el mismo escenario que BBQ en la figura 2, en el puerto P0 del conmutador #7 todas las colas excepto una estarían desperdiciadas, ya que todas los paquetes que se recibirán en este puerto siempre se le asignarán a la misma cola.

Por otro lado, si se utiliza VOQsw [11] o OBQA [12], los paquetes se asignarán a colas dependiendo del puerto solicitado por el paquete en cada conmutador. Esto reduce el impacto del *HoL-blocking* de bajo nivel, pero los flujos de paquetes comparten colas con diferentes flujos dependiendo del conmutador en el que se encuentre. Como consecuencia, los flujos calientes contribuyentes a un *hot-spot* pueden causar *HoL-blocking* de alto nivel a varios flujos, distintos en cada conmutador, reduciendo el rendimiento de estos esquemas en los escenarios *hot-spot*. Por el contrario, como se ha explicado antes, cuando se utiliza BBQ, sólo se ven afectados los paquetes dirigidos a la banda donde aparece el *hot-spot*.

En conclusión, el esquema de colas BBQ aprovecha las características de la topología KNS y de *Hybrid-DOR* para asegurarse de que todas las colas se usan para reducir el *HoL-blocking*. Cabe señalar que este esquema se basa en una simple operación (asignación del paquete a una banda), que puede ser realizada incluso antes de la inyección del paquete a la red.

IV. Evaluación

En esta sección, evaluamos la técnica BBQ mediante experimentos de simulación, en comparación con otros esquemas de colas. A continuación, describimos el modelo de simulación que hemos utilizado en nuestros experimentos y discutimos los resultados.

A. El Modelo de Simulación

El modelo de simulación que hemos utilizado en los experimentos [24] ha sido construido utilizando la plataforma OMNeT++ [25]. Básicamente, nuestro simulador es capaz de modelar varias configuraciones de redes de interconexión, mediante módulos simples y compuestos. Estos módulos de OMNeT++ definen la estructura y el comportamiento de la red, soportando varios algoritmos de encaminamiento, esquemas de colas, políticas de arbitraje y control de flujo. Esta herramienta de simulación ha sido validada utilizando redes reales.

Hemos modelado tres configuraciones de la topología KNS con diferentes tamaños para probar la escalabilidad de BBQ: #1, 8-ary 2-direct 1-indirect; #2, 16-ary 2-direct 1-indirect; y #3, 32-ary 2-direct 1-indirect. En todas las configuraciones de red, asumimos enlaces series segmentados *full-duplex* con un ancho de banda de 2.5 GB/s (20Gbps), un retardo de la propagación en el enlace de 6 nanosegundos (es decir, una longitud de 1.2 metros y un retardo de 5 ns/m, estos valores se basan en la especificación de InfiniBand [2]), tanto como para los enlaces entre conmutadores como entre nodos y conmutadores. En todos los casos la técnica de conmutación es *Virtual Cut-Through*, la política de control de flujo se basa

en créditos y el tamaño de los paquetes es de 2048 bytes. Además, el algoritmo de encaminamiento utilizado es *Hybrid-DOR*, descrito en la sección II.

Con respecto al modelo del conmutador, hemos utilizado conmutadores con 8 puertos para la configuración de red #1, 16 puertos para la configuración #2, y 32 puertos, para la #3[1]. Además, la arquitectura del modelo del conmutador tiene colas solo en los puerto de la entrada (*Input-Queued*), es decir, las memorias RAM sólo se encuentran en los puertos de entrada. La organización de estas memorias depende del esquema de colas. Hemos modelado los siguientes esquemas de colas (descritos en las secciones II-B y III) centrados en la prevención del *HoL-blocking*:

- **Cola única** (1Q). Se configura una única cola en cada RAM de los puertos de entrada con 128 KB. No hay política de reducción del *HoL-blocking*, por lo que evaluamos el rendimiento del algoritmo *Hybrid-DOR* sin mitigarlo.
- **Virtual Output Queues a nivel de conmutador** (VOQsw). En cada puerto de entrada hay *buffers* de 128 KB, divididos estáticamente en un número de colas igual al grado del conmutador, en la configuración #1 se utilizan 8 colas; en la #2, 16; y en la #3, 32.
- **Virtual Output Queues a nivel de red** (VOQnet). Esta técnica requiere memorias mayores porque cada cola necesita un tamaño mínimo[2]. Los tamaños de las RAM para las configuraciones #1, #2 y #3 son 256 KB, 1024 KB y 4096 KB, respectivamente. Aunque este esquema es inviable para redes de tamaño medio o grande, lo incluimos para mostrar el máximo teórico de prevención del *HoL-blocking*.
- **Destination-Based Buffer-Management** (DBBM). Los *buffers* por puerto de entrada son de 128 KB, dividido en partes iguales y estáticas entre 2 ó 4 colas.
- **Band-Based Queuing** (BBQ). Los *buffers* de los puertos de entrada también tienen 128 KB, divididos de manera estática y equitativa entre 2 ó 4 colas.

Los nodos finales se conectan a los conmutadores mediante los *Host Channel Adapters* (HCAs), modelados con tantas colas de admisión como nodos finales hay en la red. Cada paquete generado se almacena en la cola de admisión asignada a su destino, por lo que se elimina el *HoL-blocking* previo a la inyección de tráfico. Los paquetes se transfieren desde las colas de admisión a las de inyección, que se organizan de acuerdo al esquema de colas establecido en los puertos de entrada de los conmutadores (por ejemplo, en BBQ, cada HCA tiene 2 ó 4 colas de inyección). Las colas de inyección se controlan a nivel de flujo desde los puertos de entrada de los conmutadores.

En lo que se refiere al modelado del tráfico, hemos utilizado un patrón *hot-spot*: un 25 % de nodos generan tráfico dirigido a un único destino (*hot-spot*), mientras que el resto de los nodos generan tráfico uniforme. La tasa de generación de tráfico se incrementa en simulaciones sucesivas desde el 0 % hasta el 100 % del ancho de banda del enlace, evaluando en la eficiencia normalizada (cantidad de tráfico entregado por unidad de tiempo, normalizado respecto a la cantidad máxima de tráfico que pueden generar los nodos).

B. Resultados de tráfico hot-spot

La figura 3 muestra la eficiencia de la red, para las tres configuraciones de red, con el patrón de tráfico *hot-spot*. Téngase en cuenta que como el 25 % de los nodos envía a un único destino, sólo se puede lograr como máximo un 75 % de eficiencia la red.

En la figura 3a (64 nodos), 1Q apenas consigue un 30 % de eficiencia para una tasa de generación del 100 %, mientras que la eficiencia ideal está sobre el 75 %, alcanzada por VOQnet. VOQsw alcanza una eficiencia máxima del 45 %, debido al *HoL-blocking* de alto nivel que aparece en los escenarios *hot-spot*. Por contra, BBQ y DBBM con sólo 2 colas alcanzan un mejor rendimiento que VOQsw (que requiere 8), mientras que si utilizan 4, la eficiencia de red mejora aún más. Además, BBQ sobrepasa a DBBM en un 10 % cuando se inyectan en la red altas cargas de tráfico. Esto es debido a que BBQ separa de una manera eficaz los flujos calientes, utilizando todas las colas disponibles, mientras que DBBM las desaprovecha en los conmutadores de la dimensión Y. Es más, BBQ con 2 colas obtiene resultados similares a DBBM con 4, cuando se inyecta tráfico al 100 %.

Se llegan a conclusiones similares a partir de las figuras 3b y 3c; aunque VOQnet, utilizando 256 ó 1024 colas, mejora un 15 % a BBQ, con sólo 4. Además, BBQ supera en un 15 % a DBBM y en un 50 % a VOQsw, es decir, BBQ alcanza con diferencia el mejor resultado de los esquemas factibles.

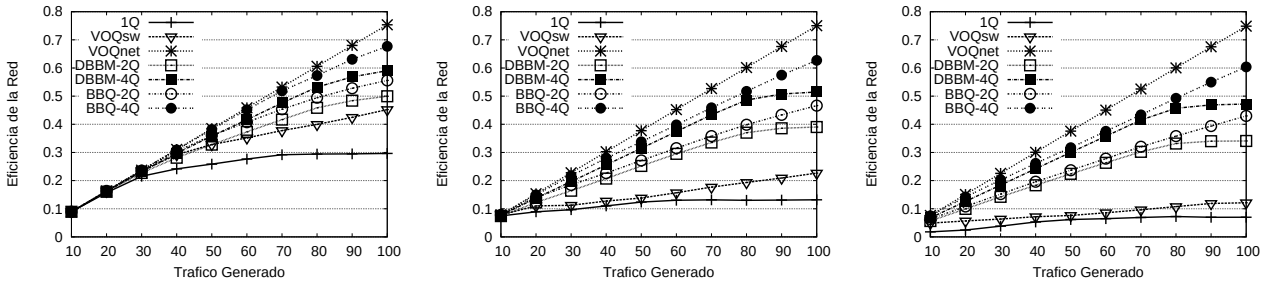
En resumen, BBQ alcanza el mejor rendimiento, a excepción de VOQnet, en estos escenarios, con menos o las mismas colas (sólo 4) que otros esquemas.

V. Conclusiones y Trabajo Futuro

El efecto *HoL-blocking* puede degradar drásticamente el rendimiento de las redes de interconexión de altas prestaciones actuales si no se soluciona adecuadamente. Aunque se han propuesto muchas técnicas para solucionar este problema en los conmutadores actuales, sólo son viables algunas de ellas, la mayoría basadas en esquemas de colas. Muchos de estos esquemas no tienen en cuenta la topología de red o el algoritmo de encaminamiento, por lo que en ciertos escenarios no aprovechan todas las colas disponibles. En este artículo hemos descrito y evaluado *Band-Based Queuing* (BBQ), un esquema de colas sencillo para la topología KNS k-ary n-direct s-indirect que reduce eficientemente el *HoL-blocking* en escenarios con tráfico uniforme y *hot-spot*, requiriendo un pequeño número de colas por puerto. BBQ mejora las

[1]Téngase en cuenta que estos números de puertos son comunes en los conmutadores comerciales

[2]Considerando las restricciones del control de flujo, el tamaño de paquete, el ancho de banda del enlace y el retardo del enlace, el tamaño mínimo de una cola es 4 KB (2 paquetes).



(a) Configuración de red #1. (b) Configuración de red #2. (c) Configuración de red #3.

Fig. 3. Eficiencia de red vs Tráfico generado (Tráfico Hot-Spot)

prestaciones de otros esquemas que necesitan un número similar de colas, y alcanzan prestaciones ligeramente menores que estos esquemas que necesitan muchas más colas. Por lo tanto, como BBQ se basa en una política simple pero inteligente de asignación de paquetes a colas, se puede implementar fácilmente en redes reales, como hemos explicado en el artículo y tenemos pensado hacer en el futuro.

Agradecimientos

Este trabajo ha sido parcialmente financiado por MINECO y por la JCCM, mediante los proyectos TIN2012-38341-C04-04 y POII10-0289-3724, respectivamente.

Referencias

[1] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queuing on a space-division packet switch," *IEEE Transactions on Communications.*, vol. COM-35, pp. 1347–1356, 1987.

[2] InfiniBand Trade Association, *InfiniBand architecture specification volume 1. Release 1.2.1.*

[3] Myrinet, ," https://www.myricom.com/.

[4] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Architecture," in *Proc. 1st HiPEAC Conf.*, 2005, pp. 266–285.

[5] M. Jurczyk and T. Schwederski, "Phenomenon of Higher Order Head-of-Line Blocking in Multistage Interconnection Networks under Nonuniform Traffic Patterns," *IEICE Trans. on Information and Systems*, vol. E79-D, no. 8, pp. 1124–1129, Aug. 1996.

[6] W. Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *6th Hot Interconnects*, 1998, pp. 41–50.

[7] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Efficient, Scalable Congestion Management for Interconnection Networks," *IEEE Micro*, vol. 26, no. 5, pp. 52–66, 2006.

[8] Jesus Escudero-Sahuquillo, Pedro J. Garcia, Francisco J. Quiles, Jose Flich, and Jose Duato, "An Effective and Feasible Congestion Management Technique for High-Performance MINs with Tag-Based Distributed Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, 2012.

[9] Y. Tamir and G.L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 725–737, June 1992.

[10] William Dally, "Virtual-Channel Flow Control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.

[11] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Trans. on Computer Systems*, vol. 11, no. 4, pp. 319–352, Nov 1993.

[12] J. Escudero-Sahuquillo, P. J. García, Francisco J. Quiles, J. Flich, and J. Duato, "OBQA: Smart and cost-efficient queue scheme for Head-of-Line blocking elimination in fat-trees," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1460–1472, 2011.

[13] J. Escudero-Sahuquillo, P. J. García, Francisco J. Quiles, J. Flich, and J. Duato, "Cost-effective queue schemes for reducing head-of-line blocking in fat-trees," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 17, pp. 2235–2248, 2011.

[14] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," in *Workshop CAC in conjunction with the IPDPS*, March 2007, p. 235.

[15] R. Penaranda, C. Gomez, M.E. Gomez, P. Lopez, and J. Duato, "A New Family of Hybrid Topologies for Large-Scale Interconnection Networks," in *Network Computing and Applications (NCA), 2012 11th IEEE International Symp. on*, Aug. 2012, pp. 220–227.

[16] J. Kim and W.J. Dally, "Flattened butterfly: A cost-efficient topology for high-radix networks," in *in Proc. of the Intl. Symp. on Computer Architecture*, 2007.

[17] J. Kim, W.J. Dally, S. Scott, and D. Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology," in *ISCA: Proc. of the 35th annual Int. Symp. on Computer Architecture*, 2008, pp. 77–88.

[18] M. Katevenis, D. Serpanos, and E. Spyridakis, "Credit-Flow-Controlled ATM for MP Interconnection: the ATLAS I Single-Chip ATM Switch," in *Proc. 4th HPCA*, 1998, pp. 47–56.

[19] W.L. Guay, S.A. Reinemo, O. Lysne, and T. Skeie, "dFtree: a fat-tree routing algorithm using dynamic allocation of virtual lanes to alleviate congestion in infiniband networks," in *Proceedings of the first international workshop on Network-aware data management*, New York, NY, USA, 2011, NDM '11, pp. 1–10, ACM.

[20] T. Nachiondo, J. Flich, and J. Duato, "Buffer Management Strategies to Reduce HoL-Blocking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, pp. 739–753, 2010.

[21] W. Olesinski, H. Eberle, and N. Gura, "Scalable alternatives to virtual output queueing," in *Proc. IEEE ICC*, 2009, pp. 1–6.

[22] W.L. Guay, B. Bogdanski, S.A. Reinemo, O. Lysne, and T. Skeie, "vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion," in *Proc. of IPDPS*, 2011, pp. 197–208.

[23] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized InfiniBand^{TM} fat-tree routing for shift all-to-all communication patterns," *Journal of CCPE*, vol. 22, no. 2, pp. 217–231, 2010.

[24] P. Yebenes, J. Escudero-Sahuquillo, P.J. Garcia, and F.J. Quiles, "Towards Modeling Interconnection Networks of Exascale Systems with OMNet++," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conf. on*, 2013, pp. 203–207.

[25] András Varga, "OMNeT++ User Manual," http://omnetpp.org/doc/omnetpp/manual/usman.html.

KNS: Familia de Topologías Híbridas para la Interconexión de Redes de Gran Escala

Roberto Peñaranda [1], Crispín Gómez [2], María Engracia Gómez [1], Pedro López [1] y Jose Duato[1]

Resumen— **En los grandes supercomutadores actuales, la topología de la red de interconexión es clave para su diseño, ya que impacta en sus prestaciones y coste. Las topologías directas proporcionan un coste hardware reducido, pero el número de dimensiones está restringido por del cableado. Por lo tanto, se utiliza un gran número de nodos por dimensión, lo que incrementa la latencia de comunicación y reduce la productividad de la red. Por otro lado, las topologías indirectas proporcionan mejores prestaciones para redes de gran tamaño, pero con un mayor número de switches y enlaces. En este trabajo, proponemos una nueva familia de topologías que combina las mejores caracteristicas de las topologías directas e indirectas. En concreto, proponemos una topología n–dimensional, donde los nodos de cada dimensión se conectan mediante una pequeña topología indirecta. De esta fusión obtenemos una familia de topologías que proporciona un alto rendimiento, con una productividad y latencia cercana a la conseguida con topologías indirectas, pero con un bajo coste. En concreto, es capaz de doblar la productividad ontenida por elemento de conmutación de las topologías indirectas. Además, el diseño de la topología es mucho más simple que en las topologías indirectas.**

Palabras clave— **topologías regulares, encaminamiento determinista, flattened–butterfly, clústers.**

I. Introducción

EL tamaño de los super-computadores está creciendo, año tras año. Las máquinas que se encuentran en lo más alto de la lista del top 500 [1] de super-computadores más potentes del mundo llegan a tener cientos de miles de nodos de procesamiento. Los principales objetivos de una red de interconexión son proporcionar comunicaciones con muy poca latencia para reducir el tiempo de ejecución de las aplicaciones, y una alta productividad para permitir tantas comunicaciones simultáneas como sea posible. Dos de los aspectos más importantes en el diseño de una red de interconexión son la topología de red y el algoritmo de encaminamiento [2], [3]. La topología define como se interconectan entre sí los nodos de procesamiento, y el algoritmo de encaminamiento determina cual va a ser el camino que recorrerá un paquete desde el nodo fuente hasta el nodo destino. La topología de una red también repercute en el coste de la red. Entre las diferentes formas de clasificar las topologías, la más utilizada las divide en topologías directas e indirectas[2], [3].

En este trabajo se propone una nueva familia de topologías híbridas, donde se combinan las mejores características de ambos tipos de topologías, obteniendo una familia de topologías que proporciona unas altas prestaciones, con productividad y latencia cercanas a las obtenidas con topologías indirectas, con un coste hardware reducido.

El resto del trabajo se organiza de la siguiente forma: la Sección II presentan las topologías directas e indirectas y trabajos actuales que se han presentado sobre esta temática, respectivamente. La Seccion III y IV describe la familia de topologías y los algoritmos de encaminamiento propuestos. En la Seccion V se evaluan y finalmente mostramos las conclusiones.

II. Topologías directas e indirectas

Las topologías directas normalmente adoptan una estructura ortogonal en un espacio n–dimensional. Este tipo de topologías son conocidas como k_d–ary n_d–cubes, donde k_d es el número de nodos en cada dimensión, y n_d el número de dimensiones de la red. Las topologías directas más utilizadas son la malla, el toro y el hipercubo. Las topologías directas están siendo usadas por las super-computadoras más potentes del momento. Por ejemplo, algunas de las primeras super-computadoras de la lista Top500 [1] en Junio del 2012 usan redes basadas en toros. Para un número dado de nodos N, las topologías directas proporcionan una mejor conectividad a medida que aumenta el número dimensiones de la red, pero esto causa un elevado coste ya que se usa más enlaces y los routers tienen un alto grado.

Por su parte, las redes indirectas más comunes son las redes multietapa (MINs). En las MINs, los switches están organizados por etapas. La MIN más utilizada en sistemas comerciales es la topología fat–tree [4]. Esta topología es usada en algunas de las supercomputadoras más potentes del momento, como por ejemplo en la supercomputadora Tianhe-2, la número 1 de la lista Top500 [1]. La implementación más extendida de la topología fat–tree es la k_i–ary n_i–tree. k_i es el número de enlaces que conectan un switch a la etapa siguiente o anterior, y n_i es el número de etapas de la red. Para una red con n nodos de procesamiento, si usamos switches de un alto grado, se necesitarán pocos switches, pero cada switch será más complejo.

Existen otros trabajos previos que proponen topologías diferentes a las presentadas, pero que nunca o pocas veces han sido usadas en productos comerciales o super-computadoras. Un ejemplo es la flattened–butterfly [5], que se obtiene fusionando los routers de cada fila de una MIN butterfly convencional, obteniendo así una red directa n–dimensional (sus parámetros serán referidos en este trabajo como k_f y n_f) donde los nodos de cada dimensión no están conectados con un anillo como en el toro, o mediante una pequeña topología indirecta como en la propuesta presentada en este trabajo, en su lugar es-

[1]Grupo de Arquitecturas Paralelas, Univ. Politécnica de Valencia, e-mail: ropeaceb@gap.upv.es.

[2]Dpto. de Informática, Univ. de Castilla La Mancha, e-mail: Crispin.Gomez@uclm.es.

tán totalmente conectados. De esta combinación sale resultante una topología muy similar a un hipercubo generalizado [6], pero conectando varios nodos de procesamiento a un mismo switch. Esta topología, como el hipercubo generalizado, tiene un alto coste, especialmente en máquinas de gran tamaño, máquinas a las que va orientada la familia propuesta en este trabajo.

III. Nueva Familia de Topologías Híbridas

En esta sección se presenta la nueva familia de topologías híbridas que combina redes directas n–dimensionales con pequeñas redes indirectas, combinando las ventajas de las topologías directas e indirectas para obtener una familia de topologías que permita conectar un gran número de nodos de procesamiento, mientras que se proporciona una latencia baja y una productividad alta con un coste más bajo que las redes indirectas.

A. Descripción de la Familia

La nueva topología propuesta organiza los nodos de procesamiento en n dimensiones, como una topología directa, pero cada nodo de procesamiento no está exclusivamente conectado a sus nodos adyacentes en cada dimensión. En este caso, todos los nodos de una dimensión dada están directamente conectadas por medio de una red indirecta. La familia propuesta se define con tres parámetros: el número de dimensiones n_n, el número de nodos por dimensión k_n y el número de etapas de la subred indirecta, referenciado como s_n. El número de etapas necesarias para interconectar los k_n nodos de procesamiento de una dimensión dada depende del número de puertos de los switches que implementa la subred indirecta, que será referenciado como d_n. Si $k_n \leq d_n$, un simple crossbar puede ser capaz de interconectar todos los nodos de la dimensión, y s_n tendrá un valor de 1. Por otro lado, si $k_n > d_n$, se necesitará una MIN para interconectar los nodos de una dimensión dada, y el número de etapas vendrá dado por $log_{\frac{d_n}{2}} k_n$. Esta familia de topologías recibe el nombre de k_n–ary n_n–direct s_n–indirect (KNS).

En este trabajo se han considerado dos MINs diferentes. La primera es la topología fat–tree, una MIN bidireccional. La segunda MIN considerada es RUFT [7], una MIN unidireccional que utiliza un algoritmo de encaminamiento determinista con equilibrado de carga [8], la cual requiere menos recursos hardware reduciendo un poco las prestaciones. La Figura 1 muestra un ejemplo de la nueva topología con 2 dimensiones ($n_n = 2$) y 4 nodos de procesamiento por dimensión ($k_n = 4$), con un total de 16 nodos de procesamiento. En este ejemplo, como tenemos 4 nodos por dimensión y el grado de los switches es 4 ($d_n = 4$), con un único crossbar es suficiente para interconectar los 4 nodos de cada dimensión. Por lo tanto, la red implementa una 4–ary 2–direct 1–indirect. Sin embargo, si tenemos un mayor número de nodos por dimensión, como por ejemplo 8, y switches del mismo tamaño, tendríamos que utilizar una MIN, por ejemplo un fat–tree o un RUFT con 3 etapas y 12 switches con 4 puertos (bidireccionales para fat–tree y unidireccionales para RUFT), implementando en ese caso una 8–ary 2–direct 3–indirect.

Es posible conectar varios nodos de procesamiento en el mismo router (lo que se conoce en la bibliografía como concentración). Esto se ve en la Figura 2, donde los nodos y los routers se muestran separados. En este caso, dos nodos de procesamiento están conectados a cada router. Sólo se muestran los nodos de las esquinas por motivos de claridad. Esta extensión introduce un nuevo parámetro en la topología, p_n, que representa el número de nodos de procesamiento conectados a cada router. En este caso el número de nodos de procesamiento totales se calcula como $N = p_n k_n^{n_n}$. En la Figura 2, se pueden distinguir dos elementos de encaminamiento de diferente tamaño. Los switches (etiquetados con una "S") con un grado de k_n, y los routers (etiquetados con "R") con un grado de $n_n + p_n$. Como se puede ver en la Figura 2, si se usa un crossbar como subred indirecta, los switches permiten a los paquetes cambiar su posición en una dimensión dada atravesando solo dos enlaces, mientras que los routers permiten cambiar de dimensión. Por lo tanto, el diámetro de la nueva topología es $2n_n$. Si se usa una MIN como subred indirecta, el diámetro de la red será el diámetro de la MIN usada multiplicado por n_n.

TABLA I
Parámetros de los diferentes tipos de topologías analizadas en este trabajo.

Topología	Parámetros	
Malla o Toro	n_d	número de dimensiones
	k_d	número de nodos por dimensión
Fat–tree	n_i	número de etapas
	k_i	aridad del switch
KNS	n_n	número de dimensiones
	k_n	número de nodos por dimensión
	s_n	número de etapas de la subred indirecta
	d_n	grado del switch
	p_n	número nodos por router

La tabla I resume los parámetros que definen las redes directas e indirectas tradicionales y además la familia híbrida de topologías propuesta en este trabajo. Cabe destacar que solo el parámetro d_n o s_n es necesario, ya que uno puede ser derivado a partir del otro y k_n.

Las topologías híbridas propuestas pueden sacar ventaja del alto número de puertos de los switches actuales. Estos pueden tener hasta 64 puertos [9], aunque en recientes trabajos se han propuesto switches de hasta 144 puertos [10]. Con estos switches, con una pequeña MIN con sólo 2 ó 3 etapas o con solo un simple switch sería posible conectar todos los nodos de cada dimensión en la mayoría de los casos.

Las topologías híbridas presentan varias ventajas principales. La primera, que permite reducir el diámetro comparado con las topologías directas. Con esto se espera que aumenten las prestaciones de la red, ya que se incrementa la productividad y disminuye la latencia. Además, el número de switches y enlaces se reduce si lo comparamos con una topología indirecta, como se mostrará en la sección V. Con todo esto, se espera que la nueva familia de topologías pueda reducir los costes de la red.

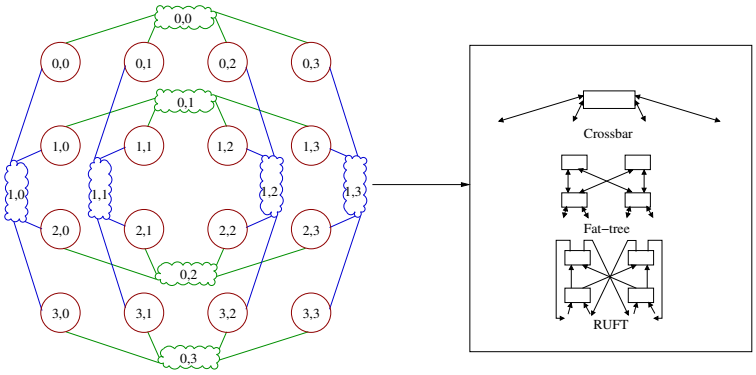


Fig. 1
4-ary 4-direct s-indirect.

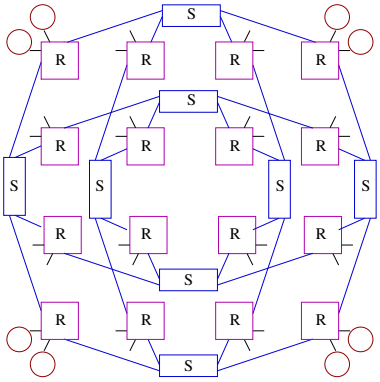


Fig. 2
4-ary 4-direct 1-indirect con $p_n = 2$.

IV. Algoritmo de Encaminamiento para la Nueva Familia de Topologías

En esta sección se describe el algoritmo propuesto para las topologías KNS que utilizan crossbars, y después se describirán los algoritmos para el caso general, es decir, para las topologías KNS con MINs como subredes indirectas.

A. Encaminamiento en k_n–ary n_n–direct 1–indirect

Cada router será etiquetado como en una malla o en un toro, con tantas componentes o coordenadas como dimensiones tenga la red $\langle r_{n_n-1}, r_{n_n-2}, ..., r_1, r_0 \rangle$. Cada coordenada representa la posición de cada router en cada una de las dimensiones. Por otro lado, los switches están etiquetados mediante una dupla *[d,p]*, donde d corresponde a la dimensión en la cual está localizado el switch, y p a la posición que tiene el switch en la dimensión en la que se encuentra (ver Figura 1).

El algoritmo de encaminamiento determinista para las topologías k_n–ary n_n–direct 1–indirect, que será referenciado como Hybrid–DOR, es una variante del algoritmo de encaminamiento determinista DOR para mallas. En Hybrid–DOR, las dimensiones de la red también son cruzadas en un orden establecido para garantizar un algoritmo libre de bloqueos. Por lo tanto, en Hybrid–DOR, los routers envían los paquetes directamente por el único enlace de la dimensión que se tiene que atravesar, y este enlace está conectado al correspondiente crossbar. Cabe destacar que en estas topologías no se requiere elegir el sentido en cada dimensión, ya que solo se dispone de un enlace que conecta con el crossbar. Respecto a cómo encaminan los paquetes los crossbars, éstos envían los paquetes a través del enlace indicado por la componente del destino en la correspondiente dimensión, necesitando solo un salto para alcanzar el siguiente router.

B. Encaminamiento en k_n–ary n_n–direct s_n–indirect

En las topologías k_n–ary n_n–direct s_n–indirect, los crossbars son reemplazados por pequeñas MINs.

Para poder identificar los switches dentro de un fat-tree o RUFT, se han extendido las coordenadas clásicas de las MINs, incluyendo las coordenadas correspondientes de la MIN en la topología global. De esta manera, las coordenadas del switch en estas topologías KNS vienen dadas por una tupla *[d,p,e,o]*, donde d es la dimensión en la que se encuentra la MIN, p es la posición de la MIN en esta dimensión, e es la etapa del switch dentro de la MIN y o es la posición del switch en esa etapa. Hay que recordar que d y p son las coordenadas del crossbar equivalente en las topologías KNS que utilizan crossbars.

Dado que los routers son los mismos independientemente de la topología indirecta que se use, su algoritmo de encaminamiento es el mismo que para las topologías KNS que utilizan crossbars. Sin embargo, el algoritmo de encaminamiento de los switches depende de la red indirecta que se esté usando.

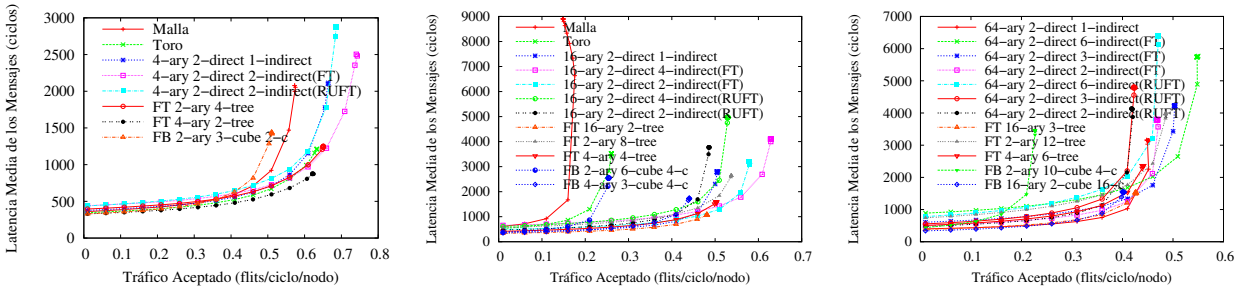
Primero nos centraremos en la topología KNS que usa fat–trees en las subredes indirectas. A pesar de que un fat–tree tiene varios caminos para cada par fuente–destino, lo que permite encaminamiento adaptativo, se propone utilizar el algoritmo de encaminamiento determinista presentado en [8], ya que es simple y permite superar al encaminamiento adaptativo para muchos patrones de tráfico. En las subredes fat–trees de las topologías KNS, el algoritmo de encaminamiento es el mismo, pero solo se usa la parte del identificador del mismo correspondiente a la dimensión en la que se encuentra el fat–tree (x_d). De esta manera, el paquete es entregado al mismo router que sería alcanzado en el caso de usar un crossbar, en una topología KNS que utilizan crossbars.

En RUFT, solo hay un camino entre cada par fuente–destino, y los paquetes viajan subiendo todas las etapas, alcanzando la última etapa, que está directamente conectada a los nodos de procesamiento. El enlace que tiene que ser utilizado en un switch en particular es dado por la componente del destino correspondiente a la etapa donde el switch esta localizado. Véase [7] para más detalles.

V. Evaluación

Hemos evaluado la nueva familia de topologías mediante simulación. Se han utilizado switches con colas de entrada y salida con virtual cut-through y un control de flujo basado en creditos. Los valores de temporización de la red son los del modelo de red Myrinet en [11].

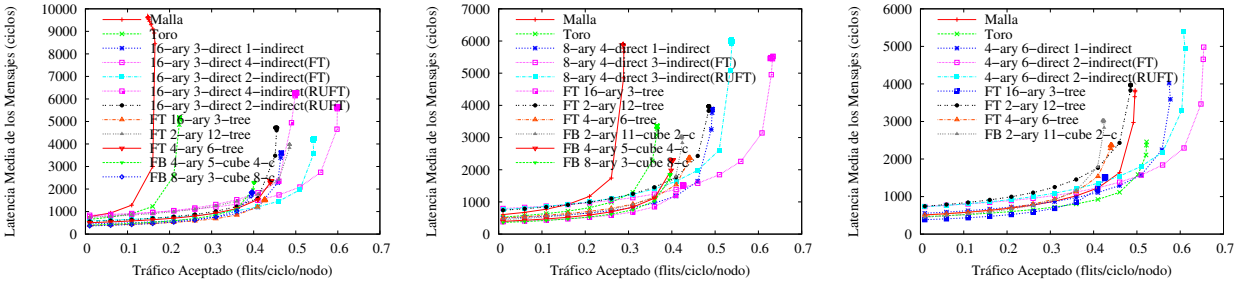
La Figura 3.(a) muestra los resultados para una red muy pequeña (16 nodos) con tráfico uniforme. Como se puede ver, la topología flattened–butterfly es de las que menos productividad alcanza. La malla es la segunda peor topología seguida del toro. Las diferentes configuraciones de la familia propuesta y los



(a) (b) (c)

Fig. 3

Tráfico uniforme para diferentes topologías y 2 dimensiones con: (a) 16, (b) 256 y (c) 4096 nodos.



(a) (b) (c)

Fig. 4

Tráfico uniforme para diferentes topologías y 4096 nodos con: (a) 3, (b) 4 y (c) 6 dimensiones.

fat–trees obtienen una productividad muy similar.

Al incrementar el número de nodos por dimensión dejando constante el número de dimensiones en las topologías n–dimensionales, como se puede ver en las Figuras 3.(b) y 3.(c), la productividad disminuye en todas las topologías. En el caso de las topologías malla y toro, la productividad decae en mayor medida, ya que la distancia media entre dos nodos crece mucho más que en las demás topologías. De hecho, en la Figura 3.(c) no se muestran las topologías malla y toro porque su productividad es muy baja (un 89 % y un 84 % mas baja que la KNS con crossbars, respectivamente). Como se puede ver, la topología KNS es mucho más escalable que la malla y el toro. En todos los casos, la topología con mejores prestaciones es una de la nueva familia. En concreto, la que usa fat–trees como subred indirecta. En las Figuras 3.(b) y 3.(c) también se puede ver la diferencia de usar más etapas en las MINs de las topologías KNS. Para un mismo número de nodos por dimensión, cuanto más etapas tenga la subred indirecta, más latencia se obtendrá, ya que se tienen que cruzar más switches. Pero este hecho hace que se tenga más productividad, que se explica por la reducción del efecto del HoL (Head of Line) blocking.

Respecto a la latencia base, en una KNS con crossbars es siempre el mismo, pues el número de saltos entre nodos no depende del número de nodos por dimensión. Sin embargo, en la KNS con MINs como subredes, y los fat–trees, la latencia base aumenta porque aumenta el número de etapas, y por tanto, el número de switches que se tienen que cruzar para mandar un paquete de un nodo a otro. En el caso de los toros y mallas, la latencia base está fuertemente ligada con el número de nodos por dimensión, ya que la distancia media entre nodos aumenta.

La Figura 4 analiza el impacto del número de dimensiones en las diferentes topologías. Se han analizado redes de 4096 nodos implementadas con diferentes números de dimensiones. Se puede distinguir tres comportamientos distintos. Las topologías malla y toro tienen un comportamiento similar. Cuantas más dimensiones y menor número de nodos por dimensión, mayor productividad se alcanza. También disminuye la latencia, porque la distancia media se reduce. Para las KNS que utilizan crossbars, la productividad aumenta con el número de dimensiones porque el tamaño de los crossbars es reducido, y por lo tanto, el pernicioso efecto del HoL blocking se reduce. Sin embargo, la latencia base no mejora con el incremento de dimensiones. Porque el número de saltos que el paquete tiene que hacer crece con el número de dimensiones. En las topologías KNS con MINs tenemos un comportamiento similar, pero con una diferencia. La latencia base, en este caso, disminuye cuando incrementamos las dimensiones, ya que hay menos nodos por dimensión. Por lo tanto, las subredes indirectas tienen menos etapas que cruzar. La configuración seleccionada para la topología flattened–butterfly (la única con un coste hardware similar al de la nueva familia propuesta) proporciona una productividad intermedia.

A. Análisis Coste–Prestaciones

Esta sección estima y compara el coste hardware de cada topología evaluada en este trabajo. En particular, se analiza el número de enlaces, el número de switches y el número de elementos de conmutación. En este trabajo, consideramos el elemento de conmutación como componente básico de un multiplexor, de tal forma que para un multiplexor de n entradas requiere n elementos de conmutación. Además, se introducirán dos índices más con el fin de eva-

luar mejor de forma combinada las prestaciones con el coste hardware de una topología. El primero es la productividad por elemento de conmutación. Cuanto mayor sea este parámetro, mejor relación prestaciones/coste tendrá la topología. El otro parámetro es el producto de la latencia base y el número de elementos de conmutación. Cuanto mayor sea, peor será la topología.

La Tabla II muestra las fórmulas para calcular el número de enlaces, switches y elementos de conmutación para cada topología. En el cálculo de los enlaces no están incluidos los enlaces entre los nodos de procesamiento y los switches, ya que son los mismos para todas las topologías. Otro aspecto considerado es que en todas las topologías, los enlaces son bidireccionales pero, cuando se usa RUFT como subred indirecta, estas subredes utilizan enlaces unidireccionales. Además, el número de elementos de conmutación en las subredes RUFT es menor que en una subred fat–tree, debido a que se usan switches unidireccionales.

La Tabla III muestra en la parte de arriba los resultados para las topologías de 16 nodos. Las redes son de dos dimensiones en el caso de las topologías KNS, malla y toro. También se ha considerado una red de dos dimensiones para la topología flattened–butterfly y otra configuración de tres dimensiones que posee un coste hardware similar a la familia propuesta, con el mismo número de nodos de procesamiento. Como se aprecia, la topología flattened–butterfly de dos dimensiones es la que mayor productividad alcanza, seguida de la KNS usando fat–trees como subredes, pero con un gran número de elementos de conmutación. Las siguientes son la KNS con RUFT y crossbar como subredes indirectas.

Ahora comparemos los costes de las distintas topologías. La topología flattened–butterfly es la que menos elementos de conmutación utiliza. Después le sigue las topologías KNS que utilizan crossbars y KNS utilizando RUFT como subred indirecta, las cuales tienen el mismo número de elementos de conmutación. Aunque la última necesita más switches para ser implementada, estos switches son mucho más simples. Por otro lado, las topologías flattened–butterfly de 2 dimensiones y KNS usando fat–trees como subredes indirectas, en ese orden, son las topologías más costosas, respecto a número de elementos de conmutación. Sin embargo, cuando se considera la relación productividad/coste, y en concreto el parámetro productividad/elemento de conmutación, las conclusiones son totalmente diferentes. En este caso, la mejor topología es la flattened–butterfly de 3 dimensiones, seguida muy de cerca por KNS con RUFT y KNS con crossbars. Por otro lado, los fat–trees son menos interesantes debido a su mayor complejidad. Conforme va subiendo el número de etapas de las subredes indirectas, se puede ver que empeora la relación latencia–coste. Sin embargo, si pensamos solo en la productividad, una MIN con más etapas es mejor.

La Tabla III muestra también los resultados para redes más grandes. Como se puede observar, cuando el número de nodos por dimensión aumenta, las prestaciones de red de las topologías malla y toro empeoran en gran medida. Respecto a las topologías híbridas que se han presentado en este trabajo, los resultados mostrados confirman lo que se dijo anteriormente. Las mejores prestaciones las obtienen las topologías flattened–butterfly y las configuraciones de KNS (subredes fat–tree), con una ventaja mayor para las topologías propuestas en este trabajo a medida que vamos aumentando el número de nodos por dimensión. Desde el punto de vista coste–prestaciones, las topologías con RUFT como subredes indirectas son la mejor opción. La tabla IV muestra los resultados para las mismas grandes redes (4096 nodos) pero ahora considerando más dimensiones (3 y 6). Como se puede observar, cuando el número de dimensiones utilizadas aumenta se consiguen mejoras, especialmente para las topologías malla y toro. Sin embargo, la topología que siempre obtiene mejor productividad es la flattened–butterfly, seguida de la configuración KNS (fat–trees), aunque con un mayor coste en el caso de la primera. La mejor relación coste–prestaciones la consigue la topología KNS (RUFT).

VI. Conclusiones

En este trabajo se ha presentado una nueva familia de topologías híbridas para interconectar redes de gran escala con el objetivo de combinar las ventajas de las topologías directas e indirectas.

Esta combinación proporciona un conjunto de topologías que proporciona altas prestaciones, con latencias y productividades cercanas a las redes indirectas, pero con un menos coste hardware. En concreto las nuevas topologías que usan fat-trees como subredes indirectas son las topologías que alcanzan la mayor productividad. Sin embargo, desde el punto de vista coste/prestaciones, las nuevas topologías con RUFT como subredes indirectas obtienen mejores resultados, siendo capaces de obtener el doble de la productividad por elemento de conmutación comparado con las topologías indirectas, y esta ventaja es aún mayor para las topologías directas. Además, el diseño de la topología es mucho más simple que en las topologías indirectas.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO) y por los fondos del Plan E mediante la Ayuda TIN2009-14475-C04-01 y TIN2012-38341-C04-01, y mediante las Ayudas para Primeros Proyectos de Investigación de la Universitat Politècnica de València con referencia 2370.

Referencias

[1] "TOP500 Supercomputer Site," `http://www.top500.org`.
[2] William Dally and Brian Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
[3] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
[4] Charles E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985.
[5] J. Kim, W.J. Dally, and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," in *Proceedings of the 34th annual international symposium on*

TABLA II

Comparación analítica de las diferentes topologías. k_i en las topologías KNS se refiere a la aridad de los switches indirectos.

	Malla	Toro	Fat–tree	Flattened–Butterfly	KNS FT	KNS Xbar y RUFT
Switches	N	N	Nn_i/k_i	$k_f^{n_f}$	$N/p_n((m_n n_n/k_i)+1)$	$N/p_n((m_n n_n/k_i)+1)$
Elementos de conm.	$N(2n_d+1)^2$	$N(2n_d+1)^2$	$4k_i n_i N$	$k_f^{n_f}(n_f(k_f-1)+N/k_f^{n_f})^2$	$N/p_n((4m_n n_n k_i+(n_n+p_n)^2)$	$N/p_n(m_n n_n k_i+(n_n+p_n)^2)$
Enlaces	$2(k_d-1)k_d^{n_d-1}n_d$	$2k_d^{n_d}n_d$	$2N(n_i-1)$	$k_f^{n_f}(k_f-1)n_f$	$2(k_n^{n_n}n_n+(m_n-1)Nn_n/p_n)$	$2k_n^{n_n}n_n+(m_n-1)Nn_n/p_n$

TABLA III

Resultados para diferentes topologías 2–D con tráfico uniforme y encaminamiento determinista.

#Nodos	Topología	Lat. base	Prod.	Enlaces	Switches	E.C.	Prod./E.C.	Lat. base*E.C.
16	Flattened–Butterfly 2-ary 3-cube 2-c	329,00	0,51133	24	8	200	0,0025567	65800
16	4–ary 2–direct 2–indirect(RUFT)	437,01	0,69003	96	48	272	0,0025369	118866
16	4–ary 2–direct 1–indirect	375,48	0,66259	64	24	272	0,0024360	102131
16	Toro	345,30	0,63534	64	16	400	0,0015883	138120
16	Malla	360,38	0,57466	48	16	400	0,0014367	144150
16	Fat-Tree aridad 2 etapas 4	415,57	0,65030	96	32	512	0,0012701	212773
16	Fat-Tree aridad 4 etapas 2	327,46	0,62456	32	8	512	0,0012198	167661
16	4–ary 2–direct 2–indirect(FT)	439,24	0,74223	128	48	656	0,0011314	288139
16	Flattened–Butterfly 4-ary 2-cube 1-c	328.47	0,81492	96	16	784	0,0010394	257524
4096	64–ary 2–direct 6–indirect(RUFT)	770,56	0,47438	57344	28672	135168	0,0000035	104155095
4096	64–ary 2–direct 3–indirect(RUFT)	546,71	0,42546	32768	10240	135168	0,0000031	73898072
4096	64–ary 2–direct 2–indirect(RUFT)	463,11	0,41971	24576	6144	167936	0,0000025	77772700
4096	64–ary 2–direct 6–indirect(FT)	866,40	0,55148	98304	28672	430080	0,0000013	372620624
4096	64–ary 2–direct 3–indirect(FT)	591,82	0,50569	49152	10240	430080	0,0000012	254531077
4096	Fat-Tree aridad 2 etapas 12	865,20	0,48493	90112	24576	393216	0,0000012	340210585
4096	Fat-Tree aridad 4 etapas 6	561,23	0,43872	40960	6144	393216	0,0000011	220683070
4096	Flattened–Butterfly 2-ary 10-cube 4-c	439,94	0,22688	10240	1024	200704	0,0000011	88297728
4096	64–ary 2–direct 2–indirect(FT)	499,96	0,46912	32768	6144	561152	0,0000008	280553722
4096	64–ary 2–direct 1–indirect	389,29	0,44818	16384	4224	561152	0,0000008	218448488
4096	Toro	1292,88	0,07323	16384	4096	102400	0,0000007	132390963
4096	Flattened–Butterfly 16-ary 2-cube 16-c	338,47	0,40332	7680	256	541696	0,0000007	183349068
4096	Fat-Tree arity 16 stages 3	388,45	0,41601	16384	768	786432	0,0000005	305487961
4096	Malla	1606,12	0,05084	16128	4096	102400	0,0000005	164467087
4096	Flattened–Butterfly 64-ary 2-cube 1-c	339,65	0,60032	516096	4096	66064384	0,0000000	22435873084

TABLA IV

Resultados para topologías de 4096 nodos con tráfico uniforme y encaminamiento determinista.

#DIM	Topología	Lat. base	Prod.	Enlaces	Switches	E.C.	Prod./E.C.	Lat. base*E.C.
3	16–ary 3–direct 4–indirect(RUFT)	766,14	0,50088	61440	28672	163840	0,0000031	125524540
3	16–ary 3–direct 2–indirect(RUFT)	544,01	0,45590	36864	10240	163840	0,0000028	89130980
3	16–ary 3–direct 1–indirect	443,32	0,46498	24576	4864	262144	0,0000018	116213728
3	16–ary 3–direct 4–indirect(FT)	815,39	0,60257	98304	28672	458752	0,0000013	374060903
3	Fat-Tree arity 2 stages 12	865,20	0,48493	90112	24576	393216	0,0000012	340210585
3	16–ary 3–direct 2–indirect(FT)	576,95	0,50412	49152	10240	458752	0,0000011	264674815
3	Torus	654,40	0,22376	24576	4096	200704	0,0000011	131340348
3	Fat-Tree arity 4 stages 6	561,23	0,43872	40960	6144	393216	0,0000011	220683070
3	Flattened–Butterfly 4-ary 5-cube 4-c	396,56	0,40179	15360	1024	369664	0,0000011	146595250
3	Flattened–Butterfly 8-ary 3-cube 8-c	361,82	0,39680	10752	512	430592	0,0000009	155798223
3	Mesh	762,46	0,16553	23040	4096	200704	0,0000008	153029131
3	Fat-Tree arity 16 stages 3	388,45	0,41601	16384	768	78432	0,0000005	305487961
3	Flattened–Butterfly 16-ary 3-cube 1-c	365,00	0,63116	184320	4096	8667136	0,0000001	3163526160
6	4–ary 6–direct 2–indirect(RUFT)	714,24	0,61761	73728	28672	299008	0,0000021	213564147
6	4–ary 6–direct 1–indirect	546,78	0,57877	49152	10240	299008	0,0000019	163492186
6	Fat-Tree arity 2 stages 12	865,20	0,48493	90112	24576	393216	0,0000012	340210585
6	Flattened–Butterfly 2-ary 11-cube 2-c	450,03	0,42385	22528	2048	346112	0,0000012	155761542
6	4–ary 6–direct 2–indirect(FT)	715,60	0,65648	98304	28672	593920	0,0000011	425011201
6	Fat-Tree arity 4 stages 6	561,23	0,43872	40960	6144	393216	0,0000011	220683070
6	Torus	463,94	0,52038	49152	4096	692224	0,0000008	321150900
6	Mesh	502,79	0,49203	36864	4096	692224	0,0000007	348041789
6	Fat-Tree arity 16 stages 3	388,45	0,41601	16384	768	78432	0,0000005	305487961
6	Flattened–Butterfly 4-ary 6-cube 1-c	416,37	0,66666	73728	4096	1478656	0,0000005	615671623

Computer architecture, New York, NY, USA, 2007, ISCA '07, pp. 126–137, ACM.

[6] L.N. Bhuyan and D.P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," *Computers, IEEE Transactions on*, vol. C-33, no. 4, pp. 323–333, april 1984.

[7] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato, "RUFT: Simplifying the Fat-Tree Topology," in *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, dec. 2008, pp. 153–160.

[8] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, march 2007, pp. 1–8.

[9] S. Scott, D. Abts, J. Kim, and W.J. Dally, "The BlackWidow High-Radix Clos Network," *SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 16–28, May 2006.

[10] N. Binkert, Al Davis, N.P. Jouppi, M. McLaren, N. Muralimanohar, R. Schreiber, and Jung Ho Ahn, "The role of optics in future high radix switch design," *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 437–448, June 2011.

[11] J. Flich, M.P. Malumbres, P. López, and J. Duato, "Improving Routing Performance in Myrinet Networks," *Parallel and Distributed Processing Symposium, International*, p. 27, 2000.

Deterministic versus adaptive routing in direct topologies

Roberto Peñaranda [1], Crispín Gómez [2], María Engracia Gómez [3], Pedro López [3], y Jose Duato[3]

Abstract— **Routing is a key design factor to obtain the maximum performance out of interconnection networks. Depending on the number of routing options that packets may use, routing algorithms are classified into two categories. If the packet can only use a single predetermined path, routing is deterministic, whereas if several paths are available, it is adaptive. It is well-known that adaptive routing usually outperforms deterministic routing. However, adaptive routers are more complex and introduces out-of-order delivery of packets. In this paper, we take up the challenge of developing a deterministic routing algorithm for direct topologies that can obtain a similar performance than adaptive routing, while providing the inherent advantages of deterministic routing such as in-order delivery of packets and implementation simplicity. The key point of the proposed deterministic routing algorithm is that it is aware of the HoL-blocking effect, and it is designed to reduce it, which, as known, it is a key contributor to degrade interconnection network performance.**

Keywords— **Interconnection networks; Direct topologies; Deterministic routing; Adaptive routing.**

I. Introduction

THE interconnection network performance strongly impacts on the performance of large parallel computers. Latency and throughput are the key performance metrics of interconnection networks [1], [2]. To achieve the required performance level, the designer manipulates three main parameters [1], [2]: topology, routing and switching. The switching mechanism decides how resources are allocated to the messages while they advance through the network. Topology usually adopts a regular structure that simplifies routing, implementation and expansion capability. Among the different taxonomies of regular topologies, the most commonly one divides them into direct and indirect topologies. Taking into account that many of the very large machines of the top500 list [3] adopt a direct network topology, in this paper, we will focus on direct networks, although its conclusions could be extrapolated to indirect networks. Routing is a critical design issue of interconnection networks [1]. Routing algorithms can be deterministic or adaptive. In deterministic routing schemes, an injected packet traverses a unique, fixed, predetermined path between source and destination, while in adaptive routing schemes, several paths are available. However, as several routes are possible, a choice or selection of the path that will be finally used is required, which makes routing operation more complex. In addition, several concerns about deadlock-freedom must be taken into account. Moreover, adaptive routing introduces the problem of out-of-order delivery of packets, which occurs when a packet sent from a given source arrives to a given destination before another one sent previously from the same source to the same destination. In-order packet delivery is important for cache coherence protocols and communication libraries. While there are solutions to this problem (for instance by using a reordering buffer at destinations [4]), they are not simple, as they require the use of storage resources and control packets. Instead, deterministic routing guarantees in-order packet delivery by design.

Another issue to consider when designing routing algorithms is to avoid interferences among packets destined to different nodes [5], [6], since the Head-Of-Line (HoL) blocking effect may limit the throughput of the switch up to 58% of its peak value [7], [8], [9].

In this paper, we explore the behavior of both deterministic and adaptive routing on direct topologies, also proposing a new deterministic routing algorithm that takes advantage of virtual channels to reduce the HoL blocking effect.

The rest of the paper is organized as follows. Section II introduces some background. In Section III, we present the new HoL-blocking-aware deterministic routing algorithm with virtual channels (VCs). Section IV evaluates different topologies (torus and mesh) with different routing algorithms. Finally, some conclusions are drawn.

II. Direct topologies

The most important regular direct topology is the k–ary n–cube, which has k nodes in each of its n dimensions, connected in a ring fashion. In this topology, nodes are labeled by an identifier with as many components as dimensions in the topology $\langle p_{n-1}, ..., p_0 \rangle$, and the value of the component associated to each dimension ranges from 0 to $k-1$ (i.e., nodes are numbered from $\langle 0, 0, ..., 0 \rangle$ to $\langle k-1, k-1, ..., k-1 \rangle$). This topology is popularly known as torus. A mesh is a particular case where the k nodes of each dimension are connected by a linear array (i.e. without wraparound links).

Deterministic routing in meshes and tori can be implemented with the DOR (dimension-order routing) routing algorithm [1]. This algorithm routes packets by crossing dimensions in strictly increasing (or decreasing) order. Despite that DOR is deadlock-free in meshes, it is not in tori due to the wraparound

[1]Grupo de Arquitecturas Paralelas, Univ. Politécnica de Valencia, e-mail: ropeaceb@gap.upv.es.
[2]Dpto. de Informática, Univ. de Castilla La Mancha, e-mail: Crispin.Gomez@uclm.es.
[3]Grupo de Arquitecturas Paralelas, Univ. Politécnica de Valencia, e-mail: {megomez,plopez,jduato}@disca.upv.es.

links. Channel dependence graph cycles has to be broken by splitting each physical channel into two VCs [10]. Another technique used to avoid deadlocks in tori with deterministic routing is the bubble flow control mechanism [11].

Adaptive routing can be achieved in meshes and tori by allowing packets to cross dimensions in any order. According to Duato's theory [12], we add a set of VCs that may be used to cross network dimensions in any order. Deadlock freedom is achieved by providing a *escape path* to packets, by means of using a deadlock-free routing algorithm (for instance, DOR) in other virtual channel.

III. A HoL-blocking-aware Deterministic Routing Algorithm

Adaptive routing requires more resources (i.e. VCs) than deterministic routing. Indeed, more VCs imply not only more buffers but also increasing the crossbar size and the routing algorithm complexity. We will analyze alternatives to improve network performance also based on the use on VCs but trying to reduce this complexity.

A first approach is to use deterministic routing with several VCs, which offer as many routing options as VCs, also requiring a selection function as adaptive routing does. Although this routing algorithm improves network performance over the baseline deterministic routing (see Section IV), switch complexity and therefore routing time are still increased. Moreover, deterministic routing with several VCs may also introduce out-of-order delivery of packets. For this reason, we will refer to this mechanism as OODET (Out-of-Order DETerministic routing).

What is actually needed is a criterion to assign packets to VCs. We propose to classify them depending on their destinations. If a packet destined to a given node is only assigned to one VC, always the same, the result is a deterministic routing algorithm, with all its advantages (simpler, faster and in-order delivery of packets). In particular, we propose to assign VCs to packets according to the component of its destination node in the dimension in which the packet is being routed, modulo the number of VCs. As a consequence, the proposed mechanism classifies destinations among the VCs, thus contributing to reduce the interference among destinations, and, therefore, to reduce the HoL-blocking effect. A nice property of this mechanism is, that, as only one routing option is provided for each destination, it preserves, by design, in-order delivery of packets and simplicity. On the other hand, VC selection is easily done, as only the LSBs of a component from destination identifier are required. In contrast to OODET, we will refer to this mechanism as IODET (In-Order DETerministic routing).

In [13], a similar mechanism (DBBM) was proposed, but considering the whole destination identifier modulo the number of VCs to assign packets to VCs. The fact of only considering the LSBs of the destination provides lower opportunities of classifying packets among VCs (see Section IV). DBBM is a simplification of VOQnet [14], which needs as many VCs as nodes in the network, and associates each VC to a different destination. Another scheme is VOQsw [15], in which VCs are selected according to the next output port the packet will use. VOQsw and VOQnet are not scalable as the number of VCs they require depends on the system or switch size, respectively.

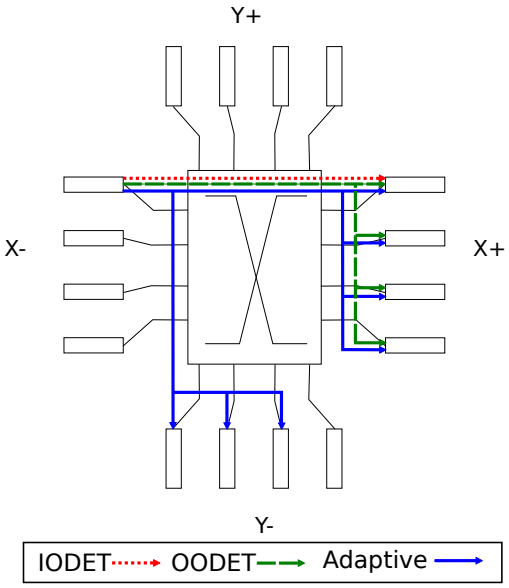


Fig. 1
VCs that can be chosen in IODET, OODET and adaptive.

Figure 1 shows one router of a 2D-network with four VCs per physical channel. We can see which output channels can be used by adaptive routing, OODET and IODET for a packet that has to traverse both dimensions. In the case of adaptive routing, the packet can be forwarded to any of the output VCs of both $X+$ and $Y-$ ports, but the escape path in $Y-$ since it would introduce a cycle in the dependency graph. In deterministic routing, as DOR is used and the packet has to still cross dimension X, packets can only be forwarded to $X+$ port. The difference between OODET and IODET relies in the possible output VCs in port $X+$. In OODET, all the VCs of this port can be used by the packet, while the packet can only be forwarded to one VC in IODET, selected according to its destination. Notice that both adaptive routing and OODET needs a selection function, while IODET does not require it.

IV. Experimental Evaluation

In this section, we compare by simulation adaptive versus deterministic routing in meshes and tori. Each node has a switch based on a full crossbar with queues of two packets both at their input and output ports. Switch and link bandwidth is one flit per clock cycle, and link fly time is 1 clock cycle. We also assume that each node require 20 clock cycles to apply the routing algorithm in the baseline deterministic routing (with one VC). To consider the increased complexity of adaptive routing, we have assumed a higher routing time. In particular, we have

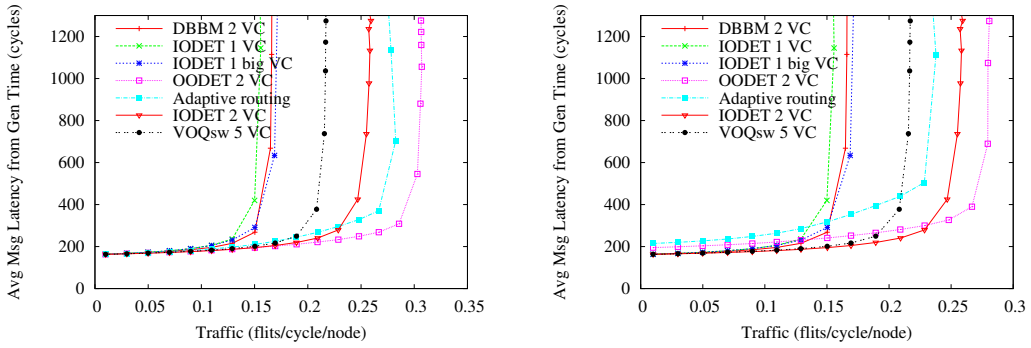


Fig. 2
AVERAGE PACKET LATENCY VS. ACCEPTED TRAFFIC. 64-NODE 2D-MESH. UNIFORM TRAFFIC. ROUTING TIMES ARE SCALED IN (B).

used Chien's model ([16], [17]). This model states that routing time depends on the degree of freedom (i.e., the number of routing options) of the routing algorithm. It is the sum of a constant time (i.e. the one required to apply the routing function) plus a component that grows logarithmically with the number of routing options (i.e. the selection function delay). If this fact is taken into account, OODET and adaptive routing would take more clock cycles to route a packet than IODET, DBBM, and VOQsw.

A. Performance analysis

First, we evaluate the mesh topology. Figure 2.(a) shows results for a 8 × 8 mesh. IODET 1 VC is the baseline deterministic routing algorithm. As expected, adaptive routing doubles network performance over the baseline deterministic routing algorithm. However, this is not a completely fair comparison since adaptive routing uses two VCs. The deterministic routing algorithm with double buffer space (referred to as *big VC* in the figures) improves throughput by 11% compared with deterministic routing with one VC, but it is not even near to adaptive routing. Therefore, resources have to be used in a smarter way in deterministic routing to significantly increase its performance.

In particular, IODET 2VC dramatically increases network throughput, obtaining a 91% of the one achieved by adaptive routing. Notice that DBBM does not improve very much the performance of deterministic routing. The reason is that, as it always uses the modulo of the whole packet destination identifier, packets may not be classified in all dimensions. In fact, in some dimensions, packets may use the same VC, wasting the other VCs. In this case, for each column of the second dimension, only one VC is used. In the case of OODET 2VC, it even gets better results than adaptive routing (9% higher throughput). This behavior is due to the fact that, as meshes are not regular, adaptive routing algorithms tend to concentrate traffic in the central part of the network, thus reducing channel utilization of the borders of the mesh. Finally, VOQsw requires 5 VCs, but in spite of using 5 VCs, VOQsw obtains only a 77,35% of the adaptive routing throughput.

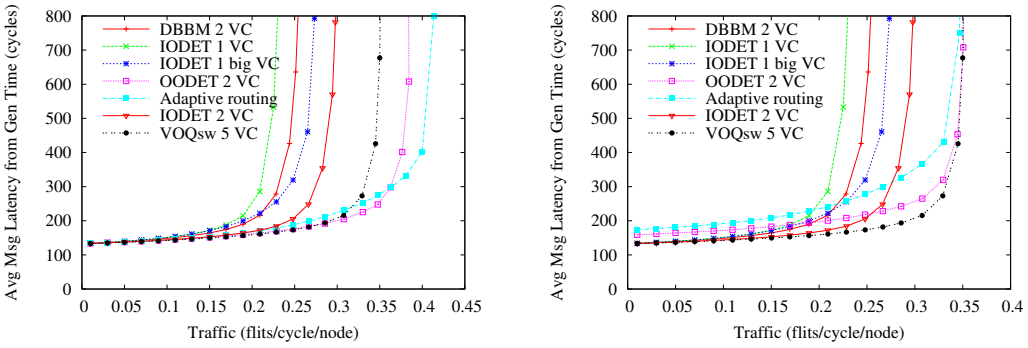
As stated above, providing several routing options and selecting one of them increases the complexity of the router and, hence, its delay. To take this into account, we have use a simple model ([16], [17]) to estimate the differences in routing delays of the different algorithms. This model assumes that routing delay depends on the degree of freedom (i.e., the number of routing options) of the routing algorithm. Taken this into account, adaptive routing would take more clock cycles to complete the routing of a packet than OODET, and OODET more than IODET, VOQsw and DBDM. The more options we have in the routing function, the more clock cycles required for routing. Figure 2.(b) presents the same results as Figure 2.(a) but now scaling the routing time as shown in Table I, computed according to the model (IODET, VOQsw and DBBM are not shown since they have the base delay of 20 cycles). As it can be seen, adaptive routing and OODET 2 VC are no longer as good as they were. Indeed, IODET 2 VC obtains a 9% better throughput than adaptive routing.

The behavior of tori is very different when compared to meshes. Figure 3.(a) clearly shows that adaptive routing is better than any deterministic routing, including OODET 2VC. Concerning the IODET 2VC algorithm, it improves performance over the baseline deterministic routing, but it is far from achieving the throughput of adaptive or OODET 2VC routing algorithms. If we scale the routing delay accordingly to the routing complexity we obtain the results shown in Figure 3.(b). In this case, the difference between deterministic OODET 2VC and adaptive routing decreases. In fact they obtain roughly the same throughput. The IODET 2VC routing algorithm proposed in this paper still obtains a lower throughput, but the difference is narrowed. Notice, though, that it is able to improve the throughput of the baseline deterministic routing algorithm by 40%, and also remember that it does not introduce out-of-order delivery of packets. Indeed, it is the best of the deterministic routing algorithms but VOQsw, that requires 5VCs, which increases switch cost as discussed in Section IV-B. It is worth to highlight that the higher routing delays

TABLE I

COMPARISON OF ROUTING TIMES (IN CYCLES) FOR ADAPTIVE AND OODET SWITCHES.

Dimensions	Virtual channels	OODET	Adaptive	Dimensions	Virtual channels	OODET	Adaptive
2	2	25	28	2	6	33	38
3	2	25	30	3	6	33	40
4	2	25	32	4	6	33	42
6	2	25	34	6	6	33	45
2	4	30	34	2	8	35	40
3	4	30	37	3	8	35	43
4	4	30	39	4	8	35	45
6	4	30	42	6	8	35	48



(a) (b)

Fig. 3

AVERAGE PACKET LATENCY VS. ACCEPTED TRAFFIC. 64-NODE 2D-TORUS. UNIFORM TRAFFIC. ROUTING TIMES ARE SCALED IN (B).

of both adaptive routing and OODET 2VC lead to an increase of network latency.

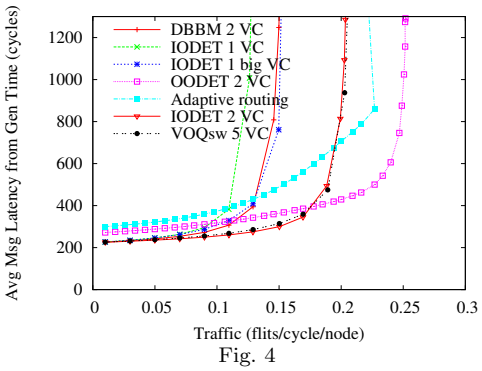


Fig. 4

AVERAGE PACKET LATENCY VS. ACCEPTED TRAFFIC. 256-NODE 2D-TORUS. UNIFORM TRAFFIC WITH SCALED ROUTING TIMES.

We have also analyzed the impact of increasing the number of nodes per dimension (Figure 4). As we increase k, the average distance traversed by packets also increases. Therefore, as expected, all routing algorithms reduce its performance. The best results are obtained by the OODET 2 VC routing algorithm. IODET 2V obtains performance results that are halfway between the baseline deterministic routing and OODET 2 VC.

A.1 Impact of the Number of VCs

In this section we analyze how the number of VCs impacts the different routing configurations. Figure 5 shows results for a 256-node 2D torus with 4, 6 and 8 VCs considering the impact of the complexity of each routing algorithm on the routing delay. As it can be seen, as the number of VCs is increased, the throughput of the deterministic routing proposed in this paper, IODET, is progressively increased, being able of outperforming all other configurations with 8 VCs. Notice that using more than 8 VCs in this network makes no sense.

Most important, message latency is strongly reduced (almost halved for low traffic rates and reduced to one third for high traffic rates and 8 VCs). The explanation of this behavior is twofold. On one hand, it is due to the reduced routing delay of IODET. On the other hand, IODET has been specially designed to reduce the HoL-blocking effect, classifying packets when routing them and according to the component of the destination in the current dimension. In fact, for the analyzed network size ($k = 16$), 8 VCs allow packets to be classified without interference. Indeed, using more than 8 VCs in this network makes no sense. Remember that the maximum distance in any dimension of a k-ary n-cube is $\frac{k}{2}$, 8 in our network. Therefore, there are not more than 8 different destinations in each dimension. In general, the more VCs, the better the classification of packets. About DBBM, we can see that it improves performance when the number of VCs goes up to 4 or 8, but this improvement is lower than the one shown by IODET. Counterintuitively, it works quite well with 6 VCs. The explanation is that the modulo operation it applies affects not only one but different components of the node identifier, and, thus, traffic is classified not only in dimension 0. In general, we have seen that DBBM works bet-

ter when the number of VCs is not a power of two. However, the implementation of the modulo operation by non-power of two numbers is more complex. In fact, with power of two numbers, it is as simple as performing a bitwise and operation.

B. Cost Analysis

This section estimates the cost measured in switching elements per switch for the different configurations. In order to perform such estimation, we compare the number of connections required in the crossbar for each configuration. Although a full crossbar (i.e. with a number of ports equal to the product of the number of physical channels by the number of VCs) is able to cope with any of the analyzed routing algorithms, some connections are not actually required, which may lead to simplify it. For instance, with DOR, packets may only be forwarded to dimensions higher than the one they enter the router. As a consequence, the output multiplexers of the crossbar corresponding to higher dimensions will have more inputs (and hence, more switching elements) than the ones located in the lower dimensions' channels. On the other hand, with adaptive routing, a packet entering through a channel may be forwarded to any other output channel. When several VCs are used, the number of options grows considerably. In addition, the injection and ejection channels must be also taken into account.

We assume that a $n-$input multiplexer needs n switching elements. Notice that it depends on the number of dimensions and number of VCs. Table II shows the number of required switching elements for adaptive routing and two deterministic routing configurations considered in this paper, OODET and IODET. We have not included DBBM due to the complexity of implementing the modulo operator by non-power of two numbers. Notice that VOQsw needs the same number of switching elements as OODET if the same number of VCs are used, because in a given dimension a packet in VOQsw can change the VC at each hop, like OODET. As it can be seen in the table, IODET requires a crossbar with fewer connections than the other configurations. That is, it requires not only a cheaper but also a simpler crossbar which may also help in reducing switch delay (not considered in this paper). This is due to the fact that messages traversing a given dimension cannot return to the previous dimensions, since DOR routing is used. Therefore, we could dispose those switching elements that connect with lower dimensions. In IODET, in addition to using DOR, the VC used by a given packet does not change along the path in the same dimension. Hence, there is no need to have a crossbar connection (and the corresponding switching elements) to allow packets to change from this VC to other VCs of the same dimension. Opposite to that, in OODET, the crossbar would require those connections since packets are allowed to change from one VC to other in the same dimension. Finally, adaptive routing would require a crossbar that allows almost all combinations of physical and VCs (the only exception are connections related to escape paths, which must be traversed in order).

In the table, we can see that, as the number of VCs is increased, the number of switching elements required in the switches further increases, specially for adaptive routing with a high number of dimensions, which more than doubles the required number of switching elements over IODET.

C. Performance/Cost Comparison

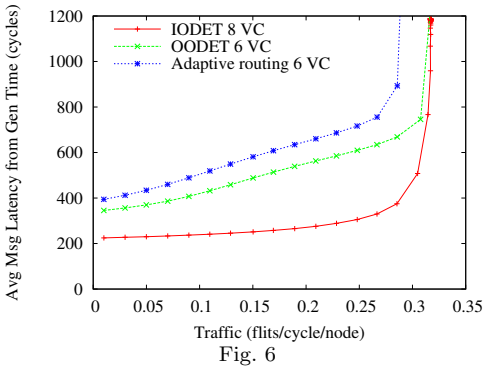


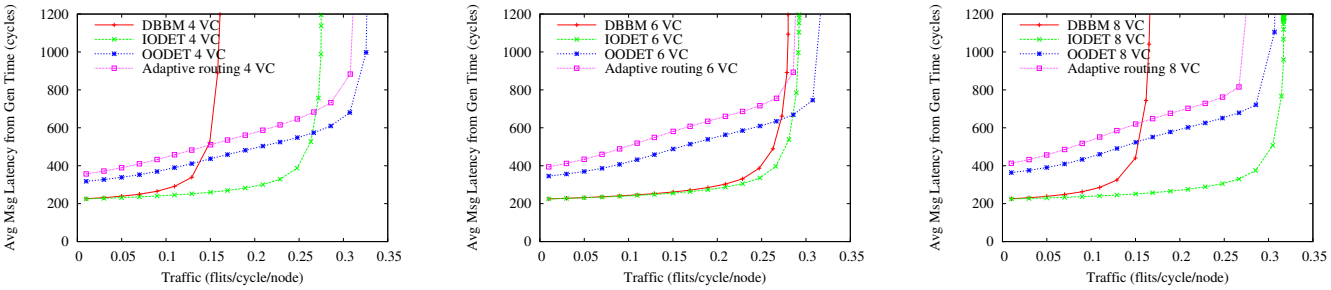
Fig. 6

Average packet latency vs. accepted traffic. 256-node 2D-torus. Uniform traffic. Different #VCs.

As stated in Section IV-A.1, as we increase the number of VCs, IODET becomes the best routing configuration. However, taking into account that it also has a lower cost in terms of number of switching elements, we could compare configurations of similar complexity. This means adding more VCs to IODET to have a cost similar to the other configurations (OODET and adaptive). For example, in Figure 5.(b) we obtained that the IODET with 6 VCs achieved a similar performance as adaptive routing with 6 VCs, but worse than OODET with 6 VCs. In this case, as we can see in Table II, IODET uses switches with 216 switching elements, adaptive routing uses 480 and OODET uses 336. If we use switches with 8 VCs in IODET, it requires 352 switching elements per switch, which is a similar cost as OODET with 6 VCs, and it is still smaller than the one of adaptive routing. Indeed, as stated above, using a number of VCs that is a power of two is mandatory to get a simple implementation of the modulo operation. Taking this into account, a fairer comparison among all the analyzed alternatives is shown in Figure 6, where IODET 8VC obtains the best absolute results both in terms of throughput and latency. Moreover, it must be taken into account that, contrary to the other ones, IODET routing algorithm provides in-order delivery of packets by design.

V. Conclusions

This paper proposes a new HoL-blocking-aware deterministic routing algorithm (IODET) for direct regular topologies. It uses virtual channel flow-control, and assigns packets to VCs according to a subset of bits of the destination identifier (i.e., the component that corresponds to the dimension the packet is traversing). The result is a deterministic



(a) (b) (c)

Fig. 5

Average packet latency vs. accepted traffic. 256-node 2D-torus. Uniform traffic. (a) 4 VCs, (b) 6 VCs and (c) 8 VCs.

TABLE II

Comparison of the number of switching elements

Dimensions	Virtual channels	OODET	Adaptive	IODET	Dimensions	Virtual channels	OODET	Adaptive	IODET
2	2	48	64	40	2	6	336	480	216
3	2	96	144	84	3	6	720	1152	540
4	2	160	256	144	4	6	1248	2112	1008
6	2	336	576	312	6	6	2736	4896	2376
2	4	160	224	112	2	8	576	832	352
3	4	336	528	264	3	8	1248	2016	912
4	4	576	960	480	4	8	2176	3712	1728
6	4	1248	2208	1104	6	8	4800	8640	4128

routing algorithm which exhibits its well-known advantages over adaptive routing: simplicity, low routing times and in-order delivery of packets. If routing times are scaled according to the number of routing options of each routing algorithm, IODET is able to outperform adaptive routing in meshes, while it reaches a performance half-way between the baseline deterministic and adaptive routing algorithms in torus. In addition, IODET also simplifies switch design. This combination of a moderate improvement in performance with a simple implementation makes IODET an interesting alternative to consider when selecting the routing algorithm.

Acknowledgment

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and Plan E funds, under Grants TIN2009-14475-C04-01 and TIN2012-38341-C04-01, and by Ayudas para Primeros Proyectos de Investigación from Universitat Politècnica de València under grant ref. 2370.

References

[1] J. Duato, S. Yalamanchili, and N. Lionel, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Publishers Inc., USA, 2002.

[2] W.J. Dally and B. Towles, *Principles and practices of interconnection networks*, Morgan Kaufmann, 2004.

[3] "TOP500 supercomputer site," `http://www.top500.org`.

[4] J. C. Martínez, J. Flich, A. Robles, P. López, and J. Duato, "In-order packet delivery in interconnection networks using adaptive routing," in *IEEE International Parallel and Distributed Processing Symp*, 2005.

[5] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato, "Deterministic versus adaptive routing in fat-trees," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, march 2007, pp. 1 –8.

[6] Xuan-Yi Lin, Yeh-Ching Chung, and Tai-Yi Huang, "A multiple lid routing scheme for fat-tree-based infiniband networks," in *IPDPS*, 2004.

[7] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *Communications, IEEE Transactions on*, vol. 35, no. 12, pp. 1347 – 1356, dec 1987.

[8] J.C.R. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *Networking, IEEE/ACM Transactions on*, vol. 7, no. 6, pp. 789 –798, dec 1999.

[9] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100 in *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, mar 1996, vol. 1, pp. 296 –302 vol.1.

[10] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *Computers, IEEE Transactions on*, vol. C-36, no. 5, pp. 547 –553, may 1987.

[11] V. Puente, R. Beivide, J.A. Gregorio, J.M. Prellezo, J. Duato, and C. Izu, "Adaptive bubble router: a design to improve performance in torus networks," in *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, 1999, pp. 58 –67.

[12] J. Duato, "A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 841–854, 1996.

[13] T. Nachiondo, J. Flich, and J. Duato, "Buffer management strategies to reduce hol blocking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 739–753, 2010.

[14] L. Dennison W.J. Dally, P. Carvey, "Architecture of the avici terabit switch/router," *in:Proceedings of Hot Interconnects*, vol. 6, Aug. 1998.

[15] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, Nov. 1993.

[16] A.A. Chein, "A cost and speed model for k-ary n-cube wormhole routers," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 2, pp. 150 –162, feb 1998.

[17] J. Duato and P. López, "Performance evaluation of adaptive routing algorithms for k-ary n-cubes," in *Parallel Computer Routing and Communication*, Kevin Bolding and Lawrence Snyder, Eds., vol. 853 of *Lecture Notes in Computer Science*, pp. 45–59. Springer Berlin, Heidelberg, 1994, 10.1007/3-540-58429-3_27.

Vehicular Networks: embracing wireless heterogeneous communications through Vertical Handover

Johann Marquez-Barja[1] , Carlos T. Calafate, Juan-Carlos Cano, Pietro Manzoni[2] and Luiz DaSilva[3]

Abstract— **By taking advantage of the ever growing deployment of wireless networks, the automotive industry is increasingly enabling vehicles to communicate with one another and with the infrastructure, with benefits to information delivery and safety on the road. Vertical Handovers can be used to ensure that the Quality of Service demanded by applications (*e.g.,* throughput, latency) is met while the vehicle is changing its position. In this paper we present a Vertical Handover Decision Algorithm empowered by the IEEE 802.21 standard and its services. Our proposed algorithm uses the vehicle's on-board unit features and considers the geolocation, the car navigation and a realistic propagation model for heterogeneous underlying networks such as Wi-Fi, WiMAX, and UMTS. Our results demonstrate, through a case of study, that QoS can be guaranteed when location and networking parameters are jointly considered when performing vertical handover.**

Keywords— **Vehicular networks, heterogeneous wireless networks, geolocation, IEEE 802.21, Wi-Fi, WiMAX, UMTS, ns-2, GPS.**

I. Introduction

NOWADAYS, cars are being empowered by advanced On-Board Units (OBUs), which are enhanced by features such as faster processors, high definition displays, and multiple wireless networking technologies, offering full connectivity inside and outside of the car in order to deliver and to access different content (*e.g.,* safety messages, advertisements, or a movie) while the car is moving. Thus, these units offer not only voice services, but also infotainment.

Due to wireless impairments and to mobility issues, the Quality of Service (QoS) can be affected or the link may be lost while crossing different coverage areas, whether the areas are covered by the same technology or not. To be able to switch from one network technology to another without affecting the QoS (*e.g.,* bandwidth, packet latency), Vertical Handover (VHO) techniques are required. The IEEE 802.21 protocol [1] has been developed to improve VHO processes, by offering a homogeneous middleware that can be accessed by applications to communicate with heterogeneous network interfaces, contributing to simplify the complexity of the multi-interface management. Through this middleware, different services can be accessed in order to obtain context information (such as Point of Attachment (PoA) information and geolocation, local information, network information) as well as to interact and perform different actions on the interfaces, based on such information.

Concerning the Vehicular Networks (VNs) context and VHO processes involved, many mobility and location issues must be considered. The intrinsic characteristics of the VNs, such as dynamism, speed, and intensely changing contexts, present a challenge for VHO. Other features of the VNs, such as availability of geolocation through the Global Positioning System (GPS), and the lack of power restrictions due to the continuous energy source, allow the devices to improve the gathering of context information in order to perform an accurate decision on choosing the most suitable candidate network to hand over to, thus improving the VHO process.

In this work, we present a Vertical Handover Decision Algorithm (VHDA) which combines GPS information (both geolocation and navigation), underlying network information (based on realistic propagation models), as well as network architecture information, in order to optimize the network selection process, a critical element of VHO.

II. Related Work

One of the first approaches using GPS systems for improving handover was presented by Dutta *et al.* in [2]; the authors present a methodology related to GPS-IP discovery for intra-technology handovers considering layer 2 detection, IP address assignment, and duplicate address detection. Yliant-tila *et al.* [3] present work based on GPS support for inter-technology handovers considering Wireless Fidelity (Wi-Fi) and Universal Mobile Telecommunications System (UMTS) technologies. The authors propose a location-aware architecture to perform vertical handovers based on location management and resource allocation taking into account the mobility in and out of the cells.

A method to enhance link and network layers handover by predicting the future position via GPS was presented by Montavont *et al.* in [4]. The authors combine Wi-Fi and GPS information in order to perform the network selection process. Concerning Worldwide interoperability for Microwave Access (WiMAX) technologies, Hsiao *et al.* in [5], make use of a combination of GPS information and WiMAX interface status information to avoid scanning the

[1]CTVR / the telecommunications research centre, Trinity College Dublin, Ireland. e-mail: marquejm@tcd.ie
[2]Universitat Politècnica de València, Spain. e-mail: {calafate, jucano, pmanzoni}@disca.upv.es.
[3]CTVR / the telecommunications research centre, Trinity College Dublin, Ireland and Virginia Tech, USA. e-mail: dasilval@tcd.ie

channel and to pre-connect to the PoA, hoping to provide stable WiMAX service.

III. Proposed VHDA

The proposed algorithm takes advantage of the car's features, such as powerful OBUs, GPS (geolocation and navigation information), different networking interfaces, and context information provided by the IEEE 802.21 standard. However, to design an accurate VHDA able to perform the handoff not only considering the most adequate candidate network to switch to, but also considering the time to leave the current PoA and join the target one, we must estimate the packet loss conditions associated with the different networks at different distances between vehicle and PoA. To obtain a valid network conditions model we have performed several measurements within the Polytechnic University of Valencia campus and the University of Murcia campus, observing real Wi-Fi and WiMAX behavior [6], respectively.

The proposed algorithm is composed of three tasks: Networking, Neighborhooding, and Decision-making tasks. Figure 1 presents the flow diagram of the algorithm.

The wireless network sensing process is performed in the networking task. It periodically sends and receives information about the network status (*e.g.,* Router Advertisement (RA) and Router Solicitation (RS)). The IEEE 802.21 services, *i.e.,* Media Independent Event Service (MIES) and Media Independent Command Service (MICS), check the link status and the reports received, notifying the upper layers for taking further actions.

Regarding to the Neighborhooding task, there are two data storage elements that collect the surrounding information related to the Current Neighborhood and the Future Neighborhood. Both storage elements are periodically filled-in with information about the current and future PoAs available, respectively. By consulting the GPS module, the current and future geolocation within the map and route (navigation information) is stored and used in order to access the PoA information database, powered and made available by the Media Independent Information Service (MIIS) of IEEE 802.21. A list of current and future available PoAs is retrieved and locally stored at the OBU to be used by the decision-making tasks. Based on the MIIS information, this task also calculates the useful coverage time under each PoA coverage by combining the GPS information about the route on the map and the MIIS information. The useful coverage time is affected by different issues, such as how tangentially the route crosses the coverage area, the times for reaching/leaving a coverage area, the existence of overlapping coverage areas along the path, and the cell coverage at a given target QoS, which we refer to as the 'QoS border' considered by the decision-making task, as shown in Figure 2.

Finally, the selection of the target network is made by the decision-making task. This process is in charge of evaluating all the gathered information and, based on a Multiple Criteria Decision-Making (MCDM)-based evaluation [7], the candidate PoA which best fits the application's requirements is chosen. The main decision logic of our proposed algorithm, whose aim is to guarantee the QoS, considers the cell coverage time and guaranteed QoS border of the cell, and allows handovers to take place only when there is alternative useful coverage available, considering also the distance to the QoS cell border of the PoA involved in the decision process. As shown in Figure 2, when a vehicle arrives to the coverage area A, the wireless network interface triggers a Link Detected event, starting the VHDA process. If we based the decision on the Time Coverage A, considering neither the immediate future nor the QoS border, we could make a mistaken decision, since the vehicle will soon leave the coverage area A to join the coverage area B, and so the VHO would be worthless.

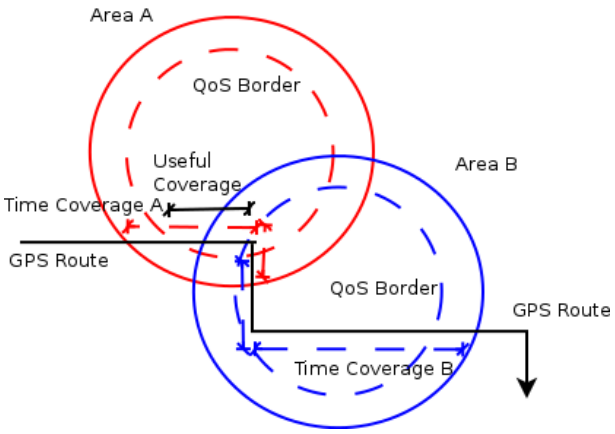


Fig. 2. Usefull coverage and QoS example.

To complete the whole VHO process, different notification-update processes are triggered at the server to handle the mobility issues by using Mobility support for Internet Protocol v.6 (MIPv6), OBU's wireless interfaces, and networking elements, thereby redirecting the flows and keeping the connection alive.

IV. Simulation Scenario

For our simulations, we have used a well-known simulator within the wireless networks area: the Network Simulator (ns-2) [8]. Moreover, we have used the *NIST mobility package for ns-2* [9], in conjunction with EURANE [10], taking advantage of the many capabilities and features offered to simulate Wi-Fi, WiMAX, and UMTS technologies, and to perform VHO among them. Furthermore, the NIST add-on also enables the MIES and MICS services of the IEEE 802.21 standard to interact with heterogeneous network interfaces under homogeneous standard primitives. Concerning the Media Independent Information Service (MIIS), we have used our version previously developed in [11], as well as the Global Positioning System (GPS) add-on module for the ns-2 presented in [6] for geolocation capabilities.

In our simulation, we have modelled vehicles mov-

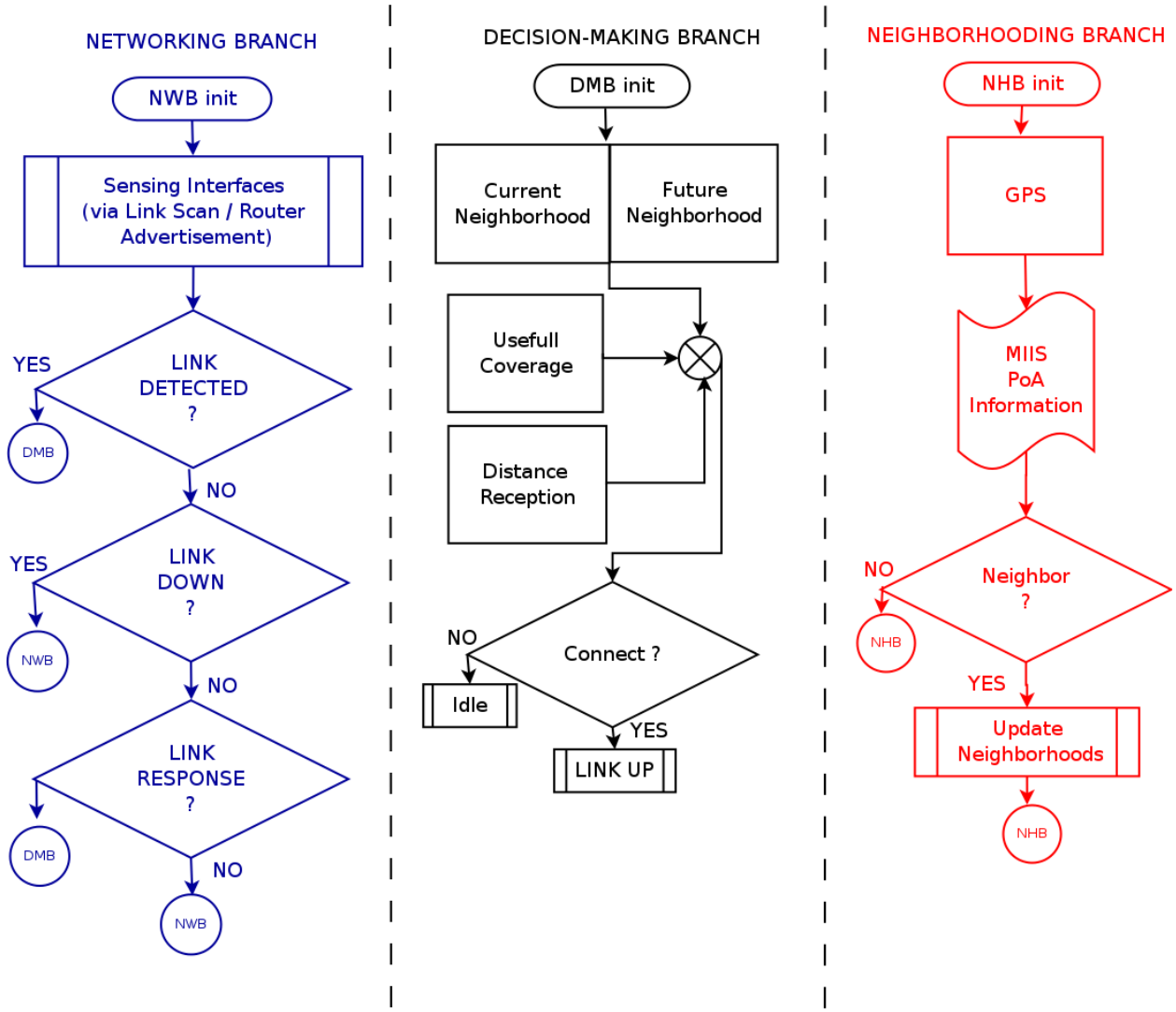


Fig. 1. Proposed VHDA algorithm.

ing at 32 Km/h from Universitat de Valencia Campus to Universitat Politècnica de Valencia Campus in the city of Valencia, Spain. Figure 3 shows the route from one place to another, involving a distance of 5.5 km in a 3.75 km^2 area. In this context we deployed 1 UMTS, 8 Wi-Fi, and 3 WiMAX PoAs covering different areas and with heterogeneous capacity, as illustrated in figure 4. It is important to point out that UMTS covers the whole area, meaning that the UMTS technology is always the backup connectivity technology for this set of experiments. Table I presents the main configuration parameters for the experiments. In the experiments, the OBU requests a 1.48 Mbps Constant Bit Rate (CBR) video traffic stream.

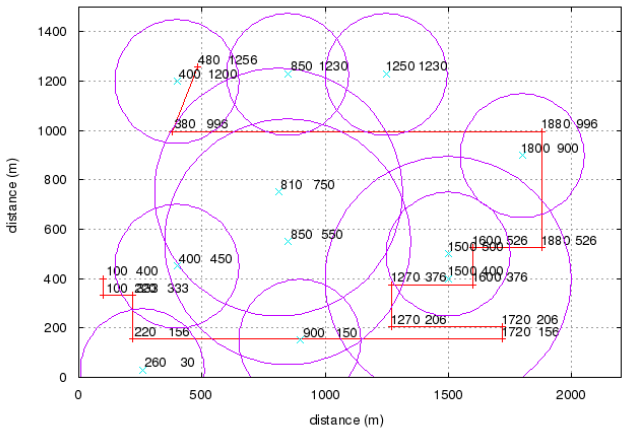


Fig. 4. Coverage grid.

V. Performance evaluation

To evaluate the performance of the proposed VHDA, we have also compared it against two different VHDAs. We briefly describe the main decision-making process of those alternative algorithms:

- **Tech-Aware VHDA**. It takes into account only the theoretical bandwidth offered by the underlying technologies. So, whenever the car finds a new coverage area of a technology with higher theoretical bandwidth, the VHDA connects to the new PoA. It makes use of two IEEE 802.21 services: MICS and MIES.
- **Cell coverage-based VHDA** . It uses the car's current and future geolocation, neighbor-

TABLE I
SCENARIO COMPONENTS.

Component	Wi-Fi	WiMAX	UMTS
Access Point	8	3	1
Theoretical Bw (Mbps)	54	70	5
Bw offered (Mbps)	28.2	16.3	2.7
VHO latency (ms) [12], [13]	1080	2665	-
Advertisement Interval (ms)	100	5000	-
Coverage (m)	500	1000	5000

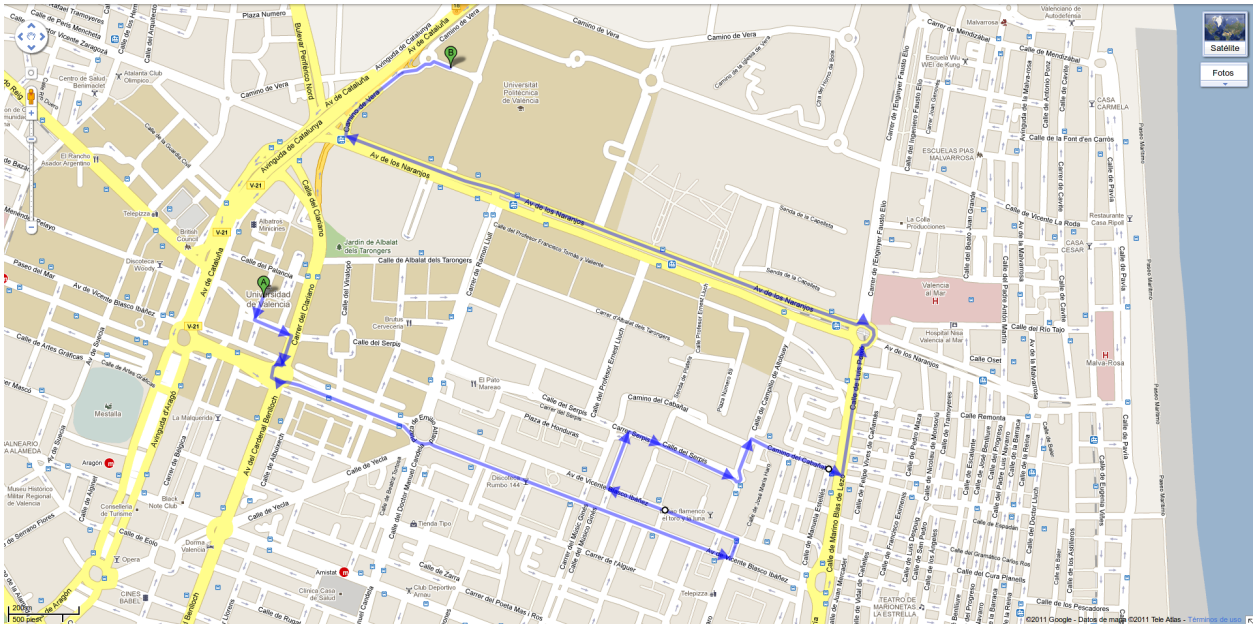


Fig. 3. Map layout.

hooding, and the three IEEE 802.21 services: MICS, MIES and MIIS. It considers the useful coverage time depending on the PoA coverage area and the route calculated by the GPS.

The handovers performed by the evaluated VHDAs are presented in Figure 5. As we can observe, Figure 5(a) shows the coverage resulting from the Tech-Aware VHDA and Cell coverage-based VHDA. Tech-Aware VHDA performed up to 18 VHO events due to its decision-making policy. Cell coverage-based VHDA performed about 15 VHO events, being more selective when switching from one network to another. In Figure 5(b), our proposed algorithm behaves in the same manner, performing up to 11 handovers, whether the minimum packet delivery threshold is set to 40% or 60%. The main difference consists on the geolocation (QoS border) where the handover has occurred.

Figure 6 presents the technology use dwell-time per VHDA. It summarizes the interfaces connectivity depending on the decision-making by the VHDA.

We have also evaluated the mean throughput obtained by each algorithm. Table II shows that the throughput per technology is also increased when applying more sophisticated VHDAs; as switching to a

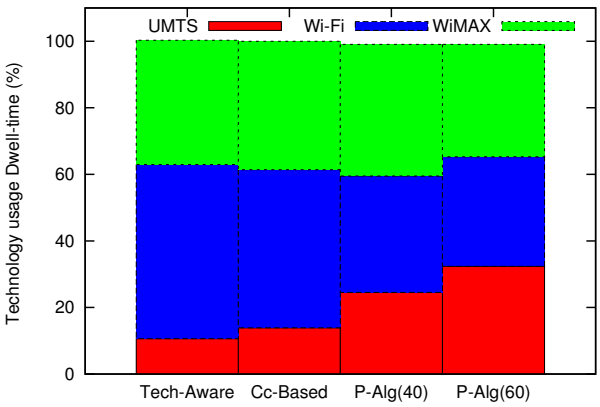
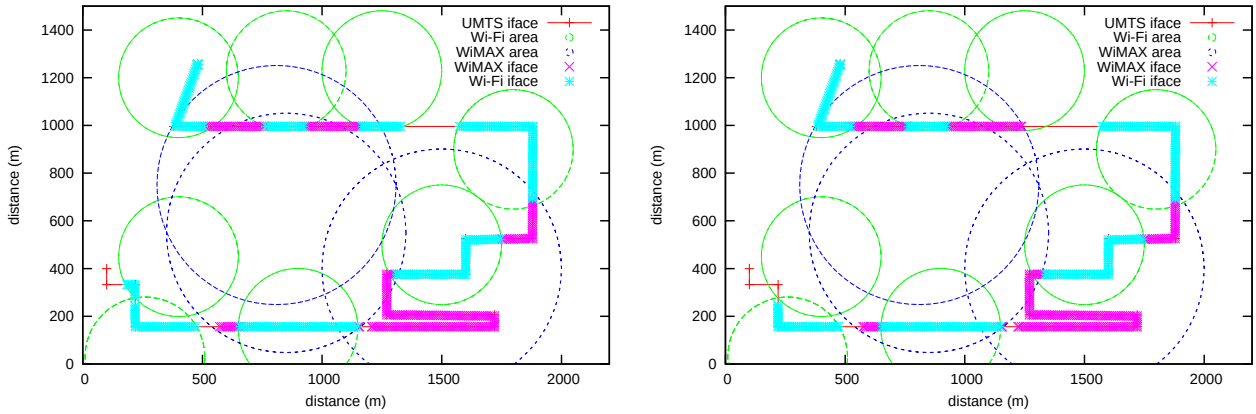


Fig. 6. Dwell-time comparison

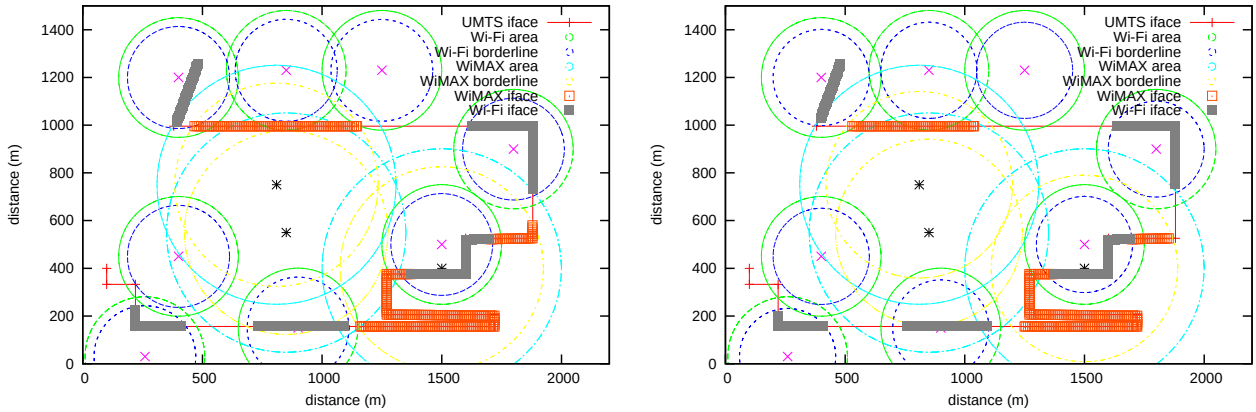
new network only occurs when the QoS is guaranteed by taking into account the QoS border, as per our proposed algorithm.

VI. CONCLUSIONS

We have presented and evaluated a Vertical Handover Decision Algorithm that allows heterogeneous wireless vehicular communications. The algorithm considers the geolocation and navigation information and network context to guarantee the QoS of



(a) Tech-Aware and Cell coverage-based



(b) The proposed algorithm under (40%) and (60%) minimum packet delivery thresholds

Fig. 5. VHDAs Handover comparison

TABLE II
MEAN THROUGHPUT COMPARISON (MBPS)

Technologies	Tech-aware	Cell coverage-based	Proposed algorithm 40%	Proposed algorithm 60%
Wi-Fi	1.1436	1.239	1.446	1.441
WiMAX	1.090	1.072	1.181	1.232
UMTS	1.406	1.407	1.411	1.421

the chosen candidate by taking into account realistic propagation models for underlying wireless networks such as Wi-Fi, WiMAX and UMTS, thus improving the correct flow transition from one PoA to another. Through a case of study, we have demonstrated that the proposed algorithm guarantees data flow switches to a similar or better network than the current one used, thus boosting performance in comparison to other solutions.

ACRONYMS

CBR Constant Bit Rate . . . 3
GPS Global Positioning System . . . 1
MCDM Multiple Criteria Decision-Making . . . 2
MICS Media Independent Command Service . . 2
MIES Media Independent Event Service . . . 2
MIIS Media Independent Information Service . 2
MIPv6 Mobility support for Internet Protocol v.6 . . . 2
ns-2 Network Simulator . . . 2
OBU On-Board Unit . . . 1
PoA Point of Attachment . . . 1
QoS Quality of Service . . . 1
RA Router Advertisement . . . 2
RS Router Solicitation . . . 2
UMTS Universal Mobile Telecommunications System . . . 1
VHDA Vertical Handover Decision Algorithm . . 1
VHO Vertical Handover . . . 1
VN Vehicular Network . . . 1
Wi-Fi Wireless Fidelity . . . 1
WiMAX Worldwide interoperability for Microwave Access . . . 1

Acknowledgments

This work has been sponsored by the Ministry of Science and Innovation of Spain through the Walkie-Talkie project (TIN2011-27543-C03-01), and by the Universitat Politècnica de València through the ABATIS project (PAID-05-12).

We also acknowledge support from the Science Foundation Ireland under Grants No. 10/CE/I1853 and 10/IN.1/I3007.

References

[1] "IEEE standard for local and metropolitan area networks- part 21: Media independent handover," Tech. Rep., 2009.

[2] A. Dutta, S. Madhani, W. Chen, O. Altintas, and S. Cai, "GPS-IP based Fast-hanoff for Mobiles," in *3rd New York Metro Area Networking Workshop*, Sept. 2003.

[3] M. Ylianttila, J. Makela, and K. Pahlavan, "Analysis of handoff in a location-aware vertical multi-access network," *Elsevier Computer Networks*, vol. 47, no. 2, pp. 185–201, Feb. 2005.

[4] J. Montavont and T. Noel, "IEEE 802.11 Handovers Assisted by GPS Information," in *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications.*, 2006, pp. 166–172.

[5] W. D. Hsiao, Y. X. Liu, and H. C. Chao, "An intelligent WiMAX mobile network handoff mechanism with GPS consideration," in *International ACM Conference on Mobile Technology, Applications, and Systems*, New York, NY, USA, 2008, ACM.

[6] J. Marquez-Barja, C. T. Calafate, J. C. Cano, and P. Manzoni, "A geolocation-based Vertical Handover Decision Algorithm for Vehicular Networks," in *IEEE 37th Conference on Local Computer Networks (LCN)*, Oct. 2012, pp. 360–367.

[7] E. Stevens-Navarro and V. W. S. Wong, "Comparison between Vertical Handoff Decision Algorithms for Heterogeneous Wireless Networks," in *63rd IEEE Vehicular Technology Conference*, May 2006, vol. 2, pp. 947–951.

[8] K. Fall and K. Varadhan, "ns Notes and Documents.," The VINT Project. UC Berkeley, LBL, USC/ISI, and Xerox PARC, June 2009.

[9] Advanced Network Technology Division- National Institute of Standards and Technology, "Seamless and Secure Mobility," http://www.antd.nist.gov/seamlessandsecure/.

[10] B. V. Ericsson Telecommunicatie, "EURANE - enhanced UMTS radio access network extensions for ns-2," .

[11] J. Marquez-Barja, C. T. Calafate, J. C. Cano, and P. Manzoni, "MACHU: A novel vertical handover algorithm for vehicular environments," in *IEEE Wireless Telecommunications Symposium (WTS 2012)*, Apr. 2012.

[12] S. L. Tsao, Y. L. Chen, and C. H. Chang, "Evaluation of Scan and Association Process for Real-Time Communication in Mobile WiMAX," *IEEE Transactions on Wireless Communications*, vol. 9, no. 11, pp. 3320–3323, Nov. 2010.

[13] S. J. Yoo, D. Cypher, and N. Golmie, "Predictive link trigger mechanism for seamless handovers in heterogeneous wireless networks," *Wirel. Commun. Mob. Comput.*, vol. 9, no. 5, pp. 685–703, 2009.

Broadcast Schemes for Disseminating Safety Messages in VANETs

Julio A. Sanguesa[1], Manuel Fogue[1], Piedad Garrido[1], Francisco J. Martinez[1], Juan-Carlos Cano[2], Carlos T. Calafate[2], y Pietro Manzoni[2]

Abstract—**In Vehicular ad hoc Networks (VANETs), the efficient dissemination of messages is a key factor to speed up the development of useful services and applications. In this paper, we present the Optimal Broadcast Selection algorithm, a novel proposal that automatically chooses the best broadcast scheme trying to fit the warning message delivery policy to the current characteristics of each specific vehicular scenario. Our mechanism uses as input parameters the vehicular density and the topological characteristics of the environment where the vehicles are located, in order to decide which dissemination scheme to use. Simulation results demonstrate the feasibility of our approach, which is able to support more efficient warning message dissemination in vehicular environments.**

Keywords— **Vehicular ad hoc networks, warning message dissemination, adaptive systems.**

I. Introduction

VEHICULAR ad hoc Networks (VANETs) are wireless communication networks supporting cooperative driving among vehicles on the road. Vehicles act as communication nodes and relays, forming dynamic vehicular networks together with other nearby vehicles.

In a VANET, any vehicle detecting an abnormal situation (e.g., accident, slippery road, etc.) rapidly starts notifying the anomaly to nearby vehicles to spread the alert information in a short period of time. Thus, broadcasting warning messages can be useful to alert nearby vehicles. However, this dissemination is strongly affected by: (i) the signal attenuation due to the distance between the sender and receiver (especially in low vehicular density areas), (ii) the effect of obstacles in signal transmission (very usual in urban areas, e.g., due to buildings), and (iii) a reduced message delivery effectiveness due to serious redundancy, contention, and massive packet collisions provoked by simultaneous forwarding, usually known as broadcast storm (prone to occur in highly congested areas) [9]. Therefore, knowing the density of vehicles and the characteristics of the area where the vehicles are moving (e.g., in terms of topological complexity) in a vehicular communications environment is important, as better opportunities for message delivery can show up.

In this paper, we propose an adaptive algorithm that automatically chooses the best dissemination scheme to adapt the warning message delivery policy to each specific scenario. Our mechanism uses as input parameters the vehicular density and the topological characteristics of the environment where the vehicles are located, using them to decide which dissemination scheme to use. The main goal is to maximize the message delivery effectiveness while generating a reduced number of messages and, thus, avoiding or mitigating broadcast storms. In addition, we also propose the Nearest Junction Located (NJL), our novel warning message dissemination scheme specially designed for being used in highly congested urban areas.

The paper is organized as follows: in Section II we introduce our novel NJL scheme, and the optimal broadcast selection algorithm. Section III shows the simulation environment used to validate our proposal. Section IV presents and discusses the obtained results. In Section V we review previous works closely related to our proposal. Finally, Section VI concludes this paper.

II. Selecting the Optimal Broadcast Scheme in VANETs

Over the years, several dissemination schemes have been proposed to address the broadcast storm problem in vehicular networks. Some of the most representative ones are presented in detail below.

A. Broadcast Schemes

- The *Counter-based scheme* [9]. Initially proposed for Mobile Ad Hoc Networks (MANETs), this scheme aims at mitigating broadcast storms by using a threshold C and a counter c to keep track of the number of times a broadcast message is received. Whenever $c \geq C$, rebroadcast is inhibited.
- The *Distance-based scheme* [9]. This scheme accounts for the relative distance d between vehicles to decide whether to rebroadcast or not. When the distance d between two vehicles is short, the additional coverage (AC) area of the new rebroadcast is lower, and so rebroadcasting the warning message is not recommended.
- The *enhanced Street Broadcast Reduction* (eSBR) [4]. This scheme is specially designed to be used in VANETs, taking advantage of the information provided by maps and built-in positioning systems, such as the GPS. Vehicles are only allowed to rebroadcast messages if they are located far from their source ($> d_{min}$), or

[1]Dpto. de Informática e Ingeniería de Computadores, Universidad de Zaragoza, e-mail: {jsanguesa, mfogue, piedad, f.martinez}@unizar.es

[2]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: {jucano, calafate, pmanzoni}@disca.upv.es

if the vehicles are located in different streets, giving access to new areas of the scenario.

- The *enhanced Message Dissemination for Roadmaps* (eMDR) [2]. As an improvement to the eSBR scheme, eMDR increases the efficiency of the system by avoiding multiple forwardings of the same message if nearby vehicles are located in different streets. Specifically, vehicles use the information about the junctions of the roadmap, and only the vehicle closest to the geographic center of the junction, according to the geopositioning system, is allowed to forward the messages received.

B. Nearest Junction Located: our Novel Broadcast Scheme

The eMDR and eSBR schemes proved to be specially effective in sparse urban environments. However, the number of messages produced may become excessive in scenarios with a high vehicle density. To cope with this deficiency, in this paper we proposed a novel dissemination scheme called *Nearest Junction Located* (NJL) that is completely based on the topology of the roadmap, allowing vehicles to rebroadcast a message only if they are the nearest vehicle to the geographical coordinates of any junction obtained from the integrated maps. Although the performance of this algorithm is not optimal in sparse environments, it performs quite well in high-density scenarios where the dominant factor to improve the dissemination process is the position of the vehicles, achieving results similar to those obtained by the eMDR and eSBR schemes, while requiring only a fraction of the messages.

C. Optimal Broadcast Selection Algorithm

During a warning message dissemination process, the most important objective to accomplish consists on informing the highest possible number of vehicles in the shortest time. Hence, a critical metric to be used is the percentage of informed vehicles at different time instants (Inf_T). We propose to measure the percentage of vehicles receiving warning messages after 10, 30, and 120 seconds since the time when the dangerous situation started being notified, providing information about both the speed and completeness of the dissemination process. The first 10 seconds provide a good reference of the dissemination speed, the second period (30 seconds) offers a balance between dissemination speed and the completeness, and the state of the scenario after 120 seconds shows the stationary value when no evolution is observed.

These three values were combined using a weighted average, thereby obtaining a single value representing the efficiency of the dissemination process (P_{inf}). In our results, the weights applied to the values collected during the different time intervals are 0.5 (10 seconds), 0.3 (30 seconds), and 0.2 (120 seconds), respectively, since the stationary values of the different broadcast schemes do not tend to vary significantly, and the most noticeable differences occur during the first seconds of the process.

Another important metric for the dissemination schemes is the number of messages produced (M_{recv}). If the wireless channel is saturated with packets, the high contention and the occurrence of collisions will reduce the performance of the process, producing broadcast storms. Thus, the number of messages must remain as low as possible without compromising the efficiency of the dissemination.

Our Optimal Broadcast Selection Algorithm makes use of these two metrics (P_{inf} and M_{recv}) to select the scheme to be used on each particular situation. Specifically, it works following a three step process, as shown in Algorithm 1:

- *Step 1*: For each considered broadcast scheme, the first metric (P_{inf}) is computed, and the schemes with the highest percentage of informed vehicles are selected. Due to the importance of this metric, only the dissemination schemes with a deviation lower than 10% with respect to the best one are considered for the second step of the algorithm, and they are stored in set $\mathbb{C}$.
- *Step 2*: Considering only the broadcast schemes in $\mathbb{C}$, the scheme producing the lowest number of messages received per vehicle (M_{recv}) is obtained, in order to reduce the probability of broadcast storms, and the percentage variation with respect to this value is computed for each scheme.
- *Step 3*: The optimal scheme will be selected as the one minimizing the deviation with respect to both the maximal P_{inf} and the minimal M_{recv}. Depending on the vehicle density, it may become more important to minimize the number of messages for high densities, and in that case our algorithm varies the degree of importance of the two metrics by using the K value, calculated as follows:

$$K = \frac{100}{density\ of\ vehicles} \tag{1}$$

In particular, we used the value of reference 100 to compute K, since our experiments showed that the differences in terms of informed vehicles decrease noticeably for densities above 100 vehicles/km^2 (see Figure 1), and, hence, a higher weight is assigned to the number of messages received when this density is exceeded.

III. Simulation Environment

Our optimal broadcast selection algorithm was tested using the ns-2 simulator, modified to consider the IEEE 802.11p standard[1]. In terms of the physical layer, the data rate used for packet broadcasting is 6 Mbit/s, as this is the maximum rate for broadcasting in 802.11p.

The simulator was also modified to make use of our *Real Attenuation and Visibility* (RAV) scheme

[1]All these improvements and modifications are available in http://www.grc.upv.es/software/

Algorithm 1: Optimal Broadcast Selection

input : $\mathbb{B}$: set of broadcast schemes
input : $Inf_{10}(b), Inf_{30}(b), Inf_{120}(b)$: percentage of informed nodes after 10, 30, and 120 seconds
input : $M_{recv}(b)$: number of messages received per vehicle
output : $Optimal_{bcast}$: optimal scheme in terms of informed vehicles and messages received

/* Step 1: Maximize percentage of informed vehicles */
forall $b \in \mathbb{B}$ **do**
  $P_{inf}(b) = Inf_{10}(b) \cdot 0.5 + Inf_{30}(b) \cdot 0.3 + Inf_{120}(b) \cdot 0.2$;
$max_{inf} = max(P_{inf}(b)) \ \forall \ b \ \in \mathbb{B}$
$\mathbb{C} = \{\}$
forall $b \in \mathbb{B}$ **do**
  if $(max_{inf} - P_{inf}(b)) < 10\%$ **then** $\mathbb{C} = \mathbb{C} \cup \{b\}$
/* Step 2: Minimize received messages */
$min_{recv} = min(M_{recv}(b)) \ \forall \ b \ \in \mathbb{C}$
forall $b \in \mathbb{C}$ **do**
  $dev_{inf}(b) = max_{inf} - P_{inf}(b)$
  $dev_{recv}(b) = \frac{M_{recv}(b) - min_{recv}}{min_{recv}}$
/* Step 3: Selection of the optimal broadcast scheme */
$Optimal_{bcast} = \arg\min_{b \in \mathbb{C}} (dev_{inf}(b) \cdot K + dev_{recv}(b)) \ \forall \ b \ \in \mathbb{B}$

TABLE I
PARAMETER SETTINGS IN THE SIMULATIONS.

Parameter	Value
roadmaps	Rome, Valencia, Sydney, Amsterdam, Los Angeles, San Francisco, Madrid
number of vehicles per km^2	$[25, 50, 100, 150, 200, 250]$
number of collided vehicles	3
roadmap size	$2000m \times 2000m$
warning message size	$256B$
beacon message size	$512B$
interval between messages	1 second
MAC/PHY	802.11p
radio propagation model	*RAV* [5]
mobility model	*Krauss* [3]
channel bandwidth	$6Mbps$
max. transmission range	$400m$
d_{min} (used in distance-based, eSBR, and eMDR schemes)	$200m$

[5], which proved to increase the level of realism in VANET simulations using real urban roadmaps in the presence of obstacles. As for vehicular mobility, it has been obtained with CityMob for Roadmaps (C4R) [1], a mobility generator able to import maps directly from OpenStreetMaps, and make them available for being used by the ns-2 simulator. All the results represent an average of over 50 repetitions with different random scenarios, obtaining for all of them a degree of confidence of 90%; each simulation run lasted for 120 seconds. Table I shows the parameters used for the simulations.

The roadmaps used in the simulations were selected in order to have different profile scenarios (i.e., with different topology characteristics). Table II shows the main features of the cities simulated. Note that we added a column labeled as *SJ Ratio*, which represents the result of dividing the number of streets between the number of junctions.

IV. Simulation Results

In this work we performed a total of 10,500 experiments. Due to space restrictions, it is not possible to present the results of all of the cities simulated, so in some cases we only included the results obtained for San Francisco and Valencia since, according to our previous work [6], the simulation results obtained in these roadmaps are closer to the average ones.

TABLE II
MAP FEATURES.

Map	Streets	Junctions	SJ Ratio
Rome	1655	1193	1.387
Valencia	2829	2233	1.267
Sydney	872	814	1.071
Amsterdam	1494	1449	1.031
Los Angeles	287	306	0.938
San Francisco	725	818	0.886
Madrid	628	715	0.878

TABLE III
SIMULATION RESULTS IN SAN FRANCISCO AFTER 120 SECONDS.

	25 veh./km^2		250 veh./km^2	
	% inform.	mess./veh.	% inform.	mess./veh.
eSBR	89.9%	345	99.9%	4661
eMDR	89.5%	301	99.9%	4275
NJL	83.9%	174	99.9%	2184

Tables III and IV compare the simulation results after 120 seconds in two different maps (San Francisco and Valencia). The values of the *percentage of informed vehicles* and the *number of messages received per vehicle* are shown. As can be seen, the NJL scheme allows informing about 3-6% less vehicles under low densities (25 veh./km^2) in both maps, but the percentage of informed vehicles when the vehicle density is high is the same. However, the number of messages received per vehicle is reduced by half in all the scenarios tested using the NJL scheme. This makes the NJL scheme specially suitable for scenarios with a high density of vehicles where broadcast storms are prone to occur.

A. Comparison in Terms of Percentage of Informed Vehicles

Figure 1 presents the evolution of the dissemination process in terms of notified vehicles for the maps of San Francisco and Valencia under three different vehicle densities: 25, 100, and 250 vehicles/km^2. It is noticeable how the topology of the area and the number of vehicles are determinant factors affecting the performance of the broadcast scheme. The dissemination process develops faster in every situation when the vehicle density increases. For sparse networks, the counter-based scheme provides the best results in terms of informed vehicles, whereas for densities above 150 vehicles/km^2, the process presents a very similar behavior for all the selected schemes. The exception is the distance-based scheme in the map of Valencia, which proved to be very inefficient due to the high amount of obstacles interfering with the radio signal.

In addition, we corroborated that simple and regular city profiles like San Francisco allow an easier propagation of the radio signal, increasing the number of informed vehicles at a given time. The most restrictive schemes, such as the NJL, require a very high density of vehicles to achieve an efficiency similar to other dissemination schemes.

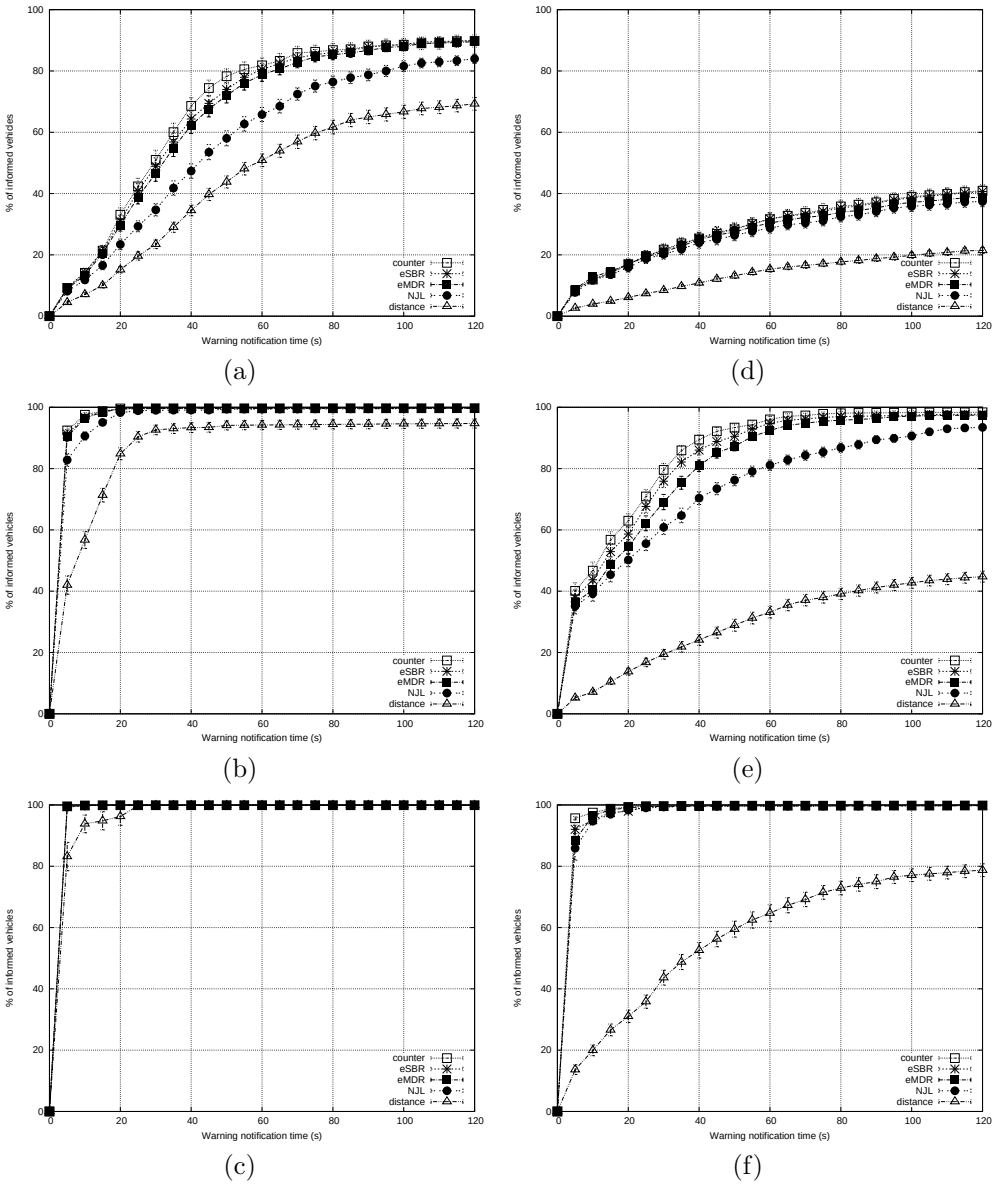


Fig. 1. Percentage of informed vehicles in San Francisco for: (a) 25, (b) 100, and (c) 250 vehicles/km^2, as well as in Valencia for: (d) 25, (e) 100, and (f) 250 vehicles/km^2.

TABLE IV
SIMULATION RESULTS IN VALENCIA AFTER 120 SECONDS.

	25 veh./km^2		**250 veh./km^2**	
	% inform.	mess./veh.	% inform.	mess./veh.
eSBR	40.6%	55	99.7%	3360
eMDR	38.8%	48	99.7%	2451
NJL	37.4%	41	99.7%	1521

B. Comparison in Terms of Messages Received per Vehicle

The number of messages produced by a given dissemination scheme may become very important in VANETs due to the high number messages sent and received by the vehicles involved. This could increase channel contention and the frequency of collisions.

Figure 2 shows the number of messages received per vehicle in two of the maps under study. As shown, the selected dissemination scheme presents a determinant influence over the amount of messages produced; some of them produce only a fraction of the messages required by other schemes. In general, the counter-based scheme produces the highest number of messages, whereas the distance-based is the most restrictive one. The NJL scheme produces the smallest amount of messages of all the schemes which used the information topology of the map to select the forwarding nodes. Again, the features of the map are determinant for the performance of the system. Simple maps allow a faster dissemination at the cost of noticeably increasing the number of messages received per vehicle, thereby increasing the probability of broadcast storms. Thus, more restrictive schemes are recommended for this kind of roadmaps.

C. Optimal Broadcast Scheme Selection

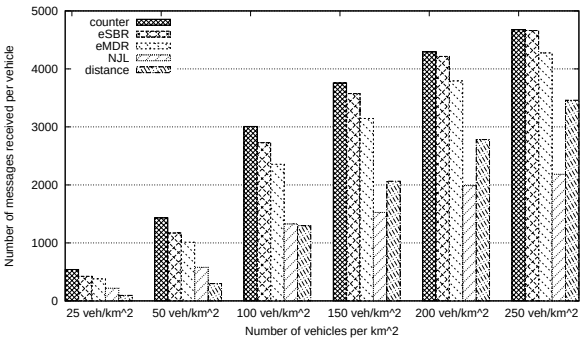
Table V contains an example of the performance of our broadcast scheme selection algorithm presented in Section II-C. Specifically, it shows the results obtained for Valencia when simulating 100 vehicles/km^2. All the values are obtained as the average of 50 repetitions for each configuration. It is noticeable how only three of the available schemes

TABLE V
SIMULATION RESULTS FOR 100 VEHICLES/KM^2 IN VALENCIA.

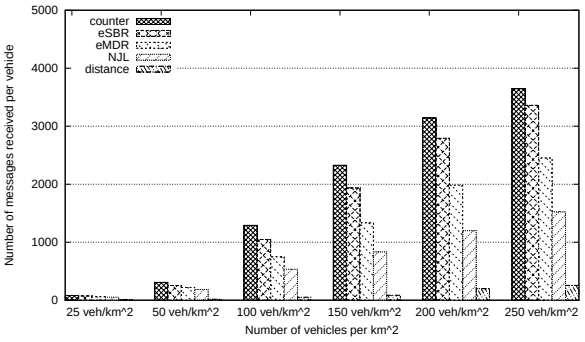
Broadcast	Inf_{10}	Inf_{30}	Inf_{120}	P_{inf}	dev_{inf}	$\mathbb{C}$ (Step 1)	M_{recv}	dev_{recv} (Step 2)	dev_{Tot}	Optimal (Step 3)
Counter	46.6%	79.5%	98.3%	66.81%	**0%**	✔	1196	**77.9%**	**75.55%**	✘
Distance	7.10%	19.4%	44.7%	18.31%	**72.59%**	✘	-	-	-	-
eSBR	43.7%	75.8%	97.7%	64.13%	**4.01%**	✔	940	**39.87%**	**43.89%**	✘
eMDR	40.4%	69%	97.4%	60.38%	**9.62%**	✔	672	**0%**	**9.62%**	✔
NJL	39.2%	60.8%	93.4%	56.52%	**15.4%**	✘	-	-	-	-

TABLE VI
BROADCAST SCHEME SELECTED ACCORDING TO OUR OPTIMAL BROADCAST SELECTION ALGORITHM.

City	SJ Ratio	Vehicle Density (veh./km^2)					
		25	50	100	150	200	250
Rome	1.387	eSBR	eSBR	eSBR	eSBR	NJL	NJL
Valencia	1.267	eMDR	eMDR	eMDR	eMDR	NJL	NJL
Sydney	1.071	eMDR	eMDR	eMDR	NJL	NJL	NJL
Amsterdam	1.031	eMDR	eMDR	NJL	NJL	NJL	NJL
Los Angeles	0.938	eMDR	eMDR	NJL	NJL	NJL	NJL
San Francisco	0.886	eMDR	eMDR	NJL	NJL	NJL	NJL
Madrid	0.878	Counter	eMDR	NJL	NJL	NJL	NJL



(a)



(b)

Fig. 2. Number of messages received per vehicle when varying the broadcast scheme and the vehicular density in: (a) San Francisco and (b) Valencia.

are considered after the first step of the algorithm: i.e., the counter-based, the eSBR, and the eMDR broadcast schemes. Since the eMDR produces the lowest number of messages while maintaining a high percentage of informed vehicles in a small time period, our algorithm considers it as the optimal broadcast scheme for this specific situation.

Table VI shows the selected broadcast scheme for each of the simulated scenarios according to our proposed Optimal Broadcast Selection Algorithm. Notice that the proposed NJL scheme is selected as the optimal one in most cases, especially under high vehicle densities or simple maps with a small SJ ratio, where the radio signal can reach long distances and broadcast storms are prone to occur. On the contrary, eMDR and eSBR schemes offer better results in scenarios where broadcast storms are not a problem, and the main objective is informing as many vehicles as soon as possible.

It is remarkable that almost all the schemes selected by our proposed algorithm rely on topology information to select the most appropriate forwarding vehicle, highlighting the importance of this factor in the warning dissemination process. In fact, broadcast schemes that only make use of the distance between the sender and the receiver, or which only focus on avoiding repeated messages, present a worse trade-off between performance and the amount of messages required. We also observed an anomaly in the results obtained in Table VI corresponding to the map of Madrid. The selected scheme when simulating 25 vehicles/km^2 is the counter-based one, while the overall trend indicates that the chosen one should be the eMDR scheme. This is due to the thresholds selected for Step 1 of the algorithm, where only those schemes with less than 10% variation with respect to the maximum value are considered. The eSBR and eMDR schemes achieve a value of 10.2% and 10.51% variation, respectively, which causes them to be ignored after the first step of the selection algorithm. This indicates that the use of fixed thresholds may lead to inaccurate decisions in some specific cases. We consider that a possible improvement of the broadcast selection algorithm could be using fuzzy logic to decide upon protocol adequacy, thereby avoiding those cases where values close to the threshold are completely ignored.

V. RELATED WORK

In the networking literature we can find several works that present adaptive mechanisms specially designed to enhance message dissemination in vehicular communications. In this section we present

some of the most representative works.

Xue-wen et al. [10] proposed the Transmission Range Adaptive Broadcast (TRAB) algorithm for VANETs. Considering the transmission ranges of vehicles together with the inter-vehicle distances, TRAB calculates the waiting time to select the relay vehicles in accordance with the additional coverage area of adjacent vehicles to ensure that fewer relay vehicles will be used to forward the emergency packets. However, this scheme is designed to obtain efficient propagation of warning messages in highway scenarios alone, making it unsuitable for scenarios with complex topologies where we would want to disseminate warning messages in all directions surrounding the critical area.

Slavik et al. [8] proposed the Rate-Adaptive Broadcast (RAB) protocol for information dissemination in VANETs. RAB adapts to the network conditions, although it does not require any knowledge of network topology. By assuming a VANET dissemination application with fixed periodic updates, RAB is able to use a decision threshold control algorithm based on the rate of both messages. If the new message rate dips below its long-run average, the decision threshold is adjusted to improve message propagation. Otherwise, RAB adjusts the decision threshold to keep the duplicate message rate within an efficient range. Unlike the TRAB scheme, the use of RAB is not restricted to highways; nevertheless, the roadmap layout is not used to select the vehicles to forward the messages.

Schwartz et al. [7] proposed a data dissemination protocol for VANETs that distributes data utility fairly over vehicles while adaptively controlling the network load. The protocol relies only on local knowledge to achieve fairness with concepts of Nash Bargaining from game theory. Simulation results show that their algorithm presents a higher fairness index, and it maintains a high level of bandwidth utilization efficiency compared to other approaches. However, the vehicular density of the scenarios where their proposal was tested was very low (i.e., only 20 $vehicles/km^2$). Additionally, it is not clearly explained if their simulations accounted for the effect of obstacles in wireless signal propagation, and the benefits of their proposal in terms of vehicles informed.

As shown, existing adaptive dissemination techniques for VANETs usually consider features related to vehicles in the scenario, such as their density, speed, and location, to adapt the performance of the dissemination process. However, most of the works in the literature are designed for highway scenarios where messages are only propagated in one direction, or focused on end-to-end routing. Additionally, most of them do not account for the effect of buildings and other obstacles during the dissemination of messages, which may lead to wrong conclusions.

VI. Conclusions

In this paper we proposed an adaptive algorithm that allows selecting the optimal broadcast scheme in a VANET scenario depending on two different metrics: (i) the percentage of informed vehicles, a particularly determinant factor in warning message dissemination, and (ii) the number of messages received by each vehicle, an important factor which indicates the channel contention and the possibility of broadcast storms during the dissemination of alert messages. In addition, we presented a new broadcast scheme called Nearest Junction Located (NJL), which was specially designed for scenarios presenting high vehicular densities or simple topologies, where broadcast storms are prone to occur. The NJL scheme reduces the number of messages received per vehicle without noticeably affecting the percentage of informed vehicles.

VII. Acknowledgments

This work was partially supported by the *Ministerio de Ciencia e Innovación*, Spain, under Grant TIN2011-27543-C03-01, as well as by the Fundación Universitaria Antonio Gargallo and the Obra Social de Ibercaja, under Grant 2013/B010, and by the Government of Aragon and the European Social Fund (T91 Research Group).

Referencias

[1] M. Fogue, P. Garrido, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni. A Realistic Simulation Framework for Vehicular Networks. In *5th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2012), Desenzano, Italy*, pages 37–46, March 2012.

[2] M. Fogue, P. Garrido, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni. Evaluating the impact of a novel message dissemination scheme for vehicular networks using real maps. *Transportation Research Part C: Emerging Technologies*, 25:61–80, December 2012.

[3] S. Krauss, P. Wagner, and C. Gawron. Metastable states in a microscopic model of traffic flow. *Physical Review E*, 55(5):5597–5602, 1997.

[4] F. J. Martinez, M. Fogue, M. Coll, J.-C. Cano, C. T. Calafate, and P. Manzoni. Evaluating the impact of a novel warning message dissemination scheme for VANETs using real city maps. In M. Crovella, L. Feeney, D. Rubenstein, and S. Raghavan, editors, *NETWORKING 2010*, volume 6091 of *Lecture Notes in Computer Science*, pages 265–276. Springer Berlin / Heidelberg, 2010.

[5] F. J. Martinez, M. Fogue, C. K. Toh, J.-C. Cano, C. T. Calafate, and P. Manzoni. Computer simulations of VANETs using realistic city topologies. *Wireless Personal Communications*, 69(2):639–663, 2013.

[6] J. A. Sanguesa, M. Fogue, P. Garrido, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni. An infrastructureless approach to estimate vehicular density in urban environments. *Sensors*, 13(2):2399–2418, 2013.

[7] R. S. Schwartz, A. E. Ohazulike, C. Sommer, H. Scholten, F. Dressler, and P. Havinga. Fair and adaptive data dissemination for traffic information systems. In *IEEE Vehicular Networking Conference (VNC)*, pages 1–8, 2012.

[8] M. Slavik, I. Mahgoub, and M. Alwakeel. Adapting statistical multi-hop wireless broadcast protocol decision thresholds using rate control. In *9th International Conference on High Capacity Optical Networks and Enabling Technologies (HONET)*, pages 32–36, 2012.

[9] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless Networks*, 8:153–167, 2002.

[10] W. Xue-wen, Y. Wei, S. Shi-ming, and W. Hui-bin. A transmission range adaptive broadcast algorithm for vehicular ad hoc networks. In *Proceedings of the 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing - Volume 01*, NSWCTC '10, pages 28–32, Washington, DC, USA, 2010. IEEE Computer Society.

An Evaluation of HEVC using Common Conditions

P. Piñol[1], A. Torres[2], O. López[1], M. Martínez[1], and M.P. Malumbres[1]

Abstract— **A new video coding standard, HEVC (High Efficiency Video Coding), has been recently developed. In this paper we will analyze its performance and how it behaves under packet loss conditions, which is a typical scenario for multimedia transmission over wireless networks. We have modified the reference software in order to make it resistant to data loss and we have measured the impact of these losses at different coding settings. For the sake of comparability we have used the Common Conditions which are oftenly used in the developement process to measure the improvements that new contributions produce in the standard.**

Keywords— **HEVC, packet loss, common conditions, error resilience.**

I. Introduction

EARLY this year (25th January 2013), the Joint Collaborative Team on Video Coding (JCT-VC) at their meeting in Geneva agreed on a new video coding standard, informally known as High Efficiency Video Coding (HEVC)[1][2][3]. JCT-VC is formed by members of ISO/IEC Motion Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG). This new standard will need half the bit rate than the previous video coding standard H.264/AVC to achieve the same video quality. This increase in compression capability will make High Definition (HD) video feasible for low bandwidth connections. HEVC is also capable to work with video formats that go beyond HD resolution, reaching Ultra High Definition (UHD) video, like 4K UHD (2160p) and 8K UHD (4320p). This new formats and target frame rates, that can reach up to 120 frames per second, bring the need of higher compression efficiency.

In this paper we will evaluate the performance of HEVC for different compression settings. We have modified the reference software [4] to make it resistant to data loss in order to evaluate losses in the bit stream. The original reference software crashes when decoding a bitstream with missing parts. But with our modified decoder version we have been able to test how HEVC bevahes under packet loss conditions and how different settings affect to the reconstructed video quality.

For the sake of comparability we have used the Common Conditions [5] that are frequently used to evaluate the proposed improvements introduced by contributions to the standard. By using this common conditions it is easier to compare the results of video quality and video compression efficiency.

[1]Departamento de Física y Arquitectura de Computadores, Universidad Miguel Hernández de Elche, e-mail: {pablop, otoniel, mmrach, mels}@umh.es

[2]Department of Computer Engineering, Universidad Politécnica de Valencia, e-mail: atcortes@batousay.com

II. HEVC and Common Conditions

In [6] the authors provide a good overview of the new HEVC standard. HEVC is similar in several aspects to H.264/AVC but it also introduces some improvements that lead it to its great performance. It follows the same hybrid compression scheme than H.264/AVC, based on motion or intra estimation/compensation, domain transform followed by coefficient scaling and quantization and, at last, entropy coding.

Some of the novelties in this new video coding standard are the following. In H.264/AVC, the unit used for coding samples is the MacroBlock (MB). A MB consists of 16x16 luma samples and its corresponding chroma samples. In HEVC Coding Tree Units (CTU) may contain Coding Tree Blocks (CTB) of 16x16, 32x32 or 64x64 luma samples and their corresponding chroma samples. Bigger sizes usually lead to improvements in compression, especially in large video formats. Intra prediction (prediction of pixels based on previously coded pixels of the same picture) now supports up to 35 modes instead of the 10 modes of H.264/AVC. This can lead to a better estimation and compensation and can produce smaller residuals which will need less bits to be encoded. A new process is now included in the video coding loop, named Sample Adaptive Offset (SAO). SAO is performed after deblocking. Its aim is to reduce distortion by classifying reconstructed pixels into different categories according to their intensity and edge parameters and then applying a different offset for each category. Also a new structure has been introduced in HEVC: the Video Parameter Set (VPS). VPS permits to send information which is common to the entire video sequence. It works in a similar way to Sequence Parameter Sets (SPS) and Picture Parameter Sets (PPS) which are available in both standards. VPS improves compression and can also be used for error resilience (ER) purposes. Another new feature is the coding using Tiles. Tiles are square regions of a picture that can be decoded indenpendently. This new structure has been introduced in the standard with the goal of facilitating parallel processing. Several other refinements are present in the new standard, like improving the entropy encoder CABAC (Context-Adaptive Binary Arithmetic Coder) and also new features like Wavefront Parallel Processing (WPP). For a deeper look into these and other novelties the reader can check out [6].

In [5] the JCT-VC defines the common test con-

ditions and software reference configurations to be used for HEVC experiments. In that paper it can be found a series of settings in order to evaluate HEVC video codec and to compare the different contributions made to it.

The common test conditions specify a number of video sequences for testing HEVC, grouped in several categories. Five of those categories consist of natural video sequences with different picture resolutions, ranging from 416x260 pixels to 2560x1600 pixels. Another category includes video sequences that have synthetic video in part or in their whole. Frame rates of sequences range from 20 frames per second to 60 frames per second and the bit depth is 8 bits (except for two sequences whose bit depth is 10 bits).

There are four compression modes specified. Each of them is best suited for a class of applications. All Intra (AI) mode codes every frame as an I (intra) frame, this is, no motion estimation/compensation is used. This mode can code a video sequence faster than the other three modes but the compression efficiency that motion compensation can achieve is not exploited here at all. In the other hand, errors in the bitstream are not propagated through the following frames so it provides an inherent mechanism of ER. Applications which are concerned about coding time but not about bandwidth can benefit from this coding mode. Also applications which need to treat each frame independently (like non-linear video editing) are target applications for this mode.

There are two coding modes that hugely increase the compression efficiency and at the same time are concerned about decoding time. They are called Low-Delay P (LP) and Low-Delay B (LB). This two modes use temporal prediction but reference pictures can only be chosen from previous pictures (in display order). In these two modes a coded sequence begins with an I frame and then P frames (for LP mode) or B frames (for LB mode) are inserted until the end of the sequence. P (Prediction) frames can use up to one reference frame and B (Biprediction) frames can use up to two reference frames . Every frame is coded and transmitted in display order so the decoder can display a frame just after receiving and decoding it. This two modes cannot deal with errors in the bitstream because an error in a frame will propagate through succesive frames.

There is a fourth coding mode specified in the common conditions: Random Access (RA) mode. In RA mode an I frame (more specifically, it is a CDR (Clean Decoding Refresh) frame) is inserted every (approximately) one second of video, and the rest are coded as B frames. But here, B frames have reference frames that can appear earlier or later (in display order). So coding (and also decoding) order is not the same as displaying order. This means that coding and decoding includes some delay. Coding delay is caused because the encoder has to wait for future frames (in display order) needed as a reference to encode the present frame. Decoding delay is caused because the decoder does not receive coded frames in display order and has to wait until it has received and decoded all the reference frames needed to decode the next frame. The applications that can use RA mode need to be tolerant to a small delay. On the other hand, inserting an I frame periodically allows actions like fast forwarding/reversing or navigating to a certain moment of the sequence. So this mode is appropriate for applications like video streaming of pre-recorded sequences where navigation is very useful and a little delay can be perfectly asumed.

For each of these four modes, two different bit depths are used for internal calculations: 8 bits (Main) and 10 bits (High). So we have a total of 8 different coding settings combining internal bit depths with coding modes. Configuration files are provided within reference software package [4] [7] [8].

In order to plot Rate-Distortion (RD) curves, four QP (Quantization Parameter) values are specified: 22, 27, 32, and 37. Lower QP values produce bit streams with higher bit rates. At each of this points bit rate and PSNR (Peak Signal-to-Noise) are calculated.

III. Experiments

As it was said before, the original reference software is not resistant to data loss. This means that when the decoder receives an incomplete bit stream it crashes. So, before launching any experiment regarding data losses, we first had to modify the HEVC reference decoder to avoid crashing when some data are missing.

For our experiments we have chosen the sequence named *Race Horses* with a format of 832x480 pixels and a frame rate of 30 frames per second. As dividing a frame in several slices will be used in our future work to introduce ER techniques, we have first measured the overhead produced by varying the number of slices per frame (in the absence of losses). We have performed the rest of our experiments by combining the four different modes (AI, LP, LB, RA) with different number of slices per frame (1, 2, 4, 8, 13, 26) and at the four indicated values for QP (22, 27, 32, 37). Every one of these combinations has been tested at six different packet loss rates (1%, 3%, 5%, 7%, 10%, 20%). For the sake of more realistic randomness, for each one of these loss rates, five different seeds have been used and the values obtained for each PSNR value are the mean values of those five results.

We have first measured the PSNR values obtained for every experiment and then we have implemented a simple error concealment (EC) technique in the decoder to see how it improves video quality in the presence of packet losses and then re-launched all the tests. This simple technique consists basically in filling the missing regions of the current frame with the corresponding regions of the last decoded frame.

After these experiments we also tested two new more modes that we call LP_I32 and LB_I32. These two modes are copies of LP and LB modes but in-

serting a CDR frame every 32 frames (in the same way as RA does) but keeping reference frames using past ones (in display order) as LP and LB do.

Taking into consideration Bjontegaard work [9][10] we have plotted our RD curves by using *Y-PSNR* and *log(bitrate)*. And also have computed Bjontegaard-Delta (BD) measurements (like BD-PSNR) by using cubic interpolation.

IV. Results

A. Overhead at different slices per frame

In Table I we can see the overhead introduced for every coding mode (at high and low bit rates) when dividing each frame into 2, 4, 8, 13 and 26 slices per frame with respect to using 1 slice per frame. Results are expressed in % of bit rate increase using BD-rate measurement. As it can be seen, overhead for AI mode keeps within certain limits but overhead for RA, LP and LB modes grows rapidly especially for low bit rate settings, because at low bit rates, slice headers represent proportionally a great amount of data. If we plan to divide frames into slices to use ER techniques which add redundancy to the coded video in order to recover information when data loss occurs, then this overhead may not be negligible.

TABLE I
BD-rate increase in % at different slices per frame with High and Low bitrates for each coding mode.

BD-rate	2sl	4sl	8sl	13sl	26sl
AI (High)	0,32	0,85	1,98	2,33	2,87
AI (Low)	0,74	2,05	4,68	5,54	6,97
RA (High)	0,53	1,72	3,80	4,55	6,03
RA (Low)	1,55	4,53	9,96	12,96	19,09
LP (High)	0,27	0,92	2,08	2,60	3,59
LP (Low)	1,04	3,26	7,31	9,83	15,20
LB (High)	0,22	1,00	2,47	3,00	4,12
LB (Low)	1,29	3,47	7,55	10,17	15,45

B. Evaluating HEVC modes without losses

Figure 1 shows the coding efficiency of each of the 4 coding modes specified in common conditions. As it can be seen, AI mode produces much higher bit rates at a same level of quality than RA, LP and LB modes. The most efficient of these three methods is RA. LP has an increase in bit rate of 8.65% over RA, and LB has an increase in bit rate of only 1.95% over RA. RA saves a 62.71% of bit rate with respect to AI. As it was said before, each mode has different target applications which not only depend on coding efficiency.

C. Evaluating HEVC with losses

Figure 2 shows the behavior of LP mode (equivalently LB mode) under data loss conditions. As this mode has only one I frame (the first one), errors propagate continuously through the entire sequence leading to a completely useless video sequence. It does not matter which the loss rate is, even for only 1% of losses, the quality of the reconstructed video is awful.

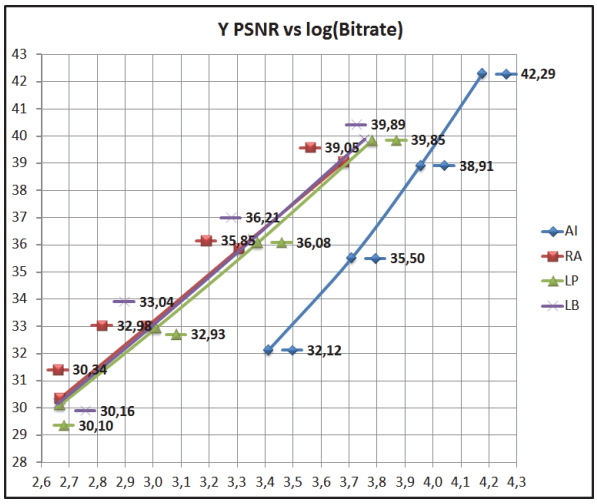


Fig. 1. Compression efficiency of the 4 coding modes.

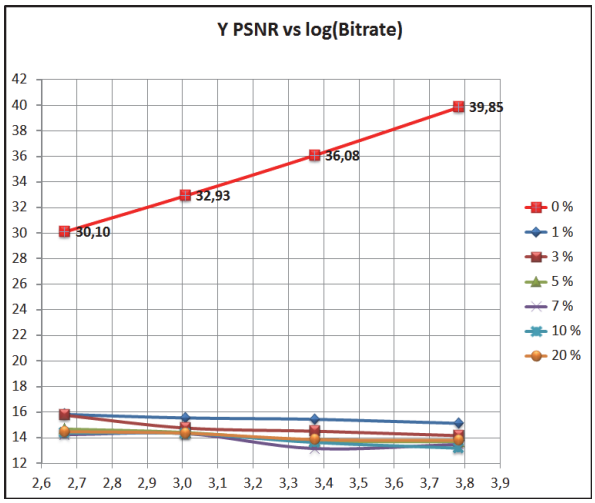


Fig. 2. LP mode at 1 slice per frame and without using EC.

On the opposite situation we find AI mode that inherently has an ER mechanism because no frame uses reference frames, so errors do not propagate at all. In Figure 3 this behavior can be seen for AI mode with 1 slice per frame at different data loss rates. RD curves from 0% to 5% are very close to each other what means that PSNR does not decrease too much.

By observing Figures 3, 4 and 5 a conclusion can be drawn, increasing the number of slices per frame reduces the PSNR value of the reconstructed sequence. Will this conclusion remain valid for sequences reconstructed using our basic EC method?

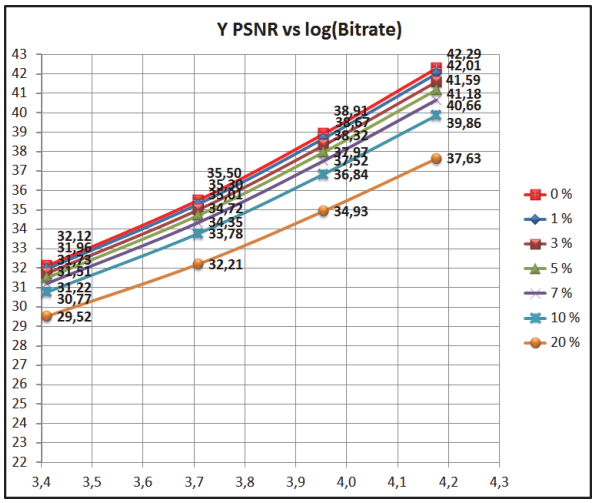


Fig. 3. AI mode at 1 slice per frame and without using EC.

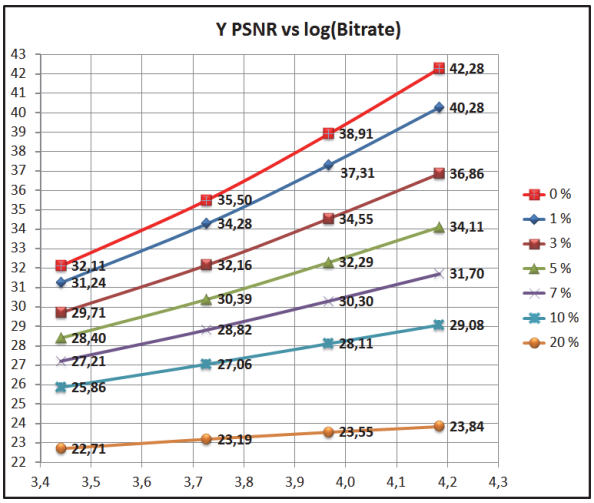


Fig. 5. AI mode at 13 slices per frame and without using EC.

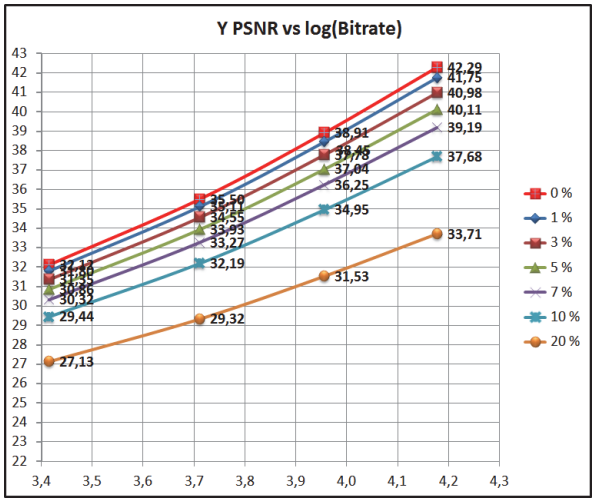


Fig. 4. AI mode at 2 slices per frame and without using EC.

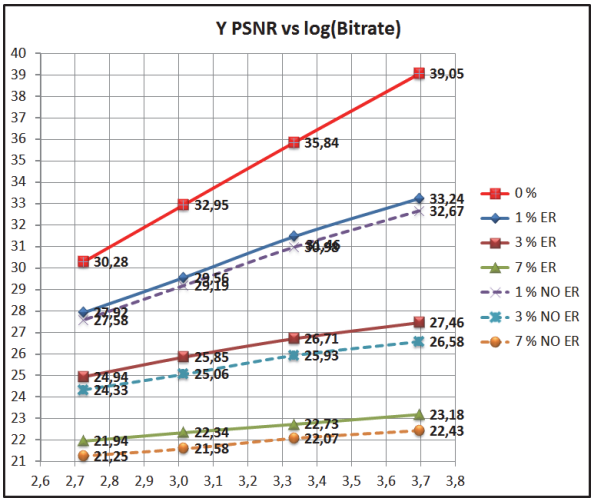


Fig. 6. Comparison of EC and non-EC decoding for RA mode at 13 slices per frame.

D. Adding a basic error concealment method

In Figure 6 we can see the improvement in PSNR that the EC method provides for RA mode at 13 slices per frame at three percentages of data loss. In Figure 7 we can see the corresponding improvements for AI mode. Comparing both figures it can be seen that the simple method implemented is much more effective in AI mode than in RA mode. In RA mode we obtain improvements of 0.44 dB, 0.78 dB and 0.71 dB for 1%, 3% and 7% data loss rates, respectively, while in AI mode we obtain improvements of 0.52 dB, 1.33 dB and 2.42 dB for 1%, 3% and 7% data loss rates, respectively.

In Figure 8 we can find the answer to the previously asked question. This plot represents the RD curves for AI mode, using the basic EC method at 3% data loss rate and varying the number of slices per frame. As in the non concealed version, an increase in the number of slices also produces a worse value of PSNR. The BD-PSNR differences with respect to encoding with 1 slice per frame are the following: -0.30 dB, -0.90 dB, -1,52 dB, -2,10 dB and -2,67 dB for 2, 4, 8, 13 and 26 slices per frame, respectively.

E. LP_I32 and LB_I32 coding modes

As we have seen in section IV-C, LB and LP modes do not recover from data losses because they only have one I frame at the beginning. From the second frame till the end of the sequence only P or B frames (which are based in previous frames) will appear. There is not any refreshing frame that can stop error drifting when data loss occurs. This is not true for RA mode, which inserts an I frame (CDR frame) every second of the sequence (for this sequence, every 32 frames). So we have created two new modes (LP_I32 and LB_I32) which are very similar to LP and LB modes but inserting an I frame

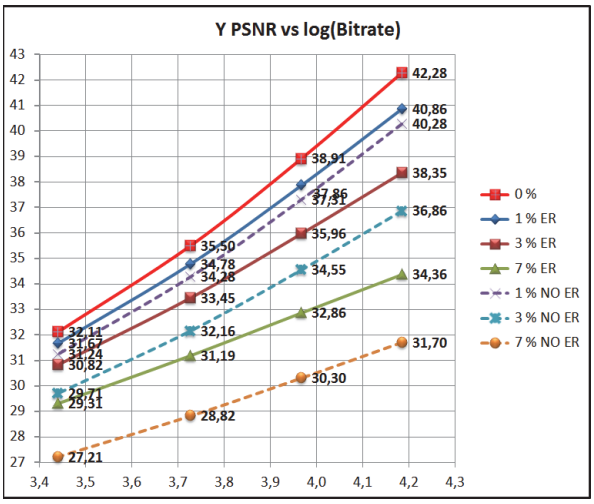


Fig. 7. Comparison of EC and non-EC decoding for AI mode at 13 slices per frame.

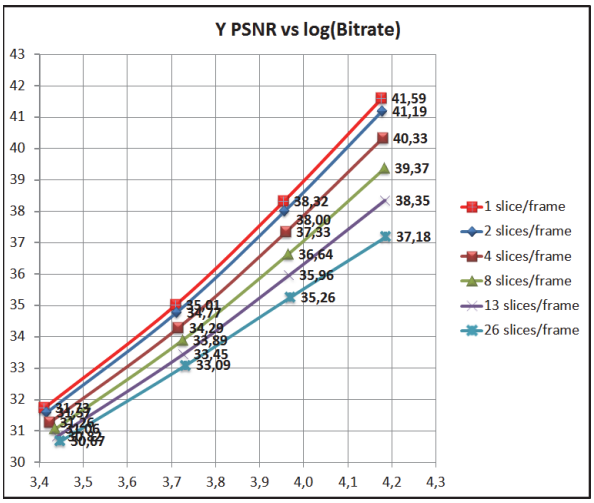


Fig. 8. AI mode at 3% loss using EC decoding for different slices per frame.

(CDR frame) every 32 frames, in a similiar way as RA does. We thought that we would get similar RD curves for LB_I32 mode than for RA mode but in Figure 9 it can be seen that there is a difference in PSNR values for these two modes. By inserting an I frame periodically, PSNR improves over the original LP and LB modes, but we have realised that there is another characteristic of RA mode that also helps to reduce drifting: prediction from future frames (in display order). In RA mode some prediction is done by using future frames, so it can benefit from future I frames to refresh a damaged region. But LB_I32 mode always uses past frames (in display order) so it cannot benefit from future refreshing frames and when an error occurs it will be probably *infect* the following frames. The difference in BD-PSNR for RA and LB_I32 modes is of 3.76 dB, 3.80 dB and 3.49 dB for 1%, 3% and 5% data loss rates.

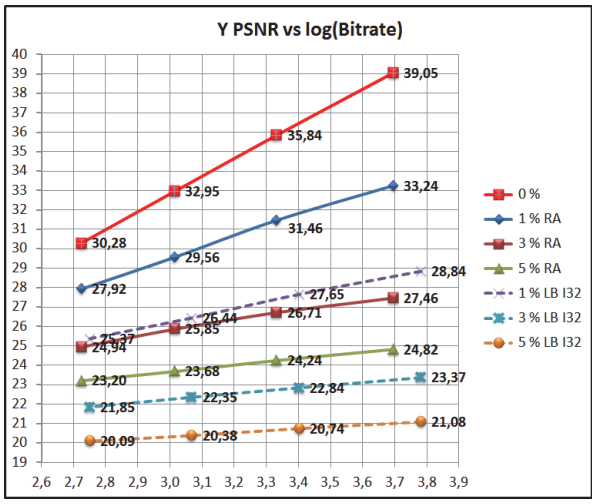


Fig. 9. RA mode versus LB_32I mode using EC decoding for different data loss rates.

V. Conclusions

In this paper we have presented an evaluation of HEVC new video coding standard. In order to make results comparable with other experiments we have used the common test conditions, which were designed to measure improvements of the contributions to the standard. As the original reference software is not resistant to data losses we have modified the reference decoder in order it does not crash when some data are missing. By using this modified version we have been able to conduct several tests and see how HEVC and the different coding modes behave under packet losses conditions. We have also implemented a basic error concealment technique in the decoder and shown the benefits of using it. At last, we have modified two of the original coding modes to study what characteristics would help to make a bit stream error resilient.

Acknowledgements

This work was supported by *Spanish Ministerio de Ciencia e Innovación* under project TIN2011-27543-C03-03 and *Consellería de Educación, Formación y Empleo de la Generalitat Valenciana* under project ACOMP/2013/003.

References

[1] High Efficiency Video Coding ITU-T Press Release, http://mpeg.chiariglione.org/sites/default/files//files/meetings/docs/w13253_0.doc, ," .

[2] High Efficiency Video Coding ISO/IEC Press Release, http://www.itu.int/net/pressoffice/press_releases//2013/01.aspx, ," .

[3] Benjamin Bross, Woo-Jin Han, Jens-Rainer Ohm, Gary J. Sullivan, Ye-Kui Wang, and Thomas Wiegand, "High Efficiency Video Coding (HEVC) text specification draft 10," Tech. Rep. JCTVC-L1003, Joint Collaborative Team on Video Coding (JCT-VC), Geneva (Switzerland), January 2013.

[4] HM Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-9.0, ," .

[5] Frank Bossen, "Common test conditions and software reference configurations," Tech. Rep. JCTVC-L1100, Joint

Collaborative Team on Video Coding (JCT-VC), Geneva (Switzerland), January 2013.

[6] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, 2012.

[7] Frank Bossen, David Flynn, and Karsten Suehring, "HM 10.0 Software Manual," Tech. Rep. JCTVC-Software Manual, Joint Collaborative Team on Video Coding (JCT-VC), 2013.

[8] Frank Bossen, David Flynn, and Karsten Suehring, "HEVC HM software development and software technical evaluation (AHG3)," Tech. Rep. JCTVC-M0003, Joint Collaborative Team on Video Coding (JCT-VC), Incheon (Korea), April 2013.

[9] Gisle Bjontegaard, "Calculation of average PSNR differences between RD-curves," Tech. Rep. VCEG-M33, Video Coding Experts Group (VCEG), Austin (Texas), April 2001.

[10] Gisle Bjontegaard, "Improvements of the BD-PSNR model," Tech. Rep. VCEG-M33, Video Coding Experts Group (VCEG), Berlin (Germany), July 2008.

Streaming Interactivo de Secuencias de Imágenes JPEG2000 de Alta Resolución

J.J. Sánchez-Hernández, J.P. García-Ortiz,
Carmelo Maturana-Espinosa y Vicente González-Ruiz
Departamento de Informática. Universidad de Almería, Spain
Campus de Excelencia Internacional Agroalimentario, ceiA3

D. Müller
European Space Agency, ESTEC
Noordwijk, Netherlands

Resumen— **El estándar internacional JPEG2000 pretende ser un sucesor del estándar JPEG en la mayoría de sus áreas de aplicación, y especialmente en el campo de la transmisión interactiva de imágenes. Para esta tarea, JPEG2000 se basa en el uso del protocolo JPIP que permite realizar un acceso espacial de forma aleatoria en tiempo real (panning, tilting y zooming) sobre la imagen que se está visualizando de forma progresiva (streaming). En este contexto, este trabajo[1] presenta una solución compatible con el estándar JPIP para el streaming interactivo de archivos JPX que son transmitidos sobre enlaces de comunicación con un ancho de banda variable. La variación impredecible del ancho de banda durante la transmisión obliga a los clientes JPIP a tener un buffer con una mínima cantidad de datos y a implementar un mecanismo de feedback que permita al servidor JPIP decidir, en tiempo real, la cantidad de code-stream que se va a transmitir de cada imagen. Nuestra propuesta garantiza que se puede mantener el frame-rate de la visualización incluso en situaciones donde se produzcan grandes variaciones de ancho de banda y se dispongan de reducidos tamaños de buffer. Como una última ventaja, queremos remarcar que nuestra solución también puede ser utilizada para realizar streaming de secuencias Motion JPEG2000.**

Palabras clave— **Streaming media, high-resolution imaging.**

I. Introducción

En muchos campos de la ciencia y la tecnología (tales como la telemedicina, telemicroscopía y astronomía) existe una necesidad de crear grandes repositorios de imágenes. Debido a razones como economizar el coste de este proceso, la maximización de la calidad de los resultados o la imposibilidad de repetir la adquisición de las imágenes, estas imágenes necesitan ser almacenadas con la mayor calidad posible. Por otro lado, como consecuencia de la alta resolución de las imágenes y el gran número de ellas, la demanda de memoria suele ser muy alta. Con el objetivo de minimizar esto, las imágenes pueden ser comprimidas, pero debido a que la información de las imágenes es importante, se utilizan codecs sin pérdida, o muy próximos a serlo.

Otro aspecto importante a considerar es que en muchos casos, los repositorios de las imágenes se encuentran físicamente alejados de la mayoría de los usuarios que necesitan trabajar con dichas imágenes. Por ejemplo, en el proyecto JHelioviewer [1], un servidor central almacena un repositorio de imágenes de alta resolución del Sol que está en constante crecimiento, y los usuarios de forma interactiva pueden obtener y visualizar colecciones de estas imágenes usando Internet como medio de transmisión. Debido al tamaño (4096×4096 píxeles) de estas imágenes, la mayoría de los usuarios no son capaces de visualizar las imágenes completas en sus navegadores y necesitan de funcionalidades como panning, tilting y zooming. Además, estas acciones necesitan ser realizadas en tiempo real mientras las imágenes están siendo descargadas (streaming). Podemos ver un ejemplo de este tipo de interacción en la Figura 1.

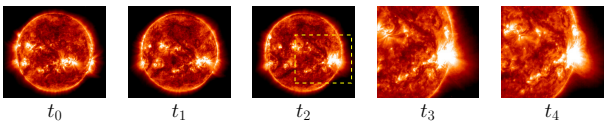


Fig. 1. Ejemplo de diferentes instantes de tiempo de una sesión de exploración de imágenes remotas con JHelioviewer. Al principio, el usuario indica la secuencia de imágenes y el frame-rate. La primera imagen se visualiza en el instante de tiempo t_0. En el instante de tiempo t_1 la imagen mostrada es ligeramente diferente de la primera, debido principalmente a la rotación del Sol. En el instante de tiempo t_2 el usuario especifica una WOI (Window Of Interest). t_3 y t_4 son los instantes de tiempo restantes, donde solamente se ha transmitido y visualizado la WOI especificada previamente.

La aplicación JHelioviewer ha sido diseñada para visualizar secuencias de imágenes de una forma continuada, como si se tratase de un vídeo. En el modo imagen, el refinamiento progresivo característico de JPEG2000 se utiliza para mostrar la WOI[2] definida por el usuario. Sin embargo, en el modo vídeo, con una cadencia que depende del frame-rate de la reproducción, se muestra la reconstrucción de una WOI de cada imagen, con una calidad que depende de la cantidad de datos recibidos para la WOI hasta el instante de la reproducción. Cuando la WOI de la última imagen de la secuencia ha sido visualizada, la reproducción vuelve a empezar desde el inicio de la secuencia. En esta segunda pasada, la calidad de la WOI de cada imagen suele ser mejor que en la

[1]Este trabajo ha sido financiado por becas del Ministerio de Ciencia e Innovación de España (TIN2008-01117 y TEC2010-11776-E), Junta de Andalucía (P08-TIC-3518 y P10-TIC-6548) y el Fondo Europeo de Desarrollo Regional (FEDER).

[2]Una WOI es una región rectangular de píxeles definida por $(s,(x,y),(w,h))$, donde s es el nivel de resolución espacial, (x,y) es la esquina superior izquierda y (w,h) es el ancho y alto de la WOI, en píxeles. s define la resolución de la imagen mostrada y estas coordenadas hacen referencia a esta imagen, que tiene una resolución de $X/2^s \times Y/2^s$, donde $X \times Y$ es la resolución original.

primera pasada porque se han recibido más datos. Este proceso de refinamiento finaliza cuando se ha transmitido todo el code-stream de la WOI o cuando el usuario selecciona una nueva WOI.

En el marco de trabajo descrito anteriormente, este trabajo presenta y evalúa una solución compatible con el estándar JPEG2000 para la tarea de obtener de forma interactiva secuencias de imágenes de alta resolución sobre enlaces de transmisión con ancho de banda variable. La sección II hace una introducción al estándar JPEG2000 y muestra algunos de los inconvenientes que presentan las técnicas más comunes utilizadas para resolver esta tarea. En la sección III, se realiza una comparativa entre nuestra propuesta y otras soluciones similares que se han realizado sobre este campo. La sección IV presenta nuestra propuesta, que es evaluada en la sección V. Este trabajo finaliza con algunas conclusiones y líneas de trabajo futuro, en la sección VI.

II. JPEG2000

El estándar JPEG2000 [2] es un codec de imágenes de dos etapas (transformada + codificación de la entropía). Primero, MCT (Multi-Component Transform) elimina la redundancia intercomponente y la transformada 2D DWT (Discrete Wavelet Transform) elimina la redundancia intracomponente. El resultado de esta transformada es un vector de matrices de conjuntos de coeficientes wavelet espacialmente no correlacionados que están organizados en un conjunto de subbandas que permiten una representación multiresolución de la imagen. En el siguiente paso, se aplica un codec de planos de bits basado en bloques llamado EBCOT (Embedded Block Coding with Optimal Truncation). El code-stream que se obtiene como resultado puede ser accedido de forma aleatoria con el objetivo de recuperar un determinado número de componentes (escalabilidad en componentes), WOI/resoluciones (escalabilidad espacial) y calidades (escalabilidad en calidad). Finalmente, si se necesita realizar un control del bit-rate, se puede utilizar una técnica como el algoritmo PCRD-opt (Post-Compression Rate-Distortion optimization) que selecciona aquellos paquetes que minimizan la distorisión de la codificación para un bit-rate específico. En la recuperación de imágenes remotas donde el enlace de transmisión suele ser el cuello de botella del sistema, los paquetes se envían siguiendo una progresión LRCP (Layer Resolution Precinct Component) porque es la progresión que ofrece menor distorsión durante la recepción del code-stream.

También se ha estandarizado [3] un protocolo de transmisión para la recuperación interactiva de imágenes remotas, llamado JPIP (JPEG2000 Interactive Protocol). Es importante resaltar que JPIP es un protocolo stateless y que los clientes no solicitan paquetes o data-bins[3], solicitan coordenadas de una WOI. Utilizando el protocolo JPIP, un cliente puede solicitar una imagen y controlar el flujo de datos (desde el servidor hacia el cliente) por medio de diferentes técnicas. Una de las más utilizadas (por ejemplo, en la implementación del estándar que ofrece Kakadu Software [4]) es una técnica stop-and-wait basada en la recuperación incremental del code-stream donde el cliente restringe el tamaño de las respuestas que quiere recibir del servidor utilizando el parámetro `len`. La Figura 2 muestra un ejemplo de uso de esta técnica, donde el cliente especifica que cada respuesta debe ser menor o igual que 2000 bytes. Cuando el cliente recibe una respuesta desde el servidor, éste vuelve a repetir el mismo proceso hasta que se hayan recibido todos los datos de la WOI seleccionada, ajustando los valores del parámetro `len` si fuese necesario.

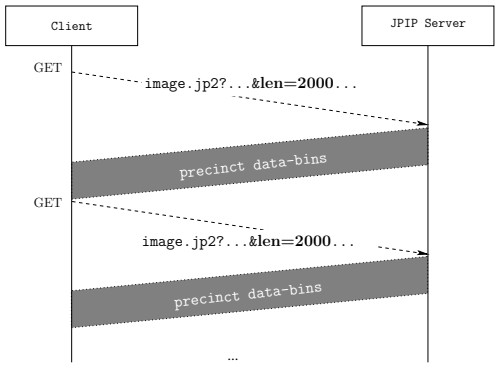


Fig. 2. Ejemplo de cómo recuperar una imagen utilizando la técnica de recuperación stop-and-wait.

Una extensión sencilla del procedimiento anterior se puede utilizar para recuperar una secuencia de imágenes si el cliente especifica en su petición un rango de imágenes. Siendo k_i el número de imágenes que se solicitan en cada i-th petición. La Figura 3 muestra un ejemplo de cómo recuperar una secuencia de 160 imágenes utilizando esta técnica. En este ejemplo, el cliente utiliza $k_i = 15$ imágenes en todas sus peticiones. Hay que tener en cuenta que dependiendo del tamaño del code-stream de las imágenes y del valor del parámetro `len`, pueden ser necesarias varias pasadas para poder transmitir completamente todas las imágenes que haya solicitado el cliente.

Claramente, k_i y $\texttt{len}_i$ tienen un alto impacto en la QoE (Quality of the Experience) del usuario quien básicamente lo que desea es poder visualizar la secuencia de imágenes con la mayor calidad posible a una velocidad de frame-rate determinada r. Tal y como se puede ver en la Figura 3, donde t_i es el tiempo de recepción del grupo de imágenes y RTT_{i+1} el Round Trip Time en la iteración $i+1$, si

$$\frac{k_i}{r} < t_i + \mathrm{RTT}_{i+1}, \tag{1}$$

el usuario sufrirá una pausa en la reproducción de la secuencia en el instante de tiempo $\sum_{x=0}^{i-1} t_x + k_i/r$.

[3] Una estructura de datos que contiene aquellos paquetes relacionados con la WOI solicitada y que son independientes, de modo que el servidor puede terminar la transmisión de la secuencia de paquetes en cualquier punto.

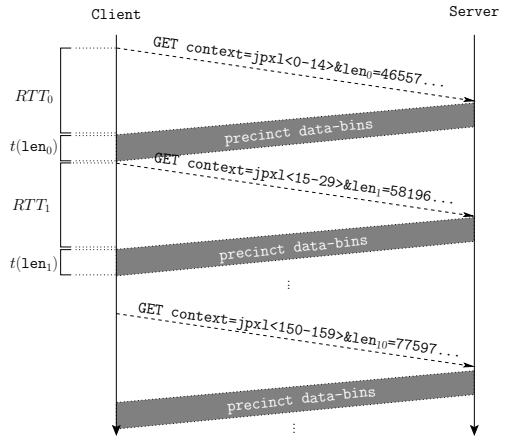


Fig. 3. Ejemplo de cómo recuperar una secuencia de imágenes utilizando la técnica de recuperación stop-and-wait.

III. Trabajos relacionados

La mayor parte de los primeros sistemas de exploración de imágenes remotas están basados en representaciones de pirámides de Gauss y Laplace y/o en el tiling en el dominio de la imagen para proporcionar operaciones de zooming, panning y tilting sobre las imágenes. Un ejemplo de esto puede ser la propuesta de Grunheit et al. [5] para realizar streaming interactivo de vistas panorámicas de alta resolución, donde las imágenes panorámicas son divididas en pequeños patches de 512×512 píxeles y son comprimidos de forma independiente con JPEG. Sin embargo, esta solución sólo tiene la capacidad de transmitir imágenes independientes (modo imagen), y no permite la transmisión de secuencias de imágenes (modo vídeo).

En [6], Naman and Taubman desarrollan un sistema interactivo escalable de vídeo basado en JPEG2000 (JSIV). Las secuencias de vídeo se almacenan como frames independientes comprimidos con JPEG2000 para proporcionar escalabilidad en calidad, espacial y en resolución, y se hace uso de predicciones y refresco condicional de code-blocks JPEG2000 para explotar la redundancia temporal. El servidor selecciona de forma óptima el mejor número de capas de calidad de cada code-block transmitido y el cliente intenta reconstruir los frames con la mayor calidad posible. Además, se presentó una extensión de este sistema en [7] basada en el uso de la compensación de movimiento para mejorar las predicciones en el lado del cliente.

Vetro et al. describen en [8] un sistema de streaming de vídeo escalable JPEG2000 sobre redes con un ancho de banda limitado, claramente orientado a sistemas de videovigilancia. Este sistema ofrece varios métodos de streaming y un algoritmo adaptativo de control del rate para realizar el transcoding JPEG2000 que adapta la calidad de la resolución de la escena en función del ancho de banda disponible.

Recientemente, en [9], Jiménez-Rodríguez et al. proponen un método de control del rate para la transmisión de secuencias de vídeo pre-codificadas con JPEG2000 sobre canales en los que puede variar su capacidad durante la transmisión del vídeo debido a congestiones de la red, fallos hardware o saturaciones en los routers.

Ninguna de las propuestas descritas anteriormente puede ser aplicada directamente sobre el contexto de JHelioviewer debido a que ninguna proporciona la flexibilidad que se requiere, porque estas técnicas están basadas en un escenario donde la secuencia de imágenes debe ser conocida previamente y no en un escenario donde el conjunto de imágenes es seleccionado de forma interactiva desde un repositorio. Otro aspecto importante a considerar en la transmisión de secuencias de imágenes (modo vídeo) es que el envío de datos sobre una red puede introducir retardos variables (jitter) que dificulten la sincronización entre el cliente y el servidor. Y. Hui et al. [10] consideran el uso de grandes buffers y un mecanismo de feedback para mantener el cliente y el servidor sincronizados. En su propuesta, el stream multimedia es modelado como un flujo de datos controlado por buffers con tres estados (high, medium, low) que determinan el rate de la transmisión que se ha solicitado y si es el rate apropiado.

En [11] se especifica un protocolo de sincronización para garantizar una adecuada visualización de un stream multimedia en el cliente. En esta propuesta, el ciente está basado en un esquema de control que utiliza un umbral superior (BOH) y un umbral inferior (BOL) para controlar el nivel de ocupación del buffer.

Ninguno de los esquemas de sincronización que se han mencionado anteriormente son compatibles con el estándar JPIP y por lo tanto no pueden ser aplicados fácilmente al escenario que estamos considerando en nuestra propuesta. Nuestra propuesta está basada en un esquema de sincronización sencillo pero eficiente totalmente compatible con el estándar JPIP donde el cliente es el único que controla el estado del buffer y envía mensajes de feedback al servidor utilizando parámetros del estándar JPIP.

IV. Propuesta

Hemos modificado la técnica stop-and-wait por otra técnica más eficiente basada en un pipeline. Por lo tanto, en el lado del cliente, cuando el usuario especifica una WOI, el cliente envía solamente una petición. Por ejemplo, para reproducir un vídeo de 160 frames a 20 fps con un ancho de banda de 10 Mbits/segundo, el cliente solamente tendría que enviar al servidor la petición que se muestra en la Figura 4.

En el lado del servidor hemos considerado que las peticiones de una WOI son acumulativas, de modo que el cliente no tiene que volver a reenviar algunos de los parámetros que ya han sido enviados en cada una de las peticiones, debido a que estos parámetros se mantienen a lo largo de toda la sesión.

Considerando los comentarios anteriores, hemos propuesto una técnica que permite controlar el flujo de datos basándonos en la estimación del ancho de

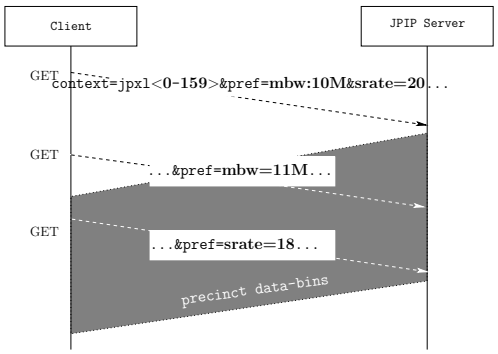


Fig. 4. Ejemplo de cómo recuperar una secuencia de imágenes utilizando una técnica basada en un pipeline.

banda disponible entre el servidor y el cliente, y la velocidad a la que se va a reproducir la secuencia en el cliente. Los resultados experimentales demuestran que nuestra propuesta es efectiva y totalmente compatible con el estándar JPIP, debido a que está basada en el uso de los parámetros `mbw` y `srate` para transmitir vídeo con el protocolo JPIP. El parámetro `mbw` (Maximum Bandwidth), indica el máximo rate con el que el cliente espera recibir los datos, expresado en bits/segundo y `srate` (Sampling Rate) se utiliza para indicar que la media de la tasa de muestreo por segundo no puede ser mayor que este valor. Estos parámetros se han definido para controlar flujo de datos de una transmisión de un vídeo Motion JPEG2000 (muy similar a la transmisión de una secuencia de imágenes JPEG2000). El parámetro `len` no se utiliza en esta técnica.

Con el objetivo de entender cómo se utilizan estos parámetros, vamos a suponer que el servidor recibe una petición para tranmitir una secuencia de imágenes que van a ser visualizados con un frame-rate de 20 imágenes/segundo (`srate`=20) y la estimación del ancho de banda disponible es de 10 Mbits/segundo (`mbw`=10). En esta situación el servidor debería enviar:

$$\frac{10\frac{\text{Mbits}}{\text{second}}}{20\frac{\text{pictures}}{\text{second}}} = 0.5 \text{ Mbits/picture.}$$

En la implementación descrita anteriormente, el cliente le comunica al servidor el ancho de banda disponible cada segundo y también le comunicará cualquier modificación que se produzca en el valor del frame-rate con el que se va a reproducir la secuencia de imágenes.

Se han establecido dos modos de funcionamiento en el cliente y en el servidor, dependiendo de los parámetros que se envían en las peticiones del cliente.

Modo Imagen. Este es el modo de funcionamiento por defecto y se activa cuando se abre el vídeo por primera vez o cuando el vídeo es pausado por el usuario. En este modo el servidor envía las imágenes completas, de manera que no envía ningún paquete de la siguiente imagen hasta que no ha enviado todos los paquetes de la imagen actual.

Modo Vídeo. Este modo se activa cuando el usuario inicia la reproducción del vídeo y necesita que que las peticiones que se envían al servidor contengan los parámetros `mbw` y `srate`. Estos parámetros sólo se pueden utilizar en el modo vídeo. En este caso, el servidor calcula el número de bytes que se deben enviar para cada imagen y envía aproximadamente la misma cantidad de bytes de cada una de ellas.

A. El Cliente

Uno de los principales problemas en el streaming interactivo es la sincronización entre el cliente y el servidor de modo que el cliente sea capaz de tener suficientes datos de las imágenes en su buffer para mantener la velocidad de reproducción de la secuencia a lo largo de toda la transmisión, debido a los retardos de la red provocados por el jitter. Para resolver este problema se ha propuesto un algoritmo de sincronización implementado en el lado del cliente basado en la estimación del ancho de banda, teniendo en cuenta las estimaciones previas y los errores cometidos. Todas las operaciones de este proceso se realizan en el lado del cliente con el objetivo de aliviar la carga del servidor liberándolo de esta tarea y por lo tanto mejorar la escalabilidad del servidor.

A continuación se presenta una descripción del algoritmo utilizado por el cliente para realizar la estimación del ancho de banda.

Entrada:

$s(R_i)$	Tamaño de la última respuesta R_i
$t(R_i)$	Tiempo necesario para recibir R_i
$w = 0.5$	Factor de corrección
λ	Tiempo de buffering
$\alpha = 0.1$	Peso para la medida del ancho de banda actual
$\texttt{mbw}_0$	Ancho de banda durante la transmisión de los metadatos
P	Velocidad de reproducción
N	Número de imágenes en el buffer

Salida:

$\texttt{mbw}_i$	Ancho de banda estimado

Algoritmo:

```
# Última medida del ancho de banda
(1) B = s(R) / t(R)
# Media ponderada del ancho de banda
(2) mbw_i = αB + (1 − α)mbw_{i−1}
# Media del error del ancho de banda
(3) ε_i = (ε_{i−1} + |B − mbw_i| / max(B, mbw_i)) / 2
# Tamaño actual del buffer en segundos
(4) λ' = N / P
# La estimación del ancho de banda ha sido muy alta
(5) if(λ' < λ) then
(6)     mbw_i = mbw_i(1 − w(λ'/λ + 1))
# La estimación del ancho de banda ha sido muy baja
(7) else if(λ' > λ) then
(8)     mbw_i = mbw_i(1 + ε_j)
(9) end if
```

En nuestra implementación, este algoritmo se ejecuta de forma periódica cada segundo, aunque este periodo puede ser ligeramente superior dependiendo de $t(R_i)$. Este tiempo depende del ancho de banda real que se ha usado en la respuesta R_i y de $s(R_i)$. El primer factor es incontrolable (es una consecuencia directa de la carga de la red) y el segundo depende del tamaño de los últimos data-bins recibidos.

B. El Servidor

Para cada petición, el servidor envía un conjunto de data-bins de un conjunto de imágenes. Estas imágenes son especificadas por el cliente haciendo uso del parámetro `context` como se puede ver en la Figura 4. El número de data-bins que el servidor envía de cada imagen depende del valor de los parámetros `srate` y `mbw`. Tal y como muestra la siguiente expresión, el servidor envía aproximadamente (dependiendo del tamaño de los data-bins) el siguiente número de bits de cada imagen: $\frac{\texttt{mbw}}{\texttt{srate}}$.

Aunque este procedimiento de control del bit-rate no es el más óptimo, debido a que con poca frecuencia los code-streams de las imágenes son truncados exactamente al final de una capa de calidad y debido a que no existe una priorización de los paquetes durante la transmisión para ofrecer la mínima distorsión en la reconstrucción de las imágenes, se ha seleccionado este método por dos razones: (1) los requisitos computacionales del servidor son muy bajos y (2) en la práctica este método funciona bastante bien porque en la mayoría de los casos el punto de truncado está muy próximo a un punto de truncado óptimo de una capa de calidad, si existen suficientes capas de calidad.

V. Evaluación

Para llevar a cabo los experimentos, se han utilizado dos escenarios de red reales con diferentes capacidades de ancho de banda. El escenario S_1 representa una conexión con alto ancho de banda entre cliente y servidor, 1.5 MB/s, y el escenario S_2 representa un escenario con poco ancho de banda, 120 KB/s. Las secuencias de vídeo utilizadas en los experimentos han sido dos archivos JPX, *Sun* y *Stockholm*, formados por un conjunto de imágenes diferentes. *Sun* es una secuencia de imágenes del Sol formada por 1000 frames de 4096×4096 píxeles con 8 bits/componente y comprimidas con JPEG2000 utilizando una compresión con path irreversible, con 8 capas de calidad y 9 niveles de resolución. *Stockholm* es una secuencia que ofrece una vista panorámica de la ciudad de Estocolmo [12]. La secuencia está compuesta por 604 imágenes de 1280×720 con 8 bits/componente, y comprimidas con JPEG2000 utilizando una compresión con path irreversible, con 8 capas de calidad y 6 niveles de resolución.

En el experimento 1 hemos estudiado el impacto del uso de precintos en la secuencia *Sun* cuando es enviada a través de los escenarios S_1 y S_2 utilizando la técnica basada en pipeline. Las imágenes han sido codificadas con precintos de 128×128 y sin precintos. En el lado del cliente se ha establecido un frame-rate de 10 frames/segundo y no se ha utilizado buffering.

La Figura 5 muestra los resultados de la exploración del impacto el uso de precintos. Se puede observar que el uso de precintos mejora el valor medio del PSNR en ambos escenarios.

La Figura 6 muestra los resultados del experimento 2, que examina el impacto del número de capas de

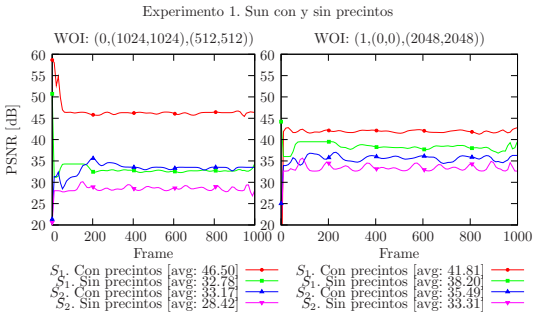


Fig. 5. Experimento 1. Estudio del impacto del uso de precintos sobre los escenarios S_1 y S_2.

calidad en la secuencia *Sun* cuando es transmitida a través de los escenarios S_1 y S_2, utilizando la técnica basada en pipeline. En el lado del cliente se ha establecido un frame-rate de 5 frames/segundo y no se ha utilizado buffering.

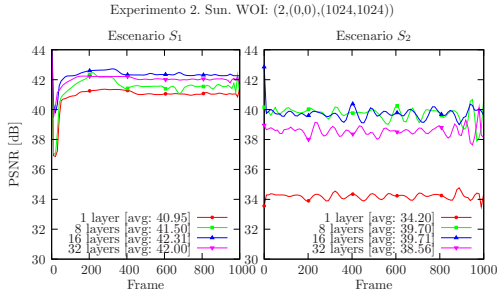


Fig. 6. Experimento 2. Estudio del impacto del número de capas de calidad sobre los escenarios S_1 y S_2.

El experimento 3 examina el comportamiento de las dos técnicas, stop-and-wait y pipeline, con el objetivo de hacer una comparación del impacto sobre la calidad de la imagen cuando el vídeo se reproduce a diferentes frame-rates y no existe buffering en el cliente. En este experimento se han transmitido ambas secuencias, a través de los escenarios S_1 y S_2. Las imágenes fueron codificadas con 8 capas de calidad y con precintos de 128×128. En el lado del cliente se ha evaluado el impacto de utilizar frame-rates de 5, 10, 15 y 20 frames/segundo.

En las Figuras 7 y 8 se puede observar que la técnica propuesta puede mejorar la calidad de las imágenes respecto a la técnica stop-and-wait, para todos los frame-rates que han sido testeados.

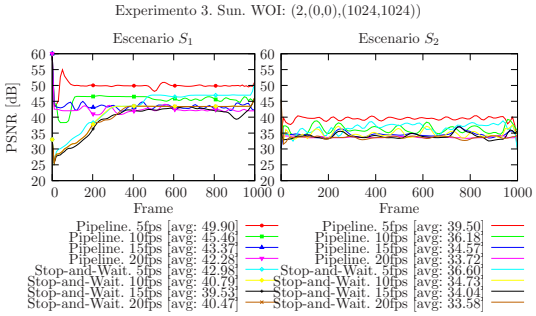


Fig. 7. Experimento 3. Una comparativa entre las dos técnicas en ambos escenarios S_1 y S_2.

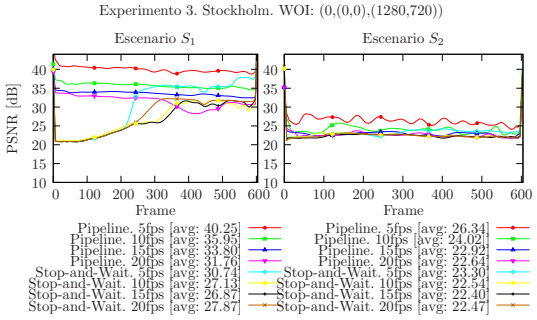


Fig. 8. Experimento 3. Una comparativa entre las dos técnicas en ambos escenarios S_1 y S_2.

El experimento 4 examina el impacto del uso de un buffer con diferentes tamaños, cuando ambas secuencias de imágenes son transmitidas sobre los escenarios S_1 y S_2, utilizando la técnica basada en pipeline. Las imágenes han sido codificadas con 8 capas de calidad y con precintos de 128×128. El frame-rate utilizado en el cliente ha sido de 5 y 10 frames/segundo. Las Figuras 9 y 10 muestran los resultados de cada uno de los experimentos realizados.

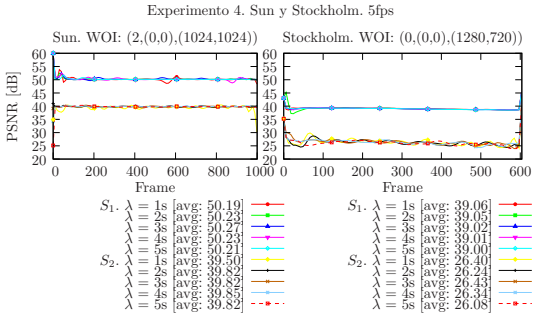


Fig. 9. Experimento 4. Estudio del impacto del uso de un buffer con diferentes tamaños.

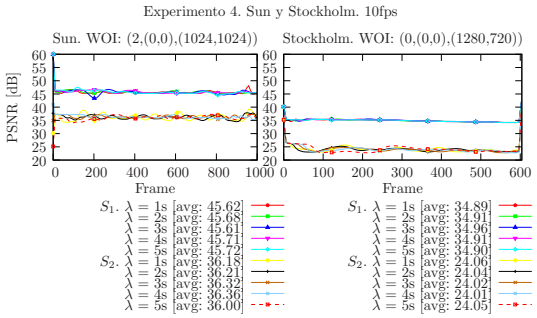


Fig. 10. Experimento 4. Estudio del impacto del uso de un buffer con diferentes tamaños.

VI. Conclusiones

Los resultados de los experimentos demuestran que nuestra propuesta garantiza el mismo frame-rate durante toda la reproducción incluso en situaciones donde se produzcan variaciones significativas del ancho de banda disponible y se disponga de un tamaño reducido del buffer. Nuestra propuesta es efectiva y totalmente compatible con el estándar JPIP.

Como línea de trabajo futuro se plantea mejorar el método que realiza el control del rate en el servidor.

Referencias

[1] D. Müeller, B. Fleck, G. Dimitoglou, B. W. Caplins, D. E. Amadigwe, J. P. Garcia Ortiz, A. Alexanderian B. Wamsler, V. Keith Hughitt, and J. Ireland, "JHelioviewer: Visualizing large sets of solar images using JPEG 2000," *Computing in Science and Engineering*, vol. 11, no. 5, pp. 38–47, September 2009.

[2] International Organization for Standardization, "Information Technology - JPEG 2000 Image Coding System - Core Coding System," ISO/IEC 15444-1:2004, September 2004.

[3] International Organization for Standardization, "Information Technology - JPEG 2000 Image Coding System - Interactivity Tools, APIs and Protocols," ISO/IEC 15444-9:2005, November 2005.

[4] "Kakadu JPEG 2000 SDK," `http://www.kakadusoftware.com`.

[5] C. Grunheit, A. Smolic, and T. Wiegand, "Efficient representation and interactive streaming of high-resolution panoramic views," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, 2002, pp. III–209 – III–212.

[6] Aous Thabit Naman and David Taubman, "Jpeg2000-based scalable interactive video (jsiv)," *IEEE Transactions on Image Processing*, vol. 20, pp. 1435– 1449, May 2011.

[7] A.T. Naman and D. Taubman, "Jpeg2000-based scalable interactive video (jsiv) with motion compensation," *Image Processing, IEEE Transactions on*, vol. 20, no. 9, pp. 2650 –2663, sept. 2011.

[8] Anthony Vetro, Derek Schwenke, Toshihiko Hata, and Naoki Kuwahara, "Scalable video streaming based on jpeg2000 transcoding with adaptive rate control," *Adv. MultiMedia*, vol. 2007, pp. 7–7, January 2007.

[9] Member Leandro Jiménez-Rodríguez, Francesc AulÃ--Llinás and Michael W. Marcellin, "Fast rate allocation for jpeg2000 video transmission over time-varying channels," *IEEE Transactions on Multimedia*, vol. 15, pp. 15–26, January 2013.

[10] Jun Li Joseph Y. Hui, Ezhan Karasan and Junbiao Zhang, "Client-server synchronization and buffering for variable rate multimedia retrievals," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 226–237, January 1996.

[11] Mourad Daami and Nicolas D. Georganas, "Client based synchronization control of coded data streams," *IEEE International Conference on Multimedia Computing and Systems*, pp. 387–394, June 1997.

[12] "Stockholm video sequence," `ftp://ftp.ldv.e-technik.tu-muenchen.de/pub/test_sequences/720p/720p5994_stockholm_ter.yuv`.

Configuración de un *cluster* InfiniBand para una gestión eficiente y abierta de los niveles de servicio

Álvaro Cebrián-García, Jesús Escudero-Sahuquillo, Pedro Javier García y Francisco José Quiles[1]

Resumen— **La especificación InfiniBand está presente en los actuales sistemas de cómputación de altas prestaciones (aproximadamente en un 45% de los sistemas de la lista TOP500). Entre sus ventajas destaca la posibilidad de configurar varios mecanismos para adaptarlos a las necesidades del sistema, aunque la implementación concreta de dichos mecanismos no siempre está completamente definida por el estándar InfiniBand. Uno de estos aspectos configurables es la gestión de los niveles de servicio (SLs) y los canales virtuales (VLs), que se usan en diversos algoritmos y técnicas orientadas a maximizar las prestaciones de la red de interconexión. En este artículo se propone una metodología de ayuda para realizar una configuración efectiva de SLs y VLs en un *cluster* InfiniBand real, basado en componentes del fabricante Mellanox. La idea básica consiste en asignar los SLs disponibles a distintos flujos de tráfico. Dicha asignación se realizará de forma dinámica, ya que la infraestructura *software* del *cluster* permite una asignación dinámica de SLs a flujos de datos en tiempo de ejecución de las aplicaciones, es decir, cuando dichos flujos de datos vayan a ser inyectados en la red. En función del SL obtenido, el sistema asignará a los flujos de tráfico el VL correspondiente. Los resultados que se muestran en este trabajo como ejemplo de esta metodología, y que han sido obtenidos mediante experimentos en un sistema real, demuestran que incluso una política muy sencilla de gestión de SLs es capaz de conseguir un 290% de mejora en ciertas condiciones de tráfico.**

Palabras clave— **InfiniBand, Niveles de Servicio, Canales Virtuales, Subnet Manager**

I. Motivación

ACTUALMENTE , puede afirmarse que InfiniBand (IB)[1] se ha consolidado como el estándar de tecnología de red de interconexión más habitualmente empleado en sistemas de computación de alto rendimiento. De hecho, es la tecnología de red empleada en aproximadamente en un 45% de los sistemas de supercomputación de la lista Top500 [2] de noviembre del año 2012). Las razones del éxito de esta tecnología son varias,destacando que los componentes basados en el estándar InfiniBand suelen disponer de varios mecanismos abiertos, cuyo uso y funcionamiento son configurables por el administrador del sistema.

De entre estos mecanismos, resultan especialmente útiles los *Virtual Lanes* (VLs),nombre dado por InfiniBand a su adaptación de los "clásicos" canales virtuales(Virtual Channels [3]), que pueden emplearse para implementar diversos algoritmos y técnicas orientadas a mejorar las prestaciones de la red de interconexión. Por ejemplo, pueden servir como "vías de escape" para garantizar ausencia de interbloqueos en el encaminamiento [4], para garantizar un determinado ancho de banda a ciertos flujos de tráfico en entornos con garantías de calidad de servicio (Quality of Service, QoS) [5], o para implementar esquemas de colas que reduzcan la probabilidad de bloqueo de cabeza de línea (*Head-of-Line blocking*) [6], [7], [8], [9], derivado de situaciones de congestión en la red. En cualquier caso, la gestión de VLs está estrechamente ligada en el estándar a los distintos niveles de servicio (Service Levels, SLs) que pueden asignarse a los distintos flujos de tráfico. Tanto la relación entre VLs y SLs, como la asignación de SLs a flujos de tráfico, son configurables, lo que permite adaptar su uso a las necesidades y características particulares del sistema. En este sentido, el estándar no vincula el uso de VLs y SLs a ninguna topología de red, algoritmo de encaminamiento o tipo de aplicaciones soportadas por el sistema.

Sin embargo, la configuración efectiva de SLs y VLs en *clusters* reales basados en componentes que se ajusten al estándar InfiniBand no resulta trivial. De hecho, el estándar no especifica prácticamente nada respecto a la implementación práctica de dicha configuración, que "de facto" queda en manos de los fabricantes de componentes IB. De cara a aportar alguna ayuda en este sentido, en este artículo se describe una metodología para configurar el uso de SLs y VLs en un *cluster* basado en componentes de red suministrados por Mellanox [10], uno de los principales fabricantes cuyos productos se ajustan al estándar InfiniBand. Téngase en cuenta que la intención principal no es describir cómo implementar una política concreta de asignación de VLs a SLs, o de SLs a flujos de tráfico, sino ofrecer unas pautas generales que faciliten a investigadores y administradores del sistema la implementación de aquellas políticas que deseen incorporar para el uso de SLs y VLs. Por tanto, la metodología descrita a continuación debe considerarse como una ayuda para la gestión de SLs y VLs en sistemas basados en componentes similares a los del *cluster* descrito.

El resto del artículo se organiza de la siguiente forma: en la sección II se revisan los principios básicos sobre la gestión de VLs y SLs según el estándar InfiniBand, y se analiza la aplicación de este estándar en los productos Mellanox. En la sección III se describen las líneas generales de la metodología

[1]Dpto. de Sistemas Informáticos, Universidad de Castilla-La Mancha. Campus universitario, s/n. 02071 Albacete. e-mail:{alvaro.cebrian,jesus.escudero, pedrojavier.garcia,francisco.quiles}@uclm.es.

seguida para configurar el uso de SLs y VLs en un *cluster* basado en componentes Mellanox. Algunos resultados obtenidos en dicho *cluster*, con una configuración de SLs y VLs concreta, se muestran en la sección IV. Finalmente, en la sección V se ofrecen algunas conclusiones.

II. Sistemas Hardware InfiniBand

InfiniBand [1] es una especificación estándar, ampliamente aceptada, que define una serie de normas para el diseño de la arquitectura de una red de interconexión. El hardware IB, cuyo principal fabricante es la empresa Mellanox [10], es capaz de aprovechar al máximo los enlaces conmutados y punto a punto de la red, consiguiendo velocidades de transferencia de datos de hasta 50 Gbps actualmente. La especificación InfiniBand se coordina mediante el consorcio *InfiniBand Trade Association* [11],que incluye empresas como IBM, Intel, Mellanox y HP, entre otras.

Una red InfiniBand se divide en un conjunto de una o varias subredes interconectadas mediante encaminadores (*routers*), y cada subred se compone de un conjunto de nodos finales (de procesamiento o de entrada/salida), interconectados por medio de una serie de conmutadores (*switches*) y enlaces punto a punto. InfiniBand soporta un gran número de topologías de red de interconexión definidas por parte del usuario, y ofrece gran flexibilidad y capacidad de escalabilidad.

Los conmutadores o *switches* son el componente fundamental para el encaminamiento de flujos de tráfico dentro de una subred IB. Los conmutadores tiene varios puertos de entrada salida a través de los cuales se interconectan con otros switches mediante enlaces bidireccionales punto a punto, transportando el tráfico desde los nodos origen a los nodos destino. El estandar define varias tasas de transferencia de los enlaces,que oscilan entre 2,5 Gbps y 50 Gbps en los adaptadores de red Mellanox Connect-IB. La velocidad más baja es de 1x (2,5 Gbps), pero se pueden encontrar velocidades de 4x (10 Gbps), 12x (30 Gbps) o 16x (50 Gbps).

La unidad básica de comunicación en InfiniBand es el mensaje (p.e. en la interfaz de red Connect-IB de Mellanox se manejan mensajes de hasta 1 GByte) que son divididos en unidades de transferencia más pequeñasy fácilmente manejables por la red. En este sentido la unidad máxima de transferencia (MTU) que maneja IB se puede establecer entre los 256 bytes hasta los 4 Kbytes.

En cuanto a los adaptadores de red (del inglés *host channel adapters*, HCAs), se definen como la interfaz de red que se usa en nodos de procesamiento y/o nodos Entrada/Salida. Los HCAs se encargan de generar/consumir tráfico en/de la red y soportan toda la funcionalidad establecida en los drivers IB (los *IB verbs*). Los HCAs pueden tener múltiples puertos, y a cada uno de los cuales se le asigna un identificador de red (*Local IDentifier*, LID) o un rango de LIDs; el número máximo de LIDs se limita a 48000. A cada conmutador también se le asigna un LID. A su vez, cada puerto del HCA tiene su propio conjunto de memorias o *buffers* de transmisión, de modo que cada puerto puede transmitir y recibir tráfico de forma autónoma y concurrente.

Para la lógica de encaminamiento, InfiniBand usa encaminamiento distribuido basado en tablas, en función del LID del nodo destino al que se dirige cada paquete. Cada paquete almacena un LID destino (DLID) en su cabecera, asignado por el HCA del nodo origen al generar dicho paquete. Cuando un paquete se llega a un conmutador, el puerto de salida para dicho paquete se obtiene consultado la tabla de encaminamiento incorporada en el conmutador y que contiene, para cada uno de los DLIDs posibles en la red,el puerto de salida por el cual se han de encaminar los paquetes dirigidos a cada DLID. Las tablas de encaminamiento de los conmutadores IB pueden direccionar hasta 48000 DLIDs. A su vez, la especificación InfiniBand deja la elección del algoritmo de encaminamiento al administrador del sistema.

En cada uno de los puertos de los conmutadores o de los HCAs, IB divide cada *buffer* en un número determinado de *Virtual Lanes*, VLs. El control de flujo entre dos *buffers* pertenecientes a dos conmutadores distintos que están interconectados mediante un enlace punto a punto, se configura a nivel de VL y sigue una política de créditos. InfiniBAnd define hasta 16 VLs, aunque el hardware Mellanox sólo permite hasta 8 VLs. A cada paquete se le asigna un VL determinado, en función del nivel de servicio (*service level*, SL) almacenado en su cabecera por el HCA origen en el momento de la generación de dicho paquete. En concreto, para realizar un mapeo de un paquete a un VL, cada HCA o conmutador utiliza una tabla SLtoVL que consulta cada vez que necesita conocer el VL que se asigna a un paquete, en función de su SL. La especificación propone el uso de los SLs y VLs, en principio, como medio para proporcionar garantías de calidad de servicio (*Quality of Service*, QoS). Sin embargo, aparte de esta funcionalidad, InfiniBand deja muy abierta la implementación de la política para el cálculo del SL correspondiente a un flujo de tráfico, que puede ser definida por el usuario. De hecho, hay numerosas políticas que no asignan SLs en función de las necesidades de QoS, sino en función de otros propósitos, como el control de congestión [12], la tolerancia a fallos [13] o el encaminamiento libre de bloqueos [14].

La política de asignación de SLs y el mapeo de flujos de tráfico a VLs en función de los SLs corre a cargo del llamado *Subnet Manager* (SM). El SM es un ente , que debe estar presente en cada subred, y que se encarga de la configuración y gestión de los enlaces, conmutadores y HCAs, tanto la primera vez que se “enciende” la red, como durante el tiempo en que ésta está en funcionamiento. El SM puede ser un conmutador o un nodo de procesamiento de la red. Las principales tareas del SM son inicializar y configurar la subred (que implica descubrir la topología y asignar LIDs a todos los dispositivos de la misma, configurar las tablas SLtoVL, rellenar la información

de las tablas de encaminamiento), y reconfigurar la red si hay cambios en la topología.

En este artículo se propone es una metodología para incluir en el SM una política orientada a calcular el SL correspondiente a cada flujo de tráfico que se genere en la red. Después de esto el SM rellena las tablas SLtoVL, acorde a esta política y asigna un SL determinado a cada flujo de tráfico. Posteriormente, el SM se podrá consultar para volver a calcular el SL asignado a un determinado flujo de tráfico. Una posibilidad muy común para la implementación del software del SM es usar soluciones de código abierto, como la que proporciona la *Open Fabrics Alliance* (OFA) [15]. En concreto, en se utilizado el software *Open Fabrics Enterprise Distribution v3.5* (OFED 3.5 [16]), que es una compilación de software libre que incluye los drivers para los dispositivos hardware IB y una API para la programación del kernel y de aplicaciones de usuario, además de otras aplicaciones como OpenSM, que implementa el código del SM de InfiniBand. No obstante,Mellanox también proporciona su propia implementación del SM en el paquete MLNX_OFED v2.0 [17], aunque no es código abierto.

En la siguiente sección se describe la propuesta para incluir en OpenSM la gestión de los SLs de una forma eficiente y abierta.

III. Descripción de la Metodología

La metodología propuesta se basa en modificar ciertos módulos del código fuente de OpenSM, para añadir la funcionalidad básica que permita obtener, mediante una consulta al SM, el SL que le corresponde a un flujo de tráfico determinado (según la política de asignación de SLs que desee configurar el administrador). Como se ha mencionado, dicho SL se asigna a los mensajes de dicho flujo de tráfico antes de ser inyectados en la red.Dado que el SM rellena las tablas SLtoVL en su etapa de configuración o reconfiguración, los flujos de datos se mapean de forma automática a los VLs una vez que dicho tráfico se inyecta en la red con su SL correspondiente.

A continuación, se describen las modificaciones efectuadas en OpenSM, se propone cómo configurar las aplicaciones para que obtengan el SL correspondiente a un flujo de tráfico, mediante consultas a OpenSM y, finalmente, se detalla un ejemplo específico de ejecución de OpenSM.

A. Ideas generales

De entre las aplicaciones paralelas que se pueden ejecutar en un *cluster* real, un paradigma representativo es el paso de mensajes. En este ámbito, el uso de la interfaz MPI está muy extendido en el desarrollo de aplicaciones paralelas. Además, la implementación de la interfaz de MPI en código abierto que ofrece el *driver* OpenMPI [18] también es ampliamente conocida y utilizada. Uno de los aspectos interesantes de este *driver* es que ofrece la posibilidad de activar el uso de SLs y su obtención (cálculo para cada flujo de datos) dinámica por medio de consultas al SM de InfiniBand, lo que sin duda es interesante para esta metodología.En adelante,se mostrará cómo el *driver* OpenMPI gestiona la obtención de SLs.

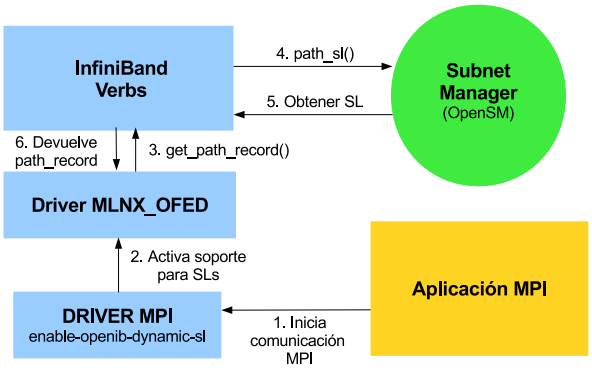


Fig. 1. Diagrama de obtención del SL en una aplicación MPI

El uso de SLs en las comunicaciones derivadas de aplicaciones paralelas, supone la intervención de una serie de módulos, tal y como se muestra en la Fig.1. Cuando una aplicación MPI desea establecer una conexión con otra aplicación, se comunicará con el *driver* MPI. Si el driver MPI está configurado para la obtención dinámica del SL a asignar a la comunicación, solicitará al *driver* IB, instalado con MLNX_OFED [17] que active el soporte para la obtención del SL. Seguidamente, el driver MLNX_OFED necesita conocer la información relativa a la conexión que desea establecer la aplicación MPI (en la que se incluye el SL), y esa información se obtiene mediante una llamada a una interfaz de programación (API) que ofrece el propio driver MLNX_OFED: los *IB verbs*. Es importante destacar que cualquier aplicación paralela que no siga el estándar MPI puede beneficiarse de la obtención dinámica de su SL, siempre que se lo solicite a la API del *driver* MLNX_OFED. Por tanto, dicho *driver* usa los *IB verbs* para realizar la petición del SL al *Subnet Manager*. Una vez obtenido el SL, los *IB verbs* devuelven al *driver* MLNX_OFED un objeto *path_record* que contiene información sobre la conexión que quiere establecer la aplicación MPI, donde se incluye el SL que se asignará a cada mensaje MPI para, posteriormente, inyectarse en la red.

B. Modificación de OpenSM

Para la implementación de este modelo, se ha realizado una modificación sobre el código de OpenSM contenido en OFED v3.5. Para ello, se ha considerado oportuno seguir la misma filosofía que para la creación de un nuevo algoritmo de encaminamiento. Obviamente, no se ha implementado dicho algoritmo, si no que se ha añadido la funcionalidad necesaria al código fuente de un algoritmo de encaminamiento para que incorpore la gestión de los SLs.

En concreto, la implementación de un nuevo algoritmo de encaminamiento supone la creación de un fichero que contenga las funciones que serán utilizadas por el SM para poder realizar el encaminamiento. En OFED, estos ficheros tienen el nombre $osm\_ucast\_\langle nombre\_del\_algoritmo \rangle$. Dentro de este

fichero es donde se han de incluir las funciones necesarias para el cálculo del SL a utilizar por cada comunicación que inicie una aplicación paralela, y también para obtener dicho SL, mediante una consulta al SM, cuando se requiera.

Antes de describir la implementación, conviene mencionar una función que debe estar presente en todos los ficheros que implementen un algoritmo de encaminamiento. Esta función suele tener el nombre *osm_ucast_⟨nombre_del_algoritmo⟩_setup* y se invoca por parte de OpenSM al iniciarse su ejecución. Su misión es inicializar la estructura de datos *struct osm_routing_engine*, que se pasa como parámetro, y que contiene referencias a las funciones específicas a las que hay que invocar para realizar una determinada acción, como rellenar las tablas de encaminamiento, en función del algoritmo de encaminamiento utilizado.

Para almacenar la información de los SLs, se propone la utilización de una tabla global, incluida en la estructura *struct osm_routing_engine* que será rellenada, al inicializarse la red, con el SL a utilizar por cada comunicación solicitada por la aplicación paralela.

```
typedef struct routing_table {
    unsigned sl;
    unsigned sl_set;
} routing_table_t;
```

En este sentido, se puede diseñar un procedimiento que, al iniciarse la red, rellene esta tabla. La forma en la cual se rellena esta tabla se deja a criterio del programador, pero si se desea, pueden tomarse como ejemplo las implementaciones de vFtree [12], DFSSSP [13] o LASH [14], éstas dos últimos incluidas en OpenSM. Una vez se ha implementado esta función, deberá incluirse una llamada en alguna de las funciones que son invocadas por OpenSM al iniciar su ejecución.La mejor opción es incluir la llamada dentro de la función que apunta a *build_lid_matrices* o *ucast_build_fwd_tables* y que se rellenan en *osm_ucast_⟨nombre del algoritmo⟩_setup*.

A continuación, se debe implementar la función que permita obtener el SL (*funcion_get_SL*()), en función de la información que se necesite para dicho cálculo. En el caso de vFtree se utiliza el LID del nodo origen (SLID) y el DLID del nodo con el que se establece la comunicación. Esta función se ejecuta cada vez que la aplicación necesita establecer una comunicación entre dos elementos de la red, consulta la tabla previamente rellenada y devuelve el SL a utilizar.

Finalmente, se debe incluir un enlace a dicha función dentro del objeto *osm_routing_engine*, que contiene un enlace a la función que ejecutara OpenSM cuando deba de obtener el SL. El lugar en el que realizar esta asignación es en el fichero *osm_ucast_⟨nombre del algoritmo⟩_setup*. El código a incluir es:

```
r->path_sl = funcion_get_SL;
```

C. *Configuración del* driver *OpenMPI*

Para que las aplicaciones MPI puedan realizar peticiones a OpenSM y así obtener el SL es necesario configurarlas. La forma que se propone es mediante el uso del *driver* OpenMPI. Para ello, tras la instalación de OFED se ha de compilar el *driver* OpenMPI, de modo que pueda comunicarse con los drivers IB que proporciona MLNX_OFED.

La forma de configurar OpenMPI para que se puedan realizar consultas a OpenSM a través de los drivers IB es la siguiente:

```
./configure --with-openib --enable-openib-dynamic-sl
 --enable-openib-rdmacm --with-psm
```

Opcionalmente, se puede dar soporte para PBS con el parámetro `--with-tm=directorio\_src\_PBS`.La instalación de OpenMPI se completa con:

```
make all install
```

D. *Ejecución de aplicaciones*

Para la ejecución de aplicaciones paralelas que hagan uso de la implementación descrita, previamente se deberá ejecutar el código modificado de OpenSM en algún nodo de la red. La forma de lanzar OpenSM para que pueda aceptar peticiones de las aplicaciones es:

```
opensm -R < nombre del algoritmo > -Q -Y <fichero qos>
-f <fichero log>
```

Los *flags* -Q y -Y permiten activar la calidad de servicio (QoS), que por defecto está desactivada, y especificar, a través de un fichero, la configuración de QoS a utilizar. La parte más importante en este fichero es *qos_sl2vl* en la que se le indica, para cada SL, el VL al que se mapeará. Un ejemplo de fichero QoS es el siguiente:

```
qos TRUE
qos_max_vls 8
qos_high_limit 4
qos_vlarb_high 0:64,1:64,2:64,3:64,4:64,5:64,6:64,7:64
qos_vlarb_low 0:4,1:4,2:4,3:4,4:4,5:4,6:4,7:4
qos_sl2vl 0,1,2,3,4,5,6,7,0,1,2,3,4,5,6,7
```

Una vez que OpenSM está en ejecución, podrá recibir peticiones para la obtención del SL. Para lanzar una aplicación paralela que use el driver OpenMPI se ejecutará el siguiente comando:

```
mpirun -display-allocation -display-map -np 64
-machinefile<fichero_maquinas> --mca btl openib,self,sm
--mca btl_openib_ib_path_record_service_level 1
<aplicacion>
```

El `fichero_maquinas` contiene, para cada una de las máquinas que se comunicarán utilizando OpenMPI, una línea con la dirección de dicha máquina. Un ejemplo de fichero de máquinas puede ser:

```
maquina01.i3a.uclm.es
maquina02.i3a.uclm.es
```

A su vez, el parámetro `--mca` activa los módulos del *driver* MPI necesarios para la comunicación del con el driver MLNX_OFED, y el parámetro `<aplicacion>` indica la ruta del ejecutable de la aplicación.

IV. Evaluación del rendimiento

En esta sección se analizan los resultados obtenidos en un *cluster* IB real. Inicialmente, se describe dicho *cluster*, se describe la confi-guración de los experimentos y se muestran y analizan los resultados.

A. *El* cluster *IB CELLIA*

Para evaluar el rendimiento alcanzado gracias al uso de SLs, los experimentos se han realizado en un *cluster* IB real, llamado *CELLIA* (*Cluster for the Evaluation of Low-Latency Interconnection Architectures*) formado por dieciséis nodos de cómputo. Cada nodo es un servidor Fujitsu PRIMERGY RX220 que cuenta con una tarjeta de red (HCA) Mellanox ConectX3 MCX353A-QCBT QSFP, con conexión QDR IB y 10 GbE, gracias a VPI, conectada al bus PCI-EXpress x8. Para la interconexión de los nodos de procesamiento se cuenta con doce conmutadores Mellanox IS5022 de 8 puertos, cada uno con una capacidad bidireccional de 40 GB/s y soporte para 9 VLs: ocho para datos y uno para gestión. Estos nodos se encuentran conectados formando una topología fat-tree (k-ary 3-tree [19]), tal y como puede verse en la Fig.2. Para la gestión del encaminamiento, la red utiliza el algoritmo D-mod-K [20].

Cada uno de los nodos de procesamiento cuenta con un procesador 64-bit Dual-Core AMD Opteron y 4 GB de memoria RAM. Además, todos tienen instalado el sistema operativo CentOS 6.4 con la versión del núcleo 2.6.32-358.el6.x86_64. Así mismo, cada nodo tiene instalado el paquete de software Mellanox OFED 2.0 (MLNX_OFED) [17], una adaptación de uso propietario del software OFED [16].

El *Subnet Manager* (SM) de la red es una modificación de OpenSM 3.3.16 al que se le ha incluido el algoritmo vFtree [12]. En el experimento se miden las prestaciones del *cluster CELLIA* cuando vFtree está activado, y cuando no lo está (cuando sólo se ejecuta el algoritmo de encaminamiento D-mod-K). Para poder medir la eficacia en el uso de VLs, se ha utilizado la suite *Perftest*, incluida dentro de OFED 2.0[16], que permite realizar tests ping-pong y medir ciertos parámetros, como la latencia y la productividad alcanzada por los flujos de tráfico generados.

B. *Descripción de la prueba*

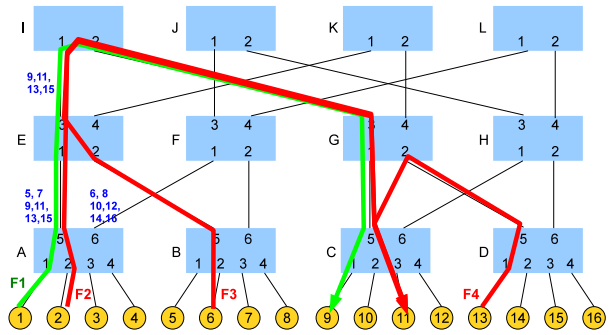


Fig. 2. Escenario de creación de un hot-spot

El experimento consiste en la generación de patrones de tráfico sintético de forma que se consiga congestionar algunos nodos de la red y, de este modo, poder medir la efectividad del uso de SLs para reducir los efectos de la congestión. Para la prueba se ha utilizado el algoritmo de encaminamiento Ftree (es decir sin soporte para SLs) y su adaptación para el uso de SLs, vFtree. La Fig. 2 muestra la configuración del patrón de tráfico en el *cluster CELLIA*. Se ha definido un conjunto de flujos de tráfico sintético desde un origen fijo a un destino fijo ($\{13 \rightarrow 11, 1 \rightarrow 9, 6 \rightarrow 11, 2 \rightarrow 11\}$), que generan un *hot-spot* en el nodo 11. Por contra, el flujo $\{1 \rightarrow 9\}$ es un flujo víctima, ya que se ve afectado por la congestión, aunque no contribuye a ella. Téngase en cuenta que la carga introducida en cada enlace por *Perftest* es de 1.25 GB/s, la máxima permitida, puesto que la configuración utilizada para cada enlace es QDR 4X que admite hasta 10 Gb/s.

C. *Resultados de la prueba*

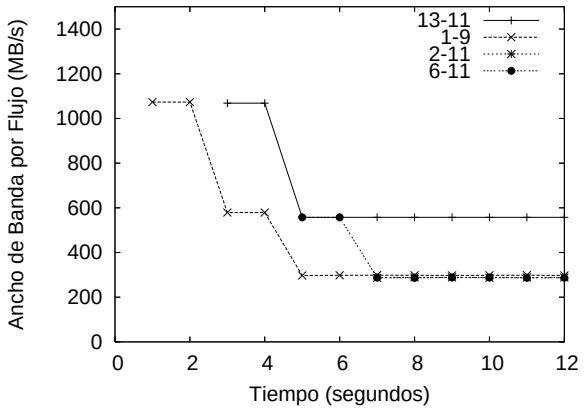


Fig. 3. Resultados de la prueba utilizando Ftree

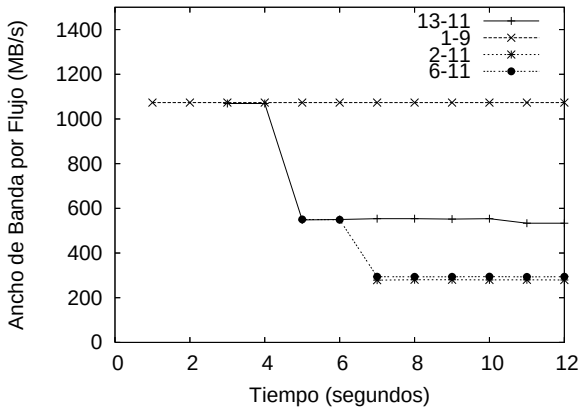


Fig. 4. Resultados de la prueba utilizando vFtree

La Fig.3 muestra el ancho de banda alcanzado por cada uno de los tráficos utilizando un único SL (Ftree). Como se puede observar, al utilizar todos los flujos de tráfico el mismo SL, la productividad de los flujos que contribuyen a la creación del *hot-spot* se ve reducida hasta los 250 MB/s, incluido el flujo víctima ($\{1 \rightarrow 9\}$) que comparte el mismo VL que

los flujos congestionados. Esta reducción del ancho de banda del flujo víctima se puede observar en los momentos en los que se incian los flujos de tráfico $\{13 \rightarrow 11\}$, en el segundo 3, y el $\{6 \rightarrow 11\}$,en el segundo 5. En la Fig.4 se observa que, para el caso de vFtree, al utilizar un VL para el flujo víctima distinto al que utilizan los flujos que contribuyen a la congestión en el nodo 11, el ancho de banda alcanzado por dicho flujo víctima se incrementa hasta los 1150 MB/s (es decir un 290%), mientras que ello no afecta al ancho de banda del resto de flujos que comparten el ancho de banda del enlace (establecido a 1,25 GB/s). Por tanto, los resultados de esta prueba tan básica permiten mostrar que el uso de los SLs es un elemento efectivo para la reducción de la congestión en la red, permitiendo mejorar el rendimiento alcanzado por ésta.

V. Conclusiones

El estándar InfiniBand es *de facto* uno de los más usados en los actuales sistemas de computación de altas prestaciones, como se puede comprobar en la lista Top500. En su última edición, InfiniBand estaba presente en aproximadamente el 45% de los sistemas. De entre las múltiples ventajas del estándar, destaca la posibilidad de configurar los niveles de servicio (SLs) y el uso de los canales virtuales (VLs) de forma abierta, y según las necesidades de los administradores de *cluster* a la hora de maximizar el rendimiento de la red de interconexión. Algunos ejemplos de dichas necesidades pueden ser el uso de encaminamiento libre de bloqueos, la calidad de servicio (QoS) o el control de la congestión.

En este artículo se ha propuesto una metodología para la gestión eficiente y abierta de los SLs, que serán asignados a los flujos de tráfico de manera, que dependiendo del SL de cada flujo de tráfico, se asignará a un determinado VL. Para realizar esto, se ha modificado el *Subnet Manager* (SM) de InfiniBand para que sea capaz de obtener el SL apropiado a un flujo de tráfico, y se ha dado servicio al *driver* OpenMPI para las aplicaciones MPI puedan obtener de forma dinámica el nivel de servicio correspondiente, mediante una consulta al *Subnet Manager*. Los resultados obtenidos mediante experimentos en un *cluster* IB real, basado en hardware del fabricante Mellanox, demuestran que con una política sencilla de asignación de flujos de tráfico a niveles de servicio, se obtiene una ganancia de prestaciones de aprovecharse un 290%, lo que justifica el uso de esta metodología.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el MINECO, así como por la Comisión Europea a través de los fondos FEDER, mediante los proyectos TIN2012-38341-C04-04. Además, también ha sido parcialmente financiado por la JCCM, mediante el proyecto POII10-0289-3724.

Referencias

[1] InfiniBand Spec. 1, *InfiniBand architecture specification volume 1. Release 1.2.1*, Nov. 2007.

[2] Top 500 List, ," Web Page at: `http://www.top500.org`, 2012.

[3] William Dally, "Virtual-Channel Flow Control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 194–205, 1992.

[4] J. Duato, Sudhakar Yalamanchili, and Lionel Ni, *Interconnection Networks An Engineering Approach*, Morgan Kaufmann, revised edition edition, 2003.

[5] Alejandro Martínez, Pedro Javier García, Francisco José Alfaro, José L. Sánchez, Jose Flich, Francisco J. Quiles, and José Duato, "A Switch Architecture Guaranteeing QoS Provision and HOL Blocking Elimination," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 1, pp. 13–24, 2009.

[6] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queuing on a space-division packet switch," *IEEE Transactions on Communications.*, vol. COM-35, pp. 1347–1356, 1987.

[7] M. Jurczyk and T. Schwederski, "Phenomenon of Higher Order Head-of-Line Blocking in Multistage Interconnection Networks under Nonuniform Traffic Patterns," *IEICE Transactions on Information and Systems*, vol. E79-D, no. 8, pp. 1124–1129, Aug. 1996.

[8] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, November 1993.

[9] William Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *Proc. of 6th Hot Interconnects*, 1998, pp. 41–50.

[10] Mellanox Technologies, ," http://www.mellanox.com/.

[11] The InfiniBand Trade Association, ," Web Page at: `http://www.infinibandta.org`, 2012.

[12] Wei Lin Guay, Bartosz Bogdanski, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie, "vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion," in *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium*, Yuan Xin, Ed. 2011, pp. 197–208, IEEE Computer Society Press.

[13] J. Domke, T. Hoefler, and W. Nagel, "Deadlock-Free Oblivious Routing for Arbitrary Topologies," in *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. May 2011, pp. 613–624, IEEE Computer Society.

[14] Tor Skeie, Olav Lysne, and Ingebjørg Theiss, "Layered Shortest Path (LASH) Routing in Irregular System Area Networks," in *IPDPS*, 2002, In proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), 15-19 April 2002, Fort Lauderdale, FL, USA,.

[15] The Open Fabrics Alliance, ," Web Page at: `https://www.openfabrics.org`, 2013.

[16] The Open Fabrics Enterprise Distribution 3.5, ," Web Page at: `http://www.openfabrics.org/downloads/OFED/ofed-3.5/OFED-3.5.tgz`, 2013.

[17] Mellanox Technologies, ," Web Page at: `http://www.mellanox.com/page/products\_dyn?product\_family=26\&mtag=linux\_sw\_drivers`, 2013.

[18] Open MPI: Open Source High Performance Computing, ," Web Page at: `http://www.open-mpi.org/`, 2013.

[19] F. Petrini and M. Vanneschi, "k-ary n-trees: High Performance Networks for Massively Parallel Architectures," in *Proceedings of International Parallel Processing Symposium*, 1997, pp. 87–93.

[20] Eitan Zahavi, Gregory Johnson, Darren J. Kerbyson, and Michael Lang, "Optimized InfiniBandTM fat-tree routing for shift all-to-all communication patterns," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 217–231, 2010.

Evaluación de Prestaciones

Evaluación de Java para Computación de Propósito General en GPU

Jorge Docampo, Sabela Ramos, Guillermo L. Taboada, Roberto R. Expósito, Juan Touriño y Ramón Doallo[1]

Resumen— **El uso de procesadores many-core como aceleradores de computación se ha visto incrementado debido a su gran capacidad para mejorar el rendimiento de cargas de trabajo altamente paralelas. La Computación de Propósito General en GPU (GPGPU) ha permitido que las unidades gráficas puedan ser empleadas con éxito en la mejora del rendimiento de aplicaciones con altos requisitos computacionales fuera del ámbito gráfico, presentando características que las hacen adecuadas para numerosas cargas de trabajo en Computación de Altas Prestaciones. Mientras que las principales librerías desarrolladas para explotar la masiva capacidad paralela de las GPUs están orientadas a lenguajes como C/C++, existen grandes esfuerzos enfocados a extender este soporte a otros lenguajes. De entre todos ellos, Java destaca por ser uno de los más extendidos y hay múltiples proyectos que permiten que Java pueda tomar provecho de la computación GPGPU. En este escenario, este artículo presenta una evaluación actualizada de las soluciones más relevantes que permiten explotar la computación GPGPU en Java.**

Palabras clave— **Java, Computación de Propósito General en GPU (GPGPU), jCuda, Aparapi, Computación de Altas Prestaciones.**

I. Introducción

El interés en explotar las Unidades de Procesamiento Gráfico (Graphical Processing Units, GPU) para cargas computacionales no gráficas está motivado por su arquitectura masivamente paralela, su capacidad de cómputo en punto flotante y su alto ancho de banda de memoria. En consecuencia, y dado que las GPUs son adecuadas para manejar cargas de trabajo paralelas típicas de aplicaciones científicas, la comunidad de la Computación de Altas Prestaciones (High Performance Computing, HPC) ha adoptado a las GPU como aceleradores many-core, y la computación de Propósito General en GPU (GPGPU) es cada vez más popular. La lista TOP 500 de los supercomputadores más potentes [1] refleja esta tendencia y, del 11,6% de los supercomputadores que usan aceleradores many-core (que eran sólo el 3,8% hace un año), el 79,3% usan GPUs.

Los principales modelos de programación para computación GPGPU fueron desarrollados como librerías para ser usadas desde C/C++. Sin embargo, ya que Java es uno de los lenguajes más relevantes, tanto en la industria como en el mundo académico, varios proyectos han surgido para hacer que Java pueda beneficiarse de la descarga de trabajo en la GPU. Java es también una alternativa en auge para la programación HPC [2], ya que su rendimiento ha mejorado en los últimos años, reduciendo la diferencia que había con respecto a lenguajes nativos compilados como C o Fortran, y presenta características que lo convierten en una opción interesante, como su facilidad de uso, orientación a objetos, soporte de red y multi-threading nativos, gestión automática de memoria, independencia de la plataforma, portabilidad y una amplia comunidad de desarrolladores.

Este artículo tiene el propósito de proporcionar una evaluación de referencia de los proyectos que permiten el aprovechamiento de la computación GPGPGU en Java, recopilando de entre las soluciones existentes las más activas y relevantes. Además, este trabajo incluye una evaluación y análisis comparativo de sus características y rendimiento.

El resto del artículo se organiza de la siguiente manera. La Sección II proporciona un análisis de los proyectos más relevantes para computación GPGPU en Java, la Sección III incluye una evaluación comparativa del rendimiento de las soluciones expuestas y la Sección IV presenta las principales conclusiones.

II. Soluciones Actuales para Computación GPU de Propósito General en Java

La arquitectura masivamente paralela de las GPUs, unida a su capacidad de cómputo en punto flotante, ha motivado el crecimiento de la computación de Propósito General en GPU (GPGPU) [3]. Las principales librerías que soportan el uso de las GPUs como aceleradores many-core en cargas de trabajo no gráficas son CUDA (Compute Unified Device Architecture) [4] y OpenCL (Open Computing Language) [5], provocando el incremento de la adopción de las GPUs como aceleradoras en entornos HPC [6], ya que muchas aplicaciones científicas presentan un enorme grado de paralelismo que permiten aprovechar las características de las GPUs.

Estos modelos GPGPU están destinados a ser usados como extensiones de códigos C/C++, mientras que otros lenguajes como Java deben recurrir a wrappers (en el caso de Java a través de Java Native Interface, JNI) para poder explotar las GPU como aceleradores. Sin embargo, el interés en Java para HPC ha ido creciendo debido a su rendimiento cada vez mayor y características como los soportes de multithreading y de red en el core del lenguaje [2]. Por lo tanto, diversos proyectos han surgido para permitir el uso de Java en la programación GPGPU. De hecho OpenJDK, AMD y Oracle están trabajando en el proyecto Sumatra [7] para dar soporte desde la JVM al uso de Java en GPUs. Sin embargo, este

[1]Grupo de Arquitectura de Computadores, Depto. de Electrónica y Sistemas, Universidade da Coruña, E-mails: {jorge.docampo,sramos,taboada,rreye,juan,doallo}@udc.es

proyecto no está disponible para su evaluación, así que nos centraremos en otras soluciones activas hoy en día.

Entre todas ellas, podemos distinguir dos acercamientos, aquellos que proporcionan acceso en Java a funciones y lenguajes de bajo nivel (CUDA, OpenCL) mediante JNI (Java bindings) o aquellos con un API cercana al usuario que abstrae la programación GPU y proporcionan un sistema que en tiempo de ejecución traduce el bytecode de Java en CUDA u OpenCL de manera transparente. Mientras que los primeros están orientados a obtener mayor rendimiento, aumentar la programabilidad hace posible encontrar un equilibrio entre alto rendimiento y productividad.

La Tabla I resume los proyectos más relevantes de computación GPGPU en Java indicando a mayores la librería nativa subyacente. En cuanto a los proyectos relacionados con CUDA, JCUDA [8] proporciona su propia interfaz para invocar ciertas funciones CUDA y kernels desarrollados por el usuario. Sin embargo, no está incluído en el grupo de proyectos *user-friendly* puesto que requiere habilidades de programación a bajo nivel y cierto conocimiento de las funciones CUDA.

jCuda [9] es el proyecto Java GPGPU más activo. Proporciona un wrapper directo al API y al driver de CUDA 4.2, permitiendo interacción directa con el dispositivo, incluyendo la gestión de memoria y permitiendo el lanzamiento de kernels CUDA desde Java. La principal fortaleza de este proyecto es que proporciona soporte a bibliotecas CUDA optimizadas como CUBLAS (CUDA Basic Linear Algebra Subprograms), CUFFT (CUDA Fast Fourier Transforms), CUDPP (CUDA Data Parallel Primitives), CURAND (CUDA Random Number Generation), CUSPARSE (CUDA Sparse Matrix) y NPP (NVIDIA Performance Primitives). El API de jCuda consiste en un conjunto de métodos estáticos que son muy similares a las funciones de biblioteca nativa, ya que el objetivo de jCuda es mantener el API tan cerca del original como sea posible, incluyendo también funciones con el fin de utilizar kernels definidos por el usuario en lenguaje CUDA, así como funciones de manejo de punteros.

En cuanto a las soluciones *user-friendly*, Java-GPU [10] introduce directivas para descargar código Java en la GPU, mientras que Rootbeer [11] proporciona una API específica de alto nivel para Java y traduce el bytecode generado a CUDA.

TABLA I
SOLUCIONES DISPONIBLES PARA LA COMPUTACIÓN GPGPU EN JAVA

	Java bindings	**User-friendly**
CUDA	JCUDA [8] jCuda [9]	Java-GPU [10] Rootbeer [11]
OpenCL	JOCL [12] JogAmp JOCL [14]	Aparapi [13]

Las soluciones Java bindings con OpenCL incluyen JOCL [12] y JogAmp JOCL [14]. La principal diferencia entre ellos es que, mientras el primero proporciona soporte para OpenCL 1.2, el último sólo maneja OpenCL 1.1.

Por último, Aparapi [13], soportado por AMD, es el proyecto más actual y activo de Java OpenCL y proporciona soporte para OpenCL 1.2. Al programador de Aparapi se le proporciona un API de alto nivel para expresar cargas de trabajo paralelas en Java, pudiendo obviar todos los detalles de implementación relacionados con la GPU. Sin embargo, es necesario conocer algunas nociones de como funciona la GPU para poder realizar una distribución de trabajo eficiente y obtener un aumento de rendimiento significativo, pero no es necesario en absoluto ningún conocimiento de OpenCL. Aparapi, en tiempo de ejecución, traducirá estas cargas de trabajo paralelas a OpenCL y las descargará en la GPU o en un pool de threads. Si por cualquiera razón el soporte para OpenCL no está disponible se ejecutará en un pool de threads de Java.

III. EVALUACIÓN DEL RENDIMIENTO

En esta sección se evalúa el rendimiento de dos proyectos Java GPGPU representativos, seleccionados por ser los más activos y representativos: jCuda, entre las implementaciones directas de wrappers sobre librerías nativas, y Aparapi, entre las enfocadas a *user-friendly*. Por otro lado, el hecho de que jCuda esté basada en CUDA y Aparapi en OpenCL, es especialmente relevante. Trabajos previos [15] que evalúan CUDA y OpenCL en la plataforma de pruebas experimental usada en esta sección han demostrado que CUDA y OpenCL son capaces de proporcionar aproximadamente el mismo rendimiento. Por lo tanto, la implementación eficiente de un código dado es la razón principal de las diferencias de rendimiento entre CUDA y OpenCL.

A. Configuración Experimental

La plataforma de pruebas usada para la evaluación de rendimiento es un servidor x86_64 con las siguientes características:

TABLA II
DESCRIPCIÓN DE LA PLATAFORMA DE PRUEBAS

CPU	1 × Intel(R) Xeon hexacore X5650 @ 2.67GHz
Rend. CPU	64.08 GFLOPS DP (10.68 GFLOPS DP per core)
GPU	1 × NVIDIA Tesla "Fermi" M2050
Rend. GPU	515 GFLOPS DP
Memory	12 GB DDR3 (1333 MHz)
OS	Debian GNU/Linux, kernel 3.2.0-3
CUDA	version 4.2 SDK Toolkit
JDK version	OpenJDK 1.6.0_24

La evaluación de GPGPU en Java ha sido realizada empleando kernels GPGPU sintéticos, fragmentos de código que representan el núcleo de distintas aplicaciones HPC ampliamente extendidas (e.g., kernel de multiplicación de matrices). Los kernels sintéticos empleados para la evaluación de GPGPU en Java han sido seleccionados de entre los diponibles en la suite de benchmarks SHOC (Scalable HeterOgeneous Computing) [16] y ejemplos correspondientes a la distribución JAVA-GPU. Por otro lado, la suite SHOC determina el rendimiento computacional del sistema con ayuda de kernels de aplicación. La Tabla III presenta los tres kernels sintéticos seleccionados. Hemos desarrollado implementaciones en Java, JCuda y Aparapi, las cuales operan sobre la CPU, y en CUDA y OpenCL en la GPU, respectivamente, permitiendo el análisis comparativo de su rendimiento.

TABLA III
KERNELS SINTÉTICOS SELECCIONADOS

Kernel	Suite	Medida	Descripción
GEMM	SHOC	GFLOPS	Multiplicación de Matrices
Stencil2D	SHOC	Time & GFLOPS	Un cálculo de stencil-2D de nueve puntos
FFT	SHOC	Time & GFLOPS	Transformada Rápida de Fourier

B. Análisis de Resultados Experimentales

Los resultados de rendimiento mostrados en las Figuras 1-3 son el resultado del promedio de 5 ejecuciones, observando poca variabilidad entre las medidas.

La Figura 1 presenta los resultados del rendimiento del kernel de multiplicación de matrices obtenidos en términos de GFLOPS, de las implementaciones en CUDA, jCuda, Aparapi y Java. CUDA consigue el mayor rendimiento, seguido de jCuda. Como ambas se basan en la misma rutina de multiplicación de matrices de CUBLAS, la diferencia de rendimiento entre ellas se debe a la sobrecarga de los movimientos de datos entre Java y la GPU. De hecho, jCuda sufre una penalización superior para el tamaño de problema 2048x2048 debido a la alta sobrecarga de inicialización de la copia de datos pero, a medida que aumenta el tamaño del problema, jCuda es capaz de obtener en torno a un 85-90% del rendimiento de CUDA.

Como las JVM estándar no son compatibles con las GPU directamente, los resultados de Java se han obtenido en la CPU, ejecutando un código Java secuencial puro (sin depender de métodos nativos) de multiplicación de matrices. Este código obtiene en torno a 1,3-1,6 GFLOPS tanto para simple como doble precisión. Aunque Java sería capaz de lograr un mayor rendimiento llamando a cualquier biblioteca BLAS con bindings a Java (por ejemplo, Intel Math Kernel Library, MKL), se opta por utilizar como referencia un código Java estándar y totalmente portable.

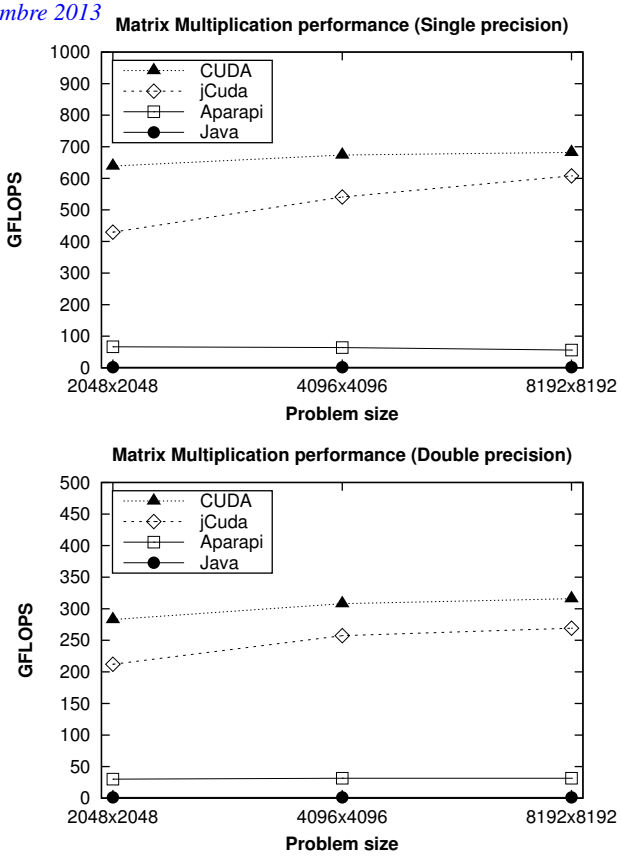
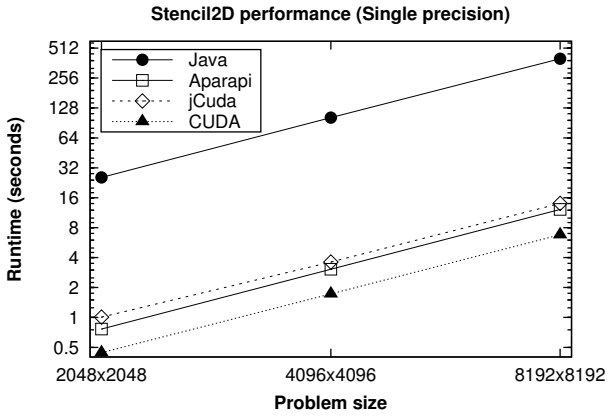


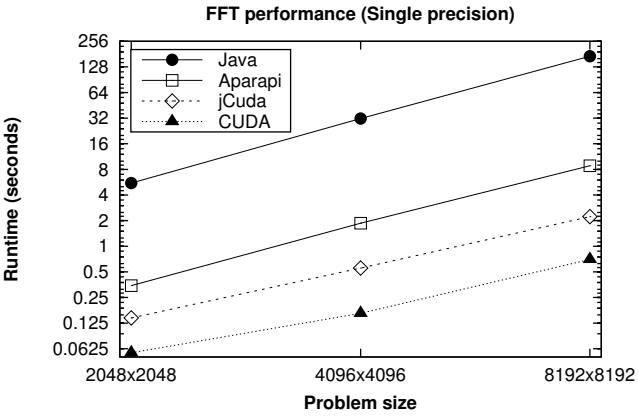
Fig. 1. Rendimiento del kernel de Multiplicación de Matrices

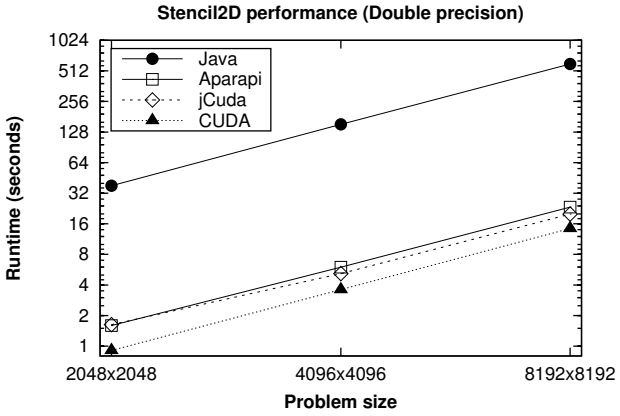
Por último, se introdujo Aparapi en la implementación Java del kernel estándar y los beneficios de rendimiento observados son muy importantes, hasta 40 veces más rápido para las ejecuciones de simple precisión (1,6 GFLOPS en Java y hasta 64 GFLOPS en Aparapi) y hasta 23 veces más rápido para las operaciones de doble precisión (1,3 GFLOPS en Java y hasta 31 GFLOPS en Aparapi). En el caso de que Aparapi no encontrase una GPU en el sistema, se ejecutaría el código utilizando la CPU, por lo que la portabilidad no está comprometida con esta solución.

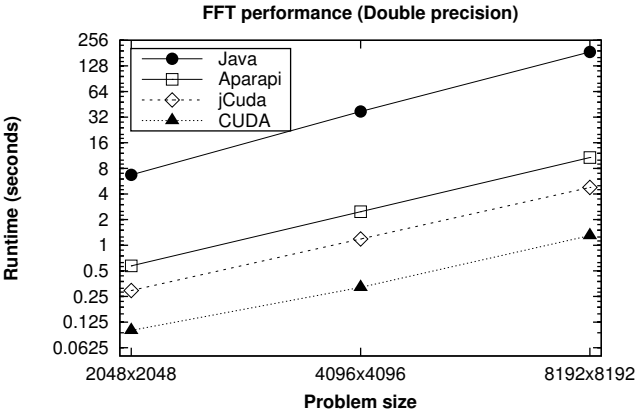
La Figura 2 presenta los resultados del kernel del Stencil2D, en términos de Tiempo (segundos) y GFLOPS, de las implementaciones en CUDA, jCuda, Aparapi y Java. Una vez más, CUDA alcanza el máximo rendimiento y Java, en CPU, es alrededor de 40-60 veces más lento. Sin embargo, jCuda y Aparapi son capaces de alcanzar alrededor del 50-75% del rendimiento nativo en CUDA, acelerando hasta en 33 veces el rendimiento de Java.

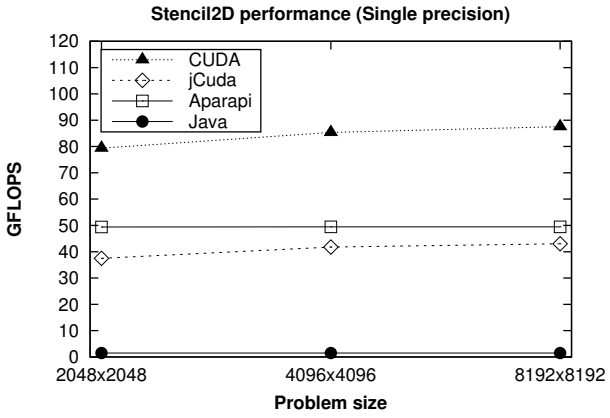
Aquí, jCuda no usa una biblioteca CUDA, por lo que su rendimiento no es tan cercano a CUDA como en el kernel de multiplicación de matrices. Además, como tanto jCuda como Aparapi aplican el mismo algoritmo, ambos logran resultados similares. Sin embargo, la productividad de Aparapi es mucho mayor, ya que ha sido mucho más fácil desarrollar el código de Aparapi que el de jCuda. Este menor tiempo de desarrollo y el significativo aumento de veloci-

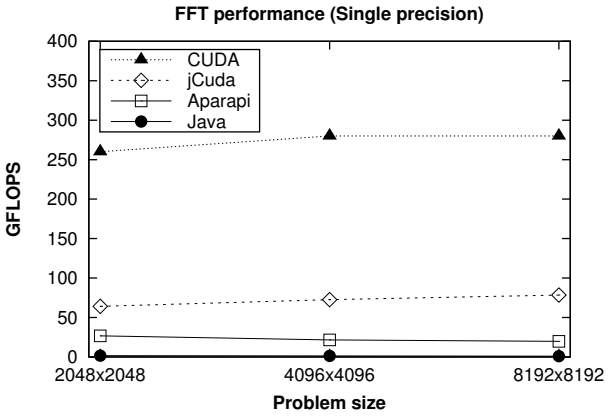


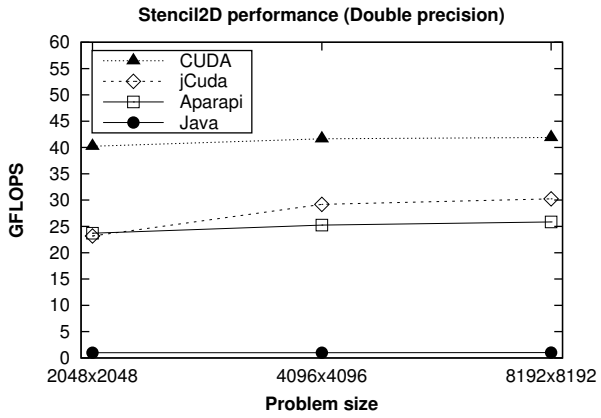












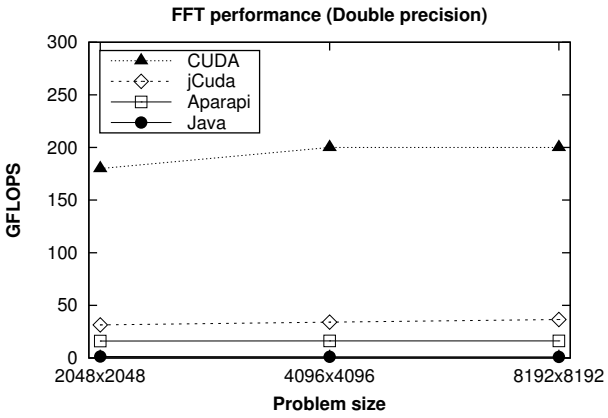


Fig. 2. Rendimiento del kernel Stencil 2D

Fig. 3. Rendimiento del kernel FFT

dad logrado sugieren que Aparapi es la mejor opción Java GPGPU cuando jCuda no puede basarse en una biblioteca CUDA optimizada como CUBLAS o CUFFT.

La Figura 3 presenta los resultados del kernel de la FFT, en términos de Tiempo (segundos) y GFLOPS, de las implementaciones en CUDA, jCuda, Aparapi y Java. En este caso, jCuda se basa en la CUFFT pero el kernel no es tan computacionalmente intensivo como la multiplicación de matrices (este algoritmo FFT tiene una complejidad de $nlog(n)$, mientras que la multiplicación de matrices tiene una complejidad de $O(n^3)$). En consecuencia, las tranferencias de datos Java-GPU tienen mayor impacto en el rendimiento global de jCuda, que alcanza alrededor del 25-30% del rendimiento bruto en CUDA.

La FFT en Aparapi es alrededor de 20 veces más rápida que Java en la CPU pero, al mismo tiempo, es alrededor de 2-3 veces más lenta que jCuda y más de un orden de magnitud más lenta que CUDA. Sin embargo, para los programadores Java, Aparapi se presenta como una opción portable y productiva para acelerar aplicaciones estándar en Java en presencia de GPU. Sin embargo, cuando el rendimiento es crítico, jCuda (e incluso escribir métodos nativos en CUDA, accesibles por JNI) es el camino a seguir, a pesar del coste que supone la pérdida de portabilidad y la necesidad de un mayor tiempo para lograr la solución.

IV. Conclusiones

En este artículo se han presentado y analizado las bibliotecas más relevantes que permiten el uso de Java en la computación GPGPU. Después de seleccionar las más representativas, jCuda y Aparapi, la evaluación de su rendimiento ha demostrado que, mientras Aparapi representa un equilibrio adecuado entre la productividad y el rendimiento, jCuda requiere un mayor esfuerzo de programación, pero, a cambio, proporciona un mayor rendimiento, especialmente cuando es posible utilizar librerías CUDA optimizadas como CUBLAS y CUFFT, siendo capaz de rivalizar con el rendimiento de CUDA. Las conclusiones derivadas de este trabajo son la clave para ofrecer a los desarrolladores de Java una información actualizada sobre las soluciones disponibles para programación GPGPU y su rendimiento, aumentando definitivamente su productividad.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación de España [Proyecto TIN2010-16735 y una subvención FPU AP2010-4348], y por la Xunta de Galicia [Programa para la Consolidación de grupos de investigación competitivos, ref. 2010/6] y la ayuda de Agrupaciones Estratégicas (CN2012/211), financiadas parcialmente con fondos FEDER.

Referencias

[1] Top 500 Supercomputers List, http://top500.org, [Last visited May 2013].

[2] G. L. Taboada, S. Ramos, R. R. Expósito, J. Touriño, and R. Doallo, "Java in the High Performance Computing Arena: Research, Practice and Experience," *Science of Computing Programming*, vol. 78(5), pp. 425–444, 2013.

[3] A. Leist, D. Playne, and K. Hawick, "Exploiting Graphical Processing Units for Data-Parallel Scientific Applications," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 18, pp. 2400–2437, 2009.

[4] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable Parallel Programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[5] J. Stone, D. Gohara, and S. Guochun, "OpenCL: a Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science and Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[6] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, "GPU Cluster for High Performance Computing," in *Proc. 16th ACM/IEEE Conf. on Supercomputing (SC'04)*, Pittsburgh, PA, USA, 2004, p. 47 (12 pages).

[7] Project Sumatra, http://openjdk.java.net/projects/sumatra/, [Last visited May 2013].

[8] Y. Yan, M. Grossman, and V. Sarkar, "JCUDA: A Programmer-Friendly Interface for Accelerating Java Programs with CUDA," in *Proc. 15th Intl. Euro-Par Conf. on Parallel Processing (Euro-Par'09)*, Delft, The Netherlands, 2009, pp. 887–899.

[9] jCuda. Java bindings for CUDA, http://jcuda.org, [Last visited May 2013].

[10] P. Calvert, "Parallelisation of Java for Graphics Processors," Part II Dissertation, Computer Science Tripos, University of Cambridge, 2010.

[11] P. Pratt-Szeliga, J. Fawcett, and R. Welch, "Rootbeer: Seamlessly using GPUs from Java," in *Proc. 14th IEEE Intl. Conf. on High Performance Computing and Communications (HPCC-ICESS'12)*, Liverpool, UK, 2012, pp. 375–380.

[12] jCuda. Java bindings for OpenCL, http://www.jocl.org/, [Last visited May 2013].

[13] Aparapi. API for data parallel Java, http://code.google.com/p/aparapi/, http://developer.amd.com/tools/hc/Aparapi/Pages/default.aspx, [Last visited May 2013].

[14] JogAmp JOCL. Java OpenCL, http://jogamp.org/jocl/www/, [Last visited May 2013].

[15] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, "General-purpose Computation on GPUs for High Performance Cloud Computing," *Concurrency and Computation: Practice and Experience*, vol. (in press), 2013.

[16] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter, "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite," in *Proc. 3rd Workshop on General-Purpose Computation on Graphics Processors (GPGPU'10)*, Pittsburgh, PA, USA, 2010, pp. 63–74.

Extensión del modelo Roofline y herramientas para su uso

O.G. Lorenzo, T.F. Pena, J.C. Cabaleiro, J.C. Pichel, F.F. Rivera [1]

Resumen— **Los sistemas actuales pueden presentar complejas jerarquías de memoria y heterogeneidad a nivel de memorias, núcleos y procesadores. Como consecuencia, su programación eficiente es un reto. Un modelo de caracterización de rendimiento, que ofrezca directrices e información sobre la ejecución de un código, es útil para la comprensión del comportamiento del mismo. En este artículo se presenta un nuevo modelo diseñado como una extensión del conocido Berkeley Roofline Model. Nuestro objetivo ha sido incluir información que tenga en cuenta las complejidades de múltiples núcleos y sistemas heterogéneos, y así, entender su influencia sobre un código cuando se ejecuta en un sistema particular. Las características incorporadas al modelo básico son: su carácter dinámico, la posibilidad de analizar cada thread individualmente y la incorporación de un nuevo parámetro, la latencia de acceso a memoria. Se han implementado una serie de herramientas para obtener y representar este modelo de manera amigable.**

Palabras clave— **Roofline Model, Rendimiento, Contadores Hardware, PEBS, NPB.**

I. Introducción

El comportamiento de los accesos a la memoria es uno de los aspectos que más influyen en el rendimiento de cualquier código. Este hecho es cada vez más relevante a medida que aumenta la influencia del "memory wall" [1]. Un área en la que la gestión de memoria y su utilización es especialmente importante es la de los sistemas paralelos y distribuidos y, en particular, en las actuales arquitecturas multinúcleo. Al entrar en la era manycore este problema se agrava al incorporar interconexiones más complejas entre los núcleos y la memoria.

Con el fin de ayudar a los programadores a entender el rendimiento de un código en un sistema particular, se han propuesto varios modelos para su caracterización. En particular, el *Berkeley Roofline Model* [2] (RM) ofrece un buen equilibrio entre simplicidad y capacidad de descripción en una gráfica 2D. Sin embargo, su propia simplicidad puede ocultar algunos cuellos de botella presentes en algunas arquitecturas modernas. En este trabajo se presenta una extensión del RM. En particular, el RM se amplió tomando diferentes mediciones durante la ejecución de una aplicación, con el fin de mostrar la evolución de diferentes fases de la misma. Hemos denominado a este modelo *Dynamic Roofline Model* (DyRM), para resaltar el hecho de que muestra la evolución en el tiempo de la ejecución de un código, hilo a hilo. Esto tiene especial importancia en sistemas multinúcleo y heterogéneos, ya que muestra claramente las diferencias en la ejecución para cada uno de los núcleos.

[1]CITIUS Centro de Investigación en Tecnoloxías da Información, Univ. of Santiago de Compostela, Spain, Email: {oscar.garcia, tf.pena, jc.cabaleiro, juancarlos.pichel, ff.rivera }@usc.es

Para poder detectar la variabilidad en las diferencias en tiempo de acceso de diferentes hilos a diferentes celdas de memoria principal, el DyRM se ha ampliado en una tercera dimensión que muestra la latencia media de los accesos a memoria principal.

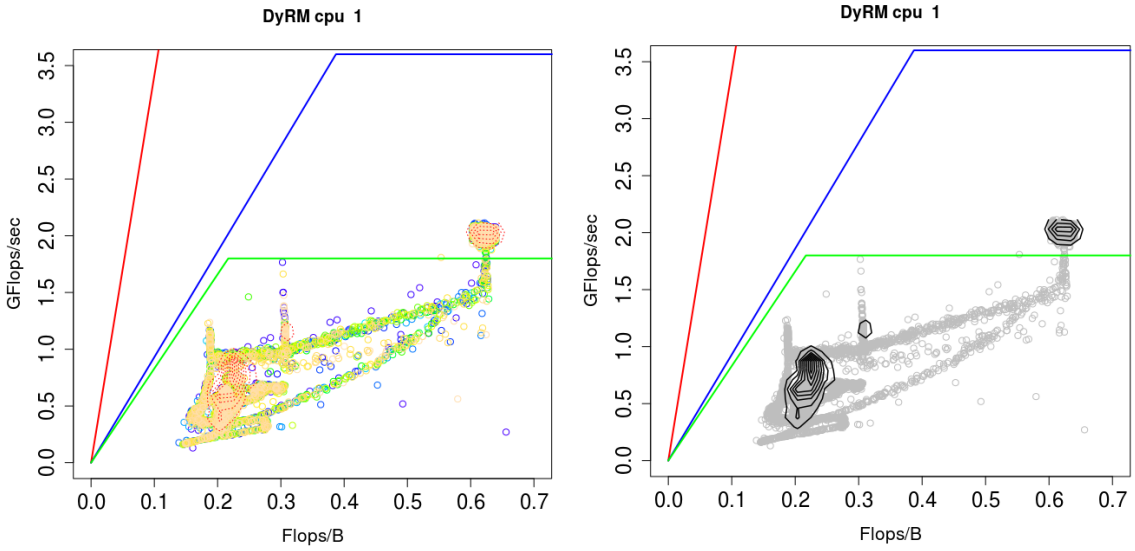
Para obtener este modelo, hemos desarrollado una herramienta que utiliza los contadores hardware presentes en los procesadores modernos. Recoge los datos proporcionados por estos contadores, los cuales son utilizados por una segunda herramienta para realizar diferentes visualizaciones útiles para el programador.

Estos contadores hardware de rendimiento son mecanismos de monitorización incluidos en la Unidad de Supervisión del Rendimiento (Performance Monitoring Unit [3]) de la mayoría de los microprocesadores modernos. Su uso está ganando popularidad como una herramienta de análisis y validación. Su efecto sobre el programa analizado es prácticamente imperceptible y su precisión se ha incrementado notablemente gracias nuevos mecanismos de muestreo como el PEBS (*Precise Event-Based Sampling* [4]) de Intel.

II. El Roofline Model

Los modelos de análisis estocásticos y estadísticos de rendimiento [5–7] pueden predecir con precisión el rendimiento del programa en multiprocesadores, pero rara vez dan una idea de cómo mejorar la eficiencia de los programas, de los compiladores y de los sistemas, siendo además difíciles de utilizar por no expertos. El RM es un modelo fácil de entender, y a partir del cual se pueden inferir directrices de mejora e información sobre el comportamiento de un programa. Ofrece información sobre cómo mejorar el rendimiento del software y hardware.

El RM utiliza un enfoque simple, en donde se pone en relieve y se cuantifica la influencia del cuello de botella del sistema. En los sistemas modernos, el cuello de botella principal es, a menudo, la conexión entre el procesador y la memoria principal. Esta es la razón por la que en el RM se relaciona el rendimiento del procesador con el tráfico de memoria fuera del chip. Se utiliza el término "intensidad operacional" como número de operaciones por byte de tráfico DRAM (medido en $Flops/Byte$). Mide el tráfico entre las memorias caché y la memoria, en lugar de entre el procesador y las caches. Por lo tanto, la intensidad operacional indica el ancho de banda de memoria DRAM que necesita un kernel en un equipo determinado. El RM une el rendimiento (medido en $GFlops/sec$), la intensidad operacional y el rendimiento de la memoria en un único gráfico 2D.



(a) DyRM sp.B, coloración temporal.

(b) DyRM sp.B, coloración de densidad.

Fig. 1. Ejemplos de Roofline Models Dinámico para la NPB benchmark sp.B.

El RM incluye la definición de límites de rendimiento, concretamente, líneas horizontales que muestran el rendimiento pico en punto flotante. De este modo, el rendimiento de punto flotante real de un kernel en particular no puede sobrepasar la línea horizontal del gráfico RM, ya que esta línea es un límite impuesto por el hardware. Una segunda línea, con una determinada pendiente, delimita el máximo rendimiento de punto flotante que el sistema de memoria puede soportar para una intensidad operacional dada. Su pendiente se correspondería con el máximo ancho de banda de memoria. Las dos líneas se cruzan en el punto de máximo rendimiento computacional y máximo ancho de banda de memoria. De esta manera, si la intensidad operacional del código está por debajo de la parte inclinada de este techo, significa que su rendimiento está limitado por la memoria. Si, por contra, está por debajo de la parte plana, estaría limitado por la computación.

El RM ofrece una representación simple del rendimiento de programas en un sistema. No obstante, en algunos casos, puede ser engañosa. Consideremos un programa que pasa por dos fases de ejecución, la primera cerca del máximo de $GFlops/sec$ y de intensidad operacional de la máquina, mientras que la segunda con un peor rendimiento y limitada por el acceso a memoria. El RM indicaría el rendimiento del código en un solo punto de la figura, tal vez entre el rendimiento de las dos fases, que no sería totalmente representativo del comportamiento real del programa. Consideremos ahora un sistema heterogéneo. El RM daría un punto de rendimiento único para todo el sistema, ocultando así la heterogeneidad de la arquitectura. Las diferencias en el interior del sistema se podrían analizar si los hilos se estudian por separado. Situaciones como éstas justifican la propuesta de un DyRM, que proporciona información a intervalos regulares de una ejecución, mostrando cada uno de los hilos de manera independiente.

III. Extensiones al Roofline Model

A. *Roofline Model Dinámico: DyRM*

El DyRM propuesto es esencialmente el equivalente a dividir en segmentos de tiempo la ejecución de un código y obtener un RM para cada uno, para, a continuación, combinarlos en un único gráfico. De esta manera se obtiene una vista más detallada del rendimiento durante toda la vida del código, mostrando su evolución y comportamiento. Con el DyRM se pueden detectar diferentes fases en la ejecución. Para mostrar la evolución del programa en el tiempo, se colorea cada punto de la gráfica secuencialmente, usando un gradiente de color. Adicionalmente, para mostrar las fases de ejecución, se puede realizar una estimación de la densidad de los puntos del DyRM. Tal estimación permite identificar las zonas en el modelo en el que el código pasa más tiempo. Los grupos resultantes se pueden resaltar en el DyRM y, con un cambio de coloración, se pueden mostrar en un gráfico de densidad, dando una mejor visión de estos. Con el uso de ambos gráficos, DyRM combina la simplicidad del RM con una vista detallada de la ejecución del programa.

Un ejemplo se muestra en la figura 1. Aquí mostramos el DyRM de uno de loas hilos de una aplicación que se ejecuta en dos procesadores Intel Xeon E5-2603. En este sistema hemos identificado tres techos, que representan el rendimiento máximo de un núcleo del procesador. El techo superior representa el pico en $GFlops/sec$ del núcleo, usando instrucciones SIMD, y el ancho de banda de memoria máximo teórico [8] (este techo alcanza los 14,4 $GFlops/s$ y se corta en la figura). El techo intermedio representa el máximo de $GFlops/sec$ sin instrucciones SIMD, considerando una operación de multiplicación y otra de suma por ciclo, y el ancho de banda de memoria máximo dado por el benchmark STREAM [9]. El techo inferior representa los $GFlops/sec$ considerando sólo una operación en punto flotante por ciclo y el peor ancho de banda de memoria dado por

el STREAM. En la figura 1(a) se muestra cómo la aplicación sitúa su comportamiento mayoritariamente bajo el techo inferior. Cada medición se colorea gradualmente según el momento en que fue tomada. En la figura 1(b) se puede ver el tipo de comportamiento, donde más tiempo pasa el programa, fundamentalmente en dos zonas. Una de estas zonas supera el techo inferior, lo que significa que hace más de una operación en punto flotante por ciclo, mediante la combinación de operaciones suma y multiplicación. La otra zona está por debajo de la parte inclinada del techo, por lo que está limitada por la memoria, y podría no beneficiarse del uso de operaciones SIMD.

B. DyRM Extendido con Latencia

El RM modela el rendimiento de la memoria de un sistema a través de la intensidad operacional. Este indicador utiliza el número de operaciones en punto flotante por byte accedido desde la memoria principal, teniendo en cuenta, así, la capacidad de la jerarquía de memoria para servir los datos necesarios a la velocidad requerida por las unidades de punto flotante. Sin embargo, puede ser insuficiente en sistemas NUMA (Non Uniform Memory Access). El RM establece límites superiores para el rendimiento, pero en un sistema NUMA, la distancia y conexión de los bancos de memoria a los diferentes núcleos puede implicar variaciones en el tiempo de acceso a memoria principal. Esta variación se muestra en el DyRM como variaciones en los *GFlops/sec* en cada núcleo, aunque cada núcleo realice las mismas operaciones, algo común en aplicaciones multihilo. De esta manera, el mismo código puede comportarse de manera diferente dependiendo de donde se haya situado para su ejecución. De este modo, la intensidad operacional puede seguir siendo la misma, ocultando el hecho de que los peores rendimientos son debidos al subsistema de memoria.0 Un programador que trate de aumentar el rendimiento de la aplicación no sabrá discernir si las diferencias en *GFlops/sec* son debidas al acceso a memoria o a otras razones, como a escalado de potencia o la presencia de otros procesos en algunos núcleos.

Proponemos ampliar el modelo DyRM con una tercera dimensión que muestre la latencia media de los accesos de memoria para cada punto en el gráfico, ya que resaltaría claramente este problema, y sería útil sobre todo en los sistemas NUMA. En la figura 2 se puede ver un ejemplo de esta representación. Se observa el mismo caso de la figura 1, pero donde cada punto del DyRM muestra ahora la latencia media de acceso a memoria durante su intervalo.

IV. Intel PEBS

La herramienta propuesta hace uso del mecanismo PEBS [4] para obtener los datos del modelo. PEBS es una función de muestreo avanzada de los procesadores Intel en la que el procesador graba directamente muestras en una región de memoria designada. Con PEBS, cada muestra contiene el estado del procesador en el momento en el que un contador hardware

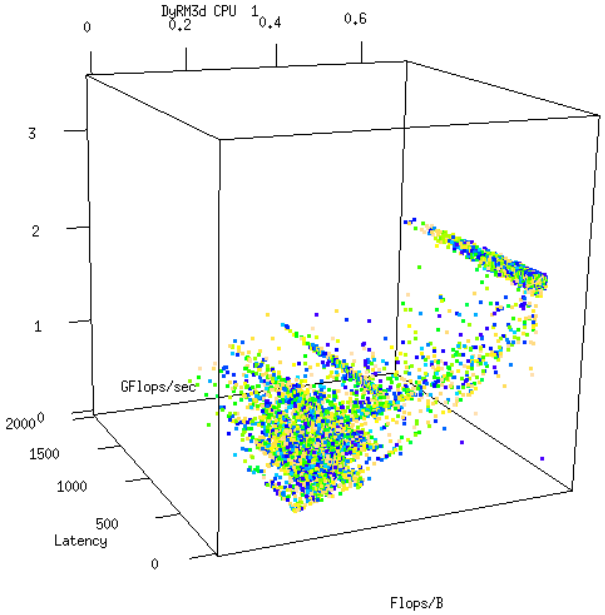


Fig. 2. DyRM3D, sp.B, GFlops/sec-Flops/B-Latecia (ciclos)

se desborda. Este contador puede medir diferentes eventos hardware y se desborda cuando alcanza un valor definido al programarlo. La precisión de PEBS proviene del hecho de que se registra el puntero de instrucción en cada muestra, y éste está a lo sumo a una instrucción de distancia de donde el contador en realidad se desbordó. Una ventaja clave de PEBS es que se minimiza la sobrecarga, debido a que el núcleo de Linux sólo está implicado cuando la memoria intermedia del PEBS se llena. Es decir, no se produce una interrupción hasta que una cierta cantidad de muestras están disponibles. Una restricción de PEBS es que sólo funciona con ciertos eventos.

En los procesadores Intel modernos, a partir de la arquitectura Nehalem, el formato de registro PEBS permite obtener información detallada sobre los accesos a memoria. Al muestrear las operaciones de memoria, se registra la dirección virtual de los datos cargados o datos guardados. Para las operaciones de carga también se registra la latencia a la que se sirven los datos (en número de ciclos), así como información detallada sobre el nivel de memoria desde donde se leen.

Con el fin de interactuar con los contadores hardware utilizamos la interfaz Linux `perf_events` [10]. Esta interfaz proporciona un medio para interactuar con el kernel de Linux a través de una API del sistema. En este caso, el buffer de muestreo se almacena en el espacio de kernel, y se puede leer en el espacio de usuario una vez que ha desbordado. La interfaz `perf_events` amplía el formato PEBS, registrando para cada muestra otros datos, como el `pid` y el `tid` del proceso activo. También ofrece la posibilidad de realizar multiplexado en el tiempo con los contadores. De este modo se pueden registrar más eventos cuasi-simultáneamente de los que permite la arquitectura hardware. Para ello registra, para cada muestra, una información temporal precisa que indica cuánto tiempo han estado funcionando los conta-

dores respecto del total.

Por el momento la interfaz `perf_events` no ofrece acceso al subsistema de medición de la latencia de lectura a memoria, y con ello el registro PEBS completo de los nuevos procesadores de Intel, de manera estable. Sin embargo existen versiones en desarrollo que sí lo hacen, y se espera que se añadirá al núcleo principal de Linux en un futuro próximo. Usando estas versiones en desarrollo, la interfaz permite realizar el muestreo de procesos individuales y sus hijos, independientemente del núcleo donde se ejecuten, o del sistema completo, es decir, en todos los núcleos simultáneamente, pero acumulando los datos de todos los procesos en cada uno.

Mediante el uso de esta interfaz aseguramos compatibilidad a través de numerosos sistemas, siempre y cuando estén basados en Linux e arquitecturas de Intel.

V. Herramientas de Captura y Análisis

Se ha implementado un conjunto de herramientas para obtener y procesar la información sobre el rendimiento de un sistema de memoria compartida. Una aplicación en línea de comandos utiliza la interfaz Linux `perf_events` para leer los contadores hardware en cada CPU y caracterizar el rendimiento de todo el sistema. A continuación, los datos se procesan en un entorno R [11].

A. Captura de Datos para DyRM

Para obtener el DyRM se necesitan los datos de las operaciones de punto flotante realizadas en cada núcleo. En los modernos procesadores Intel Sandy/Ivybridge [12] esto significa grabar hasta diez eventos de hardware diferentes (variaciones de FP_COMP_OPS_EXE y SIMD_FP_256). Si no se tienen en cuenta las instrucciones de punto flotante empaquetadas, puede ser suficiente con medir sólo dos eventos, FP_COMP_OPS_EXE:SSE_SCALAR_DOUBLE y FP_COMP_OPS_EXE:SSE_FP_SCALAR_SINGLE. Por otro lado, también es preciso medir el tráfico entre las memorias caché y la memoria principal, de modo que para cada núcleo se registran el número de líneas caché leídas desde la memoria principal (evento OFFCORE_REQUEST: ALL_DATA_READ). En todas las muestras se captura tanto información sobre el número de operaciones de punto flotante como de acceso a memoria. El recuento de instrucciones en cada núcleo se utiliza para establecer el período de muestreo, por lo que en cada desbordamiento se obtiene información sobre el número de instrucciones, operaciones de punto flotante y datos leídos. De esta manera el período de muestreo puede ajustarse fácilmente, de forma independiente del número de *Flops* del programa.

Se captura el estado de los contadores hardware en cada núcleo. Si más de un proceso o subproceso se ejecuta simultáneamente en el mismo núcleo, los datos deben ser escalados, para que cada hilo se pueda monitorizar de forma individual. Cada muestra tiene información sobre el `pid` y `tid` de la instrucción específica capturada. Esto, junto con la información de temporización proporcionada por la interfaz `perf_events`, permite escalar los datos de cada núcleo para aproximar los valores de cada hilo. De esta manera, la programación de los contadores hardware se convierte en una tarea sencilla y la ejecución de un programa se puede monitorizar de manera completa, incluyendo los procesos del sistema operativo necesarios.

TABLA I
Sobrecoste de la Captura de Datos.

Programa	Tiempo(s)	Sobrecoste(%)
bt.A	53.27	*0.23*
bt.B	218.61	*0.30*
bt.C	834.15	*0.02*
cg.A	0.70	1.07
cg.B	36.07	3.19
cg.C	96.46	*3.15*
ft.A	3.21	2.18
ft.B	41.10	1.85
lu.A	48.67	2.95
lu.B	197.41	*0.36*
lu.C	754.40	*0.46*
ua.A	41.37	1.00
ua.B	164.30	*0.77*
ua.C	629.21	*1.99*
sp.A	54.36	*<0.01*
sp.B	213.29	*0.56*
sp.C	786.36	*0.27*

B. Sobrecoste de la captura

La aplicación de captura de datos es muy ligera. El sobrecoste está determinado principalmente por las tasas de muestreo: cuanto mayor sea la resolución deseada, mayor será el sobrecoste. La mayoría de las figuras en este documento se obtuvieron tomando muestras cada 10^8 instrucciones (aproximadamente una muestra cada 5 ms para la mayoría de los códigos usados). Cuando el tiempo de ejecución de un programa supera los 50 segundos, el periodo de muestreo se sube a $3x10^8$instrucciones, así el tamaño del archivo de la traza resultante sigue siendo razonable (menos de 10 MB) y sólo afecta ligeramente a la resolución de las gráficas.

Unos valores típicos del sobrecoste se muestran en la tabla I. En esta tabla se muestran los tiempos de ejecución, en un servidor con 2 procesadores Intel Xeon E5-2603 (8 núcleos) y 16GB de RAM, de algunos benchmarks de la NAS Parallel Benchmark Suite for OpenMP (NPB-OMP) [13] con 8 hilos, compilados sin optimización, y el sobrecoste que se obtiene ejecutando los códigos junto con el programa de captura de datos. Debe tenerse en cuenta que el sobrecoste del programa de captura y toma de muestras es bajo y, en muchos casos, dentro del error de la medida. Así, en los casos en los que el muestreo se fijó a $3x10^8$ (en cursiva en la Tabla I) este sobrecoste por lo general permanece por debajo del 0,8 %.

C. Interfaz Gráfica para DyRM

Para simplificar el estudio estadístico de la información recopilada se utiliza el entorno R. Se han implementado una serie de funciones en lenguaje R para leer y procesar los archivos de datos. Para facilitar su uso se ha diseñado una interfaz gráfica de usuario (GUI) utilizando también este entrono (véase la figura 3). Esta interfaz se encarga de leer y procesar los datos, así como de la elaboración de diferentes figuras. Las figuras pueden ser desde gráficas que muestran la evolución en el tiempo del número de instrucciones ejecutadas o la latencia media de acceso a memoria, hasta otras más complejas, como los modelos DyRM y representaciones 3D incluyendo la información de latencias de acceso a memoria. También puede mostrar esta evolución dinámicamente, mediante una animación.

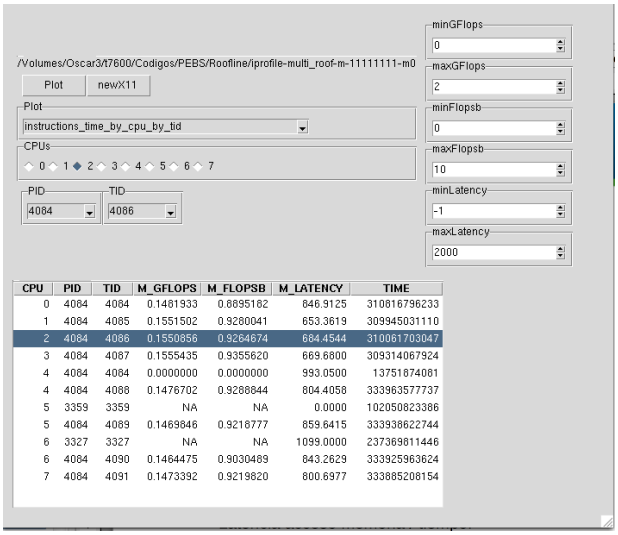


Fig. 3. La interfaz gráfica en el entorno R.

Para poder ofrecer una imagen completa del sistema, la herramienta permite seleccionar los datos recogidos de diversos modos. Es posible centrarse en el hardware, y mostrar las gráficas para cada núcleo de un procesador individualmente, usando los datos completos, sin diferenciar entre los procesos que en él se ejecutan. También es posible centrarse en los procesos, y representar modelos para cada `pid` o `tid` en ejecución. De esta manera, los usuarios tienen una visión completa del rendimiento del sistema. La propia aplicación permite mostrar figuras compuestas con varios gráficos simultáneamente. La interfaz también muestra estadísticas relevantes de los procesos, como *GFlops/sec* medios, intensidad operacional media o latencia media.

Como ejemplo de estas funcionalidades se pueden ver las figuras 4, 5 y 6. Estas muestran la ejecución de dos benchmarks del NPB-OMP ejecutados de forma consecutiva, primero `sp.A` y después `ft.A`, en un servidor con 2 procesadores Intel Xeon E5-2603 (8 núcleos en total) y 16GB de RAM. En la figura 4 se muestra la evolución completa del sistema, mostrando para cada núcleo su DyRM. En esta figura se observa cómo cada punto del DyRM se colorea respecto al tiempo total de captura de datos, cuya escala se muestra en la última subfigura. En la figura 5 se muestran los DyRM tanto del núcleo 2 como de las aplicaciones allí ejecutadas, por separado. Nótese que el coloreado del DyRM se realiza respecto del tiempo total de medición, y que la propia herramienta escala los datos para obtener una visión real del comportamiento de cada hilo por separado. Por último, en la figura 6, se puede observar el DyRM en 3D, con información de latencia, del sistema completo, con todos los núcleos combinados. Aquí la coloración se ha realizado en función del procesador, de este modo los datos de cada procesador tienen un color distinto; esto es así para poder comprobar si hay diferencias en el tiempo de acceso a memoria dependiendo del procesador (también es posible realizar la coloración temporal). En este caso la diferencia entre procesadores no es relevante, pero sí se observan diferencias entre las dos aplicaciones ejecutadas.

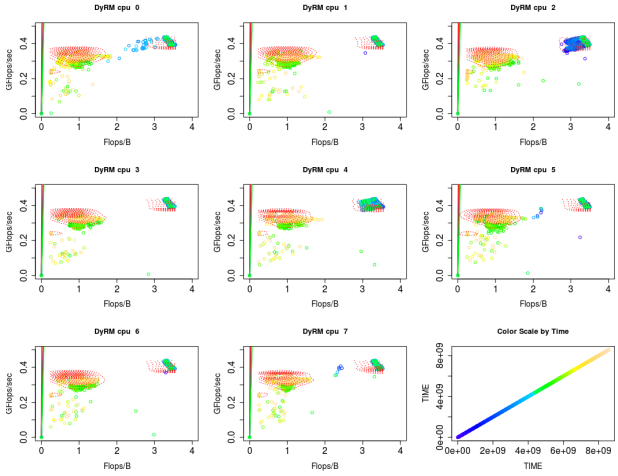
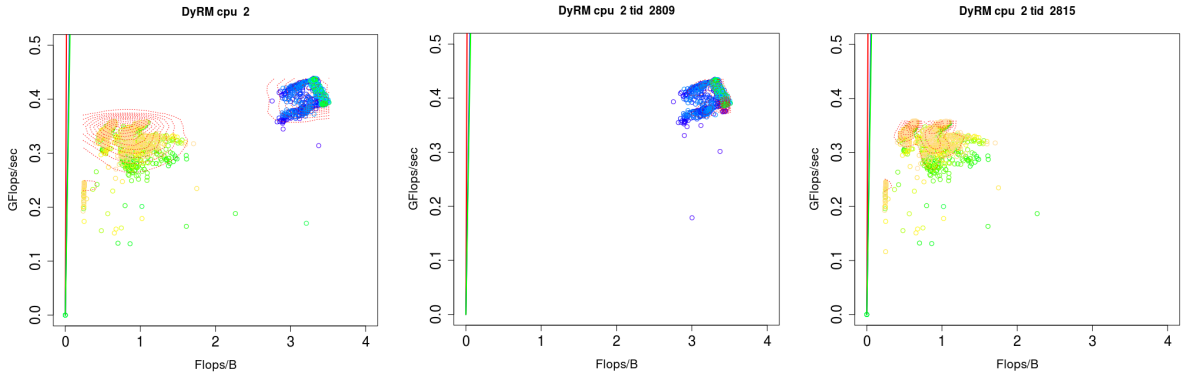


Fig. 4. Modelos DyRM para los 8 núcleos del sistema.

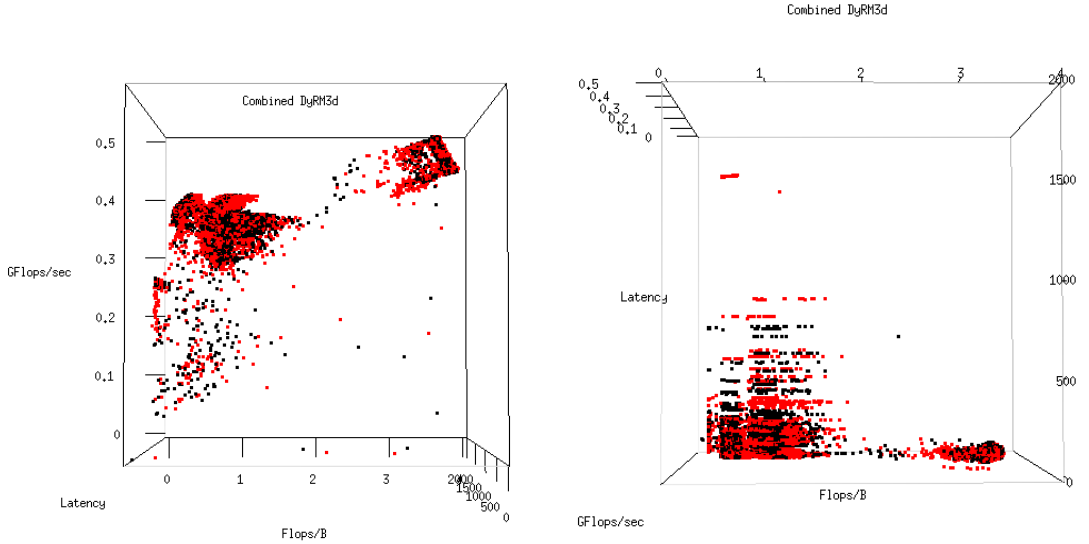
VI. Conclusiones

En este trabajo se presenta un conjunto de extensiones al Berkerley Roofline Model y mostramos su utilidad. Para ayudar a crear estos modelos se han aprovechado los contadores PEBS de los procesadores Intel para implementar una serie de herramientas que automatizan la tarea. Hemos demostrado que, mientras que el Berkerley Roofline Model sigue siendo un modelo útil y simple, esconde algunas de las características de las aplicaciones que influyen en su rendimiento en muchos sistemas, especialmente multicore o NUMA. Nuestro modelo ampliado, el DyRM, resalta comportamientos como la existencia de fases o desequilibrios, facilitando su detección. Al ampliar el DyRM con información de la latencia de acceso a memoria principal, se pueden detectar diferencias en los tiempos de acceso a memoria que pueden influir en el rendimiento de códigos paralelos. Hemos desarrollado un conjunto de herramientas que permiten obtener estos modelos de manera simple y eficiente, en sistemas basados en Linux e arquitecturas Intel. Por último se ha implementado una interfaz gráfica que permite obtener de forma simple gráficas del



(a) DyRM núcleo 2 (b) DyRM núcleo 2 tid 2809, ep.A (c) DyRM núcleo 2 tid 2815, ft.A

Fig. 5. DyRM de ep.A y ft.A. Detección de aplicación.



(a) DyRM3d, GFlops/sec Flops/B (b) DyRM3d, Flops/B Latency(cycles)

Fig. 6. DyRM3d de ep.A y ft.A. Procesador 0 (núcleos 0-3) en rojo, procesador 1 (núcleos 4-7) en negro.

rendimiento y del DyRM extendido de un programa.

Agradecimientos

Este trabajo ha estado parcialmente financiado por el Ministerio de Educación y Ciencia, fondos FEDER, bajo el contrato TIN 2010-17541, y por el proyecto de la Xunta de Galicia 09TIC002CT. Ha sido desarrollado en el marco de la red europea HiPEAC-2 y de la red española CAPAP-H.

Referencias

[1] John L. Hennessy and David A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[2] Samuel Williams, Andrew Waterman, and David Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[3] David Mosberger and Stephane Eranian, *IA-64 Linux Kernel: Design and Implementation*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[4] Precise Event-Based Sampling (PEBS), *http://perfmon2.sourceforge.net/pfmon_intel_core.html#pebs*.

[5] D. R. Martínez, V. Blanco, J. C. Cabaleiro, T. F. Pena, and F. F. Rivera, "Modeling the performance of parallel applications using model selection techniques," *Concurrency and Computation: Practice and Experience*, 2013.

[6] Xingfu Wu, *Performance, evaluation, prediction and visualization of parallel systems*, Kluwer Academic Publishers, 1999.

[7] Valerie Taylor, Xingfu Wu, and Rick Stevens, "Prophesy: An infrastructure for performance analysis and modeling of parallel and grid applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 4, pp. 13–18, 2003.

[8] Intel Ark, *http://ark.intel.com/products/64592/*.

[9] John D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[10] The Unofficial Linux Perf Events Web-Page, *http://web.eece.maine.edu/~vweaver/projects/perf_events*.

[11] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0.

[12] Intel®64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2, *http://download.intel.com/products/processor/manual/253669.pdf*.

[13] Haoqiang Jin, Michael Frumkin, and Jerry Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," Tech. Rep., Technical Report NAS-99-011, NASA Ames Research Center, 1999.

Metodología para predecir la escalabilidad de aplicaciones paralelas

Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque [1]

Resumen— **El comportamiento de las aplicaciones de paso de mensajes en un computador paralelo, a medida que estas escalan, puede variar y causar problemas a distintos niveles. Con el objetivo de proporcionar información del comportamiento de estas aplicaciones en un determinado sistema, se propone una metodología que permitirá analizar y predecir el comportamiento de la escalabilidad en aplicaciones de paso de mensajes, utilizando un tiempo acotado y un número de recursos limitados. La metodología propuesta se basa en el hecho que la mayoría de aplicaciones científicas son desarrolladas utilizando determinados patrones, tanto de comunicación como de cómputo, los cuales presentan un comportamiento específico. A medida que el número de procesos de la aplicación aumenta, estos patrones variarán su comportamiento siguiendo unas determinadas reglas, siendo funcionalmente constantes. Nuestra metodología propone caracterizar estos patrones con el objetivo de encontrar sus reglas generales de comportamiento, mediante las cuales construir una traza lógica de la aplicación con la cual poder predecir su rendimiento. La metodología utiliza la herramienta PAS2P, lo que permite realizar un análisis eficiente de la aplicación, ya que permite analizar únicamente un conjunto reducido de fases relevantes cubriendo aproximadamente un 95 % del total de la aplicación.**

Palabras clave— **Predicción de escalabilidad, Patrón de la aplicación, Aplicaciones paralelas, Sistemas HPC**

I. Introducción

DURANTE los últimos años, gracias a la constante evolución del hardware, los computadores de altas prestaciones han aumentando el número de cores significativamente. Los usuarios de estos sistemas, desean obtener el máximo beneficio a este gran número de cores escalando sus aplicaciones, ya sea reduciendo el tiempo de ejecución o incrementando su carga de trabajo.

Con el fin de lograr un uso eficiente de estos sistemas, un punto clave a considerar a la hora de ejecutar las aplicaciones, es conocer el comportamiento de la aplicación en el sistema en el cual se va a ejecutar, ya que el número ideal de procesos y recursos necesarios para ejecutar la aplicación puede variar de un sistema a otro, debido a diferencias hardware entre los sistemas. El desconocimiento de esta información, puede ocasionar un uso ineficiente del sistema, causando problemas a diferentes niveles, como puede ser el gasto económico y energético.

Con el objetivo de evitar estos problemas y hacer un uso eficiente del sistema, tanto usuarios como administradores de sistemas utilizan modelos de predicción de rendimiento, mediante los cuales buscan predecir el comportamiento de la aplicación en el sistema, con el fin seleccionar los recursos más apropiados para ejecutar la aplicación de forma eficientemente.

En este trabajo se propone una metodología, la cual nos permitirá analizar y predecir el comportamiento de la escalabilidad en aplicaciones paralelas de paso de mensajes en un determinado sistema, utilizando un tiempo acotado y un número de recursos limitados. El objetivo de la metodología es poder predecir el rendimiento que tendrá la aplicación al aumentar el número de cores, mediante la caracterización y análisis del comportamiento de los patrones de comunicación y cómputo, utilizando un conjunto reducido de cores.

La metodología se basa en el hecho que la mayoría de aplicaciones científicas han sido desarrolladas utilizando determinados patrones, tanto de comunicación como de cómputo, los cuales presentan similitud funcional a medida que el número de procesos aumenta, siguiendo unas determinadas reglas de comportamiento. Con el objetivo de caracterizar estas reglas de comportamiento, la metodología propuesta utiliza la herramienta PAS2P [1] [2]. Dicha herramienta permite reducir la complejidad del análisis de las aplicaciones mediante la creación de la firma de la aplicación, la cual contiene los patrones de comunicación y cómputo de la aplicación (fases) y su frecuencia de repetición (pesos). La utilización de la firma permite centrarse en analizar únicamente las fases relevantes de la aplicación, reduciendo la complejidad del análisis.

Las fases de la aplicación pueden ser observadas y trazadas a la hora de ejecutar la aplicación, con el fin de relacionarlas a medida que se aumenta el número de procesos y encontrar sus reglas generales de comportamiento, lo que permitirá construir una traza lógica de la aplicación independiente de la máquina. Una vez obtenida la traza lógica, se le insertarán los tiempos de cómputo y comunicación predichos para obtener la traza física de la aplicación, dependiente de la máquina, con la cual predecir el rendimiento de la aplicación.

Este trabajo se organiza de la siguiente forma: la sección II presenta los trabajos relacionados, la sección III presenta una visión general de la metodología PAS2P, la sección IV presenta la metodología propuesta, la sección V presenta la validación experimental y finalmente la sección VI presenta las conclusiones y el trabajo futuro.

[1]Dpto. de Arquitectura de Computadores y Sistemas Operativos, Universitat Autònoma de Barcelona, España, e-mail: javier.panadero@caos.uab.es, alvaro.wong@caos.uab.es, dolores.rexachs@uab.es y emilio.luque@uab.es

II. Trabajos relacionados

Hay otros trabajos los cuales están estrechamente relacionados con el estudio de predicción de escalabilidad en sistemas paralelos. Barnes et al [3] propone el estudio de escalabilidad mediante modelos de regresión separando cómputo y comunicación. Se utilizan modelos de regresión de caja negra, los cuales desconocen el comportamiento interno de las aplicaciones, tanto del cómputo como de comunicación, se basan únicamente en los parámetros de entrada (input) de las aplicaciones, los cuales tienen que ser subministrados y conocidos por el usuario. Nuestra propuesta difiere de este trabajo, ya que en lugar de utilizar modelos de caja negra, se basa en modelar la aplicación, tanto el cómputo como comunicación. Por otra parte, nuestra propuesta realiza únicamente regresiones del tiempo de cómputo, mientras que el tiempo de comunicación es medido en un simulador de red.

Snavely et al [4] por su parte propone partir de la firma de la aplicación para caracterizar el cómputo y la comunicación de la aplicación. Por otro lado, se caracteriza el rendimiento de la máquina y de la red mediante el uso de benchmarks. A partir de estas mediciones y las trazas de la firma, se aplican métodos de convolución para calcular el rendimiento de la aplicación. Este trabajo se diferencia de nuestra propuesta ya que Snavely propone utilizar métodos de convolución con el objetivo de relacionar las propiedades de la máquina con los requerimientos de la firma de la aplicación, con el fin de predecir el rendimiento de la aplicación.

Zhai et al [5] propone la herramienta PHANTOM, la cual puede ser utilizada para predecir escalabilidad de aplicaciones paralelas de paso de mensajes, suponiendo que sólo se dispone de 1 nodo, en el cual se ejecutarán los procesos representativos de la aplicación en lugar de ejecutar la aplicación completa. Finalmente, se realiza una simulación de las comunicaciones con el fin de predecir la escalabilidad de la aplicación. Esta herramienta se diferencia de nuestra propuesta, ya que analiza la aplicación completa, mientras que nuestra metodología se centra en analizar solamente las fases relevantes de la aplicación.

III. Visión general sobre PAS2P

Esta sección explica de manera resumida la metodología utilizada en la herramienta PAS2P [1]. Dicha metodología está basada en la repetibilidad de las aplicaciones y se centra en el análisis y predicción de aplicaciones paralelas mediante el uso de la firma de la aplicación. La fig. 1 ilustra los módulos de los cuales está compuesta la herramienta. Es importante destacar que la metodología está separada en dos grandes bloques. El primer bloque está dedicado a instrumentar la aplicación (Módulo de Instrumentación), analizar la aplicación, construir el modelo de la aplicación y extraer las fases significantes de la aplicación con sus respectivos pesos (Módulo de Análisis) y finalmente construir la firma de la aplicación (Módulo Generador de la Firma). Por otro lado, el segundo bloque está dedicado a ejecutar la firma de la aplicación con el objetivo de predecir la ejecución total de la aplicación en sistemas destino (Módulo de Predicción).

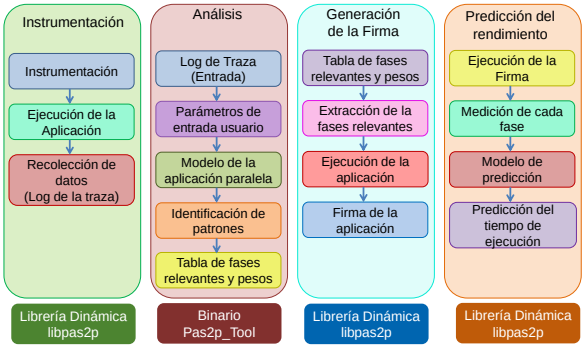


Fig. 1. Metodología PAS2P

A. Construcción de la firma

Con el objetivo de encontrar el comportamiento de la aplicación tanto en cómputo como en comunicaciones, se utiliza la interposición de funciones para interceptar y recolectar los eventos de comunicación de la aplicación paralela. Mediante estos datos recolectados, se genera un log con la traza de la aplicación. Esta traza será ordenada mediante un reloj global lógico en función de la casualidad de relaciones entre eventos. A partir de este log de eventos ordenado se obtiene el modelo independiente de la aplicación. El algoritmo para obtener este modelo está basado en el algoritmo de Lamport [6]. Una vez se ha obtenido el modelo de la aplicación, la metodología se esfuerza en identificar los patrones de la aplicación con el objetivo de encontrar el comportamiento representativo de la aplicación. Este comportamiento es encontrado a través de un procedimiento que identifica por similitud y extrae las secuencias más relevantes de la aplicación (fases) y les asigna el número de veces que ocurre en la aplicación (peso). Una vez tenemos identificadas las fases relevantes, el último paso es construir la firma de la aplicación, para ello, la aplicación se re-ejecuta para crear los checkpoints antes de que ocurra cada fase relevante.

B. Ejecución de la firma

Una vez la firma de la aplicación ha sido generada, puede ser utilizada para predecir el tiempo de ejecución de la aplicación en sistemas destino. Para ejecutar cada fase de la firma, se utiliza su checkpoint para restaurar la aplicación antes de que ocurra la fase relevante y se mide el tiempo de ejecución hasta que la fase termina. Con el objetivo de predecir el tiempo total de la aplicación, la ecuación 1 es usada, donde PAET es el tiempo de ejecución predicho, K es el número de fases, PETi es el tiempo de ejecución de la fase y finalmente Wi es el peso de la fase i.

IV. Metodología propuesta

La metodología propuesta se basa en el hecho que la mayoría de aplicaciones científicas de paso de men-

$$PAET = \sum_{i=1}^{n} (PET_i)(W_i) \quad (1)$$

sajes han sido desarrolladas utilizando determinados patrones, tanto de comunicación como de cómputo, los cuales presentan similitud funcional a medida que el número de procesos aumenta, siguiendo unas determinadas reglas de comportamiento. Estos patrones componen las diferentes fases de la aplicación, las cuales pueden ser observadas y trazadas con el fin de relacionarlas a medida que el número de procesos aumenta. Una vez relacionadas, mediante similitud funcional, se podrán encontrar sus reglas generales de comportamiento, mediante las cuales predecir el rendimiento de la aplicación.

La fig. 2 muestra las etapas de la metodología, la cual se centra en el hecho que es posible trazar el comportamiento de las fases de las aplicaciones, con el objetivo de poder relacionarlas a medida que se aumenta el número de procesos, con el fin de modelar las reglas generales de comportamiento que siguen los patrones de comunicación y cómputo de cada fase. A partir del modelado de estos patrones, será posible predecir y construir una traza lógica de la aplicación, independiente de la máquina, a la cual se le insertarán los tiempos de comunicación y cómputo predichos con el fin de predecir el rendimiento de la aplicación.

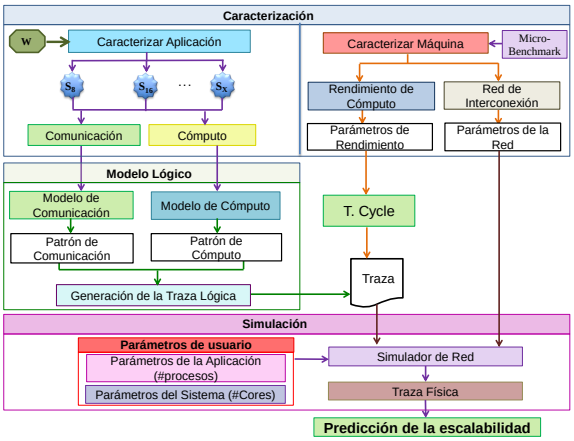


Fig. 2. Metodología propuesta

A. *Caracterización*

La etapa de caracterización consta de dos subetapas diferentes: Caracterización de la máquina y caracterización de la aplicación. Cabe destacar que la caracterización de la máquina solamente es necesario hacerla una única vez, independientemente de la aplicación que se quiera predecir su escalabilidad.

A.1 Caracterización de la máquina

Consiste en caracterizar el rendimiento de la maquina tanto en cómputo como en comunicación. Primeramente, se caracteriza el tiempo de ciclo real medio, es decir, cuando el procesador está ejecutando instrucciones de punto flotante que no presentan fallos en ningún nivel de la jerarquía de memoria. Se ha optado por caracterizar este tiempo en lugar de obtener el teórico proporcionado por el fabricante, debido a que pueden existir diferencias entre el tiempo de ciclo teórico y el tiempo de ciclo práctico. Para obtener este tiempo de ciclo, se ha diseñado un microbenchmark, el cual ejecuta un conjunto de instrucciones de multiplicación y suma en punto flotante, las cuales no causan fallo en ningún nivel de la jerarquía de memoria. Por otro lado, se caracteriza mediante la utilización de benchmarks la red de interconexión, ya que en la última etapa de la metodología es necesario proporcionarle al simulador ciertos parámetros de red, como son, la topología, la red de interconexión (Infiniband, Gyga-Ethernet,...) o la velocidad de interconexión.

A.2 Caracterización de la aplicación

La etapa de caracterización de la aplicación consiste en caracterizar el cómputo y la comunicación de la aplicación, con el objetivo de obtener información para construir su traza lógica. Para obtener esta información, se llevarán a cabo un conjunto de ejecuciones de la firma de la aplicación, para un número reducido de procesos, las cuales serán analizadas con el fin de extraer los patrones de comportamiento, tanto de comunicación como de cómputo, para cada una de las fases de las firmas. Estas firmas han sido obtenidas mediante la herramienta PAS2P [1] [2]. Se ha decidido trabajar con las firmas de la aplicación en lugar de con la aplicación, ya que la firma contiene únicamente las fases relevantes de la aplicación. Analizando este reducido conjunto de fases relevantes, estamos cubriendo aproximadamente un 95 % de la aplicación analizando una parte reducida de ella, lo que nos permite reducir el tiempo de análisis. Para este trabajo, se ha integrado PAS2P con la herramienta de rendimiento hardware PAPI [7] con el objetivo de tener información del rendimiento de cómputo a bajo nivel para cada fase la firma.

B. *Construcción de la traza lógica*

Una vez se han caracterizado las fases relevantes de la aplicación, se modelan los patrones de comunicación y cómputo para cada una de estas fases, con el objetivo de obtener las reglas generales de comportamiento con las cuales generar la traza lógica de la aplicación. La traza lógica generada es independiente de la máquina y del número de procesos de la aplicación, dependerá únicamente de la forma en la cual haya sido desarrollada la aplicación. Serán los parámetros de entrada de las reglas generales los que concretarán la generación de la traza para un número de procesos determinado. Una vez obtenida la traza lógica, se le insertarán los tiempos predichos de cómputo y comunicación para obtener la traza física de la aplicación, la cual será dependiente de la máquina y nos permitirá predecir el rendimiento de la aplicación en el sistema. Para obtener el tiempo

de cómputo predicho, se inserta en la traza el tiempo de ciclo caracterizado de la máquina, mientras que para obtener los tiempos de comunicaciones, se realizará una simulación de la traza. A continuación se explica el procedimiento para modelar los patrones de comunicación y cómputo.

B.1 Modelado del patrón de comunicación

El patrón de comunicación está compuesto por las ecuaciones generales de comunicación y su volumen de datos. Para obtener el patrón de comunicación, las fases relevantes se relacionan entre las diferentes ejecuciones y se modelan a partir de la caracterización de la comunicación, realizada en la etapa anterior. Cabe comentar que una fase puede tener de 1 a N comunicaciones, dependiendo de la aplicación. El volumen de datos predicho de cada comunicación será obtenido mediante modelos matemáticos de regresión, mientras que para obtener las ecuaciones generales de comunicación, se ha propuesto un algoritmo el cual se basa en obtener las ecuaciones de comunicación para cada una de la fases de las firmas (ecuaciones locales) y, a partir de estas ecuaciones, mediante similitud funcional, modelar las ecuaciones generales de comportamiento. La fig. 3 muestra un ejemplo de este algoritmo, se ha considerado para este ejemplo que cada fase tiene una única comunicación y por lo tanto una sola ecuación local por fase. Una vez se han obtenido las ecuaciones locales para cada una de las fases relevantes de cada firma, se analizan con el objetivo de modelar la ecuación general de comportamiento, tal y como se muestra para la fase 1.

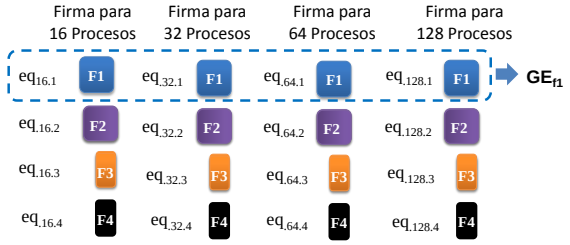


Fig. 3. Obtención de la ecuación general por similitud

El algoritmo de comunicación se divide en dos etapas, una primera etapa donde se obtienen las ecuaciones locales para cada fase y una segunda etapa donde se obtienen las ecuaciones generales de comunicación. Cabe comentar que para las dos etapas del algoritmo se convierten los procesos de la aplicación de decimal a binario para poder trabajar a nivel de bits.

La fig. 4 muestra la primera etapa del algoritmo, la cual está compuesta de dos fases, una primera fase de análisis y una segunda fase de modelado. Durante la fase de análisis, se analizan las dependencias entre procesos, el tipo de patrón y se calcula la matriz de distancias entre procesos. Esta matriz contiene la distancia de comunicación entre origen y destino para cada comunicación de la fase. Una vez se ha obtenido esta información para cada una de las fases de las firmas, se introduce como entrada en la fase de modelado. En esta segunda fase, se identifica cada patrón de comunicación, para cada una de las comunicaciones de cada fase, y se busca la repetitividad de la secuencia de comunicación de cada patrón. Se busca esta repetitividad con el objetivo de construir ecuaciones más sencillas de modelar, ya que podrán realizarse agrupaciones de procesos que realicen la misma comunicación. Una vez se ha identificado esta información, se crea la ecuación local para cada comunicación de la fase, a la cual se le aplicará un método de compresión con el objetivo de simplificar el siguiente paso de modelado. La salida de este módulo será una estructura de comunicación que identificará cada ecuación local. Dependiendo del tipo de patrón detectado, se utilizará una de las dos estructuras propuestas. En caso de que el patrón sea de tipo dinámico, son patrones del tipo *Intercambio* o *Permutación*, se utiliza la primera estructura, mientras que si es de tipo estático, perteneciente a algoritmos de tipo *Malla* o *Anillo*, se utiliza la segunda.

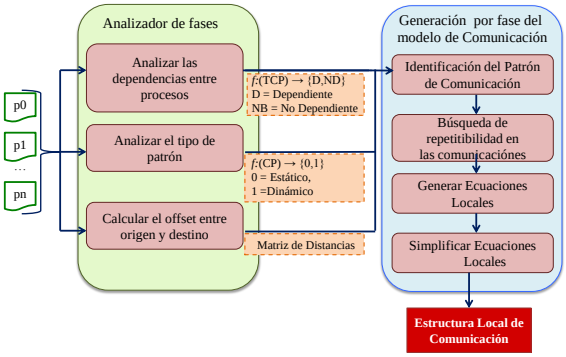


Fig. 4. Obtención de las ecucaciones locales de comunicación

La primera estructura (1), tiene como parámetros, el número de fase, el número de comunicación de la fase, el tipo de algoritmo (*Intercambio* o *Permutación*) y un vector con los bits implicados en el algoritmo. La segunda estructura (2) por su parte, tiene como parámetros, el número de fase, el número de comunicación de la fase, y un vector con la lista de repeticiones y el número de repeticiones.

1. EC1 = {#Fase, #Comunicación, Tipo , Lista de bits implicados }

2. EC2 = {#Fase, #Comunicación, list[comunicación, list[#repetición]] }

Una vez se ha obtenido la estructura de comunicación, se pasa a la segunda etapa del algoritmo con objetivo de modelar las ecuaciones generales de comportamiento, tal y como se muestra en la fig. 5. Con el objetivo de modelar las ecuaciones generales de cada fase, se analiza por similitud funcional las ecuaciones locales. Una vez identificada esta similitud, se modelan las ecuaciones generales con las cuales poder

predecir el patrón de comunicación de cada fase para un número de procesos mayor. Por último, se realiza un último paso donde se comprimen las ecuaciones generales a la mínima expresión con el objetivo de simplificar su expresión.

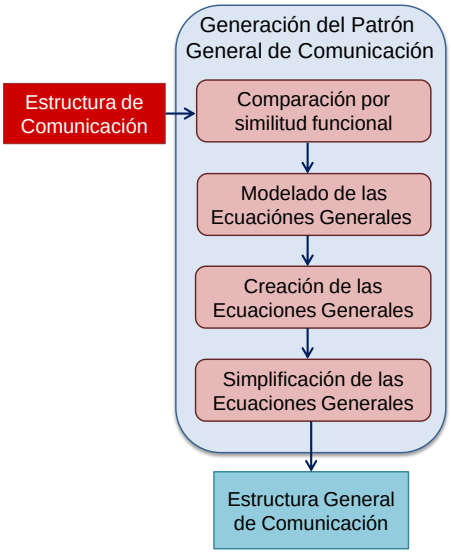


Fig. 5. Obtención de la ecuaciones generales

B.2 Modelado del patrón de cómputo

Se basa en buscar la repetitividad de las primitivas MPI para cada una de las fases con el objetivo de identificar los patrones de cómputo. Una vez identificados estos patrones se compararán mediante similitud funcional entre las diferentes ejecuciones de la firma con el fin de poder predecir el patrón de cómputo para un número mayor de procesos. La fig. 6 muestra el proceso utilizado. Se busca esta repetitividad ya que estas primitivas estarán encerradas en lazos que se irán repitiendo a lo largo de toda fase. El objetivo del algoritmo propuesto es descubrir este conjunto mínimo de primitivas y predecir su número de repetición, con el objetivo de construir la traza lógica, ya que puede variar según el número de procesos con el cual se ejecute la aplicación. Una vez caracterizado este conjunto mínimo de primitivas y sus repeticiones, se modelará el tiempo de cómputo. Este tiempo se encuentra entre las primitivas MPI caracterizadas. Con el objetivo de modelar este tiempo, se propone un modelo que se basa en dividir el tiempo de cómputo que hay entre cada primitiva MPI, entre tiempo de ejecución, el tiempo que el procesador está ejecutando instrucciones, y tiempo de stall, el tiempo que el procesador está sin ejecutar instrucciones esperando por accesos a memoria que no se han podido solapar con el cómputo. Se ha decidido separar el tiempo de cómputo ya que pueden tener tendencias diferentes. Con el fin de separar estos tiempos, el modelo utiliza la información de los contadores hardware de rendimiento, proporcionada en la etapa de caracterización. Una vez se han separado estos dos tiempos, mediante modelos estadísticos de regresión se obtienen las ecuaciones de regresión que nos permitirá predecir el tiempo de cómputo.

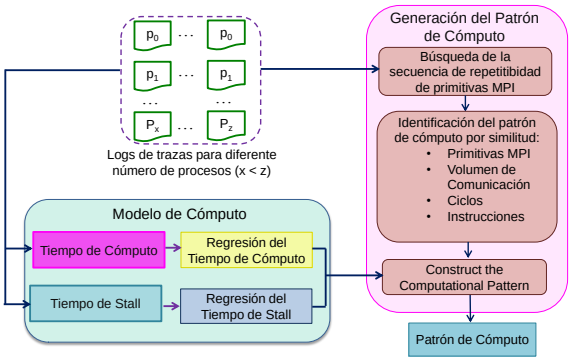


Fig. 6. Obtención del patrón de cómputo

C. Simulación

Finalmente, en la última etapa de la metodología, se llevará a cabo una simulación de la traza con el objetivo de obtener los tiempos de comunicación de los mensajes. Una vez finalizada la simulación, obtendremos la traza física, la cual contendrá los tiempos de comunicación. Esta traza física nos proporcionará el tiempo de ejecución de cada fase predicha de las cuales se compone la traza. El tiempo de cada fase será multiplicado por el peso predicho de la fase a fin de obtener el tiempo de ejecución de la aplicación para el número de procesos que queremos predecir, tal y como se muestra en la ecuación 1.

V. Evaluación experimental

En esta sección se muestra la validación experimental del modelado del patrón de comunicación. Se han utilizado los benchmarks BT y CG del conjunto de benchmarks NPB NAS, los cuales han sido ejecutados usando la clase D. Como entorno experimental se ha utilizado un cluster de 16 nodos con 16 procesadores Intel Xeon quad-core (256 cores).

Para llevar a cabo la validación experimental se han ejecutado un conjunto reducido de firmas de la aplicación para diferente número de procesos, las cuales han sido caracterizadas con el objetivo de obtener las ecuaciones locales para todas las comunicaciones de cada fase de la firma. Mediante el algoritmo propuesto, se ha obtenido la ecuación general de comunicación que nos permitirá predecir el patrón de comunicación para un número mayor de procesos de la aplicación. Para la validación del algoritmo propuesto se ha modelado la primera comunicación de la primera fase relevante de cada uno de los benchmarks utilizados.

Para el BT, se ha ejecutado la firma para 9, 16, 25 y 36 procesos y se ha predicho su comunicación para 49 procesos. Esta aplicación presenta un total de 6 fases relevantes. La fig. 7 muestra el patrón de comunicación para la primera comunicación de la primera fase, obtenido a partir de la ecuación general, la cual ha sido modelada mediante el análisis por similitud funcional de la ecuaciones locales. El tercer parámetro de la estructura de comunicación muestra

la la ecuación general de comunicación, la cual corresponde a un algoritmo estático de tipo toroidal, ya que realiza desplazamientos +1 (primer término de la ecuación), a excepción de los procesos situados en los extremos finales, los cuales conectan con los nodos iniciales (segundo término de la ecuación). La variable *desp* tiene valor 1, ya que realiza un desplazamiento de 1 entre origen y destino, mientras que la variable K indica el número de saltos para el cual se ha incrementado el número de procesos a partir del último ejecutado. En el caso del BT, se ha ejecutado hasta 36 procesos y el próximo salto incremental que nos permite la aplicación es para 49, por lo tanto K se ha incrementado 1 unidad (1 salto).

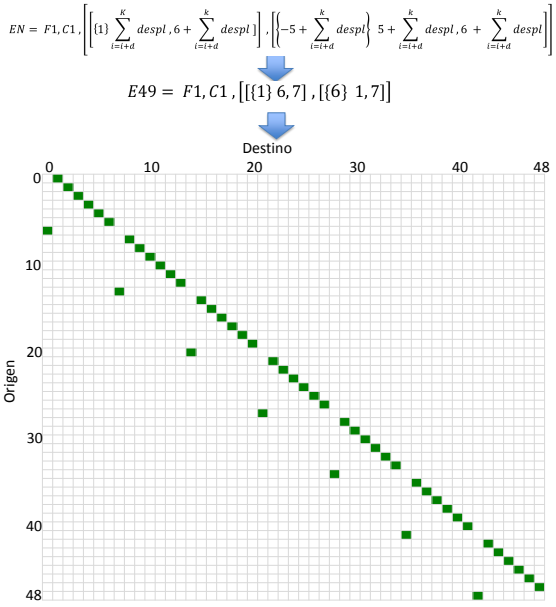


Fig. 7. Predicción patrón de comunicación fase 1 del BT

Para calcular el patrón de comunicación del NAS CG se ha ejecutado la firma para 8, 16 y 32 procesos y se ha predicho su comunicación para 64 procesos. Esta aplicación presenta un total de 3 fases relevantes. La fig. 8 muestra el patrón de comunicación predicho para la fase 1 con 64 procesos, obtenido a partir de la ecuación general, la cual ha sido modelada a partir del análisis por similitud funcional de la ecuaciones locales. Se ha obtenido un patrón de comunicación constante, a medida que se aumenta el número de procesos el patrón permuta siempre el bit 1 del origen para calcular el destino, tal y como indica la ecuación general, siendo P el tipo de intercambio (permutación) y 1 el bit implicado en la permutación.

VI. Conclusiones y trabajo futuro

Este artículo propone una metodología a partir de la cual analizar y predecir el comportamiento de la escalabilidad en aplicaciones paralelas de paso de mensajes en un determinado sistema, utilizando un número de recursos limitado y un tiempo de análisis acotado. Se ha presentado la metodología general y se ha validado experimentalmente el modelado del

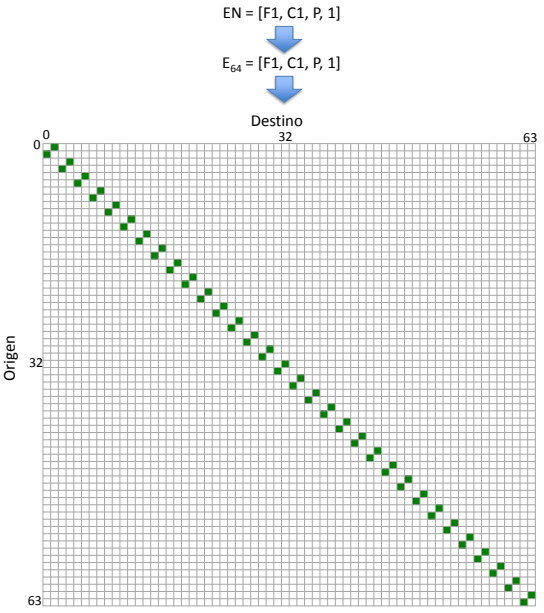


Fig. 8. Predicción patrón de comunicación fase 1 del CG

patrón de comunicación de las fases de la aplicación. Actualmente se está trabajando en terminar de validar el modelado del patrón de cómputo y la construcción de la traza lógica, con el objetivo de insertarla en el simulador de red y obtener la traza física con la cual predecir el rendimiento de la aplicación.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto MICINN Spain under contract TIN2007-64974, the MINECO (MICINN) Spain under contract TIN2011-24384, the European ITEA2 project H4H, No 09011 and the Avanza Competitividad I+D+I program under contract TSI-020400-2010-120

Referencias

[1] A. Wong, D. Rexachs, and E. Luque, "Extraction of parallel application signatures for performance prediction," *High Performance Computing and Communications, 10th IEEE International Conference*, pp. 223–230, 2010.

[2] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 31 2009-sept. 4 2009, pp. 1 –4.

[3] B. J. Barnes, J. Garren, D. K. Lowenthal, J. Reeves, B. R. de Supinski, M. Schulz, and B. Rountree, "Using focused regression for accurate time-constrained scaling of scientific applications," in *IPDPS*, 2010, pp. 1–12.

[4] A. Snavely, L. Carrington, N. Wolter, and J Labarta, "A framework for performance modeling and prediction," *Supercomputing*, pp. 1–17, Jan 2002.

[5] Z. Jidong, C. Wenguang, and W. Zheng, "Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node," pp. 305–314, 2010.

[6] L. Lamport and C. Time, "The Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[7] J. Dongarra, A. D. Malony, S. Moore, P. Mucci, and S. Shende, "Performance instrumentation and measurement for terascale systems," in *European Center for Parallelism of Barcelona*, 2003, pp. 53–62.

Performance analysis in Android™

Alejandro Acosta and Francisco Almeida[1]

Resumen— **The advent of emergent SoCs and MPSocs opens a new era on the small mobile devices (Smartphones, Tablets, ...) in terms of computing capabilities and applications to be addressed. Currently these devices have multicore processors and GPUs which provide high computational power. The efficient use of such devices, including the parallel power, is still a challenge for general purpose programmers. In the last years Android has become the dominant platform in the small mobile devices. In addition, it has a large community of developers. For application development, Android provides two development kits, Software development kit (SDK) and Native Development Kit (NDK). To exploit the high computational capabilities on current devices Android provides Renderscript, an API that allows the execution of parallel applications and is designed to be used in applications that require high computing power. In this paper we address the evaluation of the performance on these target platforms. A set of benchmark applications has been implemented to evaluate two different devices. Sequential and parallel versions of the different development kits are considered in the computational experience.**

Palabras clave— **Renderscript, Android, SoC, Performance.**

I. Introduction

SoCs (Systems on Chip [1]) have been the enabling technology behind the evolution of many of todays ubiquitous technologies, such as Internet, mobile wireless technology, and high definition television. The information technology age, in turn, has fuelled a global communications revolution. With the rise of communications with mobile devices, more computing power has been put in such systems. The technologies available in desktop computers are now implemented in embedded and mobile devices. We find new processors with multicore architectures and GPUs developed for this market like the Nvidia Tegra [2] with two and five ARM cores and a low power GPU, and the OMAP™5 [3] platform from Texas Instruments that also goes in the same direction.

On the other hand, software frameworks have been developed to support the building of software for such devices. The main actors in this software market have their own platforms: Android [4] from Google, iOS [5] from Apple and Windows phone [6] from Microsoft are contenders in the smartphone market. Other companies like Samsung [7] and Nokia [8] have been developing proprietary frameworks for low profile devices. Coding applications for such devices is now easier. But the main problem is not creating energy-efficient hardware but creating efficient, maintainable programs to run on them [9].

[1]Dept. Estadística, I.O. y Computación, ETS Ingeniería Informática, La Laguna University, Spain, e-mail: aacostad@ull.es

Conceptually, from the architectural perspective, the model can be viewed as a traditional heterogeneous CPU/GPU with a unified memory architecture, where memory is shared between the CPU and GPU and acts as a high bandwidth communication channel. In the non-unified memory architectures, it was common to have only a subset of the actual memory addressable by the GPU. Technologies like Algorithmic Memory [10], GPUDirect and UVA (Unified Virtual Addressing) from Nvidia [11] and HSA from AMD [12] are going in the direction of an unified memory system for CPUs and GPUs in the traditional memory architectures. Memory performance continues to be outpaced by the ever increasing demands of faster processors, multiprocessor cores and parallel architectures.

Android is an Open Source platform with a very high level of market penetration and it has a large community of developers [13]. Usually the applications are developed in Java, using the development tools proposed by the platform. Although the actual Java compiler provides a performance similar to the compiler for native languages like C or C++, Android includes development tools to implement code sections of Android applications in native languages. Android also provides the Renderscritp language to exploit high performance computing in the devices.

This native code is executed in Android using the Java Native Interface (JNI) provided by Java. Several studies have been conducted on the performance of applications using JNI [14], [15], [16], and over the increased yield obtained when using Renderscript [17], [18], [19], but all these studies are based on very similar testing problems, which perfectly fits the programming model used.

In this paper we present a comparative performance analysis between the different programming models in Android. We have implemented a set of testing problems with different inherent features. The analysis allows to have an idea of the behaviour of the programming models and to use this experience in enventually similar problems. The future developer can refers to these results to select the programming model to use depending of the problem to be implemented trying to obtain thea best performance. Several implementations have been generated for each problem, and have been executed in two different devices, a Asus Transformer Prime TF201 and a Samsung Galaxy SIII.

The paper is structured as follows, in section II we introduce the development model in Android and the different alternatives to exploit the devices, some of the difficulties associated to the development model are shown. In section III we describe each problem

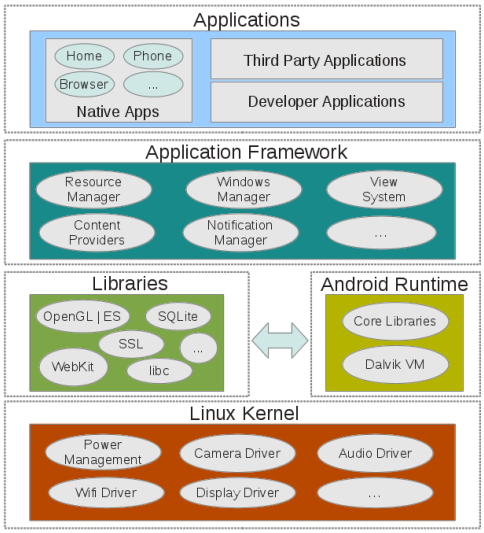


Fig. 1. The Android software stack.

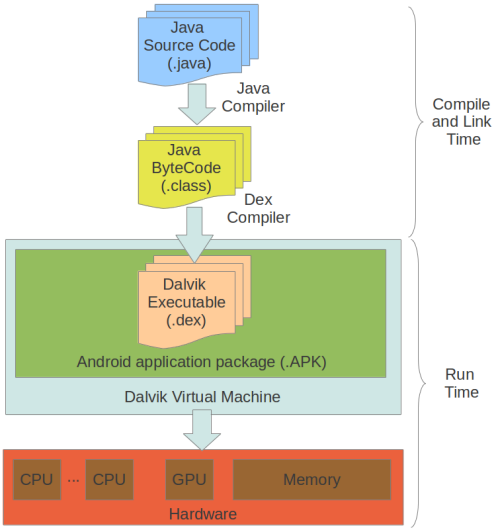


Fig. 2. Compilation and execution model of a Java Android application

and their implementations in the different programming models, the performance of these implementations are presented in section IV. The set of testing problems considers the classical matrix multiplication and the implementation of the convolution algorithm to colour images. These problems give an idea of the behaviour of the programming models because has different inherent features. Four different versions (corresponding to different programming models in Android) have been implemented, the Java original version, the native version, and two Renderscript versions, sequential parallel. The results show how the characteristics of the problem affect to the performance of the different programming models.

II. The development model in Android

Android is a Linux based operative system mainly designed for mobile devices such us mobile phones and tablet computers, although it is also used in embedded devices as smart TVs and media streamers. It is designed as a software stack that includes an operating system, middleware and key applications (see figure 1).

Applications are executed in Android under a Dalvik virtual machine Dalvik (Dalvik VM). The Dalvik VM executes files in the Dalvik Executable (`.dex`) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the `.dex` format by the included `dx` tool (see "Compile and Link time" in Figure 2). The Dalvik Executable files with any data and resource files are packaged in Android Application Package (`.apk`). All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application. Each application lives in its own security sandbox and by default they run in isolation from other applications. Every Android application runs in its own process, with its own instance of the Dalvik VM. Dalvik has been written so that a device can run multiple VMs efficiently. Android relies on Linux for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack. The Dalvik VM is supported on the Linux kernel for underlying functionality such as threading and low-level memory management.

Android applications are written in Java, and the Android Software Development Kit (SDK) provides the API libraries and developer tools necessary to build, test, and debug applications in a Software Development Kit (SDK). Figure 2 shows the compilation and execution model of a Java Android application. The compilation model converts the Java `.java` files to Dalvik-compatible `.dex` (Dalvik Executable) files. The application runs in a Dalvik VM that manages the system resources allocated to this application (through the Linux kernel)

Besides the development of Java applications, Android provides packages of development tools and libraries to developer Native applications, the Native Development Kit (NDK). The NDK enables to implement parts of the application running in the Dalvik VM using native-code languages such as C and C++. This native code executed using the Java Native Interface (JNI) provided by Java. Using Native code may introduce benefits to certain classes of applications, in the form of reuse of existing code and in some cases increased speed. Figure 3 shows the compilation and execution model of an application where part of the code has been rewritten using the NDK. The compilation process of the Java code is done using the SDK. The Native `.c` is compiled using the GNU compiler (GCC). More recent versions of the NDK allow the use of the Clang compiler to compile Native code, although the default is still to use the GCC ([20]) compiler. The Dalvik-compatible code of the application is executed in the Dalvik VM. When the Dalvik-compatible code calls a Native routine, the VM will allocate the resources needed on the system and will allow the Native code to be executed. All the resources used by the native code are controlled by the VM. Note that using native code does not result in an automatic performance increase due

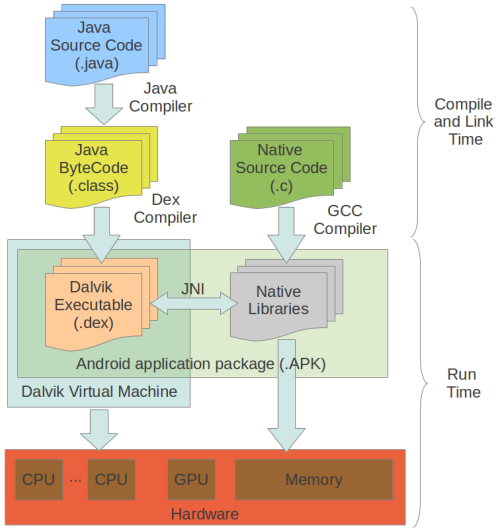


Fig. 3. Compilation and execution model for a Native application in Android

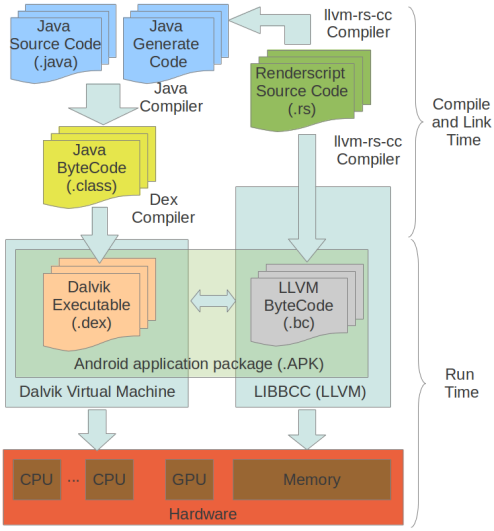


Fig. 4. Compilation and execution model of a Renderscript application under Android

to the JNI overload, but always increases application complexity, its use is recommended in CPU-intensive operations that don't allocate much memory, such as signal processing, physics simulation, and so on. Native code is useful to port an existing native codebase to Android, not for speeding up parts of an Android application.

To exploit the high computational capabilities on current devices, Android provides Renderscript, it is a high performance computation API at the native level (similar to CUDA) and a programming C language (C99 standard). Renderscript allows the execution of parallel applications under several types of processors such as the CPU, GPU or DSP, performing an automatic distribution of the workload across the available processing cores on the device. Figure 4 shows the compilation and execution model used by Renderscript. Renderscript (`.rs` files) codes are compiled using `llvm-rs-cc`, a `Clang` based compiler [21] that produces byteCode (`.bc` files) for a Low-Level Virtual Machine (LLVM), moreover, it generates a set of Java classes wrapper around the Renderscript code hiding the JNI code to the user. These Java classes, are compiled together with the rest of the Java classes of the application. The resulting Dalvik-compatible code is executed in the Dalvik VM. To execute the Renderscript (`.bc` files) byteCode is used a LLVM (`libbcc`). The Dalvik VM manages the resources of the system used by LLVM.

Aganin, the use of Renderscript code does not result in an automatic performance increasing. It is useful for applications that do image processing, mathematical modelling, or any operations that require lots of mathematical computation.

III. Implemented problems

To compare the performance of the different programming models in Android we implemented two applications a classical matrix multiplication and an implementation of the convolution algorithm to colour images. For each problem we implemented four versions of the code, correspondig to different programming models. The Java version (Java), a native C version (Native) using the NDK, and two Renderscript versions, sequential (RS Sequential) and parallel (RS Parallel).

A. Matrix multiplication

The matrix multiplication is a well known problem. This problem is similar to the above but having a higher computational load. In Listing 5 we shows the Java implementation of the standard algorithm for matrix multiplication. In this case we multiply the matrices A and B to obtain a new matrix C. The native C implementation and the Renderscript sequential version are very similar to the Java implementation. In native C code we added JNI function calls that allow to obtain the arrays in the native environment. For the Renderscript sequential version we defined the Java code to allocate the memory of matrices and we call the Renderscript function that solves the problem.

```
int i, j, k, add;
for (i = 0; i < size; i++) {
   for (j = 0; j < size; j++) {
      add = 0;
      for (k = 0; k< size; k++) {
         add += A[j*size+k]*B[k*size+i];
      }
      C[j*size+i] = add;
   }
}
```

Fig. 5. Java implementations of the matrix multiplication.

The Renderscript parallel version is shown in Listing 6, in this case we define the operation to be performed to calculate a single matrix C element, this function will be executed as many times as there are elements in the matrix C. Parameters `v_in` and `v_out` contain the elements of the matrix C that are being calculated, the variable `x` represents the element position within the linearised matrix, the value of row and column is obtained on lines 2 and 3. In line 9 the result is copied into the corresponding element of matrix C.

In listing 7 we show the Java code that performs the memory allocation for the matrices (lines 4-9), doing the copy (lines 11, 12 and 15) and getting the pointers for the memory allocated to store the ma-

```
void root(const float *v_in, float *v_out, uint32_t x) {
   int i = x%size;
   int j = x/size;
   int k;
   float add = 0;
   for (k = 0; k < size; k++) {
      add += A[j*size+k]*B[k*size+i];
   }
   *v_out = add;
}
```

Fig. 6. Parallel Renderscript implementations of the Matrix multiplication.

trices (lines 13 and 14). The method that performs the matrix multiplication is called in line 17 using the function `forEach_root` with two parameters, the pointers to the allocated memory of input and output vectors. This function calls the root method (Listing 6) for each element of the matrix C. Finally we obtain the result stored in the reserved memory for the matrix C and it is copied to the matrix C defined in Java (line 18).

B. Convolution

The convolution is a technique for filtering digital images where the filtering is performed directly in each of the pixels of the image (Linear Spatial Filtering). The convolution is to multiply the coefficients of a convolution matrix (kernel) for each image pixel and its neighbors, the values resulting from this operation are summed to obtain the new pixel value. Note that when we change the size of the kernel, the computational load varies. In listing 8 we show the Java implementancion of this problem. In lines 8 and 9 we obtain the vectors that contain all the pixels of the input image (`scrPxs`) and output image (`outPxs`). The loops on lines 11 and 12 traverse all pixels of the image, for each pixel is calculated the border neighboring pixels (lines 14-17) and multiplies each pixel in the neighborhood by the coefficient of the corresponding kernel (windows) (lines 18 -25). Note that this operation is performed for each of the colors that make up the pixel. Finally we check that the calculated values are within the range of valid values (lines 26-31) and assigns the value of the new pixel to the output image (line 32). As in the previous problem, native C and Renderscript sequential implementations are similar to the Java implementation.

In listing 9 we show the parallel implementation of the Renderscript code . As in the previous problems,

```
mRS = RenderScript.create(this);
mScript=new ScriptC_matrix(mRS,getResources(),R.raw.matrix);

Allocation AAllocation = Allocation.createSized(mRS,
                              Element.F32(mRS), size*size);
Allocation BAllocation = Allocation.createSized(mRS,
                              Element.F32(mRS), size*size);
Allocation CAllocation = Allocation.createSized(mRS,
                              Element.F32(mRS), size*size);

AAllocation.copyFrom(A);
BAllocation.copyFrom(B);
mScript.bind_A(AAllocation);
mScript.bind_B(BAllocation);
mScript.set_size(size);

mScript.forEach_root(CAllocation, CAllocation);
CAllocation.copyTo(C);
```

Fig. 7. Java code for parallel Renderscript execution.

```
int width = mBitmapIn.getWidth();
int height = mBitmapIn.getHeight();
int halfWin = kernelSize/2;
int x0, x1, y0, y1;
float R, G, B;
int pixel;

int scrPxs[] = new int[width*height];
int outPxs[] = new int[width*height];
mBitmapIn.getPixels(scrPxs, 0, width, 0, 0, width, height);
for(int x = 0; x < width; x++) {
   for(int y = 0; y < height; y++) {
      R = G = B = 0;
      x0 = Math.max(x-halfWin, 0);
      x1 = Math.min(x+halfWin, width-1);
      y0 = Math.max(y-halfWin, 0);
      y1 = Math.min(y+halfWin, height-1);
      for(int i = x0; i <= x1; i++) {
         for(int j = y0; j <= y1; j++) {
            pixel = scrPxs[j*width + i];
            R += (float)((pixel) & 0xff) * kernel[...];
            G += (float)((pixel >> 8 ) & 0xff)*kernel[...];
            B += (float)((pixel >> 16) & 0xff)*kernel[...];
         }
      }
      if(R < 0) { R = 0; }
      else if(R > 255) { R = 255; }
      if(G < 0) { G = 0; }
      else if(G > 255) { G = 255; }
      if(B < 0) { B = 0; }
      else if(B > 255) { B = 255; }
      outPxs[y*width + x] = ((int)R)+(((int)G) << 8)+...;
   }
}
mBitmapOut.setPixels(outPxs, 0, width, 0, 0, width, height);
```

Fig. 8. Java implementations of the convolution.

this function defines the operation that calculates a single pixel of the image, making a call to this function for each pixel of the image. Parameters `v_in` and `v_out` contain the pixel that is being calculated. In this case we work with a two-dimensional image, the parameters `x` and `y` represent the position of the pixel in the two dimensions of the image. In line 14 we can see the use of vector operations in Renderscript, in this case the variable `pixel` is of type `float4`, a vector of four elements that contains the pixel colours. The clamp function on line 17 checks that the pixels are within the range of correct values.

```
void root(const uchar4 *v_in, uchar4 *v_out,
                              uint32_t x, uint32_t y) {
   float4 pixel;
   int halfWin = kernelSize/2;
   int x0, x1, y0, y1;

   pixel = 0;
   x0 = max((int32_t)x-halfWin, 0);
   x1 = min((int32_t)x+halfWin, mImageWidth-1);
   y0 = max((int32_t)y-halfWin, 0);
   y1 = min((int32_t)y+halfWin, mImageHeight-1);
   for(int i = x0; i <= x1; i++) {
      for(int j = y0; j <= y1; j++) {
         pixel+=convert_float4(gInPixels[...])*kernel[...];
      }
   }
   pixel = clamp(pixel, 0.f, 255.f);
   *v_out = convert_uchar4(pixel);
}
```

Fig. 9. Parallel Renderscript implementations of the convolution.

The Java code that performs the function call to the Renderscript parallel code is shown in listing 10. In Lines 1-7 we do the memory allocations for the input and output images, and for the convolution matrix. The copy of the kernel, variables values, pointers and allocation are performed in lines 9-14. The method that performs the convolution is called in line 16 using two parameters, the input image `mInAllocation` and the output image

Asus Transformer Prime TF201				
Size	Java	Native C	RS Sequential	RS Parallel
100x100	39	56	15	15
500x500	5884	8033	1618	587
1000x1000	63916	79561	21016	7672
Samsung Galaxy SIII				
Size	Java	Native C	RS Sequential	RS Parallel
100x100	52	79	17	12
500x500	6941	9522	1434	701
1000x1000	84882	94336	17338	5737

TABLA I
PERFORMANCE OF MATRIX MULTIPLICATION

`mOutAllocation`. Finally we copied the filtered image as output image (line 17).

IV. COMPUTATIONAL RESULTS

To compare the performance of the different programming models in Android, we implemented four versions of the problems presented, a Java version (Java), a version native C using the NDK (Native), and two Renderscript versions, sequential (RS Sequential) and parallel (RS parallel). We execute these problem over two SoCs devices running Android a Samsung Galaxy SIII (SGS3) and an Asus Transformer Prime TF201 (ASUS TF201). The Samsung Galaxy SIII is composed of an Exynos 4 (4412) holding a Quad-core ARM Cortex-A9 processor (1400MHz), 1GB of RAM memory and a GPU ARM Mali-400/MP4. The Asus Transformer Prime TF201 is composed of a NVIDIA Tegra 3 holding a Quad-core ARM Cortex-A9 processor (1400MHz, up to 1.5 GHz in single-core mode), 1GB of RAM memory and a GPU NVIDIA ULP GeForce. Both devices have installed Android version 4.1. In all cases, the Java version will be used as the reference to calculate the speedup.

Table I shows the execution times in milliseconds for matrix multiplication. To simplify the problem we use square matrices of size 100, 500 and 1000. In this case we observe that the Renderscript implementations presented the best results, while the native C implementations gave the worst performance. As for Renderscript implementations, we do not observe differences between sequential and parallel versions for small problem sizes, when the problem size increases the parallel version is faster.

Figures 11 and 12 show the speedup of each implementation using all available devices. This problems have a high computational load and the Renderscript implementations get a good speedup, particularly the parallel implementation. Among the devices used, the SGS3 obtained a better speedup than the Asus TF201.

```
mInAllocation = Allocation.createFromBitmap(mRS, mBitmapIn,
                    Allocation.MipmapControl.MIPMAP_NONE,
                                  Allocation.USAGE_SCRIPT);
mOutAllocation = Allocation.createTyped(mRS,
                                  mInAllocation.getType());
mKernelAllocation = Allocation.createSized(mRS,
                      Element.F32(mRS), windows*windows);

mKernelAllocation.copyFrom(kernel);
mScript.bind_kernel(mKernelAllocation);
mScript.bind_gInPixels(mInAllocation);
mScript.set_mImageWidth(mBitmapIn.getWidth());
mScript.set_mImageHeight(mBitmapIn.getHeight());
mScript.set_kernelSize(kernelSize);

mScript.forEach_root(mInAllocation, mOutAllocation);
mOutAllocation.copyTo(mBitmapOut);
```

Fig. 10. Java code for parallel Renderscript execution.

Table II shows the execution times in milliseconds for the convolution. For this problem we use two sized images 640×480 and 1024×768. Each of these images we applied the convolution algorithm using kernels sizes of 3×3, 5×5, 7×7 and 9×9. In all cases analysed the best results are obtained using Renderscript implementations, being the parallel version the fastest.

Figures 13 and 14 show the speedup of each implementation using all available devices. The computational load of this problem depends on the window size used, when the window size is increased, the computational load of the algorithm is also increased. In all cases the Renderscript implementations presented the best speedup, when using larger window sizes, the speedup increases. Note that, both Renderscript implementations used vector operations, making this problem to get the best speedups of all the problems implemented.

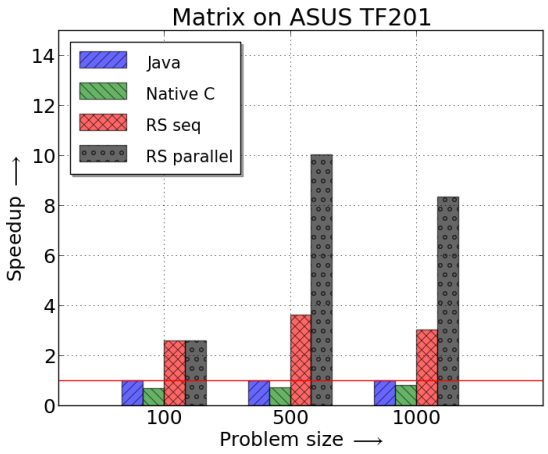


Fig. 11. Speedup of Matrix Multiplication on ASUS TF201

Asus Transformer Prime TF201					
Size	Window Size	Java	Native C	RS Seq	RS Parallel
640 x 480	3x3	430	590	102	69
	5x5	972	1462	186	96
	7x7	1761	2735	309	112
	9x9	2840	4416	469	163
1024 x 768	3x3	1200	2064	412	96
	5x5	2652	4928	637	168
	7x7	4759	9293	1014	236
	9x9	7971	15158	1729	465
Samsung Galaxy SIII					
Size	Window Size	Java	Native C	RS Seq	RS Parallel
640 x 480	3x3	476	819	129	53
	5x5	1169	2035	226	73
	7x7	2117	3853	379	139
	9x9	3411	6228	572	198
1024 x 768	3x3	1447	2520	415	128
	5x5	3271	6087	759	227
	7x7	5830	11245	1300	340
	9x9	9451	18333	2053	558

TABLA II
PERFORMANCE OF THE CONVOLUTION PROBLEM

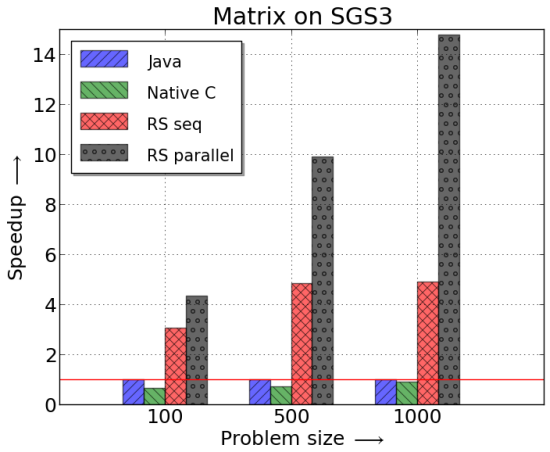


Fig. 12. Speedup of Matrix Multiplication on SGS3

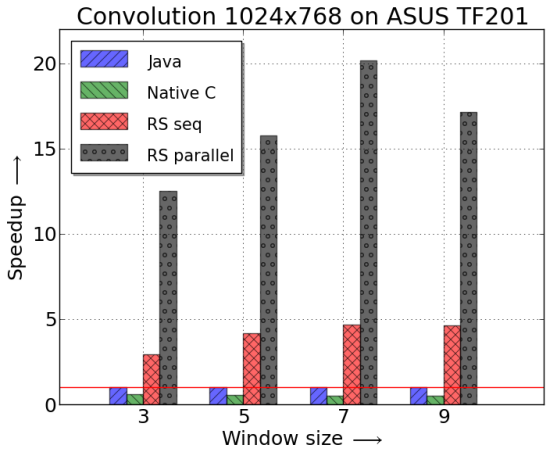


Fig. 13. Speedup of the Convolution Problem on ASUS TF201

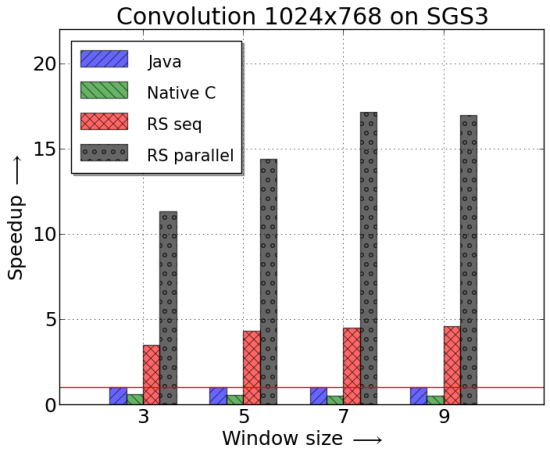


Fig. 14. Speedup of the Convolution Problem on SGS3

V. Conclusion

We conducted a performance comparison between the different programming models available in Android, each model has been analysed under different conditions using two test problems. These problems have different characteristic and cover a wide range of situations that a developer can found when implementing their applications. The results show that Java is the better option for the problems with low computational load because of the costs associated with data transfers in Renderscript and JNI models. If the computational load is high, the Renderscript implementations are the best options. The performance is increased in this class of problems when vector operators can be introduced. The Native C model is useful for reuse existing code libraries written in native languages but the cost associated to JNI penalizes the performance.

Acknowledgment

This work has been supported by the EC (FEDER) and the Spanish MEC with the I+D+I contract number: TIN2011-24598

Referencias

[1] SoCC, "IEEE International System–on–Chip Conference," http://www.ieee-socc.org/, Sept. 2012.
[2] NVIDIA, "NVIDIA Tegra mobile processors: Tegra2, Tegra 3 and Tegra 4," http://www.nvidia.com/object/tegra-superchip.html.
[3] Texas Instruments, "OMAP™Mobile Processors : OMAP™5 platform," http://www.ti.com/omap5.
[4] Google, "Android mobile platform," http://www.android.com.
[5] Apple, "iOS: Apple mobile operating system," http://www.apple.com/ios.
[6] Microsoft, "Windows Phone: Microsoft mobile operating system," http://www.microsoft.com/windowsphone.
[7] Samsung, "Bada: Samsung mobile operating system," http://developer.bada.com.
[8] Nokia, "Nokia Belle: lastest Nokia symbian platform," http://www.developer.nokia.com/.
[9] Alastair D. Reid, Krisztián Flautner, Edmund Grimley-Evans, and Yuan Lin, "SoC-C: efficient programming abstractions for heterogeneous multicore systems on chip," in *Proceedings of the 2008 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES'08*, Erik R. Altman, Ed., Atlanta, GA, USA, Oct. 2008, pp. 95–104, ACM.
[10] Memoir Systems, "Algorithmic Memory ™technology," http://www.memoir-systems.com/.
[11] Nvidia, "GPUDirect Technology," http://developer.nvidia.com/gpudirect.
[12] Anandtech, "AMD Outlines HSA Roadmap: Unified Memory for CPU/GPU in 2013, HSA GPUs in 2014," http://www.anandtech.com/show/5493/.
[13] Android, "Android, the world's most popular mobile platform," http://developer.android.com/about.
[14] O. Cinar, *Pro Android C++ with the NDK*, Apress, 2012.
[15] Steve Wilson and Jeff Kesselman, *Java Platform Performance - Strategies and Tactics*, Addison-Wesley, 2000.
[16] Dawid Kurzyniec and Vaidy Sunderam, "Efficient cooperation between java and native codes jni performance benchmark," in *In The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2001.
[17] H. Guihot, *Pro Android Apps Performance Optimization*, Apressus Series. Apress, 2012.
[18] Jason Sams, "Levels in renderscript," http://android-developers.blogspot.com.es/2012/01/levels-in-renderscript.html.
[19] Jason Sams, "Evolution of renderscript performance," http://android-developers.blogspot.com.es/2013/01/evolution-of-renderscript-performance.html.
[20] Android, "Native development kit (ndk)," http://developer.android.com/tools/sdk/ndk/index.html.
[21] clang, "a c language family frontend for llvm," http://clang.llvm.org/.

Tecnología Grid, Clúster y Plataformas Distribuidas

Selección Eficiente de Recursos en Entornos Grid. Optimización Basada en Redes de Mundo Pequeño

María Botón-Fernández[1], Francisco Prieto-Castrillo[1] y Miguel A. Vega-Rodríguez[2]

Resumen— **El fenómeno de *mundo pequeño* se encuentra presente en un amplio rango de redes naturales, biológicas, sociales o de transporte. La idea principal es que nodos aparentemente distantes están unidos por un pequeño camino de saltos. En este estudio se aplica dicho fenómeno para resolver el problema de selección de recursos en entornos grid. El modelo propuesto identifica, en un número pequeño de pasos, aquellos recursos que mejor se adapten a las condiciones de una determinada aplicación. De este modo, se dota a la aplicación de una capacidad auto-adaptativa a los cambios del entorno. Finalmente, el modelo de selección es testado en una infraestructura grid real. A partir de los resultados obtenidos se deduce que se consigue tanto una reducción en el tiempo de ejecución de la aplicación como un incremento de la tasa de tareas finalizadas con éxito.**

Palabras clave— **Fenómeno mundo pequeño, Optimización, Auto-adaptabilidad, Computación grid**

I. Introducción

LOS entornos de computación grid [1][2] son sistemas distribuidos capaces de coordinar recursos heterogéneos, que no están sujetos a un control centralizado, mediante el uso de interfaces y estándares abiertos. Por tanto, un sistema grid interconecta recursos geográficamente distribuidos con diferentes dominios administrativos, respetándose las políticas de seguridad y gestión internas. En concreto, este tipo de infraestructuras satisface las demandas de computación masiva.

Una de las áreas de investigación más importantes dentro de la comunidad grid es la gestión eficiente de recursos. Una característica importante a considerar sobre los entornos grid es la naturaleza dinámica y cambiante de sus recursos, ya que provoca variaciones de la disponibilidad y rendimiento de los mismos en el tiempo. Este hecho afecta negativamente la ejecución de las aplicaciones. Por ello, se ha convertido en un reto realizar de manera eficiente los procesos de descubrimiento, asignación, planificación y monitorización de recursos.

En un principio, este área de investigación se centró en gestionar de la mejor manera la heterogeneidad de los recursos. Sin embargo, no se ha encontrado un estándar capaz de resolver este problema. En los últimos años se está haciendo uso del concepto de *adaptación* para desarrollar soluciones eficientes. Se han presentado muchas estrategias en esta línea: frameworks que gestionan eficientemente las tareas, sistemas grid autónomos y adaptativos, nuevos procesos de descubrimiento y monitorización de recursos, etc. A pesar de la rica proliferación de soluciones, ninguna se ha establecido de forma generalizada entre las múltiples plataformas grid, de manera que el problema persiste. Por este motivo, proponemos un modelo de selección eficiente *ERS* (*Efficient Resources Selection*) que mejora la productividad de la infraestructura sin modificarla.

Nuestro modelo es definido a nivel de usuario para guiar a las aplicaciones durante su ejecución. Para este propósito se fijaron dos objetivos: reducir el tiempo de ejecución de la aplicación y mejorar la tasa de tareas finalizadas con éxito. Esto significa que estamos interesados en descubrir los mejores recursos de la infraestructura en el menor tiempo posible. Estas consideraciones nos han motivado a utilizar el fenómeno de *mundo pequeño* [3]-[5] durante el proceso de selección.

Los matemáticos Watts y Strogatz [3] llevaron a cabo un análisis sobre cierto tipo de redes/grafos aleatorios caracterizadas por unas peculiares propiedades de conectividad. En dicho trabajo, los autores definen las *redes de mundo pequeño* como aquellas que tienen una distancia promedio (*path length*) pequeña, al igual que los grafos aleatorios [6], y un coeficiente de agregación alto (*clustering coefficient*). En nuestro modelo, se espera que aplicando este algoritmo se encuentren los recursos más eficientes en un número pequeño de pasos.

Finalmente, cabe destacar que el modelo ha sido testado en un entorno grid real, perteneciente a la Infraestructura Grid Europea (*European Grid Infrastructure*[1]). En concreto, las pruebas se realizan a través de la Iniciativa Grid Nacional (*NGI*) en España (*Spanish National Grid Initiative*[2]). Durante la fase de evaluación se establecieron dos escenarios de pruebas para verificar que se alcanzaban los objetivos fijados.

El resto del articulo ha sido estructurado de la siguiente manera. La Sección II presenta trabajos relacionados con el tema de estudio. En la Sección III se describe la estrategia de selección propuesta, incluyendo un breve resumen de conceptos grid para un mejor entendimiento del modelo. A continuación, se presenta la etapa de evaluación con los resultados obtenidos en la Sección IV. Finalmente, las conclusiones y trabajos futuros se exponen en la Sección V

[1]Dpto. de Ciencia y Tecnología, Ceta-Ciemat, e-mail: {maria.boton, francisco.prieto}@ciemat.es
[2]Dpto. de Tecnología de Computadores y Comunicaciones, Univ. Extremadura, e-mail: mavega@unex.es

[1]http://www.egi.eu/about/ngis/
[2]http://www.es-ngi.es/

II. Estado del Arte

Como se ha mencionado en la sección anterior, existen trabajos dentro de la comunidad grid que aplican el concepto de *adaptación* para hacer frente a la naturaleza dinámica del entorno. En este sentido, en [7] se describe el proyecto *AppLes*, cuyo objetivo es proporcionar adaptabilidad a los sistemas grid. El proyecto propone una metodología para realizar un scheduling adaptativo.

Por su parte, en [8] se describe un framework basado en Globus[3] diseñado para gestionar los trabajos de manera eficiente. Para mantener un rendimiento adecuado durante la ejecución de la aplicación, es posible adaptar la ejecución de las tareas a los cambios del entorno utilizando técnicas de planificación.

El estudio en [9] se centra en mejorar los procesos de descubrimiento y monitorización. Es por ello que se presenta una estrategia para evitar la sobrecarga del *Sistema de Información IS* y mejorar su rendimiento. Se definen dos algoritmos adaptativos para notificaciones: *sink-based algorithm* y *utilization-based algorithm* (ambos basados en la disponibilidad del *IS* y en requisitos de los datos).

Un sistema grid autónomo capaz de ajustarse al paralelismo de la aplicación dinámicamente, considerando los cambios en los recursos, es expuesto en [10]. En este caso se presentan dos políticas de rescheduling, una basada en el concepto de migración y otra en el de suspensión. Se va midiendo progresivamente el ratio entre el tiempo de ejecución actual y el tiempo de ejecución predicho. Cuando el valor medio de dichos ratios supere el umbral de tolerancia establecido se lanzarán ambas políticas.

En [11] se realiza una investigación relativa a la adaptación en las aplicaciones. Este trabajo recoge información sobre comunicación entre recursos y tiempos de procesamiento de forma periódica. Con esta información se hace una estimación de los requisitos de la aplicación en cada momento, reemplazando los recursos sobrecargados y reduciendo los cuellos de botella. Finalmente, en [12] se ha realizado una comparativa de sistemas adaptativos en computación grid.

Los trabajos comentados en esta sección tienen una característica común: mejorar el rendimiento de la infraestructura grid. Diferentes técnicas son empleadas para tal fin (scheduling, rescheduling, políticas de notificación ec.). Todas ellas han sido diseñadas desde el punto de vista del sistema (administración). Sin embargo, en este trabajo se propone un modelo auto-adaptativo para selección de recursos que ha sido diseñado desde el punto de vista del usuario. Este modelo no modifica ni controla el comportamiento de la infraestructura, es decir, no usa técnicas de planificación ni migración, ni altera el comportamiento de los recursos. Además, se espera mejorar la productividad de la infraestructura a la vez que se guía a las aplicaciones para manejar los cambios del entorno.

[3]http://www.globus.org/

III. Definición del Modelo Eficiente

En este trabajo se propone una formulación matemática para medir la eficiencia de los recursos grid de la infraestructura. Esta formulación en combinación con el algoritmo *Small-world* da lugar al modelo auto-adaptativo de selección eficiente. Dicho modelo, como se indica en la Sección I, se denota como *ERS*. Antes de continuar con la descripción del mismo, se resumen los elementos y conceptos básicos de una infraestructura grid, de manera que sea más fácil entender nuestra estrategia.

A. Conceptos Grid Básicos

En una infraestructura grid típica los usuarios interaccionan con diferentes componentes a través de la *Interfaz de Usuario* (*UI*). Esta máquina es el punto de acceso a la infraestructura, por lo que, en ella se realizan los procesos de autenticación y autorización. Una vez se ha accedido a la *UI*, los usuarios pueden enviar sus trabajos a la infraestructura; estos trabajos son gestionados por el *Sistema de Gestión de Carga* (*WMS*). Este servicio está alojado en una máquina física conocida como *Planificador de Recursos* (*RB*), cuya función es determinar a qué *Planificador de Site* (*CE*) enviar las tareas. También se encarga de registrar el estado y la salida de los trabajos (en este punto también interactúa el *Sistema de Información IS*). el proceso que utiliza el *RB* para escoger el planificador *CE* al que enviar las tareas es conocido como *match-making* y está basado en criterios de disponibilidad y cercanía. Finalmente, el *CE* es el planificador del site que determina en qué *Nodo de Cómputo* (*WN*) se ejecutarán las tareas.

El modelo propuesto en esta contribución ha sido definido a nivel de usuario, por lo que la interacción con la infraestructura se lleva a cabo utilizando únicamente los comandos propios de los usuarios. Por otro lado, los recursos que el modelo monitoriza, clasifica y selecciona en base a su eficiencia son los *CEs*. En las siguientes subsecciones se describe nuestra propuesta.

B. Midiendo la Eficiencia de los Recursos

El modelo *ERS* escoge los recursos que mejor se ajusten a los requisitos de la aplicación monitorizando de forma contínua la eficiencia de los mismos. Para obtener este valor de eficiencia se definen dos parámetros, que podemos considerar como los *principales* del modelo: el valor histórico de tareas finalizadas[4] ϵ_i para el recurso *i-ésimo* y, por otro lado, el valor histórico de tiempo de procesamiento μ_i empleado en dichas tareas. Ambos parámetros son calculados para cada *CE* durante la ejecución de la aplicación.

[4]Toda tarea cuyo estado registrado en el *IS* sea DONE o ABORTED se considerará como tarea finalizada/procesada. Las tareas cuyo *tiempo de vida* finaliza antes de que lleguen a completarse serán también consideradas como tareas procesadas. El concepto de *tiempo de vida* fue introducido para evitar sobrecarga de recursos.

El parámetro ϵ_i es definido como el ratio entre la tasa de tareas finalizadas con éxito SFt_i y el número de tareas totales At_i que le fueron asignadas al recurso i, como se observa en la ecuación Eq. 1

$$\epsilon_i = SFt_i/At_i \ . \quad (1)$$

En cuanto al parámetro μ_i, por cada tarea j se mide el tiempo de procesamiento consumido por el recurso i para procesarla (Eq. 2). El tiempo de procesamiento $T_{i,j}$ considera tanto el tiempo de comunicación del recurso $Tcomm_{i,j}$ con otros servicios grid como el tiempo de computación $Tcomp_{i,j}$ para la tarea j.

$$T_{i,j} = Tcomm_{i,j} + Tcomp_{i,j} \ . \quad (2)$$

Todos estos valores de $T_{i,j}$ que se obtienen $\{T_{i,1}, T_{i,2}, ..., T_{i,SFt_i}\}$ sirven para calcular el valor medio de tiempo de procesamiento $\bar{\chi_i}$ del recurso *i-ésimo* (Eq. 3) para las tareas procesadas SFt_i en un instante dado.

$$\bar{\chi_i} = (\sum_{j=1}^{SFt_i} T_{i,j})/SFt_i \ . \quad (3)$$

Una vez obtenidos estos valores, el histórico μ_i es calculado empleando tanto el valor normalizado de $\bar{\chi_i}$ como el *tiempo de vida* asignado a la tarea (Eq. 4).

$$\mu_i = (lt - \bar{\chi_i})/lt \ . \quad (4)$$

Por último, la eficiencia de un recurso i es medida utilizando los dos parámetros *principales* ϵ_i y μ_i, a los que se les ha asociado un peso o valor de *relevancia* (parámetros a y b en Eq. 5). Estos parámetros fueron introducidos en el modelo para permitir a los usuarios especificar las condiciones y prioridades de sus experimentos.

$$E_i = (a \cdot \epsilon_i + b \cdot \mu_i)/(a + b) \ . \quad (5)$$

C. Selección Eficiente. Combinando la Formulación Matemática con el Fenómeno de Mundo Pequeño

Como se ha mencionado anteriormente, el modelo *ERS* tiene dos objetivos: por un lado reducir el tiempo de ejecución de las aplicaciones y, por otro lado, mejorar la tasa de tareas procesadas con éxito. Se espera, además, que ambos objetivos incrementen la productividad de la infraestructura.

El modelo se basa en el mapeo de dos espacios de trabajo: **1)** un espacio de tareas J constituido por las n tareas independientes y paralelas que forman la aplicación (las tareas sólo difieren en los valores de los parámetros de entrada). **2)** Un espacio de recursos dinámico y heterogéneo donde se incluyen los recursos disponibles en la infraestructura. Se puede decir que este mapeo es una relación del tipo *muchos-a-uno*, es decir, una o más tareas pueden ser asignadas a un mismo *CE*. Como se aprecia en la Figura 1, al comienzo de la ejecución el modelo genera un subconjunto de J - denominado como T - de manera que envía dicho conjunto a ejecución. Esto se hace así para fomentar un aprendizaje más rápido por parte del modelo, de manera que se conozca de forma más rápida los recursos eficientes del entorno. Los recursos seleccionados para procesar este subconjunto pertenecen a R y forman el conjunto que denotaremos como RT. Cabe destacar que, al no tener métricas de eficiencia en este momento el modelo escoge dichos recursos de forma aleatoria.

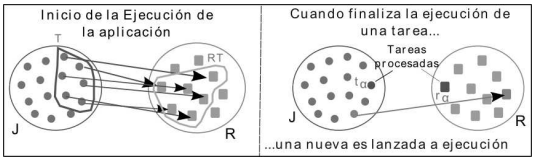


Fig. 1. Representación de la gestión de los espacios de trabajo J y R durante la ejecución de la aplicación. Puede verse que al inicio de la misma ambos espacios tienen una gestión especial respecto al resto de la ejecución.

Durante el resto de la ejecución el modelo gestiona los espacios de la siguiente manera: cuando una tarea $t_\alpha \in J$ termina su ejecución se medirá el grado de eficiencia del recurso asociado $r_\alpha \in R$. A continuación, y basándose en el fenómeno *mundo pequeño*, un recurso será escogido de manera eficiente para ejecutar una nueva tarea.

Una vez establecidas las reglas que gobiernan la gestión de los espacios J y R, el siguiente paso consiste en describir con mayor detalle el proceso de selección eficiente que caracteriza a nuestra propuesta.

En una red de *mundo pequeño* se combinan una búsqueda local junto con una excursión aleatoria de largo recorrido. A este tipo de algoritmos se les conoce como *Algoritmos de Optimización de Mundo Pequeño* (*SWOA*) [13]. Los elementos que han sido incorporados a ambos procesos de búsqueda en nuestro modelo son descritos a continuación:

- Un umbral de carga de trabajo ϖ utilizado para pasar de una búsqueda local a una búsqueda de largo recorrido. Por tanto, se calcula la carga de trabajo[5] real de cada recurso monitorizado. El modelo asume que un recurso está sobrecargado cuando su valor de carga es igual o superior a ϖ.
- Conjunto de recursos evaluados S_E: vector ordenado de menor a mayor valor de eficiencia que es utilizado durante la búsqueda local. Este vector está compuesto por aquellos recursos cuya eficiencia ya ha sido monitorizada (recursos ya utilizados durante la ejecución).
- Conjunto de recursos no evaluados S_{UE}: vector que incluye los recursos no evaluados del espacio R (aquellos que no han sido seleccionados ninguna vez hasta la fecha).

Búsqueda Local

El objetivo de esta búsqueda es escoger, de una manera eficiente, un recurso perteneciente a la vecindad

[5]Para obtener este valor se mide no sólo la carga real producida por nuestra aplicación sino también la carga local (propia de otros experimentos que están explotando los recursos). Dicho valor es normalizado.

de r_α. Se va a considerar como *vecindad* de un recurso r_α al conjunto formado por los dos vecinos más próximos a él (v_1 y v_2 en la Figura 2); es decir, aquellos dos recursos con valores de eficiencia próximos a r_α serán considerados sus vecinos. Existen dos casos especiales donde la vecindad de r_α está compuesta por un único recurso: cuando su valor de eficiencia es el más alto o el más bajo de todos.

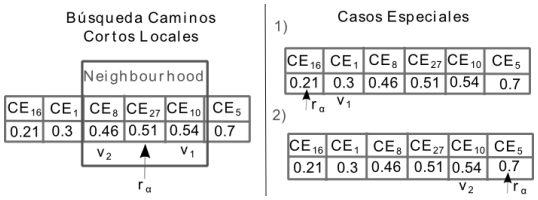


Fig. 2. Proceso de Búsqueda Local para el modelo *ERS*. Dentro de este proceso se tienen en cuenta la eficiencia y carga de trabajo de los recursos.

Cada vez que se calcula el valor de eficiencia de r_α será insertado de manera ordenada en el conjunto S_E. A continuación, el modelo selecciona uno de sus vecinos para ejecutar la siguiente tarea t_α. Como primera opción se considerará al vecino más eficiente v_1 y, en caso que esté sobrecargado (en base a ϖ), se escoge el segundo vecino v_2. Si este recurso estuviese también sobrecargado, el modelo procedería a utilizar la excursión aleatoria de largo recorrido (se trata de un salto en la búsqueda motivado por ϖ).

Excursión Aleatoria de Largo Recorrido
En cuanto a este salto largo que se produce en el proceso de búsqueda, se hace uso del conjunto S_{UE}. El modelo selecciona aleatoriamente un recurso perteneciente a dicho conjunto. Este recurso sólo debe cumplir un requisito: no estar sobrecargado; de manera que esta excursión aleatoria es repetida hasta dar con un recurso que lo satisfaga.

Al igual que ocurre en la búsqueda local, se tienen dos casos especiales: que todos los recursos de S_{UE} estén sobrecargados o que S_{UE} esté vacío. En ambos casos la excursión aleatoria pasaría a realizarse sobre el conjunto S_E.

Para finalizar, vamos a resumir el comportamiento del modelo. En primer lugar, un subconjunto de J denominado T es lanzado a ejecución. Las tareas que forman dicho subconjunto son monitorizadas hasta que terminan su ejecución. Cuando una tarea t_α es procesada, el recurso asociado a la misma r_α pasa a ser evaluado. Este recurso es insertado en orden en S_E en base a su valor de eficiencia. A continuación, el modelo aplica la propiedad de *mundo pequeño*: se realiza una búsqueda local para ver si es factible utilizar algunos de los vecinos de r_α. En caso de que los dos vecinos más próximos estén sobrecargados el algoritmo realiza un salto de largo recorrido. Cuando la búsqueda finaliza el modelo asigna una nueva tarea t_α al recurso seleccionado de manera eficiente. Estos procesos de monitorización, evaluación y selección son repetidos hasta que todas las tareas $t_\alpha \in J$ son procesadas. Cabe destacar que el modelo registra toda la información generada durante la ejecución de la aplicación en unos ficheros de salida.

IV. Evaluación del Modelo

El modelo es evaluado en una infraestructura grid real perteneciente a la Infraestructura Grid Europea (*EGI*)[6]. La Iniciativa Grid Nacional en España (*ES-NGI*)[7] está compuesta por una amplia variedad de proyectos, cada uno de ellos utilizando un entorno de computación específico dentro de la infraestructura. En concreto, nosotros nos hemos afiliado a una de las *VO* del proyecto Ibergrid[8] que tiene alrededor de 30 *CEs* (un valor de recursos razonable para evaluar nuestra propuesta).

Dos escenarios de experimentación han sido definidos para verificar que los objetivos fijados (reducir el tiempo de ejecución total de la aplicación e incrementar la tasa de tareas finalizadas con éxito) se han cumplido. En ambos escenarios el presente modelo *ERS-SW* (*Modelo de Selección Eficiente* basado en el fenómeno de *Mundo Pequeño*) es comparado con la *Selección Tradicional de Recursos* (*TRS*) en las infraestructuras grid europeas. Esta selección está basada en criterios de cercanía y disponibilidad de los recursos.

A. Escenario 1

En este primer escenario se pretende evaluar la capacidad de aprendizaje del modelo *ERS* y hasta qué punto se ve influenciado por el tamaño del subconjunto T. El escenario está formado por 5 pruebas y, por cada una de ellas, se realizan 10 ejecuciones de ambos algoritmos (10 para cada test de *TRS* y otros 10 para cada test de *ERS-SW*). El tamaño del espacio J es fijado a 200 tareas en todos los tests. Por su parte, el tamaño del subconjunto T varía desde 5 hasta 40 tareas (como se observa en la Figura 3).

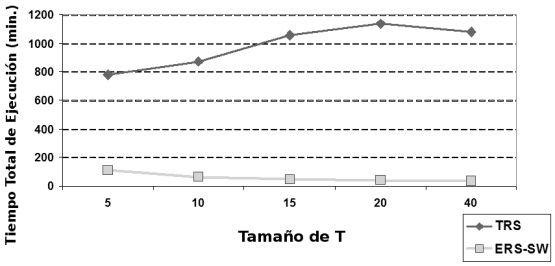


Fig. 3. Resultados obtenidos en los diferentes tests que se llevan a cabo en el primer escenario de evaluación.

Como se observa en la Figura 3, el modelo propuesto *ERS-SW* reduce el tiempo de ejecución respecto a *TRS*. Puede apreciarse, además, que a medida que el tamaño de T aumenta, crece la diferencia de tiempos de ejecución entre ambas versiones. Esto quiere decir que para valores altos de T el modelo encuentra los recursos eficientes más rápidamente (aprendizaje del modelo más rápido). Por otra parte,

[6]http://www.egi.eu/about/ngis/
[7]http://www.es-ngi.es/
[8]http://www.ibergrid.eu/

si nos fijamos en la tasa de tareas finalizadas por ambas versiones (Figura 4), el modelo *ERS-SW* nuevamente supera a *TRS*.

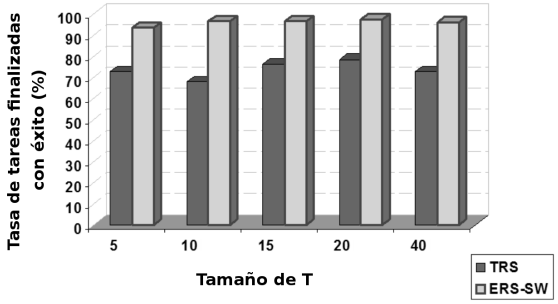


Fig. 4. Tasa de tareas finalizadas con éxito durante las pruebas realizadas en el primer escenario para ambas versiones. El modelo *ERS-SW* obtiene mejores resultados.

B. Escenario 2

En este segundo escenario el objetivo es determinar el rango de aplicaciones donde es posible aplicar nuestro modelo. En este caso, fijamos el tamaño de T a 10 ya que a partir de los resultados anteriores (escenario 1) se observa que es el menor valor con el que se alcanza un buen aprendizaje del modelo. En esta ocasión se van a realizar 6 tests, de manera que el tamaño del espacio J varía de unos a otros (desde 50 hasta 500 tareas a ejecutar).

Al igual que ocurría en el primer escenario, el modelo *ERS-SW* mejora los resultados obtenidos por *TRS* (Figura 5). Para los dos primeros valores de J la diferencia de tiempo de ejecución entre ambas versiones no es muy grande. Sin embargo, esta diferencia va aumentando progresivamente (se hace más significativa) a medida que aumenta el tamaño de J. Por ejemplo, para un tamaño de J de 500 tareas, *TRS* necesita unos 3019 minutos (unas 50 horas) mientras que *ERS-SW* sólo utiliza 108 minutos.

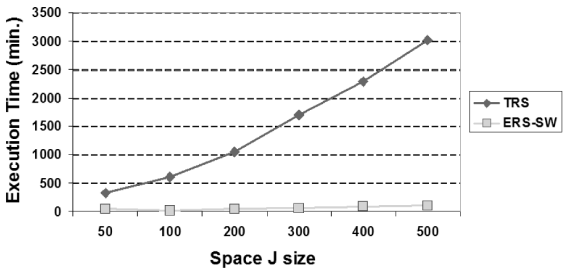


Fig. 5. Resultados obtenidos para las 6 pruebas realizadas en este segundo escenario. *ERS-SW* nuevamente supera a la versión *TRS*.

En cuanto a la tasa de tareas finalizadas con éxito (ver Figura 6), en este segundo escenario es mejorada un 25%. *TRS* tiene una tasa de éxito del 70% mientras que *ERS-SW* la tiene del 95%. Se incluye, además, una gráfica (Figura 7) donde se muestra el tiempo medio empleado por cada versión en el procesamiento de una tarea para los diferentes tests realizados. Puede apreciarse que, en general, *ERS-SW* tarda alrededor de 1 minuto en ejecutar una tarea mientras que *TRS* necesita entre 5 y 7 minutos. En conclusión, se puede decir que es beneficioso utilizar el modelo auto-adaptativo propuesto para aplicaciones de computación masiva en entornos grid.

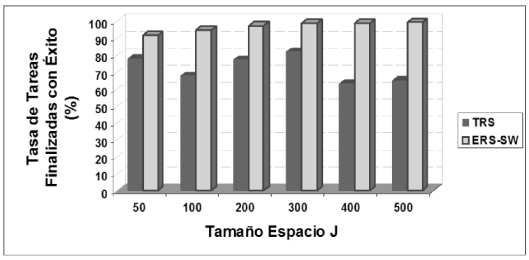


Fig. 6. Tasa de tareas ejecutadas correctamente en este segundo escenario. Puede observarse que *ERS-SW* está muy próximo a un valor de tasa del 100%.

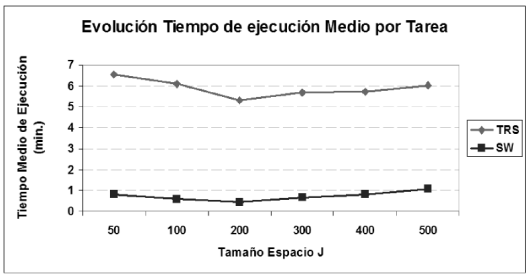


Fig. 7. Tiempo medio de ejecución de tareas de las dos versiones comparadas, *TRS* y *ERS-SW*.

V. Conclusiones

En la presente contribución se propone un modelo de selección eficiente para resolver el problema de gestión de recursos en entornos grid. El modelo *ERS* ha sido diseñado desde el punto de vista del usuario y está basado en el fenómeno de *mundo pequeño*. Dicho modelo proporciona una capacidad auto-adaptativa, permitiendo a las aplicaciones hacer frente a los cambios del entorno.

De los resultados obtenidos en la fase de evaluación se puede deducir que nuestro modelo es una beneficiosa alternativa para los usuarios grid, ya que, reduce los tiempos de ejecución de las aplicaciones e incrementa la tasa de tareas finalizadas con éxito.

En cuanto a trabajo futuro, se enriquecerá el modelo mediante la aplicación de nuevos algoritmos, se reforzarán algunas partes del mismo y se tendrán en cuentan nuevos servicios grid.

Agradecimientos

La investigación de María Botón-Fernández está subvencionada mediante una beca de Formación de Personal Investigador otorgada por el Ministerio de Economía y Competitividad a través del Centro de Investigaciones Energéticas, Medio Ambientales y Tecnológicas (CIEMAT). Los autores quieren agradecer también el soporte de los Fondos Europeos para el Desarrollo Regional (FEDER).

Referencias

[1] I. Foster, *What is the Grid? A three Point Checklist*, GRIDtoday, Vol. 1, No. 6, pp. 22-25, 2002.

[2] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the GRID. Enabling Scalable Virtual Organizations*, Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L.(eds.) Euro-Par 2001, LNCS, Vol. 2150, Springer-Verlag, Heidelberg, pp. 1-4, 2001.

[3] D.J. Watts, and S.H. Strogatz, *Collective Dynamics of Small-world Networks*, Nature Vol. 393, Nature Publishing Group, pp. 440-442, 1998.

[4] J. Kleinberg, *The Small-world Phenomenon: an Algorithm Perspective*, proceedings of The Thirty-second Annual ACM Symposium on Theory of Computing, pp. 163-170, 2000.

[5] M. Newman, A.-L. Barabási, D.J. Watts, D.J. *The Structure and Dynamics of Network*, Princeton University Press, 2006.

[6] P. Erdos, and A. Rény, *On the Evolution of Random Graphs*, Publications of the Mathematical Institute of the Hungarian Academy of Sciences, Vol. 5, pp. 17-61, 1960.

[7] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, *Adaptive Computing on the Grid Using AppLeS*, IEEE Transactions on Parallel and Distributed Systems, Vol. 14, Issue 4, pp. 369-382, April 2003.

[8] E. Huedo, R.S. Montero, and I.M. Llorente *A Framework for Adaptive Execution in Grids*, Software-Practice & Experience, Vol. 34(7), pp. 631-651, 2004.

[9] H.N.L.C. Keung, J.R.D. Dyson, S.A. Jarvis, and G.R. Nudd, *Self- Adaptive and Self-Optimising Resource Monitoring for Dynamic Grid Environments*, DEXA ?04 Proceedings of the Database and Expert Systems Applications, 15th International Workshop, Washington DC, USA, pp. 689-693, 2004.

[10] S.S. Vadhiyar, and J.J. Dongarra, *Self Adaptivity in Grid Computing*, Concurrency and Computation: Practice and Experience, Vol. 17(2-4), pp. 235-257, 2005.

[11] G. Wrzesinska, J. Maasen, and H.E. Bal, *Self-adaptive Applications on the Grid*, Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Jose, California, USA, pp. 121-129, 2007.

[12] D.M. Batista, and L.S. Da Fonseca, *A Survey of Self-adaptive Grids*, IEEE Communications Magazine, Vol. 48, Issue 7, IEEE Press Piscataway, NJ, USA, pp. 94-100, 2010.

[13] H. Du, X. Wu, and J. Zhuang, *Small-world Optimization Algorithm for Function Optimization*, ICNC, Part II, LNCS 4222, pp. 264-273, Springer-Verlag Berlin Heidelberg, 2006.

Solución Auto-adaptativa para Selección Eficiente de Recursos en Entornos Grid. Aplicando Inteligencia Colectiva

María Botón-Fernández[1], Miguel A. Vega-Rodríguez[2] y Francisco Prieto Castrillo[1]

Resumen— **Los algoritmos de inteligencia colectiva son muy utilizados para simular el comportamiento de sistemas auto-organizativos y no centralizados, los cuales pueden ser naturales o artificiales. En este sentido, en el presente trabajo aplicamos el algoritmo de Colonia Artificial de Abejas (*ABC*) para mejorar el proceso de selección en entornos grid (sistemas distribuidos compuestos por recursos heterogéneos y geográficamente dispersos). El modelo propuesto se ha diseñado para seleccionar los recursos más eficientes durante la ejecución de una aplicación. Para ello, los recursos son considerados como fuentes de alimento con una gran concentración de néctar. Cabe destacar que el modelo proporciona, además, una capacidad auto-adaptativa permitiendo que las aplicaciones hagan frente a los cambios del entorno. Para finalizar, el modelo es evaluado en una infraestructura grid real. De los resultados obtenidos puede deducirse que no sólo se consigue una reducción en el tiempo de ejecución sino que se mejora la tasa de tareas finalizadas con éxito.**

Palabras clave— **Colonia Artificial de Abejas, Optimización, Computación Grid, Auto-adaptación, Inteligencia Colectiva**

I. Introducción

LOS entornos de computación grid [1] [2] son sistemas distribuidos cada vez más utilizados por la comunidad científica en la última década. Este tipo de infraestructuras se caracteriza por estar compuestas por recursos heterogéneos que, a su vez, tienen una localización geográfica distinta. Además, los recursos grid son gestionados a través de Organizaciones Virtuales (*VO*) a las que deben afiliarse los usuarios para hacer uso de las infraestructuras grid. Cada *VO* por lo general está asociada a un proyecto de investigación.

A pesar de las ventajas de este tipo de infraestructuras, presentan varios problemas relacionados con las características y naturaleza grid. Existen dos niveles de heterogeneidad en los sistemas grid. Por un lado, los recursos del entorno pertenecen a diferentes centros, cada uno de ellos con diferentes sistemas operativos y distintos sistemas administrativos. Como los principios fundamentales de las infraestructuras grid permiten el uso de diferentes políticas de gestión, nos encontramos con que cada centro maneja/mantiene sus recursos acorde a sus propias políticas. Por otro lado, los recursos pueden ser agrupados en base a su funcionalidad en el sistema. Pero, incluso dentro de cada grupo los recursos tienen distintas características a nivel de hardware y de software (por pertenecer a distintos centros). A todo esto debemos añadir que el permitir diferentes dominios administrativos provoca una variación en el rendimiento y la disponibilidad de los recursos a lo largo del tiempo. Es por todo ello que las aplicaciones grid necesitan conocer el estado de la infraestructura en tiempo real. Pero también, ciertos procesos como los de descubrimiento, monitorización y selección de recursos deben ser mejorados para conseguir un sistema más autónomo.

En los últimos años, la comunidad grid se ha centrado en diseñar y desarrollar soluciones adaptativas para abordar los problemas anteriormente descritos. En ese sentido, en el presente trabajo proponemos una solución centrada en mejorar el proceso de selección eligiendo los recursos más eficientes durante la ejecución de la aplicación. Este modelo propuesto proporciona así una capacidad auto-adaptativa a las aplicaciones. Para conseguir esa selección eficiente nos hemos basado en el algoritmo de colonia de abejas [3] (más comúnmente conocido como *ABC - Artificial Bee Colony*). Se trata de un algoritmo del campo de la *Inteligencia Colectiva*, muy utilizado en problemas de optimización y basado en el comportamiento de las abejas en la búsqueda de fuentes de alimentos. A este modelo lo hemos llamado *Modelo de Selección Eficiente basado en Colonia Artificial de Abejas* (*ERS-ABC*). En la fase de definición se establecieron dos objetivos: reducir el tiempo de ejecución de las aplicaciones y aumentar la cantidad de tareas acabadas con éxito.

El modelo es evaluado en una infraestructura grid real perteneciente a la Infraestructura Grid Europea (*EGI*)[1]. En la fase de evaluación dos escenarios se han establecido para verificar que los objetivos fueron alcanzados. Para ello, *ERS-ABC* es comparado con el proceso de selección estándar de las infraestructuras grid (basadas en el middleware de gLite[2]), al que nosotros denotamos como *TRS* (Selección Tradicional de Recursos).

El resto del artículo se ha estructurado de la siguiente manera. La Sección II presenta el trabajo relacionado. En la Sección III se describen todas las consideraciones del modelo junto con la formulación matemática para medir la eficiencia de los recursos. Por su parte, la Sección IV especifica cómo se adapta el algoritmo *ABC* en el modelo para realizar

[1]Dpto. de Ciencia y Tecnología, Ceta-Ciemat, e-mail: {maria.boton, francisco.prieto}@ciemat.es
[2]Dpto. de Tecnología de Computadores y Comunicaciones, Univ. Extremadura, e-mail: mavega@unex.es

[1]http://www.egi.eu/
[2]http://glite.cern.ch/

la selección eficiente. La etapa de evaluación queda recogida en la Sección V. Finalmente, la Sección VI concluye el artículo.

II. Estado del Arte

Como se ha comentado anteriormente, existe un número de investigaciones que utilizan el concepto de *adaptación* para resolver los problemas relacionados con la gestión de recursos grid. En [4] se describe el proyecto *AppLes*, cuyo objetivo es proporcionar una capacidad adaptativa a los sistemas grid. En el proyecto se crea una metodología para desarrollar y desplegar aplicaciones distribuidas de alto rendimiento, es decir, se propone una planificación adaptativa.

El estudio en [5] describe un framework para ejecuciones adaptativas en grid. Este framework está basado en Globus[3] y ha sido diseñado para gestionar las tareas eficientemente. Mediante nuevas políticas de planificación buscan mantener un nivel de rendimiento adecuado.

Otro trabajo, presentado en [6], se centra en mejorar los procesos grid de descubrimiento y monitorización. Se introduce una aproximación que evita la sobrecarga del Sistema de Información (IS) mediante el diseño de dos algoritmos de notificación: *sink-based algorithm* y *utilization-based algorithm*.

El sistema autónomo que se expone en [7] se ajusta dinámicamente al paralelismo de la aplicación utilizando dos políticas de planificación: una para manejar la suspensión de tareas y otra para controlar la migración de la aplicación. Se utiliza un umbral de tolerancia para determinar cuál de las dos políticas aplicar en cada momento.

Para finalizar, en [8] se realiza un resumen de los sistemas adaptativos existentes. En base a los mismos, los autores añaden sugerencias para alcanzar ese sistema autónomo grid.

Los trabajos descritos anteriormente tienen una característica común: buscan mejorar el rendimiento de la infraestructura grid. Además, todas las soluciones comentadas utilizan técnicas (planificación, migración, cancelación, etc.) que han sido diseñadas desde el punto de vista del sistema. Sin embargo, nosotros proponemos un modelo eficiente y autoadaptativo para seleccionar de manera eficiente los recursos. El modelo no controla ni modifica el comportamiento de los elementos grid (no se utilizan técnicas de planificación, migración ni de asignación). Nuestra estrategia ha sido diseñada desde el punto de vista del usuario, utilizando los comandos básicos de los usuarios. Se espera que el modelo mejore la productividad de la infraestructura a la vez que guía a las aplicaciones para hacer frente a los cambios del entorno.

III. Modelo de Selección Eficiente de Recursos

Como se ha mencionado en la Sección I, en este trabajo proponemos un modelo de Selección Eficiente de Recursos (*ERS*) que permita a las aplicaciones auto-adaptarse a las cambiantes condiciones del entorno. Se define una formulación matemática que permita medir la eficiencia de los recursos considerando los objetivos de tiempo y tareas establecidos. Esta formulación ligada a un proceso de selección basado en el algoritmo ABC da lugar a nuestro modelo eficiente.

Se van a manejar dos espacios de trabajo: un espacio de tareas J, compuesto por las n tareas independientes y paralelas de la aplicación, y un espacio dinámico y heterogéneo de recursos R, en el que se incluyen todos los recursos disponibles de la infraestructura. En concreto, se mide la eficiencia de un tipo de planificador grid conocido como *Computing Element* (*CE*). Este componente interacciona con los nodos de cómputo, decidiendo en cuál de ellos serán ejecutadas las tareas. La filosofía de la grid permite a los usuarios interaccionar con este tipo de elementos y especificar con qué *CE* quieren trabajar. Por este motivo se decide monitorizar este tipo de componente grid.

En cuanto a la gestión de las tareas, al inicio de la ejecución el modelo lanza un subconjunto de J al que hemos denotado como T. De esta forma se espera motivar o agilizar el aprendizaje del modelo, determinando más rápidamente cuales son los recursos más eficientes. Las tareas de T son asignadas a una serie de recursos (RT) escogidos aleatoriamente, ya que al inicio de la ejecución de la aplicación no se tienen medidas de la eficiencia de los recursos.

A continuación, el modelo monitoriza las tareas de manera que cuando una tarea t_α termina su ejecución, se mide la eficiencia del recurso asociado y se elige uno nuevo de manera eficiente (aplicando ABC) para ejecutar una nueva tarea. Todos estos pasos de monitorización, clasificación y selección se repiten hasta que todas las tareas de J son procesadas. Cabe destacar que cada tarea tiene asociado un tiempo de vida lt, de esta forma el modelo no espera de forma indefinida por recursos sobrecargados.

En cuanto al valor de eficiencia de un recurso i, se tendrán en cuanta los siguientes parámetros para calcularla. Por un lado se tiene un histórico de tareas procesadas[4] ϵ_i y por otro lado el histórico del tiempo de procesamiento empleado para procesar esas tareas μ_i.

El valor de ϵ_i depende de la cantidad de tareas que fueron finalizadas con éxito SFt_i y del número de tareas totales asignadas a dicho recurso At_i (ver Eq. 1).

$$\epsilon_i = SFt_i/At_i \ . \tag{1}$$

Antes de definir μ_i vamos a especificar cómo se calculan los parámetros en los que se basa. En primer lugar, el tiempo de procesamiento $T_{i,j}$ del recurso para una tarea j se obtiene aplicando Eq. 2.

[3] http://www.globus.org/toolkit/

[4] Toda tarea cuyo estado registrado en el *Sistema de Información* (*IS*) sea *Done* o *Aborted* se considerará como tarea procesada. También aquellas tareas cuyo tiempo de vida se agotó antes de completarse se consideran procesadas.

Este tiempo se basa en el tiempo de comunicación $Tcomm_{i,j}$ entre el recurso i y otros servicios grid y en el tiempo de computación $Tcomp_{i,j}$ que utiliza para procesar la tarea j.

$$T_{\{i,j\}} = Tcomm_{\{i,j\}} + Tcomp_{\{i,j\}} \ . \quad (2)$$

A continuación, se calcula el valor medio de tiempo de procesamiento $\bar{\chi}_i$ del recurso (Eq. 3). El parámetro SFt_i se utiliza para especificar el número de tareas que se han terminado con éxito en un momento dado.

$$\bar{\chi}_i = (\sum_{j=1}^{SFt_i} T_{\{i,j\}})/SFt_i \ . \quad (3)$$

Por tanto, el histórico de tiempo de procesamiento queda definido como se especifica en la ecuación Eq. 4. Este parámetro depende de $\bar{\chi}_i$ y del tiempo de vida lt de las tareas.

$$\mu_i = (lt - \bar{\chi}_i)/lt \ . \quad (4)$$

Para finalizar, la eficiencia se calcula utilizando ambos valores históricos pero también se añaden dos parámetros de relevancia a y b. Ambos parámetros son fijados por los usuarios, de manera que puedan especificar las condiciones de sus experimentos o sus prioridades en los mismos.

$$E_i = (a \cdot \epsilon_i + b \cdot \mu_i)/(a + b) \ . \quad (5)$$

IV. Proceso de Selección Eficiente Basado en ABC

El algoritmo de colonia de abejas *ABC* es un método evolutivo introducido por Dervis Karaboga [3] compuesto por tres tipos diferentes de abejas: obreras, observadoras y exploradoras. La forma de funcionar del algoritmo es la siguiente: una vez que varias fuentes de alimento han sido explotadas (fuentes de néctar), las abejas obreras regresan a la colmena y realizan un baile para indicar al resto la calidad de las fuentes encontradas. A continuación, tanto las abejas obreras como las observadoras salen a explotar fuentes de alimento conocidas (explotadas anteriormente), utilizando la experiencia adquirida por la colonia. Por su parte, las abejas exploradoras eligen fuentes de alimento de manera aleatoria, sin tener en cuenta el conocimiento adquirido. De este modo la colmena aprovecha las fuentes con mayor cantidad de néctar a la vez que explora nuevas posibilidades.

En cuanto al proceso de selección de nuestro modelo *ERS-ABC*, a continuación se indican las consideraciones establecidas para hacer uso de tal algoritmo.

- Las abejas de nuestro modelo van a buscar los recursos más eficientes, aquellos con un valor de eficiencia próximo a 1 (lo que significaría una fuente de alimento con mucho néctar).
- Una solución dentro del modelo *ERS-ABC* es un recurso eficiente.
- Las abejas obreras explotan los q recursos más eficientes. El modelo agrupa estos recursos en un conjunto conocido como *conjunto de obreras* S_E.
- Para cada recurso de tipo obrero el modelo estima la probabilidad de que vuelva a ser explotado por la colmena. Esta probabilidad se basa en el valor de eficiencia de los recursos.
- Las abejas observadoras dependen de la información recogida por las obreras (su experiencia). Al igual que ocurre con las obreras, el modelo gestiona un *conjunto de observadoras* S_O.
- Cada abeja exploradora escoge un recurso de manera aleatoria para explotarlo.

A continuación se describe el flujo de ejecución del modelo. Cuando un recurso r_α procesa una tarea, el modelo determina qué tipo de abeja es la que ha explotado esta fuente (recurso). Si el recurso pertenece al conjunto de obreras se pasa a valorar la calidad de mismo, es decir, cómo de eficiente ha sido. Si el valor de calidad obtenido es mejor que el de otras fuentes memorizadas por las obreras (mejor que otros valores de eficiencia registrados en su conjunto de recursos) r_α permanece en el *conjunto de obreras*. En otro caso, el modelo busca un recurso eficiente que no sea parte de S_E y descarta a r_α (*mutación*).

Respecto al *conjunto de observadoras* S_O, al principio de la ejecución se trata de una réplica de S_E. Durante la ejecución de la aplicación es actualizado cada vez que se modifica el conjunto de obreras. El método de actualización utiliza la probabilidad de explotación como se aprecia en la Figura 1. Se hace uso, además, de S_E en la actualización debido a que al inicio ambos conjuntos son iguales. También se aplica un proceso de mutación para obtener los nuevos recursos de las observadoras: un recurso de S_O es reemplazado por su vecino más cercano que sea más eficiente que él dentro del espacio R. Cabe destacar que R es ordenado de mayor a menor valor de eficiencia durante la ejecución de la aplicación, de manera que los recursos más eficientes de un recurso r_α están colocados a su izquierda.

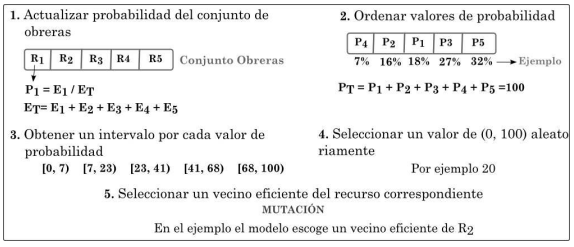


Fig. 1. Método de actualización del conjunto de observadoras basado en la probabilidad de explotación. Los puntos 4 y 5 se repiten por cada recurso de S_O.

Por último, en lo que se refiere a las abejas exploradoras, cada vez que un porcentaje w de J es procesado una exploradora entra en juego en el modelo. Esta abeja escoge un recurso de R de manera aleatoria procurando, mientras sea posible, que no haya sido explotado anteriormente por la colmena.

Los recursos gestionados por todas las abejas de la colmena (recursos de S_E más S_O más exploradora) forman un conjunto de candidatos a ser solución. La nueva solución (el siguiente recurso que procesará una tarea) es escogido aplicando la técnica *Round Robin* sobre los candidatos.

En la Figura 2 se resumen las acciones principales del flujo de ejecución de *ERS-ABC* comentadas. En la fase de inicialización (puntos 1-4) se crean los dos espacios de trabajo y el conjunto T es lanzado a ejecución. A continuación, el modelo monitoriza las tareas correspondientes. Cuando una tarea t_α termina su ejecución el valor de eficiencia del recurso asociado r_α es actualizado. Tras esto, el proceso de selección basado en *ABC* es aplicado y un nuevo recurso eficiente es escogido para procesar una nueva tarea. Estos pasos se repiten hasta que todo J es procesado.

PSEUDOCODIGO: ALGORITMO ERS-ABC

Input: tareas, recursos infraestructura
Output: conjunto solución

1. Determinar espacios J y R;
2. Preparar conjunto T;
3. Escoger conjunto RT aleatoriamente para T;
4. Lanzar T en ejecución;
5. mientras haya tareas sin procesar **hacer**
  5.1. Monitorizar tareas;
  5.2. Si una tarea acaba **entonces**
    5.2.1. Actualizar eficiencia del recurso;
    5.2.2. Aplicar el proceso de selección ABC;
    5.2.3. Lanzar una nueva tarea;
6. Fin mientras

Fig. 2. Pseudocódigo de *ERS-ABC* donde los principales pasos son indicados.

V. Evaluación del Modelo

Como se comentó en la Sección I, en la fase de evaluación los experimentos son llevados a cabo en una infraestructura grid real europea que forma parte de la Iniciativa Grid Nacional Española (*ES-NGI*)[4]. En concreto, estamos afiliados a una de las *VO* del proyecto Ibergrid[5] que contiene unos 30 *CEs* (un número de recursos razonable para evaluar el modelo).

Dos escenarios han sido definidos para determinar si se han alcanzado los objetivos establecidos: una reducción del tiempo de ejecución de la aplicación y un incremento en el número de tareas finalizadas con éxito. En ambos escenarios *ERS-ABC* es comparado con la técnica de selección estándar de las infraestructuras grid europeas y que nosotros hemos denotado como *TRS* (*Traditional Resources Selection*). Esta técnica de selección se basa en criterios de proximidad y disponibilidad para elegir los *CEs*. El método que realiza esta selección es conocido en el ámbito grid como *match-making*.

[4]http://www.es-ngi.es/
[5]http://www.ibergrid.eu/

A. Escenario 1

En este primer escenario se pretende determinar la influencia del tamaño del subconjunto T en el aprendizaje del modelo. Por este motivo se definen 5 pruebas distintas dentro del mismo con las siguientes características: el tamaño de J se fija a 200 tareas mientras que vamos variando el tamaño de T en cada una de las pruebas desde 5 hasta 40 (5, 10, 13, 20, 40). Los experimentos de cada prueba son ejecutados 10 veces en la infraestructura para cada versión (*ERS-ABC*, *TRS*). Por tanto, cada punto representado en la gráfica de la Figura 3 es el valor medio de los experimentos realizados. Con esta variación en T se espera que a medida que aumente de tamaño el aprendizaje del modelo sea más rápido, mejorando los tiempos de ejecución.

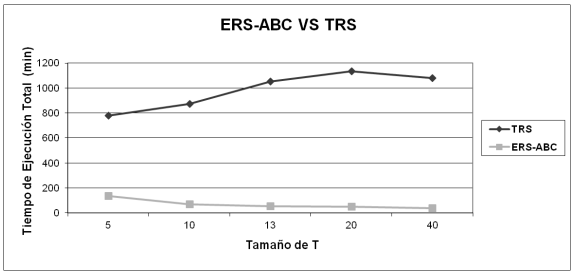


Fig. 3. Resultados obtenidos en el primer escenario de pruebas. Nuestra propuesta obtiene mejores resultados.

De los resultados obtenidos podemos concluir que *ERS-ABC* consigue mejores valores de tiempo de ejecución para la aplicación respecto a *TRS*. Además, a medida que T aumenta de tamaño la diferencia entre ambas versiones crece, como se esperaba. Luego el tamaño de este subconjunto de tareas influye en el aprendizaje del modelo. Cuando enviamos un mayor número de abejas a explotar las fuentes de alimento (recursos), como ocurre en los últimos tests, se consigue un conocimiento más profundo de la infraestructura en un menor tiempo.

Por otra parte, es importante remarcar que el tiempo de ejecución total mostrado para *ERS-ABC* incluye no solo el tiempo de ejecución de la aplicación sino también el tiempo que utiliza el modelo para monitorizar, clasificar y seleccionar los recursos eficientemente.

Sin embargo, vemos que *TRS* tiene un comportamiento diferente a nuestra propuesta. En los primeros tests (tamaños de 5 y 10) obtiene sus valores de tiempo de ejecución más bajos. A partir de entonces comienza a ser cada vez más lento en la ejecución de la aplicación. Al ser una técnica basada en criterios de cercanía y proximidad, probablemente en las últimas pruebas son muy pocos los recursos disponibles (la mayoría estarán siendo usados para ejecutar las tareas) y algunos de ellos pueden estar incluso sobrecargados.

B. Escenario 2

El objetivo de este segundo escenario es determinar el rango de aplicaciones en las que se puede

aplicar nuestro modelo de manera satisfactoria. Las aplicaciones que suelen ejecutarse en infraestructuras grid están formadas por un número de tareas considerable. Por este motivo, en los diferentes tests que se definen en este escenario variamos el tamaño de J desde 50 hasta 500 (50, 100, 200, 300, 400, 500). En total se tienen 6 tests donde el tamaño de T se ha fijado a 10. Al igual que en el primer escenario, se realizan 10 ejecuciones de cada experimento por cada tamaño de J.

Como se muestra en la Figura 4, *ERS-ABC* nuevamente obtiene mejores resultados, consiguiendo mejores tiempos a medida que la cantidad de tareas totales aumenta. Por el contrario, *TRS* consigue peores tiempos de ejecución para aplicaciones de mayor número de tareas a ejecutar.

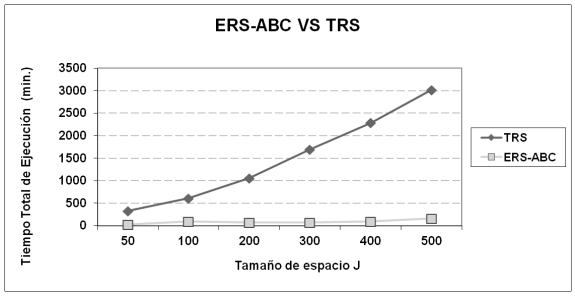


Fig. 4. Segundo escenario de pruebas del modelo *ERS-ABC*. Nuevamente nuestro modelo supera a la técnica de selección de recursos estándar en grid.

En cuanto al segundo objetivo fijado, incrementar la tasa de tareas acabadas con éxito, *ERS-ABC* consigue mejorar los resultados de *TRS*. Finalmente, se puede afirmar que el modelo es una solución factible para las aplicaciones grid.

VI. Conclusiones

El presente trabajo propone un modelo de selección eficiente que mejore dicho proceso en los entornos grid. Para ello, el modelo ha sido diseñado desde el punto de vista del usuario y haciendo uso del algoritmo *ABC*. De esta manera se dota de una capacidad auto-adaptativa a las aplicaciones, permitiéndolas enfrentarse a los cambios del entorno.

El modelo es evaluado en una infraestructura grid real perteneciente a una iniciativa europea. Es más, es comparado con la selección estándar de las infraestructuras europeas para verificar que se alcanzan los hitos establecidos. De los resultados obtenidos se deduce que se trata de una solución factible y beneficiosa para las aplicaciones grid, ya que disminuye su tiempo de ejecución e incrementa su tasa de tareas finalizadas con éxito.

Como trabajo futuro se espera enriquecer el modelo con otras funcionalidades, probar nuevos algoritmos que se adapten mejor al problema y considerar otros servicios grid.

Agradecimientos

La investigación de María Botón-Fernández está subvencionada mediante una beca de Formación de Personal Investigador otorgada por el Ministerio de Economía y Competitividad a través del Centro de Investigaciones Energéticas, Medio Ambientales y Tecnológicas (CIEMAT). Los autores quieren agradecer también el soporte de los Fondos Europeos para el Desarrollo Regional (FEDER).

Referencias

[1] I. Foster, *What is the Grid? A three Point Checklist*, GRIDtoday, Vol. 1, No. 6, pp. 22-25, 2002.

[2] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the GRID. Enabling Scalable Virtual Organizations*, Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L.(eds.) Euro-Par 2001, LNCS, Vol. 2150, Springer-Verlag, Heidelberg, pp. 1-4, 2001.

[3] D. Karaboga, *An Idea based on Honey Bee Swarm for Numerical Optimization*, Technical Report-tr06, Erciyes University. Turkey, 2005.

[4] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov *Adaptive Computing on the Grid Using AppLeS*, IEEE Transactions on Parallel and Distributed Systems, Vol. 14, Issue 4, pp. 369-382, 2003.

[5] E. Huedo, R.S. Montero, and I.M. Llorente, *A Framework for Adaptive Execution in Grids*, Software-Practice & Experience, Vol. 34(7), pp. 631-651, 2004.

[6] H.N.L.C. Keung, J.R.D. Dyson, S.A. Jarvis, and G.R. Nudd, *Self- Adaptive and Self-Optimising Resource Monitoring for Dynamic Grid Environments*, DEXA 04 Proceedings of the Database and Expert Systems Applications, 15th International Workshop, Washington DC, USA, pp. 689-693, 2004.

[7] S.S. Vadhiyar, and J.J. Dongarra, *Self Adaptivity in Grid Computing*, Concurrency and Computation: Practice and Experience, Vol. 17(2-4), pp. 235-257, 2005.

[8] D. M. Batista, and L.S. Da Fonseca, *A Survey of Self-adaptive Grids*, IEEE Communications Magazine, Vol. 48, Issue 7, IEEE Press Piscataway, NJ, USA, pp. 94-100, 2010.

Comparison of Auto-scaling Techniques for Cloud Environments

Tania Lorido-Botran[1] Jose Miguel-Alonso[2] Jose A. Lozano[3]

Abstract— **Cloud computing environments offer the user the capability of running their applications in an elastic manner, using only the resources they need, and paying for what they use. However, to take advantage of this flexibility, it is advisable to use an auto-scaling technique that adjusts the resources to the incoming workload, both reducing the overall cost and complying with the Service Level Objective. In this work we present a comparison of some auto-scaling techniques (both reactive and proactive) proposed in the literature, plus two new approaches based on rules with dynamic thresholds. Results show that dynamic thresholds avoid the bad performance derived from a bad threshold selection.**

Keywords— **Cloud computing, scalable applications, auto-scaling, service level objectives**

I. Introduction

CLOUD computing is a popular technology, characterized by its elastic nature. This means that users only acquire the resources they need (on-demand) and pay only for what they use (a pay-as-you-go scheme). Resources are provided in the form of VMs. There are two types of scaling: horizontal and vertical. Horizontal scaling (or scaling out/in) consists of adding or removing VMs, and vertical scaling (scaling up/down) consists of dimensioning the resources assigned to a particular VM (e.g. CPU or memory). Vertical scaling is not supported for all operating systems; for this reason, many cloud providers only offer horizontal scaling.

Adapting resources to application needs is a challenge, due to the constant changes in the input workload. Manual scaling would require constant supervision and, probably, will cause a bad performance in the application. Instead, an auto-scaling technique is a suitable way to automate resource adaptation. Cloud providers usually offer a rule-based auto-scaling mechanism, which is simple to set up but difficult to tune. The main drawback is its reactive nature: it is unable to cope with sudden workload bursts and also with the boot up time of VMs (up to 10 minutes). A proactive approach tries to overcome these problems, forecasting future resource needs.

Authors in the literature have proposed many reactive and proactive auto-scaling techniques, related to control theory, time series analysis, reinforcement learning and so on. Due to the heterogeneity of the techniques and the testing environment they use, it is impossible to carry out a proper comparison to decide the best auto-scaling technique for a particular scenario. For this reason, we have developed a cloud simulator based on CloudSim [1], specifically designed as a workbench for studying auto-scaling techniques. In this paper we use it to test several representative techniques, analyze the impact of parameter definition and compare techniques in terms of VM cost and SLO violations. Note that this is a preliminary, not exhaustive study.

Cloud providers usually bill on an hourly basis and partial hours are accounted as full hours. In this case, VM cost is accounted as the number of VM running hours, multiplied by the hour fee. The Service Level Agreement (SLA) is the contract between the cloud provider and the customer, in which the Quality of Service (QoS) is defined. The SLA is defined in terms of one or more Service Level Objectives (SLO). In this paper, a single SLO is considered, based on the service time required by incoming requests.

The rest of the paper is organized as follows. The target scenario is described in Section II. Then, each of the auto-scaling techniques is described and tested on the cloud simulator (Section III). Section IV presents some performance results and makes a comparison of the different auto-scaling techniques. Finally, the conclusions extracted, along with some future work lines, are presented in Section V.

II. Scenario

Our target scenario is a web-type application, deployed over a pool of homogeneous VMs. We will focus on the load balancer and the business tier. User requests arrive the application following a given pattern (in this case, based on a real workload). The load balancer will distribute requests among the VMs, based on different policies: random, round-robin or least-connection. Round-robin policy distributes requests in turns, whilst the least-connection one consists of assigning requests to the least loaded VM, i.e., the VM that is executing the least number of tasks.

Each task is assigned to a single VM. Request execution time is denoted as its *expected service time*, and in our simulated environment it is known *a-priori*. VMs adopt a time-sharing policy, so all incoming tasks will be accepted, but service time will increase accordingly to the number of tasks in the CPU. In this context, the *service time* is the lapse since a task is assigned to a VM, until the response is received back.

The auto-scaler will perform horizontal scaling actions: adding (scaling out) or removing VMs (scaling in). Many performance metrics could be applied to

[1]Dep. of Computer Architecture and Technology, U. of the Basque Country, UPV/EHU, e-mail: tania.lorido@ehu.es
[2]Dep. of Computer Architecture and Technology, U. of the Basque Country, UPV/EHU, e-mail: j.miguel@ehu.es
[3]Dep. of Computer Science and Artificial Intelligence, U. of the Basque Country, UPV/EHU, e-mail: ja.lozano@ehu.es

trigger auto-scaling actions. The most typical ones in the literature are CPU load, service time and input request rate, but many others have been proposed such as memory used by the VMs, or available bandwidth. Scaling decision may be taken based on a single metric or a combination of two or more. For simplicity, this study focuses on the use of a single metric.

The performance of each technique may be evaluated in terms of the total VM cost and the total number (or ratio) of SLO violations. Each cloud provider uses its own billing scheme. VM hours can be charged per minute or per hour (and in that case, partial hours are accounted as full hours). The cost may include not only VM running hours, but also other resource usage (e.g. disk or bandwith) and services (e.g. monitoring system).

Figure 1 shows an example of a real workload of one week duration. The number of VMs (represented with a thin line) varies according to the system load (number of requests per minute), from 10-15 VMs, up to 50 VMs at workload peaks.

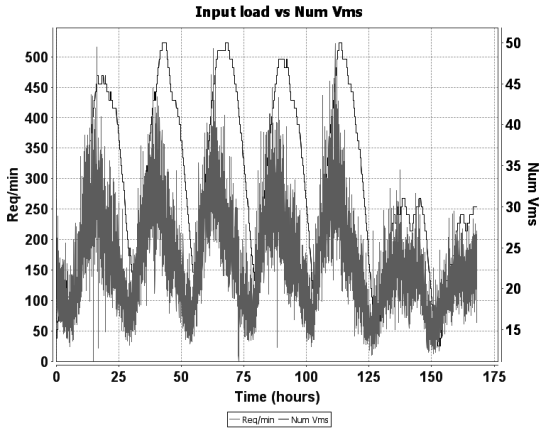


Fig. 1. ClarkNet trace (the workload) vs. number of VMs assigned

III. Description of the Auto-scaling Techniques

Many diverse auto-scaling techniques have been proposed in the literature. In [3], a taxonomy consisting of five categories was defined: static threshold-based rules, time series analysis, control theory, reinforcement learning and queuing theory. In the current study we compare techniques from the first three categories.

Static threshold-based rules are typically used by cloud providers such as Amazon EC2. A simple example: if $CPU > 70\%$, then scale out; if $CPU < 30\%$, then scale in. It is quite difficult to set the correct thresholds and this must be done manually. An incorrect adjustment will cause oscillations in the number of VMs, and therefore, lead to bad performance. After each scaling decision, a cooldown period can be applied, during which no scaling will be performed. This cooldown period reduces the oscillations in the number of VMs and can be applied not only to rules, but to any auto-scaling technique. In RightScale's [4] variation of rules, each VM votes independently, based on rules like those explained before, whether to scale or not. Then, a simple democratic voting is performed to decide the scaling action.

Time series analysis includes a number of methods that use a past history window of a given performance metric in order to predict its future values (*proactive techniques*). In this case, we consider three methods: moving average, exponential smoothing and linear regression. Moving average calculates the mean of the n last values. Exponential smoothing assigns exponentially decreasing weight to each value in the time series. Last, linear regression tries to fit a linear equation to the last values (where x is the time and y is the performance metric value), and then, it estimates a future value.

Control theory has been applied to automate the management of different systems. The typical controller is the Proportional Integral Derivative (PID) controller. In this study, we have implement a simplified version called I (Integral) controller that obeys to the following equation:

$$u_k = u_{k-1} + K_i(y_k - y_{ref}) \quad (1)$$

where u_k is the new value for the manipulated variable (new number of VM); y_k is the performance metric value (CPU load), y_{ref} is the set point or target value; and K_i is the integral gain parameter. In other words, the integral controller will adjust the number of VMs in order to maintain the performance metric value (e.g. CPU load) as closer as possible to the target value.

Lastly, we have proposed a new auto-scaling technique, based on rules, that tries to overcome the limitation of using static thresholds. In this case, the upper and lower thresholds will be adjusted based on another set of rules. First, if SLO violations occurs for a certain time (e.g. 5 minutes), the threshold range will be *widened*. For example, a 60-40% configuration for upper-lower thresholds may be adapted to 80-20%. This makes the system less reactive to workload changes, the auto-scaler will be less eager to remove VMs, and thus, the number of SLO violations will be reduced. The opposite case is when no SLO violations have taken place during the last period of time, so the threshold range is *narrowed*. Then, the system becomes more reactive. The modification to use dynamic thresholds has also been applied to RightScale's voting system.

The dynamic threshold-based rules constitute a novel contribution of this work. All the remaining auto-scaling techniques discussed in this paper, and many additional ones, are described in detail in [3].

IV. Experiments

This section presents the results for the different auto-scaling techniques and analyzes them. We have extended the functionality of the CloudSim cloud simulator to carry out the experiments. As described

TABLE I

PERFORMANCE VALUES OF LOAD BALANCING POLICIES WITH STATIC THRESHOLD-BASED RULES

Rules		R-Robin		Random		Least-con.	
Upper Thres.	Down Thres.	Cost	SLOv	Cost	SLOv	Cost	SLOv
60	20	5756.70	0.15	5967.40	4.53	5727.40	0.13
70	30	4677.00	0.86	4799.50	10.36	4625.90	0.81
80	30	4330.10	1.46	4335.30	15.50	4294.40	1.39
80	40	4037.20	3.24	4053.40	21.21	4004.70	3.15
90	30	4026.70	2.66	3907.00	23.71	4002.50	2.35
90	10	5172.80	0.49	4855.80	15.46	5151.50	0.34

TABLE II

RESULTS FOR REACTIVE TECHNIQUES, BASED ON CPU LOAD, FOR TWO DIFFERENT VALUES OF BOOT UP TIME

Technique	Parameters			0 min		10 min	
				VMcost	SLOv	VMcost	SLOv
Rules	UT: 60		DT: 20	5756.70	0.15	5575.80	0.33
	UT: 70		DT: 30	4677.00	0.86	4402.60	2.91
	UT: 80		DT: 30	4330.10	1.46	4155.80	4.92
	UT: 80		DT: 40	4037.20	3.24	3764.10	8.89
	UT: 90		DT: 30	4026.70	2.66	3914.90	7.13
	UT: 90		DT: 10	5172.80	0.49	5136.20	0.79
Dynamic thresholds	UT: 60		DT: 20	5175.50	0.50	5127.60	0.87
	UT: 70		DT: 30	5169.50	0.50	5127.50	0.87
	UT: 80		DT: 30	5168.90	0.51	5127.50	0.87
	UT: 80		DT: 40	5168.30	0.51	5126.80	0.90
	UT: 90		DT: 30	5168.70	0.51	5127.40	0.88
	UT: 90		DT: 10	5173.50	0.48	5136.30	0.79
RighstScale	UT: 60	DT: 20	Votes 75%	5813.70	0.14	5741.20	0.23
	UT: 60	DT: 20	Votes 95%	5712.00	0.11	5665.70	0.19
	UT: 70	DT: 30	Votes 75%	4656.90	0.78	4467.00	2.63
	UT: 70	DT: 30	Votes 95%	4631.10	0.84	4523.00	2.27
	UT: 80	DT: 40	Votes 75%	4001.70	3.12	3771.00	8.54
	UT: 80	DT: 40	Votes 95%	3940.20	3.14	3777.80	8.76
	UT: 90	DT: 10	Votes 75%	5134.60	0.61	5089.40	1.10
	UT: 90	DT: 10	Votes 95%	5090.40	0.87	5092.50	1.34
Dynamic thr. RightScale	UT: 60	DT: 20	Votes 75%	5138.30	0.58	5094.40	1.04
	UT: 60	DT: 20	Votes 95%	5095.00	0.78	5105.10	1.22
	UT: 70	DT: 30	Votes 75%	5136.50	0.58	5094.40	1.04
	UT: 70	DT: 30	Votes 95%	5087.90	0.81	5105.10	1.22
	UT: 80	DT: 40	Votes 75%	5114.90	0.60	5092.90	1.05
	UT: 80	DT: 40	Votes 95%	5086.00	0.82	5105.10	1.22
	UT: 90	DT: 10	Votes 75%	5131.50	0.59	5079.40	1.05
	UT: 90	DT: 10	Votes 95%	5085.30	0.83	5105.10	1.22
IController	K: -0.01		Target: 60%	6474.60	0.04	6538.50	0.29
	K: -0.01		Target: 65%	5658.00	0.16	5492.60	0.71
	K: -0.01		Target: 70%	5089.20	0.44	4906.00	1.81
	K: -0.01		Target: 75%	4602.30	1.17	4467.20	4.07
	K: -0.01		Target: 80%	4207.70	2.40	4098.40	7.55

in Section II, a web-type application is simulated, which is composed of a business tier (15 initial VMs), and a load balancer. All VMs are homogeneous: one core of 1GHz; other resources such as memory, disk or bandwith have not been considered.

Request execution time is generated based on a uniform distribution, that ranges from 3 to 7 seconds. The arrival time is taken from a real workload trace, the ClarkNet trace [5]. It contains 1654276 requests, that arrive in a clear cyclic pattern (see Figure 1): daytime has more workload than the night, and the workload on weekends is lower than that taking place on weekdays.

CPU load (mean load of all VMs) is used as the performance metric. The monitoring interval of one minute and scaling decisions are taken based on the performance metric value from the last minute. Scaling actions add or remove a single VM (other options could be considered, such as adding/removing a percentage of the current number of running VMs). Two different values of boot-up time (time to effectively put to work a new VM) are considered: 0 and 10 minutes. After a scaling decision, a cooldown period is applied. This cooldown period is equal to the boot-up time, plus an extra period of 5 minutes (that is necessary to see the effect of an scaling decision on

the system). When a scale-in decision is made, the chosen VM is not removed until all tasks that are being serviced on it finish. Therefore, in a subsequent scale-out action, instead of creating a new VM, we can use one of these VMs that are pending to be removed and avoid the boot-up time.

Each experiment is repeated 10 times. Auto-scaling techniques are compared attending to the mean VM cost (*VMcost*) and the mean number of SLO violations (*SLOv*). In this scenario, the Amazon EC2 [2] billing scheme is applied: VM hours are charged per running hour, and partial hours are accounted as full hours. Boot up time is not charged. An SLO violation occurs when the service time of a task is greater than or equal to $5 * expectedtime$.

Results are divided into several subsections. Focusing on the rules, we analyze the impact of applying different load balancing policies, and also compare static vs. dynamic thresholds. Then, several reactive and proactive techniques are compared.

A. Impact of load balancing policy

Three load balancing policies have been tested: round-robin, random and least-connection. For the comparison, we will focus on the results for static threshold-based rules, and instant boot up of VMs. Results are gathered in Table I.

Random policy shows the worst results, up to 23.71% of SLO violations. Its nature may cause an unbalanced distribution of tasks among the VMs. Round-robin and least-connection policies both show similar results, but the latter performs a little better. The performance of these two policies will depend on the homogeneity in the duration of the tasks. If all tasks have the same execution time, round robin will be the best option. Otherwise, if tasks have different execution times, least-connection policy is able to distribute the load more evenly among the VMs. For the rest of the experiments, we will use round robin as the load balancing policy.

B. Reactive techniques

Reactive techniques considered in this paper include rules, RightScale's voting system, two proposals based on dynamic thresholds and an integral controller. Results for two different values of boot-up time (0 and 10 minutes) are shown in Table II.

B.1 Static vs dynamic threshold-based rules

In this section we compare typical rules based on static thresholds (*Rules*), also combined with a voting system (*RightScale*), against our proposal of adapting thresholds depending on the number of SLO violations. Results are shown for the round-robin load balancer policy, 0 minute boot up time and CPU load as the performance metric (see Table II).

Dynamic thresholds consider 50% as the mid range, 90% as the upper limit and 10% as the lower limit. These limits have been established in order to avoid a 100-0 and/or 95-5% threshold-configurations, that lead to a situation where few scaling actions are performed and the number of VM remains mainly constant. Variations of thresholds are of 5%. Cooldown time is applied for scaling rules, but not for threshold adaptation (this is checked every minute).

Regarding both simple rules and RightScale's voting system based on static thresholds, the performance depends highly on the selection of threshold values. Some configurations result in low VM cost, but in consequence, the number of SLO violations is excessive. Despite the fact that the lowest number of SLO violations is obtained using certain values of static thresholds, our proposal based on dynamic values is able to maintain an acceptable SLO compliance (0.50% for simple rules and 0.58-0.83% for the RightScale's approach), regardless the initial threshold values. Combining dynamic thresholds with simple rules seems to be the best option (rather than using a voting system), since the number of SLO violations remains nearly constant.

B.2 Integral controller

The integral controller is different from the rest of reactive auto-scaling techniques, because it considers a target CPU value, instead of trying to keep CPU within an acceptable range (delimited by upper and lower thresholds). It directly calculates the required number of VMs (this value is rounded up). K value has been selected using a trial and error method. This parameter is really difficult to set, and a bad choice causes high oscillations in the number of VMs. Although it is the technique that achieves the lowest number of SLO violations (0.04%), it does so at a high cost and, therefore, we can not consider integral controller as the best choice.

As a general conclusion, reactive techniques cannot anticipate to changes and a boot-up time of 10 minutes causes an increase in both the VM cost and the number of SLO violations. The performance of each auto-scaling technique highly depends on parameter selection, which is difficult to be done manually. Besides, a bad configuration causes a high number of SLO violations and dissatisfied clients. Our proposal based on dynamic thresholds is able to improve the general performance, by adjusting the thresholds automatically, without human intervention.

C. Proactive techniques

Three proactive methods (based on time series analysis) are considered: moving average (MA), linear regression (LR) and exponential smoothing (ES). They predict the mean CPU load for the next period (1 minute ahead) and this value is used with threshold-based rules to decide the next scaling action. Among the threshold configurations for the scaling rules, we have selected the three that obtained the lowest number of SLO violations in the reactive case: 60-20%, 70-30% and 90-10%. Table

TABLE III
RESULTS FOR PROACTIVE TECHNIQUES, BASED ON CPU LOAD, FOR TWO DIFFERENT VALUES OF BOOT UP TIME

T.	UT LT	Par.	0 min		10 min	
			VMcost	SLOv	VMcost	SLOv
MA	60 20	W:2	5699.70	0.15	5638.80	0.24
		W:3	5659.80	0.16	5600.90	0.31
		W:5	5572.30	0.18	5551.20	0.36
		W:10	5479.00	0.23	5393.70	0.41
	70 30	W:2	4585.80	0.78	4414.00	2.59
		W:3	4535.70	0.90	4423.20	2.73
		W:5	4539.30	1.07	4502.20	2.27
		W:10	4505.20	1.43	4469.80	2.62
	90 10	W:2	5063.00	0.88	5090.90	1.27
		W:3	5008.70	1.25	5066.20	1.57
		W:5	5079.30	1.57	5050.70	2.13
		W:10	4945.20	2.37	5003.40	3.41
ES	60 20	α:.1	5364.50	0.30	5371.20	0.45
		α:.3	5544.50	0.18	5548.20	0.33
		α:.6	5751.00	0.13	5715.10	0.23
		α:.9	5735.70	0.15	5603.00	0.29
	70 30	α:.1	4453.30	1.85	4428.30	3.14
		α:.3	4565.70	1.00	4514.40	2.26
		α:.6	4596.20	0.71	4474.00	2.22
		α:.9	4662.70	0.78	4436.80	2.72
	90 10	α:.1	4801.40	3.74	4844.60	4.65
		α:.3	5092.10	1.68	5144.60	2.32
		α:.6	4997.70	1.02	5039.30	1.40
		α:.9	5099.00	0.58	5090.80	0.90
LR	60 20	W:2	5602.30	0.71	4993.90	2.23
		W:3	5594.70	0.45	5057.50	1.76
		W:5	5604.70	0.26	5345.70	0.53
		W:10	5605.60	0.15	5471.60	0.26
	70 30	W:2	4666.90	2.69	4144.70	9.70
		W:3	4691.00	1.88	4183.30	8.06
		W:5	4695.50	1.20	4261.50	5.87
		W:10	4525.30	0.97	4380.30	2.91
	90 10	W:2	4734.60	1.42	4293.70	6.23
		W:3	4836.50	1.09	4581.60	3.52
		W:5	4901.80	0.73	4794.60	1.64
		W:10	4994.70	0.98	4944.00	1.51

III contains the results for two boot-up times: 0 and 10 minutes.

Again, results highly depend on parameter configuration, history window and smoothing factor. In case of MA, the best result is obtained for a history window W equal to 2; higher values of this parameter lead to an increase in the number of SLO violations. On the contrary, for LR case, it is better to use a larger history window ($W = 10$). Last, ES is applied with different values of the smoothing factor α. The lowest number of SLO violations is obtained with $\alpha = 0.6$ (in two out of three cases), which balances the weight given to new values (0.6), and the weight assigned to the past values.

MA and ES techniques are less dependent on the parameter values than LR. The latter can vary from 9.70% of SLO violations with a $W = 2$ to 2.91% with $W = 10$ (with 70-30% thresholds and 10 minutes of boot-up time).

Looking at the number of SLO violations for both instant and 10-minute boot-up times, the lowest results are obtained by ES with $\alpha = 6$ (0.13/0.23%), nearly followed by MA with $W = 2$ (0.15/0.24%) and last, LR with $W = 10$ (0.15/0.26%). All three techniques improve the results of static threshold-based rules (0.15/0.33% for 60-20% threshold values).

D. Comparing Reactive vs. Proactive techniques

Figure 2 shows a global perspective of auto-scaling techniques. Regarding proactive techniques, only results for 60-20% threshold configuration are presented, as they obtained the lowest number of SLO violations.

In general, the performance worsens when the boot-up time of VMs is 10 minutes in terms of SLO violations. In contrast, the overall cost is quite similar for the two values of boot-up time. In almost all cases, for both reactive and proactive approaches, the number of SLO violations highly depends on the parameter configuration. The lowest number of SLO violations is achieved by the integral controller and RightScale's voting system, but both require a fine tuning of two or more parameters. However, proactive techniques (MA and ES), combined with 60-20% threshold-based rules, are less dependent on parameters, and still improve the performance of simple rules. Finally, proactive techniques have been tested with rules for simplicity, but they may be combined with any method such as an integral controller or RightScale's voting system.

V. CONCLUSIONS

Cloud computing environments have a great potential for running scalable applications, thanks to their elasticity nature. Auto-scalers are the key to the efficient use of elastic resources, because they enable the user to adjust resources to needs, while complying with SLO and reducing the cost. Using simulation, we have tested some representative auto-scaling techniques, comparing them in terms of overall cost and SLO violations. We have also proposed a method based on rules with dynamic thresholds, that improves the performance of simple, static threshold-based rules. The main conclusion extracted is that any auto-scaling method is very dependent on the parameter tuning.

As future work, we plan to improve our initial idea of using dynamic thresholds. The main reason is that nowadays, cloud providers that offer auto-scaling capabilities are based on rules. But we also want to explore the potential of other proactive techniques, such as ARMA models or reinforcement learning. ARMA techniques are able to model complex time series such as input workload to an application, in order to perform more accurate forecasts. Reinforcement learning is a different approach that learns based on experience the best scaling action to take at a given system state.

ACKNOWLEDGEMENTS

This work has been partially supported by the Saiotek and Research Groups 2013-2018 (IT-609-13) programs (Basque Government), TIN2010-

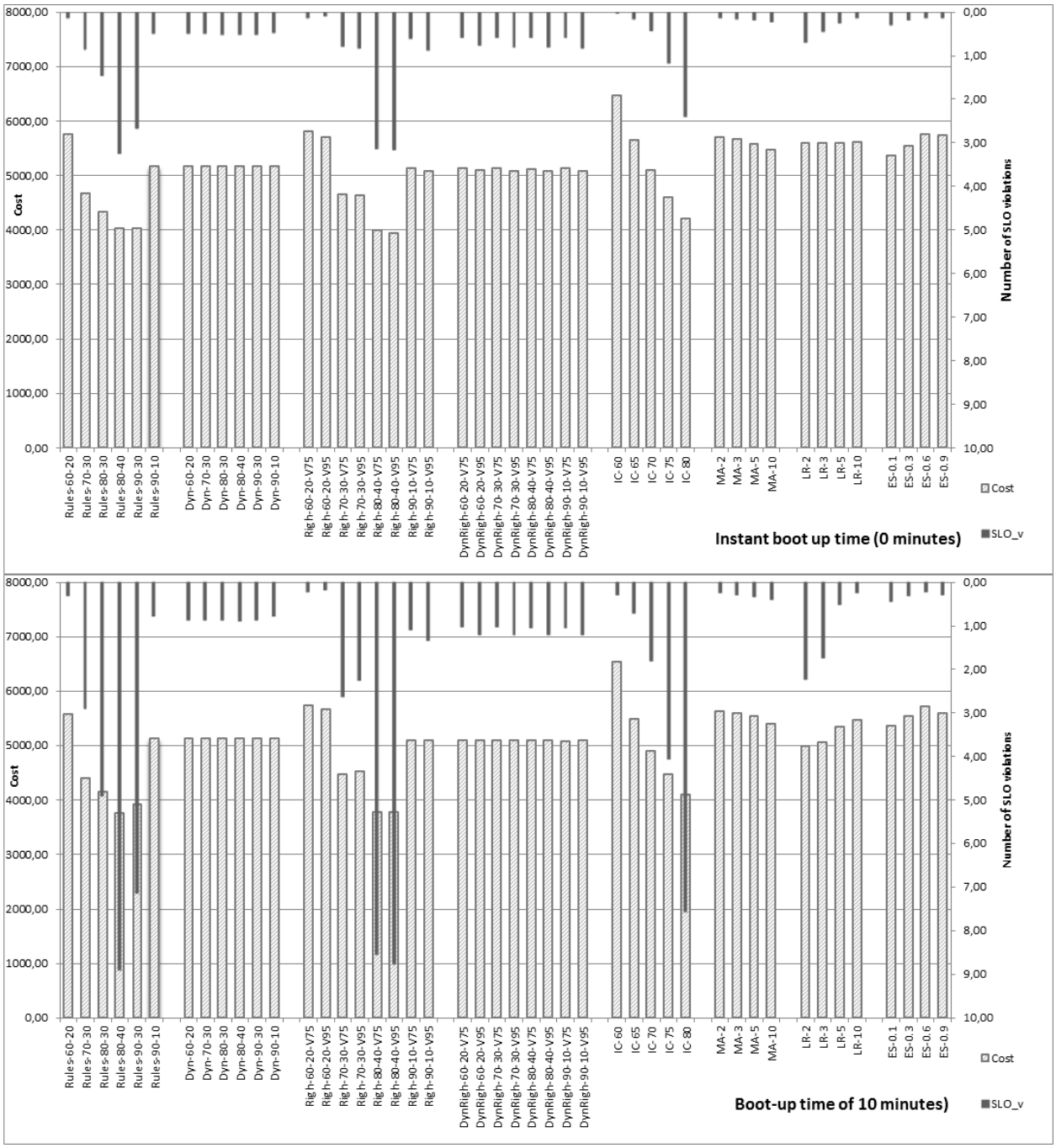


Fig. 2. VM cost and number of SLO violations for auto-scaling techniques, with instant boot-up time (top) and 10 minutes of boot-up time (bottom).

14931 (Ministry of Science and Technology), COMBIOMED network in computational biomedicine (Carlos III Health Institute), and by the NICaiA Project PIRSES-GA-2009-247619 (European Commission). Dr. Miguel-Alonso is member of the HiPEAC European Network of Excellence. Mrs Lorido-Botrán is supported by a doctoral grant from the Basque Government.

References

[1] "CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services.," `http://www.cloudbus.org/cloudsim/`, 2012, [Online; accessed 18-September-2012].

[2] "Amazon Elastic Compute Cloud (Amazon EC2)," `http://http://aws.amazon.com/ec2/`, 2012, [Online; accessed 15-May-2013].

[3] T Lorido-Botrán, J Miguel-Alonso, and JA Lozano, "Auto-scaling Techniques for Elastic Applications in Cloud Environments," Tech. Rep., University of the Basque Country UPV/EHU; EHU-KAT-IK-09-12, 2012.

[4] "RightScale Cloud Management," `http://www.rightscale.com/`, 2012, [Online; accessed 15-May-2013].

[5] "ClarkNet HTTP Trace (From the Internet Traffic Archive)," `http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html`, 2012, [Online; accessed 15-May-2013].

Un planificador de GPUs remotas para clusters HPC

Sergio Iserte[1], Adrián Castelló[1], Carlos Reaño[1], Antonio J. Peña[2], Federico Silla[3], Rafael Mayo[1], Enrique S. Quintana-Ortí[1] y José Duato[3]

Resumen— **SLURM es un planificador de recursos que permite gestionar un cluster heterogéneo, compartiendo algunos de los recursos del cluster entre los procesos que los solicitan para su ejecución. Sin embargo SLURM no está capacitado para compartir ciertos recursos genéricos, como las GPUs, entre los nodos, como lo hace con las CPUs, ya que el planificador gestiona el uso de GPUs pero sólo pueden ser utilizadas por el nodo en el que se encuentran físicamente instaladas. Esta característica de SLURM se convierte en una limitación cuando se emplean soluciones de virtualización de GPUs como rCUDA, cuyo propósito es proporcionar acceso transparente a las GPUs de un cluster aunque éstas estén instaladas en otro nodo. Para hacer compatible la planificación de SLURM con el funcionamiento de rCUDA, se ha creado un nuevo recurso compartido, la rgpu y se ha añadido al código de SLURM la lógica necesaria para que acepte este nuevo recurso y sea capaz de tratarlo como los recursos no genéricos.**

Palabras clave— **SLURM, rCUDA, GRes, CUDA, planificador de recursos, cluster, GPU**

I. Introducción

En los últimos años, la utilización de los aceleradores de cálculo en las instalaciones dedicadas a computación de alto rendimiento se ha consolidado como una clara opción para incrementar la productividad de estas instalaciones. En particular, la utilización de los procesadores gráficos (GPUs) para el cálculo de propósito general ha sido la opción más adoptada, llegando incluso a incorporarse en los grandes supercomputadores que aparecen en los primeros lugares de la lista Top500 [15]. Esta incorporación se ha debido en parte a la aparición de entornos de programación, como CUDA y OpenCL, que permiten, de una forma más o menos sencilla, la ejecución de parte del código de las aplicaciones en las GPUs.

Las GPUs son procesadores masivamente paralelos que ofrecen un gran rendimiento para aplicaciones con un alto grado de paralelismo de datos, mostrando un bajo rendimiento en aquellas partes de la aplicación que no presentan esta característica. Otro aspecto a tener en cuenta a la hora de utilizar las GPUs para cálculo de propósito general es que tienen un espacio de direccionamiento separado del de la CPU. Por ello la forma más común de utilizar las GPUS es la siguiente. Cuando se desea que una parte de la aplicación se ejecute en la GPU, se debe: a) copiar desde el espacio de memoria de la CPU al espacio de memoria de la GPU los datos que se van a tratar; b) cargar, también desde la memoria de CPU a la memoria de GPU, el código que debe ejecutar la GPU para tratar esos datos (que se suele denominar *kernel*); c) ordenar a la GPU que ejecute dicho código; y, por último, d) copiar desde el espacio de memoria de la GPU al de la CPU los resultados generados por la ejecución de dicho código. Así pues, con esta forma de funcionamiento, las aplicaciones factibles de ser aceleradas utilizando GPUs son aquellas que presentan partes de código con un alto grado de paralelismo, y en las que el tiempo necesario para las distintas transferencias entre el espacio de direccionamiento de la CPU y el espacio de direccionamiento de la GPU, suponen un pequeño coste respecto a la reducción de tiempo que se obtiene al realizar la ejecución en la GPU.

Por otra parte, la práctica totalidad de los grandes supercomputadores presenta una arquitectura de cluster, en donde un conjunto de cientos o miles de nodos colaboran en la ejecución de un código paralelo. Esta colaboración se realiza utilizando redes de altas prestaciones, entre las que destacan en la actualidad aquellas que se basan en el estandar de InfiniBand. En estos grandes supercomputadores, si se desea utilizar la potencia de cálculo de las GPUs, se debe incorporar al menos una GPU por nodo. Esto tiene diversos inconvenientes, que no pueden ser obviados:

- **Económicos**: incorporar el GPU a cada uno de los nodos incrementa TCO (*Total Cost of Ownership*), que incluye tanto los costes de adquisición como los de funcionamiento y mantenimiento de las instalaciones.
- **Ecológicos**: incorporar una GPU a cada uno de los nodos incrementa la energía necesaria para el funcionamiento del sistema, lo que no sólo repercute a nivel económico, sino también a nivel de la huella de carbono para la instalación.
- **De rendimiento**: no todas las aplicaciones son susceptibles de beneficiarse de su ejecución en una GPU. Ni siquiera aquellas que pueden obtener beneficio, lo pueden hacer durante todas las fases de su ejecución, sino que la mayor parte de veces sólo es posible en determinadas partes de la misma. Esto hace que se disponga de un hardware (GPU) que sólo se utiliza du-

[1]DICC, Universitat Jaume I (UJI), 12.071 - Castellón (España), e-mails: carregon@gap.upv.es, {siserte, adcastel, mayo, quintana}@uji.es.
[2]MCS, Argonne National Laboratory, 60.439 - Illinois (USA), e-mail: apenya@mcs.anl.gov.
[3]DISCA, Universitat Politècnica de València (UPV), 46.022 - Valencia (España), e-mails: {fsilla, jduato}@disca.upv.es.

rante un cierto tiempo, pudiendo estar inactivo durante largos periodos.

Todo esto ha hecho que en los últimos años se hayan desarrollado tecnologías que permiten la virtualización de GPUs para el cálculo de propósito general basándose en el acceso a GPUs remotas que pueden ser compartidas entre distintos nodos. Así, con un menor número de GPUs, equipando sólo a algunos nodos con GPUs, se pueden obtener los mismos beneficios que dotando a todos los nodos de GPUs. Entre las diversas tecnologías de virtualización de GPUs podemos nombrar rCUDA [3][8] y vCUDA [14]. De estas tecnologías rCUDA es la que tiene un mayor grado de madurez, permitiendo el acceso a GPUs remotas utilizando las capacidades de las redes InfiniBand con un sobrecoste muchas veces despreciable respecto a la utilización de GPUs locales.

Para que estas herramientas sean útiles en grandes instalaciones es necesario que se integren dentro de los entornos de planificación de trabajos que en ellas se utilizan. Esto es, los sistemas de colas de trabajos deben ser conscientes de la existencia de un nuevo recurso, las GPUs remotas, que puede ser compartido entre varios trabajos, y también deben poder controlar el uso que se hace de ellos. En este trabajo se presenta el desarrollo que se ha realizado para integrar rCUDA dentro de SLURM, de forma que los trabajos, además de solicitar los recuros estandar de cómputo y almacenamiento, también pueden solicitar la utilización de GPUs remotas, siendo todo ello gestionado por la herramienta de planificación de forma automática.

En la sección II se realiza una introducción a SLURM y su funcionamiento. De la misma forma, en la sección III se introduce rCUDA. En la sección IV se introducen las modificaciones que ha sido necesario incorporar a SLURM para que esta herramienta gestione también la utilización de GPUs remotas. Por último, en la sección V se presentan las pruebas de funcionamiento realizadas, y la sección VI se dedica a las conclusiones.

II. SLURM: Simple Linux Utility for Resource Management

Un gestor de recursos es una herramienta capaz de asignar los recursos disponibles dentro de un sistema, siguiendo unas directrices, a las aplicaciones que requieren el uso de esos recursos. SLURM[1][2] es un sistema de gestión de recursos y de planificación de trabajos open-source, tolerante a fallos y escalable para todo tipo de clusters. SLURM incluye gestión de trabajos, gestión de particiones y control sobre el estado de las máquinas que forman el cluster.

SLURM proporciona tres servicios clave:

- Reserva recursos de forma exclusiva o no para los usuarios durante un tiempo determinado permitiendo a éstos utilizar dichos recursos.
- Ofrece un entorno para el lanzamiento y monitorización de los trabajos que se encuentran en el sistema.
- Dispone de un arbitraje para el manejo de la cola de tareas dependiendo de los recursos requeridos por los trabajos y aquellos disponibles.

Cada trabajo que SLURM va a procesar se divide en uno o varios `steps`. Un `step` es una parte del fragmento de trabajo asignado a un nodo. El número de `steps` creados dependerá de la complejidad del trabajo.

Como se observa en la Figura 1, SLURM está formado por el demonio `slurmd` que se ejecuta en todos los nodos del sistema, un demonio `slurmctld` que se ejecuta sobre el nodo de gestión y seis comandos para la interacción con estos demonios: `srun, scancel, scontrol, sinfo, squeue y smap`. Los comandos tienen las siguientes funcionalidades:

- `srun` se utiliza para enviar un trabajo para su ejecución, reservar recursos, adjuntarlo a una reserva de recursos ya realizada o iniciar los `steps` de cada trabajo.
- `scancel` se utiliza para la cancelación de uno o varios trabajos encolados o bien la cancelación de uno o varios `steps` pertenecientes a un trabajo.
- `scontrol` es una herramienta administrativa para el control del estado de los nodos del sistema pudiendo consultarlo y/o modificarlo. La información devuelta contiene datos como: nombre del nodo, arquitectura, recursos que ofrece al cluster, memoria RAM, el estado en que se encuentra, etc.
- `sinfo` devuelve la información sobre las particiones y los nodos gestionados por SLURM.
- `squeue` muestra el estado de la cola de trabajos. Algunos de los datos devueltos son: identificador del trabajo (se utiliza como parámetro del comando `scancel`), nombre del trabajo, usuario que lo ha encolado, los nodos que utiliza y el estado del trabajo.
- `smap` muestra información sobre trabajos, particiones y los nodos gestionados por SLURM gráficamente.

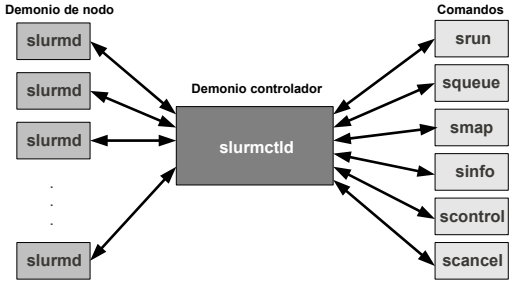


Fig. 1. Arquitectura de SLURM

Además de los anteriores, existen dos comandos adicionales que son `sbatch y salloc`. Ambos comandos forman parte de un nivel de uso superior que, después de realizar las tareas propias, utilizan

finalmente el comando `srun`.

- `sbatch` se utiliza para enviar un script creado por el usuario donde se especifican todos los parámetros que necesita el trabajo (sockets por nodo, cores por socket, etc.). Con este script, SLURM posteriormente crea la llamada al comando `srun` con la configuración correspondiente.
- `salloc` se utiliza para reservar un conjunto de recursos donde se va a realizar la ejecución interactiva de trabajos.

Una vez se lanza un trabajo para que se planifique y ejecute, se realizan una serie de envíos de mensajes entre todos los componentes que intervienen en este proceso. En la Figura 2 se muestran las comunicaciones y el orden en que se realizan.

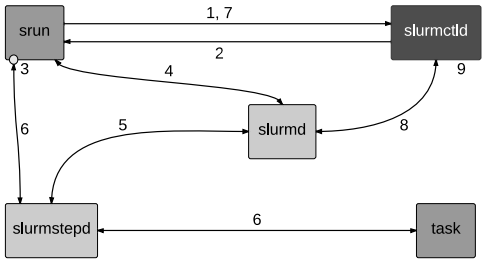


Fig. 2. Paso de mensajes en SLURM

Los mensajes intercambiados en la Figura 2 son:

1. Cuando se envía un trabajo al gestor de recursos con el comando `srun`, se crea un proceso el cual envía la solicitud al controlador `slurmctld`.
2. El controlador, después de planificar y asignar recursos al trabajo, devuelve una credencial a `srun`.
3. `srun` abre los sockets necesarios para la comunicación del trabajo.
4. `srun` propaga la credencial con la información de la tarea a los procesos `slurmd` de cada nodo.
5. Los procesos `slurmd` de cada nodo crean los procesos hijos `slurmstepd`.
6. Los nuevos procesos `slurmstepd` se conectan al socket de `srun` y, posteriormente, ejecutan el trabajo asignado (`task`). Al finalizar, los `slurmstepd` envían la información de finalización de su tarea a través del socket de `srun`.
7. `srun` avisa a `slurmctld` que el trabajo ha sido completado.
8. `slurmctld` verifica que todos los procesos `slurmd` involucrados en este trabajo hayan terminado correctamente.
9. `slurmctld` libera los recursos reservados por este trabajo, quedando éstos libres para los trabajos posteriores.

SLURM utiliza un plug-in general para la selección de aspectos como la política de asignación de recursos o la configuración de los nodos. Cuando el demonio de SLURM se inicia, lee el fichero de configuración para conocer qué módulo y plug-ins debe cargar.

III. rCUDA: Remote CUDA

rCUDA [3][4][5][6] es un entorno que proporciona acceso transparente a GPUs instaladas en nodos remotos, a aplicaciones que utilizan la tecnología CUDA. De esta forma, cada nodo de un cluster puede tener acceso a todas las GPUs instaladas en el sistema tal y como se ilustra en la Figura 3. Desde el punto de vista del nodo todas las GPUs son locales, a pesar de estar físicamente instaladas en un nodo remoto. Esta implementación elimina la restricción actual de CUDA que sólo permite a un nodo utilizar las GPUs instaladas de forma local.

Con la arquitectura cliente-servidor de rCUDA, mostrada en la Figura 4, las peticiones CUDA que realizan las aplicaciones son redirigidas por el entorno al servidor utilizando la capa de comunicaciones. El servidor se encarga de procesar y ejecutar la llamada sobre una GPU real y devolver los resultados al cliente. Para el cliente, el comportamiento es como si se ejecutaran en una GPU local.

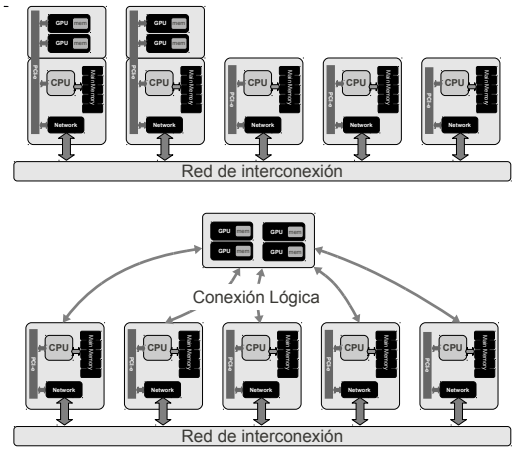


Fig. 3. Superior: sin rCUDA las GPUs son sólo accesibles desde los nodos donde están instaladas. Inferior: con rCUDA se puede acceder a las GPUs desde cualquier nodo

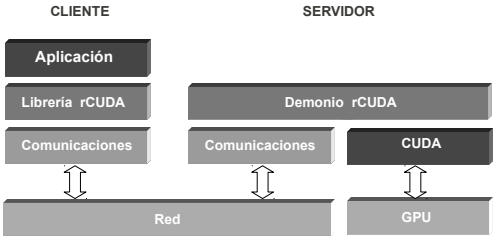


Fig. 4. Arquitectura Cliente-Servidor de rCUDA

La capa de comunicaciones que conecta el servidor con el cliente está formada por un módulo de comunicaciones distinto para cada tipo de red (actualmente TCP/IP e InfiniBand Verbs). rCUDA proporciona una API (*Application Programming Interface*)

genérica para el desarrollo de estos módulos que permite aprovechar al máximo las prestaciones de cada red de interconexión.

En cuanto al rendimiento que puede obtenerse al trabajar sobre rCUDA, éste es ligeramente inferior al ofrecido por CUDA, puesto que la conexión entre la CPU y la GPU además del bus PCIe incluye la red entre los nodos.

La versión actual de rCUDA es la 4.0.1 [9] y ofrece soporte para la API de CUDA 5.0, a excepción de los módulos de interoperabilidad con gráficos. Además de mejoras en el uso de transferencias asíncronas, también soporta el resto de bibliotecas CUDA como CUBLAS [10], CUFFT [11] o Thrust [12].

IV. Modificaciones a SLURM

En esta sección se explican a grandes rasgos las modificaciones realizadas a SLURM, así como en qué se diferencia el comportamiento con la versión original en cuanto a configuración y uso.

A. Qué se ha modificado

En primer lugar, se han añadido nuevos atributos a varias estructuras de datos para que sean capaces de gestionar la información de las GPUs en las particiones, los nodos y los trabajos.

En segundo lugar, para permitir la compartición de GPUs dentro de los nodos del cluster, se ha tenido que modificar el módulo GRes, ya que éste es el encargado de reservar y liberar recursos genéricos como las GPUs.

Además, se han implementado dos nuevos plug-ins para SLURM. Por un lado, un plug-in GRes que declara un nuevo recurso genérico en el sistema. Este plug-in se llama `"gres/rgpu"` y permite el uso compartido de GPUs (a partir de ahora nombradas como rGPUs). Por otro lado, también se ha creado un plug-in que planifica y selecciona los recursos para un trabajo. Este plug-in está basado en el funcionamiento del plug-in `"select/cons_res"`, donde añade la capacidad de planificar y compartir los recursos rGPU. Siguiendo la nomenclatura propia de SLURM, el plug-in se ha nombrado `"select/cons_rgpu"`.

Finalmente, se han modificado varios paquetes que se envían durante las comunicaciones RPC, para que admitan más campos con la información referida al resultado de la planificación de rGPUs. A su vez, se han añadido nuevas variables de entorno:

- RCUDA_DEVICE_COUNT. Utilizada para que el cliente rCUDA conozca el número de GPUs disponibles.
- RCUDA_DEVICE_X. Utilizada para que el cliente rCUDA conozca la dirección IP del nodo donde se encuentra una GPU. La X tomará los valores desde 0 hasta el número de GPUs.

Estas variables contendrán la información necesaria para poder ejecutar los trabajos sobre rCUDA.

B. Cómo se configura SLURM tras la modificación

El fichero de configuración `"slurm.conf"` debe incluir las siguientes líneas:

- `SelectTypeParameters = CR_CORE`
- `SelectType = select/cons_rgpu`
- `GresTypes = rgpu, gpu`

Estas líneas indican a SLURM que utilice el core como la unidad de recurso consumible mínima para planificar y reservar en el nuevo plug-in de selección `"select/cons_rgpu"`, ya que para asignar rGPUs a un trabajo es necesario asignar como mínimo un core para ejecutarlo. Además, se indica que existen los recursos genéricos rGPU y GPU en el sistema. En este mismo fichero también se tienen que configurar los nodos que tengan rGPUs:

```
NodeName=rcu[1-4] CPUs=8 [...] gres=rgpu:2
```

Con la línea anterior indicamos a SLURM que los nodos `rcu1, rcu2, rcu3` y `rcu4` tienen la misma configuración hardware (8 CPUs, 2 rGPUs, etc.).

Además, a estos nodos con rGPUs se les debe añadir la configuración (localización, versión y memoria) de éstas en su fichero `"gres.conf"`:

```
Name=rgpu File=/dev/nvidia0 Cuda=3.5 Mem=5G
```

Con la línea anterior indicamos a un nodo que su recurso rGPU, accesible en el path `/dev/nvidia0` tiene una Compute Capability [8] de 3.5 y una memoria de 5 Gbytes.

C. Cómo funciona SLURM tras la modificación

Al ejecutar el controlador `slurmctld`, se empieza a configurar SLURM, donde se inicializará un contador global de rGPUs.

Una vez configurado e inicializado con éxito, es el momento de enviar el trabajo al gestor de recursos. Se han implementado los tres modos de envío de un trabajo que permite SLURM, por lo que se puede utilizar cualquiera ellos. Estos modos son: `srun, salloc y sbatch`. Se va a centrar el ejemplo en el comando `srun` por ser el que se ejecuta siempre. Al enviar el trabajo habrá que indicarle el número de recursos que necesitamos y la memoria de éstos:

```
srun -N1 --gres=rgpu:4:512M script.sh
```

Con esta línea se ordena que `script.sh` se ejecute en un nodo y se informa de que va a necesitar `512MB` de cada una de las 4 rGPUs solicitadas. También se puede enviar al planificador indicándole específicamente en qué nodos se debe ejecutar el script:

```
srun -w"rcu14" --gres=rgpu:4:512M script.sh
```

En este caso `script.sh` se lanzará en el nodo `rcu14` del cluster. Nótese que en cualquiera de los dos casos, las GPUs pueden estar en cualquier nodo del cluster, al tratarse de un recurso rGPU.

Después de enviar el trabajo, entra en juego el plug-in `"select/cons_rgpu"`. Éste verificará si el trabajo no necesita rGPUs para hacer la planificación tal y como lo haría el plug-in original

"select/cons_res". En caso de que sí necesite hacer uso de rGPUs, se activará el "rgpu_mode".

Si el sistema cumple estos requisitos, el trabajo podrá mandarse a ejecución o bien encolarse en el caso de que, aunque existan recursos suficientes en el sistema, algunos de ellos no estén disponibles. La selección de los recursos `rgpu` se lleva a cabo en la parte de código de SLURM modificado mientras que el resto de recursos requeridos por el trabajo se seleccionarán del mismo modo en que SLURM lo hacía originalmente.

Acto seguido es el momento de reservar los recursos. Hasta entonces solo se ha comprobado que el trabajo no requiera más recursos de los existentes. Esta reserva se lleva a cabo en el módulo GRes modificado para compartir GPUs. En esta versión se ha implementado una política de reserva muy sencilla, en la cual se recorre una lista de rGPUs de forma cíclica seleccionando aquellos recursos que cumplan los requisitos. Este proceso terminará cuando las necesidades del trabajo hayan sido satisfechas.

Cuando termine la ejecución del trabajo, los recursos anteriormente reservados serán liberados para poder ser usados por otros trabajos.

Los cambios efectuados sobre el código de SLURM se ven reflejados en el resto de comandos. Esto significa que la modificación de los ficheros de configuración es visible a todo el sistema. Un ejemplo es la Figura 5, donde se muestra la salida del comando `scontrol` y cómo aparecen en la lista de Gres tanto las 4 GPUs físicas como las 4 rGPUs que aporta el nodo `rcu16` al sistema. A su vez se observa cómo el nodo `rcu17` no tiene ninguna GPU ni rGPU aunque en la ejecución de tareas podrá utilizar las del nodo `rcu16`.

```
NodeName=rcu16 Arch=x86_64 CoresPerSocket=4
   CPUAlloc=0 CPUErr=0 CPUTot=8 Features=(null)
   Gres=rgpu:4,gpu:4
   NodeAddr=rcu16 NodeHostName=rcu16
   OS=Linux RealMemory=7682 Sockets=1
   State=IDLE ThreadsPerCore=2 TmpDisk=0 Weight=1
   BootTime=2013-04-24T18:45:35
   SlurmdStartTime=2013-04-30T10:02:04


NodeName=rcu17 Arch=x86_64 CoresPerSocket=4
   CPUAlloc=0 CPUErr=0 CPUTot=8 Features=(null)
   Gres=(null)
   NodeAddr=rcu17 NodeHostName=rcu17
   OS=Linux RealMemory=7682 Sockets=1
   State=DOWN* ThreadsPerCore=2 TmpDisk=0 Weight=1
   BootTime=2013-04-24T18:16:04
   SlurmdStartTime=2013-04-30T10:02:04
```

Fig. 5. Salida del comando `scontrol show node` después de la modificación

Otro ejemplo es el estado de la cola de trabajos en un instante determinado, visualizado mediante el comando `squeue` que se muestra en la Figura 6. En ella se observa cómo el trabajo con `JOBID 855` utiliza 6 rGPUs desde el nodo rcu16, que solamente posee 4 GPUs.

```
JOBID NAME  USER      ST  TIME NODES NODELIST(REASON)
 858  r6g4  siserte   PD  0:00     1 (Resources)
 860  r2g1  adcastel  PD  0:00     3 (Priority)
 861  r1g2  adcastel  PD  0:00     1 (Priority)
 863  r4g0  martineh  PD  0:00     1 (Priority)
 855  r6g0  siserte   R   0:48     1 rcu16
 859  r0g0  siserte   R   0:32    16 rcu[3-18]
 862  r1g0  catalans  R   0:02    10 rcu[3-12]
```

Fig. 6. Salida del comando `squeue` después de la modificación

V. Pruebas realizadas

Las primeras pruebas de estas modificaciones se han ejecutado sobre dos clusters de tamaño reducido:

- Un cluster de 3 nodos con máquinas virtuales basadas en KVM [13].
- Un cluster de 4 nodos del grupo HPC&A de la Universitat Jaume I.

Ambos sistemas han servido para realizar todos los pasos en la implementación del código introducido a SLURM. El primero ha servido para poder realizar una gran cantidad de configuraciones en los nodos, tanto en lo referente a recursos genéricos como no genéricos. El segundo cluster se ha utilizado para probar el funcionamiento fuera del entorno de máquinas virtuales.

Se han evaluado todas las posibles configuraciones de recursos que un trabajo puede requerir:

- No se necesitan GPUs.
- Se necesita una o varias GPUs locales.
- Se necesita una o varias GPUs remotas.
- Se necesitan tanto GPUs locales como remotas.
- Se necesita una cantidad específica de memoria en la GPU.

Para poder contrastar los resultados después de la planificación, y a su vez comprobar la correcta integración del planificador SLURM con rCUDA, se han utilizado varios de los ejemplos que proporciona el SDK de CUDA como `bandwidthTest [7]`, `simpleMultiGPU [7]` y `simpleMPI [7]`.

`BandwidthTest` se ha utilizado para comprobar la correcta comunicación entre nodos y GPUs ya sean GPUs locales como remotas.

El ejemplo `simpleMultiGPU` ha sido utilizado para comprobar la comunicación entre un nodo y varias GPUs llegando a utilizar todas las GPUs disponibles en el sistema.

Con el ejemplo `simpleMPI` se ha verificado que una vez insertadas las modificaciones del código, la funcionalidad no se ve alterada.

Para mostrar el comportamiento de SLURM después de la modificación, se realizó una prueba sobre un cluster formado por 4 nodos y 3 GPUs con esta configuración.

- Nodo 0: 2 GPUs
- Nodo 1: 1 GPUs
- Nodo 2: 0 GPUs
- Nodo 3: 0 GPUs

En el test, se enviaban 5 trabajos con una espera entre ellos de un segundo. Las caracterśticas de los

trabajos son:

- Trabajo 1: Nodos: 2 GPUs: 3
- Trabajo 2: Nodos: 3 GPUs: 1
- Trabajo 3: Nodos: 1 GPUs: 0
- Trabajo 4: Nodos: 2 GPUs: 2
- Trabajo 5: Nodos: 2 GPUs: 1

Se puede observar en la Figura 7 el funcionamiento de SLURM con las modificaciones para que el planificador sea capaz de tratar las GPUs del cluster como si fueran locales a todos los nodos. En la parte inferior de la gráfica se observa el estado de la cola de trabajos en cada instante. Cabe destacar que los trabajos 4 y 5 están haciendo uso de las GPUs instaladas en el nodo 0 de forma simultánea.

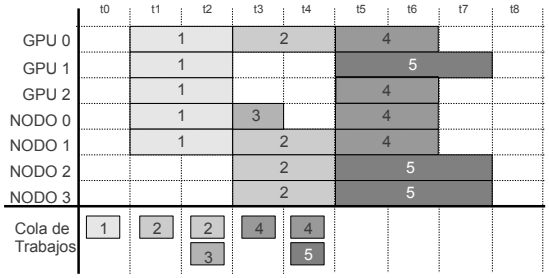


Fig. 7. Funcionamiento SLURM modificado

Como se puede ver en la Figura 8, con la versión original de SLURM no se podría realizar esta planificación puesto que el trabajo 1, que necesita 3 GPUs, no se podría ejecutar ya que ningún nodo tiene instaladas de forma local 3 GPUs. Además, al no compartir las GPUs, aunque el trabajo 1 no se ejecute, la ejecución de los trabajos tarda más tiempo que en el SLURM modificado.

Fig. 8. Funcionamiento SLURM sin modificar

Finalmente, también se han hecho pruebas a mayor escala de esta modificación de SLURM sobre un cluster del grupo Parallel Architectures Group (GAP) de la Universitat Politècnica de València. El cluster está formado por 21 nodos y 6 GPUs con distintas distribuciones Linux y distintas configuraciones de hardware. Las GPUs se reparten entre 3 nodos: 4 GPUs en el nodo 16 y 1 en los nodos 14 y 15.

VI. Conclusiones

Se ha modificado el comportamiento genérico de SLURM para que pueda realizar la planificación de los trabajos entrantes teniendo en cuenta las GPUs del sistema como si fueran recursos compartidos entre todos los nodos. Para ello se ha creado un nuevo recurso compartido (GRes) para las GPUs y se ha añadido toda la lógica necesaria para que el planificador pueda actuar sobre este nuevo recurso.

Se ha modificado el comportamiento para los comandos `srun, sbatch y salloc` y se ha desarrollado un algoritmo básico para la asignación de GPUs basándose en la ocupación de la GPU y la memoria requerida por el trabajo.

Como trabajo futuro queda la implementación de un algoritmo de asignación de rGPUs con una mayor cantidad de características, como el porcentaje de uso de GPU, la capacidad de la GPU, etc.

Agradecimientos

Los investigadores de la Universitat Jaume I de Castelló han sido financiados por el Ministerio de Economía y Competitividad español (MINECO), el programa FEDER (contrato TIN2011-23283), la Fundación Caixa-Castelló/Bancaixa (contrato P11B2011-19) y Mellanox Technologies. Por otro lado, el personal de la Universitat Politècnica de València ha sido subvencionado por el MINECO y por fondos FEDER bajo el acuerdo TIN2012-38341-C04-01. Para finalizar añadir que parte del equipo utilizado en los experimentos fue donado por Mellanox Technologies y NVIDIA Corporation.

Referencias

[1] SLURM website, http://computing.llnl.gov/linux/slurm/slurm.html
[2] Andy B. Yoo, Morris A. Jette, Mark Grondona, *"SLURM: Simple Linux Utility for Resource Management"*, in Job Scheduling Strategies for Parallel Processing, L. Rudolph and U. Schwiegelshohn, Editors. 2003, SpringerVerlag. p. 44-60.
[3] J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí y F. Silla, *"An Efficient Implementation of GPU Virtualization in High Performance Clusters"*, Euro-Par 2009, Parallel Processing - Workshops, vol. 6043 Lecture Notes in Computer Science, p. 385-394. Springer-Verlag, 2010.
[4] J. Duato, A. J. Peña, F. Silla, R. Mayo y E. S. Quintana-Ortí, *"rCUDA: Reducing the Number of GPU-based Accelerators in High Performance Clusters"*, High Performance Computing and Simulation, 2010.
[5] J. Duato, A. J. Peña, F. Silla, R. Mayo y E. S. Quintana-Ortí, *"Performance of CUDA Virtualized Remote GPUs in High Performance Clusters"*, International Conference on Parallel Processing, 2011.
[6] rCUDA Team, rCUDA User's guide v4.0.1, rCUDA, 2013.
[7] NVIDIA, The NVIDIA GPU Computing SDK, NVIDIA, 2012.
[8] NVIDIA, CUDA Toolkit Reference Manual, NVIDIA, 2012.
[9] rCUDA website, www.rcuda.net
[10] NVIDIA, CUDA CUBLAS User Guide, NVIDIA, 2012.
[11] NVIDIA, CUDA CUFFT User Guide, NVIDIA, 2012.
[12] NVIDIA, CUDA Thrust Quick Start Guide, NVIDIA, 2012.
[13] KVM website, http://www.linux-kvm.org/.
[14] L. Shi, H. Chen, and J. Sun, *"vCUDA: GPU accelerated high performance computing in virtual machines"* IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09), 2009
[15] Top500 website, http://http://www.top500.org/

Estudio y Requisitos de la Simulación de Sistemas de Cloud Computing

Francisco J. Clemente[1], Rafael Mayo[1] y Enrique S. Quintana-Ortí[1]

Resumen— **El cloud computing o computación en la nube es una tecnología reciente que permite que los recursos TI (Tecnologías de la Información) y las aplicaciones, sean ofertadas a los usuarios finales como servicios, bajo un modelo de pago por uso. Cada vez más empresas están utilizando esta tecnología para virtualizar sus aplicaciones y utilizar únicamente los recursos TI necesarios para su ejecución en cada momento, ahorrando costes de adquisición, mantenimiento y energía. Normalmente estas aplicaciones tienen un complejo despliegue y distintas configuraciones y requisitos. Evaluar y predecir el rendimiento, la asignación de recursos, las cargas de trabajo y los costes que éstas conllevan en un entorno de cloud computing, utilizando distintas configuraciones y variaciones del sistema, son tareas complejas y costosas de lograr. Para llevar a cabo estas tareas, sería conveniente utilizar un simulador de cloud computing que permitiera evaluar y predecir el comportamiento de estas aplicaciones en este entorno. En este artículo se presentan los requisitos básicos que debería tener un simulador de cloud computing, y se hace una comparativa entre varios simuladores de cloud computing existentes en la actualidad.**

Palabras clave— **Simulación, Cloud Computing, Virtualización, Centro de datos.**

I. Introducción

El paradigma de cloud computing o computación en nube está ganando actualmente mucha importancia en distintas comunidades, incluyendo investigadores, empresarios, clientes, organizaciones y gobiernos. Infraestructura, plataforma y software se complementan para ofrecer una serie de servicios, con un modelo de pago por uso, que forman lo que se denomina cloud computing. El modelo de servicio que ofrece infraestructura se denomina *Insfrastructure as a Service* (IaaS), el que ofrece plataforma se denomina *Platform as a Service* (PaaS), y el que ofrece software se denomina *Software as a Service* (SaaS). Se considera que el cloud computing será el motor de las siguientes generaciones de centros de datos, ya que ofrece un alojamiento dinámico de recursos y una gran flexibilidad [4].

Muchas empresas de TI están invirtiendo tiempo y dinero en desarrollar sus propias aplicaciones para su beneficio empresarial utilizando esta tecnología tan reciente. Mediante el uso del cloud computing, los recursos necesarios para ejecutar una aplicación se pueden adaptar a las necesidades reales que ésta necesita, reduciendo así el coste del propietario o TCO (Total Cost of Ownership). Algunas de las aplicaciones que están emergiendo del cloud computing son las redes sociales, el alojamiento web, el procesamiento de datos, y la gestión de contenidos entre otras. Todas ellas tienen distintas configuraciones, requisitos, composición y despliegue. Evaluar y predecir el rendimiento de estas aplicaciones en un centro de datos, los recursos que necesitan para su ejecución y el consumo energético que conllevan, son tareas difíciles de llevar a cabo actualmente debido a la imposibilidad de realizar dichas pruebas en un entorno real de cloud computing. Por ello, es conveniente utilizar simuladores de cloud computing que sean capaces de simular las distintas configuraciones y requisitos, tanto de las aplicaciones como de los centros de datos donde se pueden ejecutar, evaluando en cada caso su rendimiento, costes, y recursos necesarios entre otros aspectos.

Actualmente existen algunos simuladores en la literatura que simulan algunos de estos aspectos. En este trabajo se estudian cuatro simuladores: MDCSim[2], GDCSim[3], GreenCloud[4] y CloudSim[5]. Después de analizar cada uno de ellos se ha concluido que ninguno es capaz de simular todos los aspectos implicados en un entorno de cloud computing.

En este trabajo se presentan una serie de características y requisitos teóricos que debería cumplir un simulador de cloud computing para ofrecer unos resultados más realistas.

El trabajo está organizado de la siguiente forma. En la sección II se realiza un estudio de algunos simuladores de cloud computing existentes y se analizan sus características y arquitectura. En la sección III se presentan las características básicas que debería contemplar un simulador de cloud computing. En la sección IV se realiza una comparativa de los simuladores presentados con las características básicas de un simulador. Por último, en la sección V se muestran las conclusiones y el trabajo futuro a realizar.

II. Estudio de simuladores de cloud computing

El número de entornos de simulación de uso público para centros de datos dedicados al cloud computing es limitado actualmente. En este trabajo se ha realizado un estudio de algunos de los simuladores existentes. A continuación se describen dichos simuladores, analizando en cada caso su arquitectura, características y funcionamiento [1].

A. MDCSim

MDCSim [2] es una plataforma de simulación para centros de datos Multi-Tier, escrita en CSIM y desarrollada en la universidad de Pensilvania en 2009. Este simulador se presenta como una plataforma de

[1]Departamento de Ingeniería y Ciencia de los Computadores, Universidad Jaume I, 12071 - Castellón, e-mails: {fclement,mayo,quintana}@uji.es

simulación completa, flexible y escalable, que permite realizar un análisis profundo de centros de datos Multi-Tier. Esta diseñado como una arquitectura de tres capas interconectadas que permiten simular distintos paradigmas de comunicación, planificación a nivel de kernel y las interacciones a nivel de aplicación entre los tiers de un centro de datos Tier-III. La flexibilidad de este simulador yace en la posibilidad de experimentar con diferentes alternativas de diseño en las tres capas disponibles (diferentes tiers, algoritmos de planificación, tipos de interconexión, etc.), analizando en todas ellas el rendimiento y el consumo energético de cargas de trabajo reales. Los distintos experimentos se pueden realizar con un gran número de nodos en el clúster; es decir, es un simulador escalable.

La desviación media entre las medidas reales y las dadas por el simulador están entorno al 10 % para la latencia y el 3 % para el consumo energético.

En la Figura 1 se muestran los bloques funcionales de este simulador. Posee un diseño de tres capas (capa de comunicación, capa de kernel y capa nivel-usuario) para modelar la pila completa de los protocolos de comunicación de algunas aplicaciones específicas.

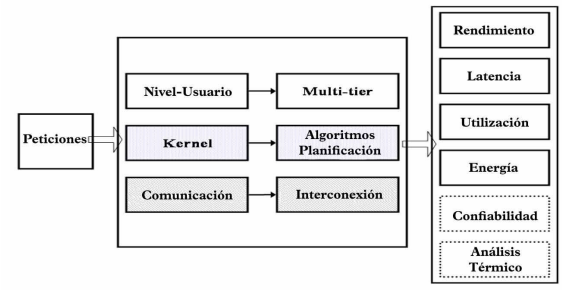


Fig. 1. Visión funcional del simulador MCDSim

La comunicación intra-cluster está modelada en la capa de comunicación. El simulador soporta Infiniband Architecture (IBA) y Ethernet sobre TCP/IP como tecnologías de interconexión. La capa de kernel se ha modelado sobre el planificador de Linux 2.6.9, que permite mantener una cola de ejecución para cada CPU en el sistema. Y la capa de aplicación o de usuario proporciona las características de la arquitectura Tier-III.

Este simulador permite analizar el comportamiento de diferentes configuraciones de un clúster para optimizar distintos objetivos funcionales sin tener que realizar grandes cambios en la configuración del simulador. Entre las aplicaciones posibles de este simulador se encuentra el análisis del rendimiento de las interconexiones; el análisis térmico y energético; la determinación de la configuración óptima de un clúster para cumplir con ciertos objetivos de rendimiento, tolerancia a fallos y fiabilidad, etc.

B. GDCSim

Green Data Center Simulator (GDCSim) [3] es una herramienta de simulación para el análisis, gestión de recursos y diseño de un centro de datos eficiente o Green Data Center. Esta herramienta se desarrolló como parte del proyecto BlueTool en el Impact Lab de la universidad del estado de Arizona en 2011.

GDCSim es un simulador que estudia la eficiencia energética de centros de datos con distintas arquitecturas y diseños, características de cargas de trabajo, esquemas de gestión de la energía, y algoritmos de planificación. Trata de unificar la simulación de técnicas de gestión con la simulación del comportamiento físico de un centro de datos.

Entre las características principales de este simulador se pueden destacar las siguientes:

- *Procesamiento automático*, que permite realizar operaciones sin la intervención del usuario.
- *Capacidad de análisis en tiempo real*, que proporciona la simulación en tiempo real de decisiones de gestión basadas en cambios en el entorno físico de un centro de datos.
- *Análisis iterativo de diseños*, que permite analizar y probar diferentes configuraciones de un centro de datos antes de desplegarlas.
- *Capacidad de análisis térmico*, caracteriza los efectos térmicos dentro de la sala de un centro de datos en un momento dado.
- *Gestión de carga de trabajo y energía*, que permite planificar las cargas de trabajo y controlar la energía de los servidores en cada caso.
- *Interdependecia entre los recursos físicos*, que proporciona información de los patrones del aire y temperatura de los nodos en un ciclo cerrado, dentro de un centro de datos, para proporcionar información a los algoritmos de gestión de la refrigeración.

En la Figura 2 se muestra la arquitectura de GDCSim. Los módulos definidos en su arquitectura son:

- *BlueSim*. Paquete de simulación desarrollado a través del proyecto *BlueTool*, que integra varias aplicaciones para la generación de diseños de centros de datos, simulaciones de dinámica de fluidos, etc. Su principal objetivo es generar un conjunto de configuraciones de centros de datos diferentes a través de una especificación XML.
- *Gestión de Entrada/Salida*. Esta herramienta tiene dos funciones principales. En primer lugar, sirve como interfaz de usuario para gestionar la simulación; y en segundo lugar sirve para almacenar una gran variedad de configuraciones de centros de datos para diferentes conjuntos de servidores activos.
- *Gestión de recursos*. Este módulo proporciona distintos algoritmos para la gestión de cargas de trabajo, la gestión energética y la gestión de la refrigeración.
- *Simulador*. El simulador está compuesto por cuatro submódulos: módulo de colas de trabajo, módulo de energía, módulo termodinámico y módulo de refrigeración. A través de los datos

que recibe como entrada realiza una simulación a través de sus submódulos y obtiene como salida unos datos de registro y una retroalimentación de información a los módulos de Gestión de Entrada/Salida y Gestión de recursos.

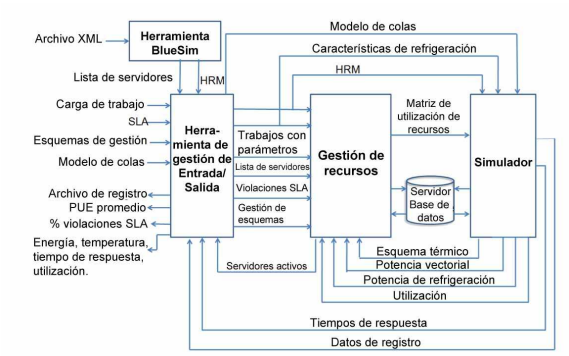


Fig. 2. Arquitectura del simulador GDCSim

C. *GreenCloud*

Green Cloud Simulator o GreenCloud [4] es un entorno de simulación energético para centros de datos dedicados al cloud computing, desarrollado en la universidad del estado Dakota del Norte en 2010. Este simulador está diseñado para obtener detalles sobre la energía consumida por los distintos componentes de un centro de datos (servidores, conmutadores, y enlaces) así como de diversos modelos de redes de comunicación con distintas configuraciones reales. Ha sido desarrollado como una extensión del simulador de redes de comunicación Ns2 [8], por tanto, implementa un modelo de referencia completo del protocolo TCP/IP, permitiendo realizar simulaciones con diferentes protocolos de comunicación como IP, TCP y UDP. Los resultados de las simulaciones incluyen información energética para arquitecturas de centros de datos Tier-I, Tier-II, y Tier-III utilizando técnicas de gestión de energía como el escalado de voltaje, escalado de frecuencia, y el apagado dinámico, que son aplicadas tanto a los componentes de computación como de red.

GreenCloud distingue tres tipos de consumo energético entre componentes: consumo computacional, consumo de interconexión, y consumo relativo a cada componente físico de la infraestructura de un centro de datos.

En la Figura 3 se muestra la arquitectura de este simulador. La estructura de GreenCloud está basada en una arquitectura Tier-III. Un servidor es el componente principal de un centro de datos, ya que es el encargado de procesar y ejecutar las tareas. En GreenCloud, los componentes de los servidores implementan nodos de un solo núcleo que tienen parámetros preestablecidos como un límite de potencia en MIPS o FLOPS, un tamaño de memoria/almacenamiento, y unas técnicas y mecanismos de planificación como round-robin, entre otros. Los submódulos de Núcleo de Red, Agregación de la Red y Acceso de Red, que incluyen conmutadores y enlaces, forman la red de interconexión entre todos los componentes del centro de datos.

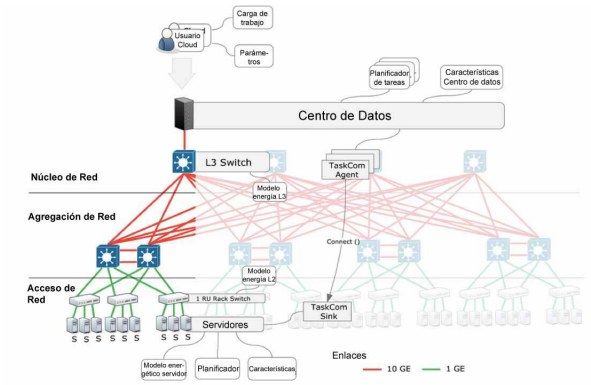


Fig. 3. Arquitectura del simulador Green Cloud Simulator

Por otra parte, las cargas de trabajo de este simulador han sido diseñadas para cubrir la mayor parte de aplicaciones típicas de cloud computing. Éste define los siguientes tipos de cargas de trabajo:

- Cargas de trabajo de computación intensiva (HPC).
- Cargas de trabajo de datos intensivos.
- Cargas de trabajo equilibradas.

D. *CloudSim*

CloudSim [5] en una herramienta de simulación desarrollada en la Universidad de Melbourne en 2010, con el objetivo de ofrecer un conjunto extensible de herramientas de simulación que permitan modelar y simular los sistemas de cloud computing y las aplicaciones asociadas a este entorno. CloudSim implementa tanto la estructura como el comportamiento de los componentes de un centro de datos dedicado al cloud computing (máquinas virtuales, políticas de asignación de recursos, etc.). Actualmente soporta entornos de simulación de cloud computing consistentes en un simple centro de datos así como federaciones de centros de datos interconectados dedicados al cloud computing. Además, incluye la posibilidad de implementar diferentes técnicas de asignación de recursos para las máquinas virtuales en diferentes tipos de escenarios.

Las principales ventajas de CloudSim son:

- *Eficiencia de tiempo*: se requiere muy poco tiempo y esfuerzo para implementar una aplicación de test y realizar pruebas con distintos entornos.
- *Flexibilidad y utilidad*: los desarrolladores pueden modelar y probar el rendimiento de sus aplicaciones en entornos heterogéneos de cloud computing (Amazon EC2, Microsoft Azure, etc.) con pequeños esfuerzos de programación.

Las principales características de CloudSim son:

- Soporte para el modelado y simulación a gran escala de entornos de cloud computing, incluyendo centros de datos con un simple nodo de cómputo.

- Modelos propios de Cloud, agentes de servicio, abastecimiento, y políticas de asignación.
- Soporte para la simulación de conexiones de red entre los componentes.
- Facilidad para simular federaciones de centros de datos de cloud computing.
- Capacidad de simular mecanismos de virtualización y gestión de servicios virtualizados.
- Flexibilidad en la simulación de técnicas de asignación de recursos entre Espacio-Compartido y Tiempo-Compartido.

En la Figura 4 se muestra la arquitectura por capas del simulador CloudSim. La capa CloudSim se encarga del modelado y simulación de entornos de virtualización de centros de datos incluyendo una gestión de interfaces de red, memoria, almacenamiento y ancho de banda de las máquinas virtuales. Tareas fundamentales como la asignación de nodos físicos a máquinas virtuales, la gestión de ejecución de aplicaciones, y la monitorización dinámica del estado del sistema, son gestionadas en esta capa.

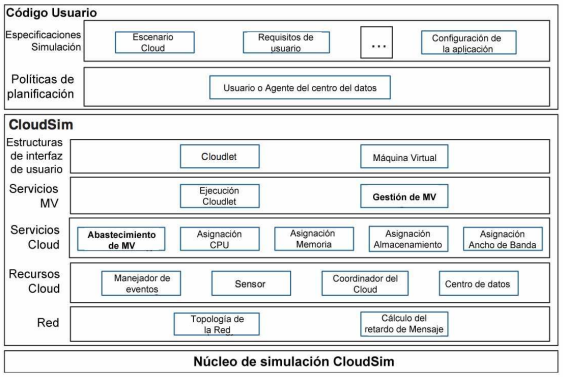


Fig. 4. Arquitectura del simulador CloudSim

La capa más alta de CloudSim es el Código de Usuario, que gestiona los parámetros básicos de los usuarios o agentes (número de máquinas, sus especificaciones, etc.), las aplicaciones (número de tareas y sus requisitos), las máquinas virtuales, el número de usuarios, y las políticas de planificación de un agente de Cloud.

III. Características básicas de un simulador de cloud computing

En esta sección se presentan una serie de características, que consideramos adecuadas, para realizar una simulación más afín a un entorno real de cloud computing [6] [7].

Un simulador de cloud computing debería ser capaz de simular todas los elementos implicados en dicho entorno, desde la simulación de la infraestructura de un centro de datos hasta la parte lógica y de aplicación que permitiera ejecutar tareas en máquinas virtuales. De esta forma, se podrían obtener datos tanto de rendimiento de máquinas virtuales y aplicaciones, como de energía consumida por los nodos físicos donde se han ejecutado, obteniendo información del coste aproximado que supondría la ejecución de una determinada aplicación en un entorno determinado. Además, de esta forma se podrían simular los costes reales de la energía consumida por todos los elementos que forman la infraestructura de un centro de datos y simular los costes de personal y de las máquinas virtuales de forma individual. Con todo ello, un administrador del cloud podría realizar una simulación de los precios que debería aplicar a la prestación de sus servicios para que fueran rentables y sostenibles.

En la Figura 5 se muestra la arquitectura propuesta en este trabajo. Se trata de un diseño modular agrupado en capas, que permite extender o ampliar cada uno de los módulos de forma individualizada, pudiendo así mejorar y adaptar cada uno de ellos según las necesidades convenientes.

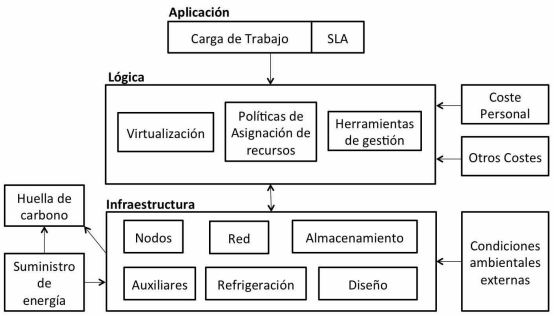


Fig. 5. Arquitectura del simulador propuesto

Esta arquitectura se ha dividido en tres capas principales: una capa de Infraestructura, que contiene toda la parte física y energética de un centro de datos; una capa Lógica, que representa toda la gestión de la virtualización y otras herramientas para el control y gestión de un centro de datos; y una capa de Aplicación, donde se encuentran las cargas de trabajo que representan el conjunto de tareas y aplicaciones que se pueden ejecutar en un entorno de cloud computing.

A continuación se describen los módulos que componen las tres capas de esta arquitectura.

A. Capa de Infraestructura

- *Suministro de energía.* Este módulo representa el tipo de fuente de energía que se utiliza en un centro de datos. Esta fuente de energía proviene de energías renovables (eólica, solar, hidráulica, etc.) o de energías no renovables (combustibles fósiles y combustibles nucleares). El suministro de energía de un centro de datos puede ser dinámico, combinando en cada caso los distintos tipos de energía.
- *Huella de carbono.* Mediante este módulo se representa la huella de carbono producida por el tipo de suministro de energía y la energía consumida por los componentes de la infraestructura.

- *Nodos.* Este módulo representa los nodos físicos que componen el centro de datos, así como todas sus características físicas de CPU, memoria, tarjeta de red (limitaciones de ancho de banda), y almacenamiento. Además, cada tipo de nodo debe contener un modelo de consumo energético que represente su consumo real aproximado en cada caso. Por otra parte, también debe simular técnicas de ahorro energético como Dynamic Voltage and Frequency Scaling (DVFS), Dynamic Power Management (DPM) o Dynamic Concurrency Throttling (DCT) .
- *Red.* La topología de interconexión del centro de datos así como sus características (tipos de conexión, tipo de dispositivos, latencia, etc.) son representadas por este módulo. Además, este módulo debe poseer un modelo de consumo energético para los distintos tipos de dispositivos que lo componen.
- *Almacenamiento.* Este módulo representa la gestión del almacenamiento de información en disco de los nodos físicos. Se pueden representar los tipos de disco utilizados (HDD, SSD, etc), así como su seguridad o tipos de interconexión, etc.
- *Auxiliares.* Todos los elementos auxiliares como la iluminación de las salas o los sensores de temperatura son representados por este módulo. Éste debe simular los costes que añaden estos tipos de elementos.
- *Refrigeración.* Este módulo representa la refrigeración aplicada en el centro de datos. Simula los tipos de refrigeración así como el consumo energético que supone su utilización en cada instante.
- *Diseño.* En este módulo se simula la distribución de todos los componentes físicos de un centro de datos, es decir, los tamaños de las salas y su número de nodos físicos, la distribución de la refrigeración, los sistemas de seguridad (replicación de componentes), etc. Mediante este diseño y la topología de la red, se podrá determinar el tipo de fiabilidad del centro de datos y su correspondiente certificación Tier.
- *Condiciones medioambientales externas.* Este módulo simula las condiciones medioambientales externas en cada instante, ya que esta información puede afectar a los sistemas de refrigeración aumentando o disminuyendo su utilización y coste.

B. Capa Lógica

- *Virtualización.* Este módulo gestiona toda la virtualización de máquinas virtuales. Contiene el tipo de hipervisor (Xen, KVM, VMWare, etc.), las características de una máquina virtual (CPU, memoria, almacenamiento y limitación de la red) y la gestión de creación, destrucción y modificación de las máquinas virtuales. Por otra parte, también debe contener un modelo de consumo energético que corresponda a cada máquina virtual.
- *Políticas de asignación de recursos.* Este módulo se encarga de asignar los recursos físicos necesarios a las máquinas virtuales para ejecutar un tipo de aplicación determinada. Un aspecto importante de este módulo es la utilización de algoritmos de reasignación dinámica de recursos o migración en vivo de máquinas virtuales para equilibrar la carga de trabajo, ahorrando costes.
- *Herramientas de gestión.* Este módulo representa todas las herramientas de gestión y control de un centro de datos como el *Data Centre Infrastructure Management* (DCIM). Además, simula el rendimiento de las máquinas virtuales y determina el cumplimiento de los acuerdos de nivel de servicio o SLAs (Service Level Agreement), establecidas por las aplicaciones o clientes.
- *Coste Personal.* En un centro de datos debe haber un personal cualificado para llevar a cabo tareas de gestión y control, comprobando en todo momento el correcto funcionamiento del mismo. Este módulo considera los costes que representa el personal así como su posible latencia ante la solución de un determinado suceso en el sistema.
- *Otros Costes.* Este modulo representa otros costes no contemplados en el resto de módulos, como los costes de subcontratación, gestión administrativa, o limpieza.

C. Capa de Aplicación

- *Carga de trabajo.* Este módulo representa las características de la carga de trabajo a procesar en un entorno de cloud computing. Simula diferentes tipos de cargas, indicando las características y requisitos que deben cumplir para satisfacer sus necesidades. Además, simula el rendimiento obtenido en cada caso así como sus costes finales, teniendo en cuenta los costes en cada etapa de procesamiento.
- *SLA.* Este módulo representa la calidad de servicio que debe ofrecer una determinada carga de trabajo en su ejecución dentro del entorno de cloud computing. Simula distintos tipos de SLAs así como posibles fallos, y su repercusión en las aplicaciones ejecutadas.

IV. Comparativa de los simuladores estudiados con las características propuestas

Algunas de las características descritas en la sección anterior han sido ya desarrolladas en diversos simuladores actuales. A continuación se realiza una breve comparativa de algunas de las diferencias entre los simuladores estudiados y las características descritas.

En la Figura 6 se muestra una tabla comparativa que señala las características que los simuladores actuales ya contemplan.

Podemos observar que GDCSim y GreenCloud ya tienen en cuenta aspectos relativos al suministro

	MDCSim	GDCSim	GreenCloud	CloudSim
Suministro Energía		✓	✓	
Nodos	✓	✓	✓	✓
Red	✓		✓	✓
Almacenamiento	✓		✓	
Auxiliares				
Refrigeración		✓		
Diseño	✓	✓	✓	✓
Condiciones externas		✓		
Virtualización				✓
Herramientas de gestión	✓	✓	✓	✓
Políticas de asginación	✓	✓	✓	✓
Costes Personal y Otros				
Cargas de trabajo	✓	✓	✓	✓

Fig. 6. Comparativa de las características propuestas

eléctrico, aunque no realizan un análisis del coste que éstos conllevan. En cuanto a la gestión de nodos, todos los simuladores estudiados realizan una caracterización de los nodos, aunque solo GreenCloud y CloudSim utilizan técnicas de ahorro energético como DVFS o DPM. Además sólo GreenCloud implementa un modelo preciso de energía de un nodo.

En cuanto a la interconexión de la red, MDCSim, CloudSim y GreenCloud ofrecen una simulación de la topología de la red, teniendo en cuenta sus tipos de conexión y latencias.

GDCSim, por su parte, es el único que tiene en cuenta algunos aspectos de la refrigeración como variable de entrada en su proceso de simulación, aunque tampoco determina su topología ni un modelo energético.

En cuanto al diseño, todos los simuladores estudiados permiten especificar algunas características del centro de datos. En el caso de CloudSim permite realizar simulaciones incluso en federaciones de clouds.

CloudSim es el único simulador que incorpora una capa de virtualización en su arquitectura para simular la gestión de máquinas virtuales en el contexto de cloud computing.

Otros aspectos comunes a los simuladores estudiados son: la simulación de herramientas de gestión, ya que en todos se puede realizar un análisis del rendimiento obtenido de sus cargas de trabajo, además de simular algunos requisitos como son las SLAs; las políticas de asignación de recursos; y las cargas de trabajo, ya que en todos ellos se pueden especificar las características que deben cumplir para satisfacer las necesidades del cliente.

Existen algunos aspectos que no han sido contemplados por ninguno de los simuladores estudiados como por ejemplo, el coste de los elementos auxiliares, el coste de personal, o la huella de carbono que producen. Estas cuestiones tienen cierta relevancia a la hora de obtener un coste más preciso de una aplicación; por tanto creemos que deben ser tenidos en cuenta.

Por otra parte, cabe destacar que los simuladores MDCSim, GDCSim y GreenCloud están más enfocados a simular el comportamiento de un centro de datos como tal, y no tanto a simular un entorno de cloud computing. En cambio, CloudSim está más enfocado a un cliente que desee simular el rendimiento y costes de una aplicación ejecutándose en un entorno de cloud computing; además simula una capa de virtualización que es una pieza fundamental en el cloud computing. Por tanto, CloudSim sería el simulador más idóneo para este fin, aunque actualmente carece de ciertas características importantes en cuanto a la parte de centro de datos y costes energéticos.

V. Conclusiones y trabajo futuro

Actualmente, existen muy pocos simuladores que ofrezcan características propias de un entorno de cloud computing. Algunos de éstos han sido estudiados en este trabajo, destacando y comparando sus ventajas e inconvenientes. En este trabajo se han propuesto una serie de características que debería cumplir un simulador para ofrecer una mayor precisión a la hora de simular el rendimiento y los costes de una determinada aplicación ejecutándose en un entorno de cloud computing. Después de comparar dichas características con los simuladores estudiados, se concluye que CloudSim sería el que más se aproxima al objetivo propuesto, ya que simula una capa de virtualización que el resto no contempla, aunque actualmente carece de ciertas características importantes que el resto sí que pueden ofrecer.

Como trabajo futuro se pretende desarrollar un simulador o extender alguno de los que ya existen, de forma que cumpla con todos los requisitos y características que se han propuesto. Se trataría de un simulador enfocado tanto al cliente como a los administradores de un centro de datos, ya que permitiría obtener información acerca del rendimiento y coste de las aplicaciones, así como de la energía y coste real que supone la ejecución de dichas aplicaciones.

Referencias

[1] Kaur, G. *Study of Comparison of Various Cloud Computing Simulators.*

[2] Lim, S. H., Sharma, B., Nam, G., Kim, E. K., and Das, C. R. (2009, August). *MDCSim: A multi-tier data center simulation, platform.*, In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on (pp. 1-9). IEEE.

[3] Gupta, S. K., Gilbert, R. R., Banerjee, A., Abbasi, Z., Mukherjee, T., and Varsamopoulos, G. (2011, July). *GDCSim: A tool for analyzing green data center design and resource management techniques.* In Green Computing Conference and Workshops (IGCC), 2011 International (pp. 1-8). IEEE.

[4] Kliazovich, D., Bouvry, P., and Khan, S. U. (2012). *GreenCloud: a packet-level simulator of energy-aware cloud computing data centers.* The Journal of Supercomputing, 62(3), 1263-1283.

[5] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.* Software: Practice and Experience, 41(1), 23-50.

[6] Tsai, W. T., Sun, X., and Balasooriya, J. (2010, April). *Service-oriented cloud computing architecture.* In Information Technology: New Generations (ITNG), 2010 Seventh International Conference on (pp. 684-689). IEEE.

[7] Zhang, L. J., and Zhou, Q. (2009, July). *CCOA: Cloud computing open architecture.* In Web Services, 2009. ICWS 2009. IEEE International Conference on (pp. 607-616). Ieee.

[8] *The Network Simulator Ns2* (2010). Disponible en: http://www.isi.edu/nsnam/ns

Plataforma de Computación Científica sobre Infraestructura Cloud Privada Oportunista

Javier Bachrachas Peterburg, Ailyn Baltá Camejo, César Cayo Ventura,
José Luis Vázquez-Poletti y José Antonio Martín H.

Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense de Madrid, 28040, Madrid (España)

Resumen— **SmartCloud es una solución completa (IaaS + SaaS) siguiendo el paradigma de cloud computing. A través de una arquitectura de capas escalable se implementa una nube privada que gestiona los elementos de una infraestructura ya existente añadiendo a los mismos la funcionalidad de nodo de procesamiento en los tiempos de inactividad sin afectar su utilización habitual. Sobre esta plataforma se ha implementado una solución de cálculo científico open source (iPython) que utiliza los recursos antes mencionados como soporte hardware para procesamiento paralelo y funcionalidades de clúster. Como resultado de esta solución se puede brindar a los usuarios un nuevo servicio sin inversión en hardware ni licencias con el valor añadido de la reutilización de infraestructuras ya existentes.**

Palabras clave— **Cloud Computing, iPython, reutilización hardware, optimización infraestructuras, educación.**

I. Introducción

La situación económica actual ha hecho que muchos grupos de trabajo en informática enfoquen sus esfuerzos en encontrar nuevas maneras de optimizar recursos o ampliar los servicios brindados a los usuarios sin que esto implique un aumento en los costes[8].

El primer enfoque que surge ante esta necesidad es migrar los sistemas hacia el software de código abierto, algo que es tendencia actual en organizaciones tanto privadas como estatales[4]. Un ejemplo de esto es la decisión del gobierno de Extremadura de migrar sus sistemas a código abierto y Linux, con un ahorro estimado de 30 millones de euros anuales[3].

En términos de la capa de aplicación, en el mundo de la investigación y la enseñanza también existen alternativas de código abierto a herramientas bien establecidas en el mercado como Matlab o Mathematica, las que al ser un software propietario tienen coste de licencia y su desarrollo está cerrado a una compañía, impidiendo las mejoras de la comunidad. En este proyecto hemos realizado una implantación de iPython[1] como alternativa a las aplicaciones antes mencionadas, permitiendo lograr una funcionalidad similar en términos de potencia de cálculo, pero adaptada a las necesidades y paradigma de este proyecto.

Un paso más en el análisis del problema, nos lleva a plantear la pregunta de cómo optimizar el soporte hardware de las aplicaciones. Nuestra aproximación a este problema es la reutilización de infraestructura ya existente para aprovechar su capacidad de procesamiento en los tiempos de inactividad[6].

Tomando como ejemplo la situación de la Facultad de Informática de la Universidad Complutense de Madrid, existen grupos de investigación que necesitan realizar procesamiento de datos y como la Facultad no puede brindarle servicios de procesamiento (o bien por qué no los tiene, o por restricciones presupuestarias), deben recurrir a proveedores externos como por ejemplo Amazon[1].

Nuestra solución a este problema es utilizar los ordenadores de los laboratorios de la Facultad cuando no están siendo utilizados, y sobre cada uno de ellos iniciar una máquina virtual con un nodo de procesamiento, lo que tiene un impacto nulo en la utilización rutinaria de los recursos y nos permite generar un clúster dinámico de procesamiento[9].

Hay que recordar los problemas que conlleva utilizar proveedores externos para el procesamiento de datos en términos de latencia y de protección de datos, por lo que hemos creado la solución "SmartCloud" como una propuesta para optimizar recursos y lograr brindar un mejor servicio a usuarios tal y como se puede apreciar en la Figura 1.

La solución se trata de una plataforma de Cloud Computing de nube privada[2] pues se lleva a cabo con los propios recursos existentes (máquinas, redes, almacenamiento, instalaciones) y su acceso está restringido a una red cerrada. Incluye un componente de Infrastructure as a Service (IaaS) y un componente de Software as a Service (SaaS) unidas por una capa de integración.

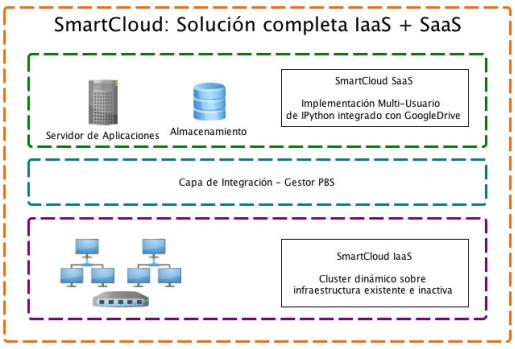


Fig. 1. Diagrama de Solución SmartCloud integrando IaaS + SaaS

[1] http://aws.amazon.com/es/ec2/
[2] http://www.postgradoeinvestigacion.uadec.mx/CienciaCierta/CC29/11.html

Se trata de una solución escalable y flexible, dado que al contar con un diseño arquitectónico por capas, se puede reutilizar o modificar la plataforma para implementar otras soluciones software sobre la misma.

El servicio brindado al usuario final se trata de una implementación de iPython multiusuario, que permite que cualquier usuario del sistema acceda a través de un navegador de internet a nuestra aplicación, permitiéndole realizar simulaciones, gráficos, cálculos, procesamiento de código Python y otras funcionalidades con la ventaja del procesamiento distribuido. El procesamiento no se realiza en el ordenador del cliente sino en nuestro servidor centralizado y en el clúster dinámico antes mencionado.

Nuestra plataforma permite adicionalmente la sincronización de archivos de usuario con Google Drive, permitiendo el trabajo colaborativo.

En esta contribución primero se estudiará el estado del arte para evaluar las alternativas existentes. Luego, se explicará la arquitectura de la solución haciendo la distinción entre diseño de alto y bajo nivel. Posteriormente se detallará el modelo de ahorro de costes e impacto económico de la solución, lo que ayuda a entender la motivación del proyecto. Para terminar se explicarán las conclusiones del trabajo y las posibles ampliaciones que puede tener.

II. Estado del arte

Con el objetivo de implementar una solución siguiendo el paradigma de Cloud-Computing, hemos basado nuestra aproximación en aplicaciones ya existentes que cuentan con una buena aceptación en la comunidad de usuarios y han demostrado buenos resultados. Se han valorado los casos particulares de Boinc [3] y Wolfram — Alpha[4].

La Infraestructura Abierta de Berkeley para la Computación en Red (BOINC) es una infraestructura para la computación distribuida, diseñada para ser una plataforma libre. Está dirigido a proyectos que requieren una gran capacidad de cálculo a través de la utilización de ordenadores personales distribuídos alrededor del mundo, realizando los cálculos en los ciclos de CPU o GPU que no son utilizados. BOINC consiste en un servidor y un cliente que se comunican para distribuir y procesar el trabajo. Cada ordenador es asignado a un único proyecto[5]. Wolfram—Alpha tiene la apariencia de motor de búsqueda pero no busca respuestas a las preguntas en un conglomerado de páginas web o documentos, sino que realiza cálculos en su servidor y envía la respuesta para que sea visualizada por el usuario a través de una interfaz web[5].

Se concluyó que no existía una solución completa que proporcionara los servicios que se querían ofrecer pero si existían soluciones que los cubrían parcialmente, por lo que se realizó un análisis de las herramientas existentes como componentes para implementar una solución completa.

Como parte del diseño arquitectónico de la aplicación surge la necesidad de utilizar: un hipervisor, un gestor de colas PBS para la capa de integración y una herramienta de cálculo potente con interfaz web.

La versión analizada de VirtualBox es de código abierto y sujeto a la licencia GPL[6]. Además, es funcionalmente idéntico en todas las plataformas de acogida y utiliza el mismo archivo y formato de imagen. Esto permite ejecutar la misma imagen de máquina virtual en instancias de VirtualBox que se ejecutan sobre diferentes sistemas operativos por lo que es fácil exportar las imágenes entre entornos, ofreciendo mayor portabilidad. VirtualBox está en constante desarrollo y actualización por lo que nuevas funcionalidades son implementadas regularmente. Además, cuenta con una API que nos permite integrarlo fácimente a nuestra solución[6].

VMware también cuenta con una versión gratis con funcionalidades similares a VirtualBox, pero al realizar una comparativa de las tecnologías se descarta dado que la filosofía de la solución es utilizar software de código abierto [7].

El gestor PBS Torque (Tera-scale Open-source Resource and Queue manager) es un administrador de recursos que nos provee de mecanismos para controlar trabajos sobre distintos nodos de un clúster. Permite gestionar el estado de los nodos y su utilización. Basado en OpenPBS, es el sistema de colas batch de código abierto líder en el mercado. Soporta el procesamiento en paralelo de múltiples batchs y colas de trabajo interactivas[8].

Otro de los objetivos de la solución es ofrecer una interfaz web. El estudio de herramientas disponibles se enfocó en encontrar un entorno interactivo que se pudiera integrar a la infraestructura existente y que contara con soporte multiusuario, pero que a la vez fuera una poderosa herramienta de cálculo. Considerando las restricciones mencionadas se evaluaron las alternativas de Sage y iPython.

Sage fue diseñado desde un principio como solución escalable a grandes servidores multiusuario centralizados (ejemplo sagenb.org con cerca de 76000 cuentas), pero con la desventja de que el mismo binario que se ejecuta en grandes servidores es utilizado para ejecuciones que no requieren tantos recursos, lo que se refleja en un bajo rendimiento en ciertos entornos. Sage tiene licencia GPL[9]. IPython es multiplataforma, software libre, tiene una enorme comunidad detrás, un desarrollo constante y bien organizado, permite paralelismo y es extremadamente potente. Nació de las manos de investigadores y en un ámbito científico, por lo que su comunidad es mayormente académica al igual que nuestros usuarios. IPython trabaja con un sistema de archivos standard accesible trivialmente desde el sistema opera-

[3] http://boinc.berkeley.edu/
[4] http://www.wolframalpha.com/
[5] http://es.wikipedia.org
[6] http://www.virtualbox.org/
[7] http://www.vmware.com/es/
[8] http://www.adaptivecomputing.com/products/open-source/torque/
[9] http://www.sagemath.org/

tivo, facilitando la realización de operaciones rutinarias en el análisis de datos y simulación numérica como acceder a scripts locales o ver archivos arbitrarios. IPython incluye una serie de opciones para dar formato a los documentos: capacidades multihoja, preámbulo de estilo LaTeX, metadatos por celda, navegación outline-level, y apoyo validación / reproducibilidad entre otras. IPython tiene licencia BSD. Además, permite añadir nuevas extensiones como Matlab, SQL y Physicsy[10].

III. Arquitectura de la solución

A. Diseño de alto nivel

Después de un análisis en profundidad llegó a la conclusión de implementar una arquitectura por capas por su beneficio modular.

- Capa Inferior: formada por los nodos de procesamiento conectados a una red para poder ser accesibles y gestionables por el servidor.
- Capa Intermedia de integración: comunica la capa superior con la inferior. Gestiona la distribución de los trabajos generados desde la capa superior.
- Capa Superior: contiene el front-end para la interacción con el usuario. En esta capa se encuentra la aplicación de cálculo que se despliega. Cuando la herramienta necesite realizar funciones sobre el cluster realizará llamadas a capas inferiores, lo que se refleja en un aumento en la carga del procesamiento del cluster.

Dada a la diversidad de alternativas para implementar estas capas, con el previo análisis de cada una, se llegó a las conclusiones de diseño:

- En la capa inferior para definir los nodos de procesamiento se decidió realizarlo mediante virtualización para homogeneizar las máquinas a usar. Como ya se ha comentado anteriormente, en cada nodo se instala una máquina virtual y sobre ésta una misma imagen. Se decidió utilizar Oracle VM VirtualBox, un hipervisor de tipo 2, dada su facilidad de instalación y configuración y sus garantías de acceso al hardware físico a través del sistema operativo host. Para el sistema operativo a virtualizar se decidió Ubuntu (Linux) por ser software libre. Ver figura 2.
- Para la capa intermedia, se decidió utilizar un gestor de trabajos PBS Torque que nos permite una gestión de procesamiento transparente. Además es el sistema de colas batch de código abierto líder en el mercado.
- Para la capa superior, se eligió iPython gracias a su entorno completo para la computación interactiva y exploratoria. Además, presenta una arquitectura robusta construida alrededor del concepto de intérprete de Python pero a la vez sencilla y eficaz[2]. También se tuvo en cuenta el interés creciente en esta herramienta (Figura 3).

[10] http://github.com/ipython/ipython/wiki/Extensions-Index#matlab

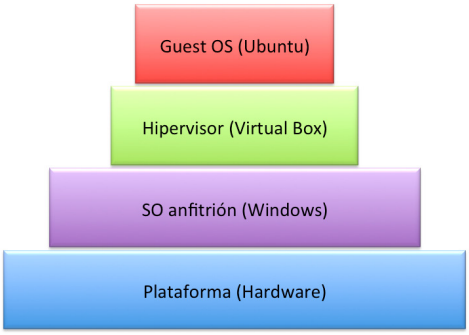


Fig. 2. Arquitectura de capas - Capa inferior

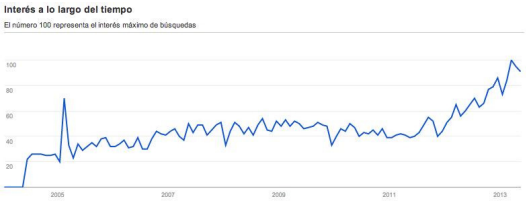


Fig. 3. Evolución del interés en iPython

B. Diseño de bajo nivel

Para la capa inferior fue necesario añadir tareas al SO que inicien la máquina virtual cuando detecte que no se estaba utilizando, la apaguen cuando reconociera un inicio de sesión e informen de su estado al cluster. En las máquinas virtuales se instala una imagen de Ubuntu 10.04 debido a su facilidad de administración. Como capa intermedia se utiliza el gestor de trabajos Torque. Se ha definido una única cola en esta herramienta en la que convergen todas las peticiones de las instancias de iPython. Al utilizar un gestor de trabajos, se simplifica la integración de ambas capas dado que es una interfaz transparente para la capa superior. En otras palabras, iPython solamente ve un único punto donde realizar peticiones/envíos de trabajos, aunque éste se encarga de distribuir los trabajos entre los recursos disponibles según políticas de optimización.
En la capa superior se define un frontend con gestión de usuarios. Para facilitar esta gestión y no guardar información de la cuenta de los usuarios, se utiliza la openID de Google y se evalúa si el usuario autenticado tiene permisos para ejecutar el notebook de iPython. Inicialmente iPython no estaba desarrollado para poder ser utilizado por varios usuarios de manera simultánea bajo un mismo servidor, razón por la cual tuvimos que modificarlo para añadir ésta funcionalidad. Dado que un usuario utiliza la cuenta de Google, tiene la posibilidad de sincronizar sus hojas con Google Drive. La figura 4 ilustra la arquitectura de la aplicación.

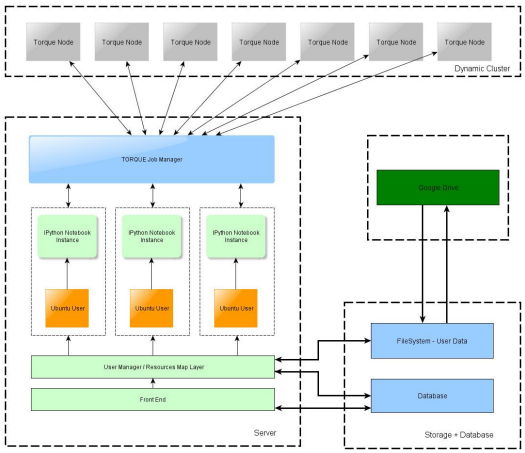


Fig. 4. Arquitectura de la solución.

IV. Estudio de costes e impacto económico

Uno de los objetivos de SmartCloud es lograr optimizar la utilización de la infraestructura existente, lo que se refleja en un ahorro en costes. Actualmente si la universidad necesita realizar cálculos que exceden sus capacidades de procesamiento debe recurrir a un proveedor externo[5]. Al utilizar el cluster dinámico de SmartCloud, se pueden ofrecer los mismos servicios de procesamiento sin tener que contratar a un proveedor externo y sin la latencia que implica el procesamiento remoto. El único sobrecoste significativo que implica esta plataforma, es el coste energético[10]. A continuación se detalla el modelo económico desarrollado para realizar estos cálculos[11].

$$C = W_l * T * F_c * F_u * C_e$$

$$O = \frac{W_l * (T_b + F_c * F_u * T_i) + W_i * (T_i * (1 - F_c * F_u))}{W_l * T_b + W_i * T_i}$$

En donde términos de costes C es el coste energético final y O el sobrecoste energético derivado de utilizar SmartCloud. W_l es el consumo energético de cada nodo con una carga standard en Kw/h y W_i el consumo energético de cada nodo estado inactivo en Kw/h. T se refiere al tiempo disponible para el procesamiento, T_b al Tiempo de utilización del clúster con carga sin utilizar SmartCloud, y T_i al tiempo de utilización del clúster en estado inactivo sin utilizar SmartCloud. Para estimar la carga se usan las variables F_c como factor de carga del clúster en el tiempo disponible, y F_u como factor de utilización de las horas disponibles. El coste de la energía en Kw/h se expresa con C_e

Se plantea un escenario de ejemplo para estudiar el problema. Se dispone de 20 ordenadores i7-3770K con 4GB de RAM y 4 cores con hyper-threading, en un laboratorio con la distribución de uso indicada en la figura 5.

En una semana hay 118 horas inactivas que podrían ser utilizadas por SmartCloud, formando un cluster de 160 cores con 80 GB de RAM.
Supongamos que se necesita utilizar todo el cluster

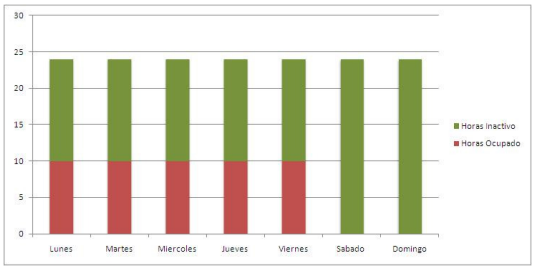


Fig. 5. Distribución de uso de ordenadores en el caso de estudio.

al 100 % de la carga, en el total de las horas disponibles. El consumo energético sería de 196,6€ lo que presenta un sobrecoste de 68 % respecto al consumo de energía actual. Ahora bien, si se recurriera a proveedores externos para realizar este procesamiento, ésto tendría un coste de media de 1508,68€. El cuadro I muestra una comparación de proveedores de servicios cloud que proporcionan un servicio equivalente [11].

Para calcular el ahorro final simplemente basta con restar al coste externo el coste energético,lo que se refleja en 1312€ de ahorro[11]. En términos de latencias, el trabajo dentro de una red local como lo que propone SmartCloud implica latencias sensiblemente menores frente a las que se experimentan trabajando con proveedores cloud. El cuadro II muestra las latencias de Amazon desde España[12].

Claramente son mayores a las latencias dentro de una red local. A su vez, iPython es código abierto, lo que comparado con herramientas como Matlab o Mathematica que tienen costes de licencias representa un ahorro significativo.

TABLA I
Comparación de proveedores Cloud.

Server Type	CPU	RAM (GB)	HDD (GB)	Base Hourly Cost (USD)	Hourly Cost (EUR)	Coste Mensual (EUR)
HP	4	4	120.0	$ 0.14	0,108 €	1.021,597 €
RackSpace	4	4	100.0	$ 0.22	0,168 €	1.583,475 €
SoftLayer	6	7.5	0.0	$ 0.23	0,178 €	1.678,338 €
AWS	4	7.5	850.0	$ 0.24	0,186 €	1.751,309 €

TABLA II
Latencias de Amazon desde España.

Service	Time (sec)	# of samples	Min (ms)	Max (ms)	Std Dev	Median (ms)	Avg (ms)
Amazon EC2	1.13	3	231	298	13.3 %	288	272.33
Amazon Cloud-Front	0.67	6	62	94	15.7 %	78	75.5

[11] http://planforcloud.rightscale.com
[12] http://www.cloudharmony.com/

V. Conclusiones y trabajo futuro

La importancia que tiene la utilización de plataformas cloud se refleja en el ahorro de costes tanto energéticos como en infraestructura tal como se refleja en el trabajo actual.

Uno de los principales aportes de este trabajo es la posibilidad de utilizar iPython en modo multiusuario, ya que actualmente la herramienta no implementa esta funcionalidad.

Al utilizar esta solución, se ofrece a los investigadores no sólo una poderosa herramienta de cálculo sino también una manera de aprovechar los recursos que existen sin costes significativos añadidos. Además, la integración de la herramienta con Google Drive facilita mucho más su usabilidad.

La plataforma resultante presenta una arquitectura de cluster. Una vez implementada, permite el despliegue de otras herramientas y recursos como apoyo a otros proyectos, nuevas aplicaciones, procesamiento batch, entre otros.

Este proyecto puede continuar para seguir explotando la escalabilidad e integración con otros recursos. Se puede dar la posibilidad de integrarlo con otros servicios cloud por ejemplo los de Amazon [13]. El diseño de una nueva solución se puede realizar desde distintos enfoques dado que se adapta a las tendencias actuales de cloud, permitiendo una mayor versatilidad[14].

Para este proyecto se ha desplegado en el cluster una herramienta en concreto, Ipython, pero cualquier aplicación distribuída podría ser desplegada, por ejemplo específicas de un proyecto o de alguna empresa que en determinado momento tenga un pico alto de consumo de recursos.

Una vez logrado el objetivo principal y consolidada la infraestructura, se podría ofrecer estos servicios de cálculo a otras facultades o entidades externas[7].

Como posible ampliación, se puede trabajar en un modelo de cluster híbrido con máquinas virtuales, servidores físicos y balanceadores de carga como alternativa a la actual implementación. También se pueden iniciar otras imágenes sobre las máquinas virtuales para implementar nuevas funcionalidades.

Un estudio en profundidad en lo referente a seguridad permitiría resolver los posibles fallos o vulnerabilidades del sistema, en el actual desarrollo solo se cubren los estándares mínimos necesarios.

Referencias

[1] F. Pérez, *The Ipython notebook: a historical retrospective*, `http://blog.fperez.org/2012/01/ipython-notebook-historical.html`

[2] F. Pérez, B. Granger, M. Ragan-Kelley, *Ipython: a very quick overview*, `http://fperez.org/talks/1010_ipython_overview.pdf`

[3] CIO, *Extremadura preveé ahorrar 30 millones de euros al año con diez proyectos de software libre*, `http://www.ciospain.es/aapp/extremadura-preve-ahorrar-30-millones-de-euros-al-ano-con-diez-proyectos-de-software-libre`

[4] A. Bernardo, *Cuáles son las universidades españolas que apoyan el software libre en 2013?*, `http://alt1040.com/2013/05/universidades-espanolas-que-apoyan-el-software-libre-2`

[5] D. Kondo, B. Javadi, P. Malecot, F. Cappello, D.P. Anderson, *Cost-benefit analysis of Cloud Computing versus desktop grids*, Parallel & Distributed Processing, Addison-Wesley, 2009.

[6] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, A. Ghalsasi, *Cloud computing - The business perspective*, Decision Support Systems, Volume 51, Issue 1, Pages 176-189, April 2011

[7] R. Buyya, C. Shin Yeo, S. Venugopal, J. Broberg, I. Brandic, *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility*, Future Generation Computer Systems, Volume 25, Issue 6, Pages 599-616, June 2009

[8] F. Etro, *The Economic Impact of Cloud Computing on Business Creation, Employment and Output in Europe. An application of the Endogenous Market Structures Approach to a GPT innovation*, Review of Business and Economics, Katholieke Universiteit Leuven, Faculteit Economie en Bedrijfswetenschappen, vol. 0(2), pages 179-208.

[9] N. Sultan, *Cloud computing for education: A new dawn?*, International Journal of Information Management,Volume 30, Issue 2, Pages 109-116, April 2010

[10] A. Berl, E. Gelenbe, M. di Girolamo, G. Giuliani,H. de Meer, M. Quan Dang, K. Pentikousis, *Energy-Efficient Cloud Computing*, The Computer Journal, Number 7, Volume 53, 2010

[11] G. D. Guerrero , R. M. Wallace, J. L. Vázquez-Poletti, J. M. Cecilia, J. M. García, D. Mozos and H. Perez-Sánchez *, A Performance/Cost Model for a CUDA Drug Discovery Application on Physical and Public Cloud Infrastructures*, Concurrency and Computation: Practice and Experience, Special Issue on Distributed, Parallel, and GPU-accelerated Approaches to Computational Biology, 2013, In Press

[13] http://ged.msu.edu/angus/beacon-2012/week1.html

[14] http://aws.amazon.com/es/ec2/

Plataforma de Computación Científica sobre Infraestructura Cloud Privada Oportunista

Javier Bachrachas Peterburg, Ailyn Baltá Camejo, César Cayo Ventura,
José Luis Vázquez-Poletti y José Antonio Martín H.

Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense de Madrid, 28040, Madrid (España)

Resumen— **SmartCloud es una solución completa (IaaS + SaaS) siguiendo el paradigma de cloud computing. A través de una arquitectura de capas escalable se implementa una nube privada que gestiona los elementos de una infraestructura ya existente añadiendo a los mismos la funcionalidad de nodo de procesamiento en los tiempos de inactividad sin afectar su utilización habitual. Sobre esta plataforma se ha implementado una solución de cálculo científico open source (iPython) que utiliza los recursos antes mencionados como soporte hardware para procesamiento paralelo y funcionalidades de clúster. Como resultado de esta solución se puede brindar a los usuarios un nuevo servicio sin inversión en hardware ni licencias con el valor añadido de la reutilización de infraestructuras ya existentes.**

Palabras clave— **Cloud Computing, iPython, reutilización hardware, optimización infraestructuras, educación.**

I. Introducción

La situación económica actual ha hecho que muchos grupos de trabajo en informática enfoquen sus esfuerzos en encontrar nuevas maneras de optimizar recursos o ampliar los servicios brindados a los usuarios sin que esto implique un aumento en los costes[8].

El primer enfoque que surge ante esta necesidad es migrar los sistemas hacia el software de código abierto, algo que es tendencia actual en organizaciones tanto privadas como estatales[4]. Un ejemplo de esto es la decisión del gobierno de Extremadura de migrar sus sistemas a código abierto y Linux, con un ahorro estimado de 30 millones de euros anuales[3].

En términos de la capa de aplicación, en el mundo de la investigación y la enseñanza también existen alternativas de código abierto a herramientas bien establecidas en el mercado como Matlab o Mathematica, las que al ser un software propietario tienen coste de licencia y su desarrollo está cerrado a una compañía, impidiendo las mejoras de la comunidad. En este proyecto hemos realizado una implantación de iPython[1] como alternativa a las aplicaciones antes mencionadas, permitiendo lograr una funcionalidad similar en términos de potencia de cálculo, pero adaptada a las necesidades y paradigma de este proyecto.

Un paso más en el análisis del problema, nos lleva a plantear la pregunta de cómo optimizar el soporte hardware de las aplicaciones. Nuestra aproximación a este problema es la reutilización de infraestructura ya existente para aprovechar su capacidad de procesamiento en los tiempos de inactividad[6].

Tomando como ejemplo la situación de la Facultad de Informática de la Universidad Complutense de Madrid, existen grupos de investigación que necesitan realizar procesamiento de datos y como la Facultad no puede brindarle servicios de procesamiento (o bien por qué no los tiene, o por restricciones presupuestarias), deben recurrir a proveedores externos como por ejemplo Amazon[1].

Nuestra solución a este problema es utilizar los ordenadores de los laboratorios de la Facultad cuando no están siendo utilizados, y sobre cada uno de ellos iniciar una máquina virtual con un nodo de procesamiento, lo que tiene un impacto nulo en la utilización rutinaria de los recursos y nos permite generar un clúster dinámico de procesamiento[9].

Hay que recordar los problemas que conlleva utilizar proveedores externos para el procesamiento de datos en términos de latencia y de protección de datos, por lo que hemos creado la solución "SmartCloud" como una propuesta para optimizar recursos y lograr brindar un mejor servicio a usuarios tal y como se puede apreciar en la Figura 1.

La solución se trata de una plataforma de Cloud Computing de nube privada[2] pues se lleva a cabo con los propios recursos existentes (máquinas, redes, almacenamiento, instalaciones) y su acceso está restringido a una red cerrada. Incluye un componente de Infrastructure as a Service (IaaS) y un componente de Software as a Service (SaaS) unidas por una capa de integración.

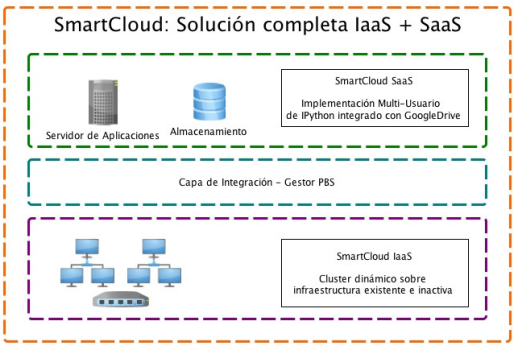


Fig. 1. Diagrama de Solución SmartCloud integrando IaaS + SaaS

[1] http://aws.amazon.com/es/ec2/
[2] http://www.postgradoeinvestigacion.uadec.mx/CienciaCierta/CC29/11.html

Se trata de una solución escalable y flexible, dado que al contar con un diseño arquitectónico por capas, se puede reutilizar o modificar la plataforma para implementar otras soluciones software sobre la misma.

El servicio brindado al usuario final se trata de una implementación de iPython multiusuario, que permite que cualquier usuario del sistema acceda a través de un navegador de internet a nuestra aplicación, permitiéndole realizar simulaciones, gráficos, cálculos, procesamiento de código Python y otras funcionalidades con la ventaja del procesamiento distribuido. El procesamiento no se realiza en el ordenador del cliente sino en nuestro servidor centralizado y en el clúster dinámico antes mencionado.

Nuestra plataforma permite adicionalmente la sincronización de archivos de usuario con Google Drive, permitiendo el trabajo colaborativo.

En esta contribución primero se estudiará el estado del arte para evaluar las alternativas existentes. Luego, se explicará la arquitectura de la solución haciendo la distinción entre diseño de alto y bajo nivel. Posteriormente se detallará el modelo de ahorro de costes e impacto económico de la solución, lo que ayuda a entender la motivación del proyecto. Para terminar se explicarán las conclusiones del trabajo y las posibles ampliaciones que puede tener.

II. Estado del arte

Con el objetivo de implementar una solución siguiendo el paradigma de Cloud-Computing, hemos basado nuestra aproximación en aplicaciones ya existentes que cuentan con una buena aceptación en la comunidad de usuarios y han demostrado buenos resultados. Se han valorado los casos particulares de Boinc [3] y Wolfram — Alpha[4].

La Infraestructura Abierta de Berkeley para la Computación en Red (BOINC) es una infraestructura para la computación distribuida, diseñada para ser una plataforma libre. Está dirigido a proyectos que requieren una gran capacidad de cálculo a través de la utilización de ordenadores personales distribuídos alrededor del mundo, realizando los cálculos en los ciclos de CPU o GPU que no son utilizados. BOINC consiste en un servidor y un cliente que se comunican para distribuir y procesar el trabajo. Cada ordenador es asignado a un único proyecto[5]. Wolfram—Alpha tiene la apariencia de motor de búsqueda pero no busca respuestas a las preguntas en un conglomerado de páginas web o documentos, sino que realiza cálculos en su servidor y envía la respuesta para que sea visualizada por el usuario a través de una interfaz web[5].

Se concluyó que no existía una solución completa que proporcionara los servicios que se querían ofrecer pero si existían soluciones que los cubrían parcialmente, por lo que se realizó un análisis de las herramientas existentes como componentes para implementar una solución completa.

Como parte del diseño arquitectónico de la aplicación surge la necesidad de utilizar: un hipervisor, un gestor de colas PBS para la capa de integración y una herramienta de cálculo potente con interfaz web.

La versión analizada de VirtualBox es de código abierto y sujeto a la licencia GPL[6]. Además, es funcionalmente idéntico en todas las plataformas de acogida y utiliza el mismo archivo y formato de imagen. Esto permite ejecutar la misma imagen de máquina virtual en instancias de VirtualBox que se ejecutan sobre diferentes sistemas operativos por lo que es fácil exportar las imágenes entre entornos, ofreciendo mayor portabilidad. VirtualBox está en constante desarrollo y actualización por lo que nuevas funcionalidades son implementadas regularmente. Además, cuenta con una API que nos permite integrarlo fácimente a nuestra solución[6].

VMware también cuenta con una versión gratis con funcionalidades similares a VirtualBox, pero al realizar una comparativa de las tecnologías se descarta dado que la filosofía de la solución es utilizar software de código abierto [7].

El gestor PBS Torque (Tera-scale Open-source Resource and Queue manager) es un administrador de recursos que nos provee de mecanismos para controlar trabajos sobre distintos nodos de un clúster. Permite gestionar el estado de los nodos y su utilización. Basado en OpenPBS, es el sistema de colas batch de código abierto líder en el mercado. Soporta el procesamiento en paralelo de múltiples batchs y colas de trabajo interactivas[8].

Otro de los objetivos de la solución es ofrecer una interfaz web. El estudio de herramientas disponibles se enfocó en encontrar un entorno interactivo que se pudiera integrar a la infraestructura existente y que contara con soporte multiusuario, pero que a la vez fuera una poderosa herramienta de cálculo. Considerando las restricciones mencionadas se evaluaron las alternativas de Sage y iPython.

Sage fue diseñado desde un principio como solución escalable a grandes servidores multiusuario centralizados (ejemplo sagenb.org con cerca de 76000 cuentas), pero con la desventja de que el mismo binario que se ejecuta en grandes servidores es utilizado para ejecuciones que no requieren tantos recursos, lo que se refleja en un bajo rendimiento en ciertos entornos. Sage tiene licencia GPL[9]. IPython es multiplataforma, software libre, tiene una enorme comunidad detrás, un desarrollo constante y bien organizado, permite paralelismo y es extremadamente potente. Nació de las manos de investigadores y en un ámbito científico, por lo que su comunidad es mayormente académica al igual que nuestros usuarios. IPython trabaja con un sistema de archivos standard accesible trivialmente desde el sistema opera-

[3] http://boinc.berkeley.edu/
[4] http://www.wolframalpha.com/
[5] http://es.wikipedia.org
[6] http://www.virtualbox.org/
[7] http://www.vmware.com/es/
[8] http://www.adaptivecomputing.com/products/open-source/torque/
[9] http://www.sagemath.org/

tivo, facilitando la realización de operaciones rutinarias en el análisis de datos y simulación numérica como acceder a scripts locales o ver archivos arbitrarios. IPython incluye una serie de opciones para dar formato a los documentos: capacidades multihoja, preámbulo de estilo LaTeX, metadatos por celda, navegación outline-level, y apoyo validación / reproducibilidad entre otras. IPython tiene licencia BSD. Además, permite añadir nuevas extensiones como Matlab, SQL y Physicsy[10].

III. Arquitectura de la solución

A. Diseño de alto nivel

Después de un análisis en profundidad llegó a la conclusión de implementar una arquitectura por capas por su beneficio modular.

- Capa Inferior: formada por los nodos de procesamiento conectados a una red para poder ser accesibles y gestionables por el servidor.
- Capa Intermedia de integración: comunica la capa superior con la inferior. Gestiona la distribución de los trabajos generados desde la capa superior.
- Capa Superior: contiene el front-end para la interacción con el usuario. En esta capa se encuentra la aplicación de cálculo que se despliega. Cuando la herramienta necesite realizar funciones sobre el cluster realizará llamadas a capas inferiores, lo que se refleja en un aumento en la carga del procesamiento del cluster.

Dada a la diversidad de alternativas para implementar estas capas, con el previo análisis de cada una, se llegó a las conclusiones de diseño:

- En la capa inferior para definir los nodos de procesamiento se decidió realizarlo mediante virtualización para homogeneizar las máquinas a usar. Como ya se ha comentado anteriormente, en cada nodo se instala una máquina virtual y sobre ésta una misma imagen. Se decidió utilizar Oracle VM VirtualBox, un hipervisor de tipo 2, dada su facilidad de instalación y configuración y sus garantías de acceso al hardware físico a través del sistema operativo host. Para el sistema operativo a virtualizar se decidió Ubuntu (Linux) por ser software libre. Ver figura 2.
- Para la capa intermedia, se decidió utilizar un gestor de trabajos PBS Torque que nos permite una gestión de procesamiento transparente. Además es el sistema de colas batch de código abierto líder en el mercado.
- Para la capa superior, se eligió iPython gracias a su entorno completo para la computación interactiva y exploratoria. Además, presenta una arquitectura robusta construida alrededor del concepto de intérprete de Python pero a la vez sencilla y eficaz[2]. También se tuvo en cuenta el interés creciente en esta herramienta (Figura 3).

[10] http://github.com/ipython/ipython/wiki/Extensions-Index#matlab

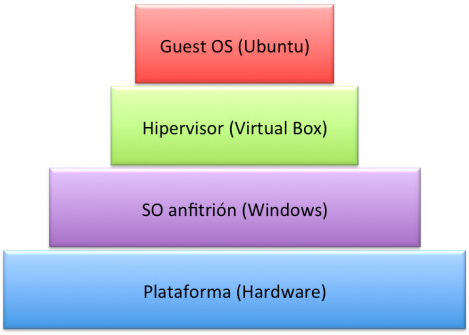


Fig. 2. Arquitectura de capas - Capa inferior

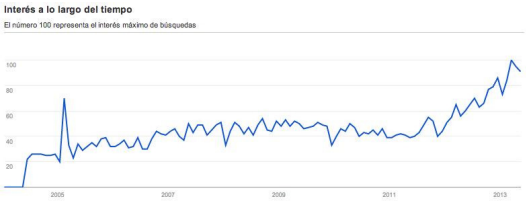


Fig. 3. Evolución del interés en iPython

B. Diseño de bajo nivel

Para la capa inferior fue necesario añadir tareas al SO que inicien la máquina virtual cuando detecte que no se estaba utilizando, la apaguen cuando reconociera un inicio de sesión e informen de su estado al cluster. En las máquinas virtuales se instala una imagen de Ubuntu 10.04 debido a su facilidad de administración. Como capa intermedia se utiliza el gestor de trabajos Torque. Se ha definido una única cola en esta herramienta en la que convergen todas las peticiones de las instancias de iPython. Al utilizar un gestor de trabajos, se simplifica la integración de ambas capas dado que es una interfaz transparente para la capa superior. En otras palabras, iPython solamente ve un único punto donde realizar peticiones/envíos de trabajos, aunque éste se encarga de distribuir los trabajos entre los recursos disponibles según políticas de optimización.
En la capa superior se define un frontend con gestión de usuarios. Para facilitar esta gestión y no guardar información de la cuenta de los usuarios, se utiliza la openID de Google y se evalúa si el usuario autenticado tiene permisos para ejecutar el notebook de iPython. Inicialmente iPython no estaba desarrollado para poder ser utilizado por varios usuarios de manera simultánea bajo un mismo servidor, razón por la cual tuvimos que modificarlo para añadir ésta funcionalidad. Dado que un usuario utiliza la cuenta de Google, tiene la posibilidad de sincronizar sus hojas con Google Drive. La figura 4 ilustra la arquitectura de la aplicación.

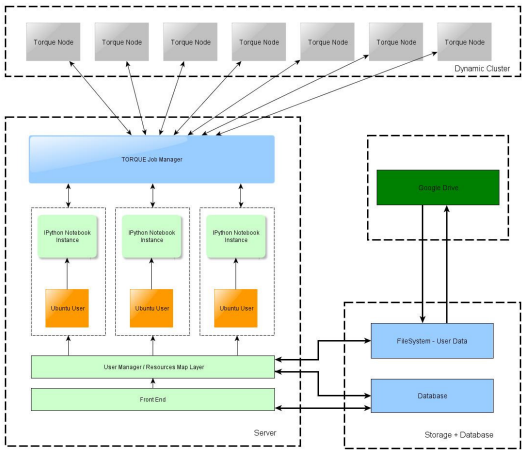


Fig. 4. Arquitectura de la solución.

IV. Estudio de costes e impacto económico

Uno de los objetivos de SmartCloud es lograr optimizar la utilización de la infraestructura existente, lo que se refleja en un ahorro en costes. Actualmente si la universidad necesita realizar cálculos que exceden sus capacidades de procesamiento debe recurrir a un proveedor externo[5]. Al utilizar el cluster dinámico de SmartCloud, se pueden ofrecer los mismos servicios de procesamiento sin tener que contratar a un proveedor externo y sin la latencia que implica el procesamiento remoto. El único sobrecoste significativo que implica esta plataforma, es el coste energético[10]. A continuación se detalla el modelo económico desarrollado para realizar estos cálculos[11].

$$C = W_l * T * F_c * F_u * C_e$$

$$O = \frac{W_l * (T_b + F_c * F_u * T_i) + W_i * (T_i * (1 - F_c * F_u))}{W_l * T_b + W_i * T_i}$$

En donde términos de costes C es el coste energético final y O el sobrecoste energético derivado de utilizar SmartCloud. W_l es el consumo energético de cada nodo con una carga standard en Kw/h y W_i el consumo energético de cada nodo estado inactivo en Kw/h. T se refiere al tiempo disponible para el procesamiento, T_b al Tiempo de utilización del clúster con carga sin utilizar SmartCloud, y T_i al tiempo de utilización del clúster en estado inactivo sin utilizar SmartCloud. Para estimar la carga se usan las variables F_c como factor de carga del clúster en el tiempo disponible, y F_u como factor de utilización de las horas disponibles. El coste de la energía en Kw/h se expresa con C_e

Se plantea un escenario de ejemplo para estudiar el problema. Se dispone de 20 ordenadores i7-3770K con 4GB de RAM y 4 cores con hyper-threading, en un laboratorio con la distribución de uso indicada en la figura 5.

En una semana hay 118 horas inactivas que podrían ser utilizadas por SmartCloud, formando un cluster de 160 cores con 80 GB de RAM.
Supongamos que se necesita utilizar todo el cluster

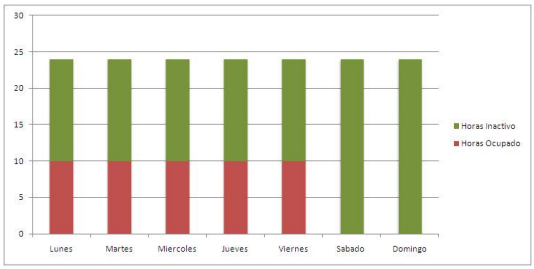


Fig. 5. Distribución de uso de ordenadores en el caso de estudio.

al 100 % de la carga, en el total de las horas disponibles. El consumo energético sería de 196,6€ lo que presenta un sobrecoste de 68 % respecto al consumo de energía actual. Ahora bien, si se recurriera a proveedores externos para realizar este procesamiento, ésto tendría un coste de media de 1508,68€. El cuadro I muestra una comparación de proveedores de servicios cloud que proporcionan un servicio equivalente [11].

Para calcular el ahorro final simplemente basta con restar al coste externo el coste energético,lo que se refleja en 1312€ de ahorro[11]. En términos de latencias, el trabajo dentro de una red local como lo que propone SmartCloud implica latencias sensiblemente menores frente a las que se experimentan trabajando con proveedores cloud. El cuadro II muestra las latencias de Amazon desde España[12].

Claramente son mayores a las latencias dentro de una red local. A su vez, iPython es código abierto, lo que comparado con herramientas como Matlab o Mathematica que tienen costes de licencias representa un ahorro significativo.

TABLA I
Comparación de proveedores Cloud.

Server Type	CPU	RAM (GB)	HDD (GB)	Base Hourly Cost (USD)	Hourly Cost (EUR)	Coste Mensual (EUR)
HP	4	4	120.0	$ 0.14	0,108 €	1.021,597 €
RackSpace	4	4	100.0	$ 0.22	0,168 €	1.583,475 €
SoftLayer	6	7.5	0.0	$ 0.23	0,178 €	1.678,338 €
AWS	4	7.5	850.0	$ 0.24	0,186 €	1.751,309 €

TABLA II
Latencias de Amazon desde España.

Service	Time (sec)	# of samples	Min (ms)	Max (ms)	Std Dev	Median (ms)	Avg (ms)
Amazon EC2	1.13	3	231	298	13.3 %	288	272.33
Amazon Cloud-Front	0.67	6	62	94	15.7 %	78	75.5

[11] http://planforcloud.rightscale.com
[12] http://www.cloudharmony.com/

V. Conclusiones y trabajo futuro

La importancia que tiene la utilización de plataformas cloud se refleja en el ahorro de costes tanto energéticos como en infraestructura tal como se refleja en el trabajo actual.

Uno de los principales aportes de este trabajo es la posibilidad de utilizar iPython en modo multiusuario, ya que actualmente la herramienta no implementa esta funcionalidad.

Al utilizar esta solución, se ofrece a los investigadores no sólo una poderosa herramienta de cálculo sino también una manera de aprovechar los recursos que existen sin costes significativos añadidos. Además, la integración de la herramienta con Google Drive facilita mucho más su usabilidad.

La plataforma resultante presenta una arquitectura de cluster. Una vez implementada, permite el despliegue de otras herramientas y recursos como apoyo a otros proyectos, nuevas aplicaciones, procesamiento batch, entre otros.

Este proyecto puede continuar para seguir explotando la escalabilidad e integración con otros recursos. Se puede dar la posibilidad de integrarlo con otros servicios cloud por ejemplo los de Amazon [13]. El diseño de una nueva solución se puede realizar desde distintos enfoques dado que se adapta a las tendencias actuales de cloud, permitiendo una mayor versatilidad[14].

Para este proyecto se ha desplegado en el cluster una herramienta en concreto, Ipython, pero cualquier aplicación distribuída podría ser desplegada, por ejemplo específicas de un proyecto o de alguna empresa que en determinado momento tenga un pico alto de consumo de recursos.

Una vez logrado el objetivo principal y consolidada la infraestructura, se podría ofrecer estos servicios de cálculo a otras facultades o entidades externas[7].

Como posible ampliación, se puede trabajar en un modelo de cluster híbrido con máquinas virtuales, servidores físicos y balanceadores de carga como alternativa a la actual implementación. También se pueden iniciar otras imágenes sobre las máquinas virtuales para implementar nuevas funcionalidades.

Un estudio en profundidad en lo referente a seguridad permitiría resolver los posibles fallos o vulnerabilidades del sistema, en el actual desarrollo solo se cubren los estándares mínimos necesarios.

Referencias

[1] F. Pérez, *The Ipython notebook: a historical retrospective*, `http://blog.fperez.org/2012/01/ipython-notebook-historical.html`

[2] F. Pérez, B. Granger, M. Ragan-Kelley, *Ipython: a very quick overview*, `http://fperez.org/talks/1010_ipython_overview.pdf`

[3] CIO, *Extremadura preveé ahorrar 30 millones de euros al año con diez proyectos de software libre*, `http://www.ciospain.es/aapp/extremadura-preve-ahorrar-30-millones-de-euros-al-ano-con-diez-proyectos-de-software-libre`

[4] A. Bernardo, *Cuáles son las universidades españolas que apoyan el software libre en 2013?*, `http://alt1040.com/2013/05/universidades-espanolas-que-apoyan-el-software-libre-2`

[5] D. Kondo, B. Javadi, P. Malecot, F. Cappello, D.P. Anderson, *Cost-benefit analysis of Cloud Computing versus desktop grids*, Parallel & Distributed Processing, Addison-Wesley, 2009.

[6] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, A. Ghalsasi, *Cloud computing - The business perspective*, Decision Support Systems, Volume 51, Issue 1, Pages 176-189, April 2011

[7] R. Buyya, C. Shin Yeo, S. Venugopal, J. Broberg, I. Brandic, *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility*, Future Generation Computer Systems, Volume 25, Issue 6, Pages 599-616, June 2009

[8] F. Etro, *The Economic Impact of Cloud Computing on Business Creation, Employment and Output in Europe. An application of the Endogenous Market Structures Approach to a GPT innovation*, Review of Business and Economics, Katholieke Universiteit Leuven, Faculteit Economie en Bedrijfswetenschappen, vol. 0(2), pages 179-208.

[9] N. Sultan, *Cloud computing for education: A new dawn?*, International Journal of Information Management,Volume 30, Issue 2, Pages 109-116, April 2010

[10] A. Berl, E. Gelenbe, M. di Girolamo, G. Giuliani,H. de Meer, M. Quan Dang, K. Pentikousis, *Energy-Efficient Cloud Computing*, The Computer Journal, Number 7, Volume 53, 2010

[11] G. D. Guerrero , R. M. Wallace, J. L. Vázquez-Poletti, J. M. Cecilia, J. M. García, D. Mozos and H. Perez-Sánchez *, A Performance/Cost Model for a CUDA Drug Discovery Application on Physical and Public Cloud Infrastructures*, Concurrency and Computation: Practice and Experience, Special Issue on Distributed, Parallel, and GPU-accelerated Approaches to Computational Biology, 2013, In Press

[13] http://ged.msu.edu/angus/beacon-2012/week1.html

[14] http://aws.amazon.com/es/ec2/

Execution of the P2PSP protocol in parallel environments

Cristobal Medina-López[1], J.A.M. Naranjo[1], Juan Pablo García-Ortiz[1], L. G. Casado[1] and Vicente González-Ruiz[1]

Resumen— **P2PSP is an application layer protocol for the real-time streaming of multimedia content over the Internet, i.e., where the users playback the stream in a synchronized way. It can be used to build a variety of live streaming services that ranges from small hangouts to large IPTV systems. Unlike in the traditional CS (Client/Server) and CDN (Content Delivery Network) based video streaming, peer-to-peer nodes contribute with their upload bandwidth to the system. For this reason, in general, P2P systems are much more scalable than those based on the client/server architecture. This work focuses on actual executions (not simulations) of P2PSP networks. Preliminary results regarding buffering time suggest that the protocol scales well up to several hundreds of peers, mainly due to its simplicity.**

Palabras clave—**Peer-to-peer, multimedia streaming, distributed algorithms.**

I. Introduction

PEER-TO-PEER video streaming technology has been an intensive research field in the last years with many new proposals appearing. Taking into account the non-commercial ones only, we find Overcast [1], PPLive [2], PPStream [3], CoolStreaming [4], ZIGZAG [5], PRIME [6], AnySee [7], HotStreaming [8], PALS [9], SplitStream [10], AnySee [7], Chainsaw [11], Chunkyspread [12], SPPM [13], DirectedPush [14] and TURINstream [15], including the PPSP IETF proposal [16] and several other approaches without a specific name or acronym. Obviously, P2PSP (Peer to Peer Straightforward Protocol) is another proposal to add to this long list of solutions. However, before describing it we highlight some of the features that make it attractive, specially for the open-source community:

1. P2PSP is not aware of the transmitted content, the bit-rate, the format, etc. Any type of multimedia stream can be transmitted without having to modify the protocol at all.
2. At least one working implementation of the P2PSP can be found in Launchpad, at `https://launchpad.net/p2psp`. Anyone can use/modify/expand it for free as long as the GNU GENERAL PUBLIC LICENSE is observed.
3. P2PSP has a layered architecture. The number of layers used depends on the final requirements of the specific instance.
4. The most basic layer (the broadcasting layer) is simple enough to run the peer process in systems with very low computing resources (for instance, running several threads or forking processes is not needed). The rest of layers add functionality to the protocol, such as parallel streaming, system integrity and information privacy. Of course, layers can be modified or new ones can be added to fulfill the requirements, always keeping the interface between them.
5. If native IP multicast is available (even locally, as happens in most of the local networks), P2PSP can use it, having the same performance as IP multicast.
6. Under unannounced peer churn, the P2PSP provides methods for error concealment in the received stream.[1]
7. Peers can be hosted in private networks, even if they are behind symmetric NATs.
8. P2PSP provides video multiresolution (both, spatially and temporally) and bandwidth-adaptive streaming services by using simulcast, scalable video coding and multiple description video coding.
9. Both, P2PSP and CS/CDN models, can be mixed in order to build massive systems.

II. The Peer-To-Peer Straightforward Protocol

A. Layers

P2PSP is organized in four layers, each one incorporating additional functionality. Figure 1 depicts them.

1. **Broadcasting Layer (BL):** This layer implements the most basic behaviour of the protocol and it has been designed to be efficient in transmitting a data-stream from a source node to the peers of the network.
2. **Integrity Layer (IL):** In some contexts, the network needs to overcome hostile peers which could produce the poisoning of the stream, a denial of service, etc. Those peers will be identified and rejected from the P2PSP overlay by using a set of rules defined in this layer.
3. **Multi-channel Layer (ML):** In contexts where there is sufficient bandwidth available, peers can decide to subscribe to more than one stream (channel) in a given time interval. A peer that implements this layer interacts with different, concurrent instances of the Broadcasting Layer.

[1]Dpto. de Informática, Univ. Almería, Agrifood Campus of International Excellence (ceiA3). Corresponding e-mail: `cristobalmedinalopez@gmail.com`.

[1]Unannounced/non-scheduled peer churn produces a loss of data in the received stream that is spread along time. Fortunately, error concealment techniques can be used easily.

4. **Privacy Layer (PL):** Finally, there is a collection of rules ensuring that the transmitted stream is played only by authorized peers. This layer can be useful to implement, for example, Pay-Per-View (PPV) streaming systems.

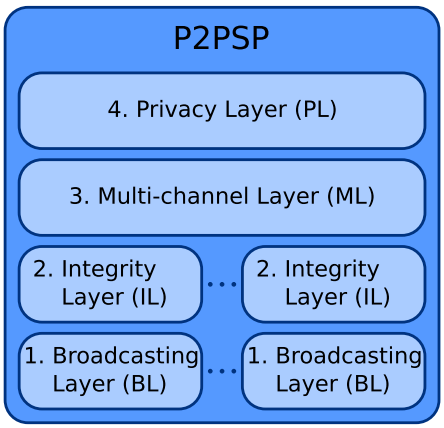


Fig. 1
P2PSP layer diagram.

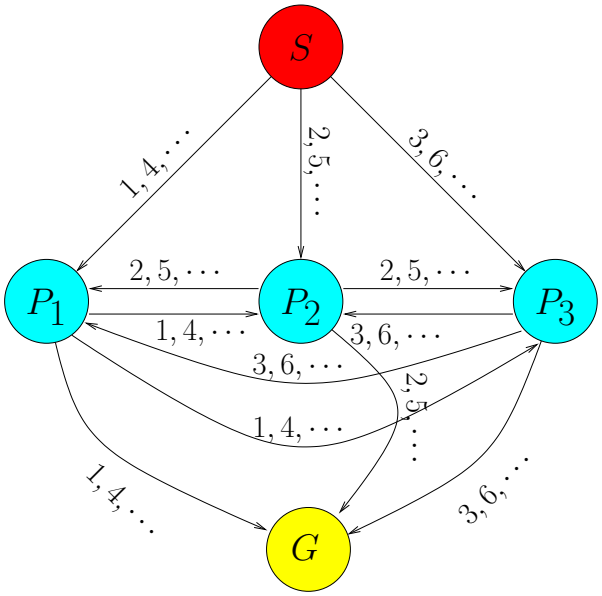


Fig. 2
A P2PSP cluster. Arrows and their labels indicate the transmission of blocks. S sends one different block to one different peer which is selected using a Round Robin model. Peers send each block received from S to each other peer in the cluster.

B. The Broadcasting Layer

This layer defines the algorithms and communication methods governing the entities of a P2PSP network. In order to describe them, it is convenient to make some preliminary definitions.

In the Broadcasting Layer, data can be transmitted in two different states:

1. As a **stream**, i.e., as an endless sequence of data that transports some kind of information. Streams are always transmitted over the TCP.
2. As a collection of **blocks**, being a block a piece of stream. All blocks have the same size. Blocks are always transmitted over the UDP. A small block minimizes the average latency of the transmission but also increments the underlying protocols (UDP/IP/data-link) overhead, and vice versa. In any case, the block size should not exceed the MTU (Maximum Transfer Unit) of the transmission links in order to avoid the overload produced by IP fragmentation.

We also define the following types of entities and groups of entities:

1. **Source (O):** It is the producer of the stream which is transmitted over the P2PSP network. Typically, it is a streaming server using the HTTP protocol such as Icecast [17].
2. **Player (L):** Players request and consume (decode and play) the stream. Usually, several players can retrieve the stream from the source, in parallel.
3. **Splitter (S):** This entity receives the stream from the source, splits it into blocks of the same size and sends the blocks to peers.
4. **Peer (P):** Receives blocks from the splitter and from other peers, ensembles the stream and sends it to a player. Some blocks are also sent to other peers (see below). Notice also that the player and the peer usually run on the same machine.
5. **Gatherer (G):** The gatherer is a special peer that sends blocks neither to other peers nor to the splitter. It only receives a sequence of blocks and (optionally) resends them as a stream to a player. Its function is explained below.
6. **Repeater (R):** Repeaters receive a stream and simply re-transmit it.
7. **Cluster (C):** A cluster is formed by a splitter, a gatherer and zero or more peers (see Figure 2).
8. **Overlay:** An overlay is the union of one or more clusters. These clusters can be connected by repeaters.

B.1 BL rules

1. **Block scheduling:** When the cluster is empty, all blocks are transmitted from the splitter to the gatherer. Otherwise, all blocks are transmitted from the splitter to the peer(s), then among peers and finally to the gatherer (see Figure 2). The splitter sends the n-th block to the peer P_i if
$$(i+n) \bmod |C| = 0, \quad (1)$$
where $|C|$ is the number of peers in the cluster. Next, P_i must forward this block to the rest of peers of the cluster and the gatherer. Blocks received from other peers are not re-transmitted.
2. **Content unawareness:** The P2PSP does not known the nature of the stream. However, at the beginning of the transmission, a stream header

can be transmitted over the TCP in case the player needs it.

3. **The list of peers:** Every node of the cluster (except for the gatherer) knows the endpoint (IP address and port) of the rest of nodes in the cluster. A list is built with this information, which is used by S to send the blocks to the peers. Peers use this list to forward the received blocks to other peers and the gatherer.
4. **IP multicast mode:** If native IP multicast is available, the cluster can be configured to use it. In this case, S uses an IP multicast address and port where all peers of the cluster and the gatherer wait to receive blocks. In this mode, the list of peers maintained by every peer is always empty and stream blocks are not forwarded by them.
5. **Free-riding control in peers:** Each peer assigns a counter to each other peer of the cluster. When a block is sent to a peer, its counter is incremented and when a block is received from it, its counter is reset. If a counter associated to a peer P_i reaches a given threshold, P_i is deleted from the list and it will not be served any more. Notice that the gatherer does not need this mechanism because it does not forward blocks.
6. **Peer arrival:** An incoming peer P_i must contact with the splitter in order to join the cluster. After that, the following steps are carried out:
   (a) The splitter appends P_i to the end of his list.
   (b) The splitter sends to P_i, using the TCP, his list of peers. P_i starts sending to the rest of peers a message (an empty block) in parallel with the reception of the list of peers, in order to introduce P_i to the cluster as soon as possible.
   (c) A peer P_j inserts the peer P_i in his list as soon it receives a message from P_i.
7. **Peer departure:** Peers are required to send a "goodbye" message to S when they leave the cluster, so S can stop sending blocks to them in a natural way. If a peer P_i leaves without notification no more blocks will be received from it. This should trigger the following succession of events:
   (a) The free-riding control mechanism will remove P_i from the list of peers at P_j, $i \neq j$.
   (b) P_j, $i \neq j$, will complain to S about blocks that likely S has sent to P_i, because they have not been received them (the complaint method is explained below).
   (c) S will delete P_i from his list (see rule 11 below).
8. **Buffering:** Blocks in transit can suffer different transmission delays due to jitter[2]. Moreover, they can arrive out of order. For this reason, S numbers every block (see Figure 3) with a 16-bit incrementing positive index. Peers and the gatherer store the received blocks in a buffer.

[2]Variations in network latency.

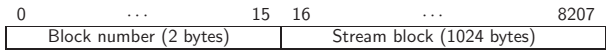


Fig. 3
A BL block. The first field (16 bits) is the block number and the second field, the stream block (in this example is 1024 bytes long).

The size in blocks of the buffer b is the same in each peer.

9. **Relation between buffer size b and block index upper bound m:** Due to practical reasons, there is an upper bound to the block index (preferably a power of two). In order to minimize the probability of receiving two or more blocks with the same number (remember that blocks can be reordered in transit), m must be a multiple of b verifying $m > b$.
10. **Block tracking:** S remembers the block index sent to each peer among the last b blocks. This is necessary to identify unsupportive peers (free-riding). On the other hand, peers can guess a lost block's index by counting buffer slots since the closest received block in the buffer.
11. **Free-riding control in S:** In an ideal scenario, every peer (and also the gatherer) should receive all blocks sent by S. However, in real P2PSP overlays, blocks can be lost due to a number of causes: noise and physical failures in the networking infrastructure, peers leaving the cluster without notification and what it is more likely, peers suffering a temporary reduction of their upload bandwidth [3]. In any of those cases, the number of lost blocks should be kept as small as possible.
    Peers become aware of a block loss at the time of sending it to the player. When a block is missing, peers send to S a message for specifying its index (obviously, G does the same). Using this information, S finds out unsupportive peers. When peer P_i accumulates more than $(|C|)/2$ complaints then P_i is deleted from the list of peers of S and it is not served any other block. This implies recalculating (1).
12. **Blocks re-transmission:** As it was explained in the previous rule, blocks sent to unsupportive peers will be most likely lost and a large part of the cluster will complain about that. S will detect this problem because it will receive a large number of messages with the same (lost) block index. Under this situation, if enough bandwidth is available, S may resend to peers, using again the Round-Robing pattern, the missing blocks just as if they were new. Note that, due to the fact that peers may be slightly asynchronized, some peers could receive these lost blocks on time and therefore would not complain.

[3]The Internet is a shared medium and therefore the load of the network influences communication capabilities of peers.

13. **Congestion control:** If IP multicast is not available, each peer sends blocks using a constant bit-rate strategy to minimize the congestion of its uploading link. The rate of blocks that arrive to a peer is a good metric for performing this control in networks with a reasonable low packet loss ratio. If multicasting is available, peers do not send blocks to the multicast channel (the splitter does all the work).

C. The Integrity Layer

The IL (Integrity Layer) is responsible for fighting against possible custom implementations of peers that might be specifically designed to attack a P2PSP network. More specifically, this layer serves as a barrier against Denial of Service attacks. We identify and neutralize two main attacks that might end in a service interruption situation.

C.1 DoS by starvation, DoS by report

An attacker (or a pool of attackers) might try to induce a denial of service situation if it joins the cluster but does not send any block. If this happened, all nodes of the cluster would complain about the lost blocks after a given period of time, that depends on the buffer size and the bitrate of the stream. Under this situation, the splitter will remove the malicious peer from the cluster and will reject further connections from the same endpoint.

Note that, when adopting such a report-based system, a complementary attack might be developed too: a pool of attackers might report a well-deliberated peer in order to expel it from the network. To prevent this, the splitter should only accept to take actions against reported peers if more than a fixed number of peers have complained (i.e. establishing a report-honesty threshold). If this threshold is set to a high value (for example, 75% of peers in the cluster), then an attacker should need to virtually control the cluster in order to expel any given victim (see Section II-B, point 11).

With the aim to make these types of attacks more difficult, the splitter allows only one connection from the same IP address. Therefore, an attacker that is behind a NAT will have only one shot (if the NAT does not renegotiate its public IP address). Obviously, this provokes that two or more well-meaning peers that are in the same private network can not connect to the same cluster. An efficient solution to this problem is to create a private cluster for all peers behind a given NAT.

C.2 DoS by stream spoiling

Another possible attack consists on the injection of fake information into the cluster. This can be caused by a custom peer sending poisoned blocks [4]. A way of tackling this problem is by inserting (and removing after a given period of time) one o more "trusted-peers". These peers are authenticated as trusted to the splitter, but not to the rest of peers (the behavior of a trusted-peer is identical to any other peer making impossible for a poisonous peer to know these special peers). The source sends to trusted-peers the hash code of randomly chosen blocks using a reliable transmission protocol (note that the poisonous peer is not aware of which blocks were selected). If a poisonous peer sends an altered block to a trusted-peer, it will detect the change using the hash code and will notify the source, which will eject the poisonous peer from the cluster.

[4] A poisoned block is a block that seems to be OK, but which the sender has changed in such a way that when played, introduces no information (for example, a block filled with zeroes) or even wrong information.

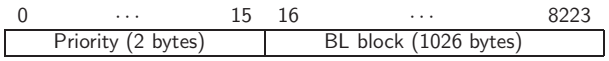


Fig. 4
A ML block is a concatenation of a BL block and a 16-bit priority number.

D. The Multi-channel Layer

P2PSP may broadcast different channels (streams) over distinct (unconnected) clusters. If the user's machine supports multiprocessing then it can run several peers in parallel, having one peer per subscribed channel. To enable this, no extra functionality needs to be added to the BL: the only essential requirement is the network providing sufficient bandwidth.

However, this condition may not be always true. If this happens, one or more (multi-channel) peers could be ejected from a cluster because they are identified as unsupportive peers. In order to minimize this drawback, the ML (Multi-channel Layer) introduces a new encapsulation scheme (see Figure 4) and a new type of node, the Multichannel Scheduler (M).

Peers that implement the ML must label each BL block. This label is a 16-bit positive integer number that represents a priority, being zero the highest one. ML blocks are sent to M which basically implements a FIFO priority queue of blocks. Each time a new ML block is received by M, it sorts them by priority and next, by block number. Thus, if there is not enough bandwidth to transmit all blocks, the user stops receiving those channels that have been assigned a lower priority.

E. The Privacy Layer

The top layer deals with privacy-related issues. Many content providers offer pay-per-view channels as part of their services. From a technical point of view, this implies having a Key Server that ciphers the stream with a symmetric encryption key and delivers such key to authorized members only. However, this is not enough: it is crucial that the Key Server renews the encryption key after the expiration of a peer's authorization period so the stream can not be decrypted any more by the peer (this feature is called *forward secrecy*). In addition, if we want to

play on the safe side then the Key Server should renew the encryption key after a peer purchases an authorization period (if the key remained the same then the peer might decrypt previously captured stream packets for a later playback): this feature is called *backward secrecy*. The associated renewal process is not trivial and is carried out by a *secure multicast protocol*. In order to alleviate the overhead incurred by avalanches of peers entering and leaving the authorized group (for example, at the beginning of a high interest event such as The Olympics) key renewal can be performed on a batch manner, i.e. renewing the key at a given fixed frequency rather than on a per arrival/exit basis. Finally, key renewal messages should be authenticated by means of a digital signature or other alternative methods [18].

Many secure multicast protocols exist in the literature, for example [19], [20], [21], [22] (see [23] for a survey). Here we suggest the implementation of a protocol by Naranjo et al [24]. In it, every authorized peer receives a large prime number from the Key Server at the beginning of its authorization period (this communication is done under a secure channel, for example SSL/TLS). For every renewal, the Key Server generates a message containing the new key to be used by means of algebraic operations: all the authorized primes are involved in this message generation process, and the key can only be extracted from the message by a peer with a valid prime. This protocol is efficient and suits P2PSP architecture in a natural way: every splitter can act as a Key Server for its own cluster. Hence, the stream would be first transmitted among splitters (possibly encrypted with a different key, shared by the splitters). Within each cluster, its corresponding splitter would control the encryption and key renewal process.

III. Experiments

We are currently in the process of testing our aforementioned implementation of P2PSP at Launchpad. The first experiments are being done in one Intel Core i5 CPU 660 @ 3.33GHz × 4 with 4 GB RAM. A single cluster is run, including a splitter, a gatherer and a group of peers, each one being a different thread within the same machine. For now, we are testing the average buffering time of a peer under *flash crowd* conditions (i.e. a large number of peers joining the cluster in a small period of time). By buffering we mean filling half the peer's buffer of stream blocks. Checking this metric for increasing cluster sizes can give an idea of the protocol's scalability. Figure 5 shows the average buffering time for different cluster sizes and different buffer lengths. Buffer lengths are measured in stream blocks, each block being 1024 bytes long. Even though real network latencies do not occur in our experiments, some properties of the protocol can be appreciated. For example, the plot makes clear that peers' buffering time is unaffected by flash crowds and growing audiences, therefore suggesting that our protocol scales well.

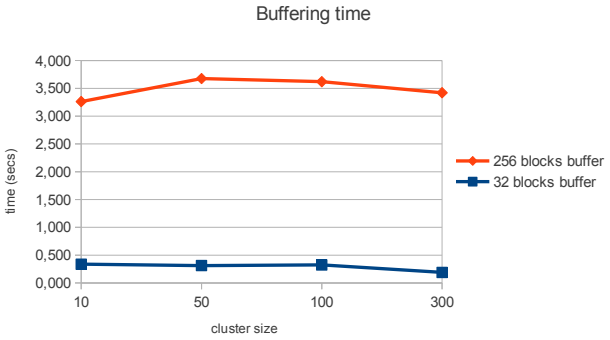


Fig. 5
Buffering time for 32 and 256 blocks buffers.

Our intention is to measure different properties of the protocol, like buffering time, presence of jitter, and quality of service (QoS) under a variety of circumstances (in stable configurations, in flash crowds, with some churn, while splitting and merging clusters, etc). Larger experiments will be made in our computer clusters at the High Performance Computing - Algorithms research group.

IV. Conclusions

Here we presented the P2PSP peer-to-peer streaming protocol. It is composed of a set of roles and rules arranged into four layers, each one with its own different purpose. Undoubtedly, the most important one is the *Broadcast Layer*, since it is at the core of the protocol and defines the rules for the most basic type of communication: one-to-many. The other layers allow to establish many-to-many communications, set different kinds of cluster configuration, provide different types of service (e.g. multilayer encoding for scalable quality, video-on-demand, etc.), assure the integrity of the streamed content and, finally, allow to establish access control rules for privacy-aware scenarios or pay-per-view services. We are in the process of testing the protocol, and show preliminary experimental results regarding the average buffering time of peers with different cluster sizes and buffer lengths. These results suggest that the protocol is scalable. Our goal is to extend our experiments to a high-performance multi-core computer in order to verify the protocol's behaviour in larger clusters.

Acknowledgements

This work has been funded by grants from the Spanish Ministry of Science and Innovation (TIN2008-01117, TIN2012-37483-C03-03, TEC2010-11776-E) and Junta de Andalucía (P10-TIC-6548 and P11-TIC7176), in part financed by the European Regional Development Fund (ERDF).

Referencias

[1] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and Jr.J.W. OŤoole, "Overcast: Reliable multicasting

with an overlay network," in *Operating Systems Design and Implementation*, 2000, pp. 197–212.
[2] "PPLive," `http://www.pplive.com`.
[3] "PPStream," http://www.ppstream.com.
[4] X. Zhang, J. Liu, B. Li, and Y.-S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming," *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 2102–2111 vol. 3, March 2005.
[5] D.A. Tran, K.A. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *IEEE INFOCOM*, April 2003, vol. 2, pp. 1283–1292.
[6] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *INFOCOM*, May 2007, pp. 1415–1423.
[7] X. Liao, H. Jin, Y. Liu, L.M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *INFOCOM*, April 2006, pp. 1–10.
[8] J.C. Wu, K.J. Peng, M.T. Lu, C.K. Lin, and Y.H. Cheng, "Hotstreaming: Enabling scalable and quality iptv services," .
[9] Reza Rejaie and Antonio Ortega, "Pals peer-to-peer adaptive layered streaming," in *International workshop on Network and operating systems support for digital audio and video*, 2003, pp. 153–161.
[10] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative environment," in *Peer-to-Peer Systems II*, February 2003.
[11] Vinay Pai, Kapil Kumar, Karthik Tamilmani, Vinay Sambamurthy, and Alexander E. Mohr, "Chainsaw: eliminating trees from overlay multicast," in *Proceedings of the 4th international conference on Peer-to-Peer Systems*, Berlin, Heidelberg, 2005, IPTPS'05, pp. 127–140, Springer-Verlag.
[12] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast," in *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, Nov., pp. 2–11.
[13] P. Baccichet, Jeonghun Noh, E. Setton, and B. Girod, "Content-aware p2p video streaming with low latency," in *Multimedia and Expo, 2007 IEEE International Conference on*, July, pp. 400–403.
[14] Guanzhong Xu, Yusuo Hu, Yao Shen, and Minyi Guo, "Directedpush - a high performance peer-to-peer live streaming system using network coding," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, Dec., pp. 292–298.
[15] A. Magnetto, R. Gaeta, M. Grangetto, and M. Sereno, "Turinstream: A totally push, robust, and efficient p2p video streaming architecture," *Multimedia, IEEE Transactions on*, vol. 12, no. 8, pp. 901–914, 2012.
[16] IETF, "Peer to peer streaming protocol (ppsp)," http://datatracker.ietf.org/wg/ppsp/charter/.
[17] The Xiph.org Foundation, "Icecast.org," `http://www.icecast.org`.
[18] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and David E. Culler, "SPINS: security protocols for sensor networks," *Wirel. Netw.*, vol. 8, pp. 521–534, 2002.
[19] Lihao Xu and Cheng Huang, "Computation-efficient multicast key distribution," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 5, pp. 577–587, May 2008.
[20] J. Lin, K. Huang, F. Lai, and H. Lee, "Secure and efficient group key management with shared key derivation," *Comput. Stand. Inter.*, vol. 31, no. 1, pp. 192 – 208, 2009.
[21] Z. Zhou and D. Huang, "An optimal key distribution scheme for secure multicast group communication," in *INFOCOM'10*, 2010, pp. 331–335.
[22] Eun-Jun Yoon and Kee-Young Yoo, "A secure broadcasting cryptosystem and its application to grid computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 620 – 626, 2011.
[23] J.A.M. Naranjo and L.G. Casado, "An updated view on centralized secure group communications," *Logic Journal of IGPL*, 2012.
[24] J. A. M. Naranjo, L. G. Casado, and J. A. López-Ramos, "Group oriented renewal of secrets and its application to secure multicast," *Journal of Information Science and Engineering*, vol. 27, no. 4, pp. 1303–1313, july 2011.

Eficiencia Energética

Estudio de la Eficiencia Energética en Big-Data Clouds Basados en Hadoop

Javier Conejero, Carmen Carrión y Blanca Caminero [1]

Resumen— **La eficiencia energética es una de las principales razones por la que empresas, universidades y centros de investigación están migrando a la computación Cloud. Es posible conseguir una mayor eficiencia energética (comparado con un despliegue local) gracias al despliegue centralizado de un Cloud en un centro de datos. Además, la tendencia creciente del precio de la energía hace que los costes operacionales de explotación de los entornos Cloud sean cada vez mayores. En este trabajo, investigamos y analizamos el consumo energético de un conjunto de máquinas virtuales haciendo uso de Hadoop, sobre un Cloud basado en OpenNebula. La carga de trabajo utilizada una aplicación que analiza los sentimientos de mensajes asociados a Tweets. Nuestro objetivo es comprender la relación entre eficiencia energética y rendimiento para este tipo de cargas de trabajo.**

Palabras clave—**Computación Cloud, Consumo energético, Hadoop, OpenNebula, Big-Data.**

I. Introducción

UNA gran cantidad de compañías (de diferente tamaño y experiencia) están adoptando o migrando a la tecnología Cloud en sus procesos de negocio, principalmente motivados por la reducción de costes por despliegue y gestión de la infraestructura de computación. Al mismo tiempo, las preocupaciones por el medio ambiente (principalmente en centros de cálculo) han motivado la necesidad de optimizar la eficiencia energética de sus infraestructuras de cómputo y considerar fuentes de energía límpias, como la solar o la eólica. Estos hechos provocan importantes retos, sobre todo en cuanto al entendimiento de cómo proveer servicios Cloud energéticamente eficientes a los usuarios finales.

Con la creciente subcontratación de servicios o capacidades computacionales, en Cloud surge la necesidad de especificar acuerdos de nivel de servicio (*Service Level Agreements* (SLAs)). Estos SLAs suelen incluir soporte para negociar el pago por el uso de los servicios ofrecidos. Hoy en dia es un reto determinar cómo y qué parámetros especificar en los SLAs, para posteriormente monitorizar su conformidad en entornos Cloud, donde es difícil garantizar el rendimiento (debido al uso de la virtualización y al mapeo variable entre recursos físicos y virtuales). La demanda para incluir métricas o parámetros de eficiencia energética en los SLAs crece cada vez más, de forma que tanto cliente como proveedor puedan negociar sus requisitos teniendo en cuenta las preocupaciones medioambientales.

Consecuentemente, hay un interés creciente en fomentar la sostenibilidad de la computación Cloud [1]. Y por lo tanto, es necesario mejorar las técnicas de eficiencia energética en todos los niveles de los centros de cálculo (desde el balanceo de carga de trabajo entre recursos hasta los sistemas de refrigeración).

Así pues, la clave para diseñar un centro de cálculo respetuoso con el medio ambiente es comprender el comportamiento de todos los sistemas que conforman la infraestructura Cloud, desde el hardware desplegado hasta las políticas de explotación de cada recursos. Esto es realmente dificil dada la complejidad de estos sistemas.

Hay también un gran interés en la utilización de las infraestructuras Cloud para realizar análisis sobre ingentes cantidades de datos (conocidos como "big–data") haciendo uso de aplicaciones como Hadoop. Hadoop [2], [3] es una herramienta para la ejecución de aplicaciones sobre "big–data" en grandes clusters haciendo uso del paradigma Map/Reduce [4]. Se ha convertido una herramienta muy popular para abordar los problemas de escalabilidad del análisis a traves de grandes cantidades de datos, las cuales no pueden ser procesadas por paradigmas o tecnologías tradicionales. Además, se ha convertido en una herramienta importante para la evaluación de rendimiento (*benchmark* [5]) de sistemas distribuidos debido a sus altos requisitos de almacenamiento, cómputo y red.

Comprender cómo desplegar Hadoop de forma eficiente a través de un entorno Cloud sigue siendo un reto importante, debido fundamentalmente a que los parámetros de la infraestructura Cloud y configuraciones de clusters virtuales son muy variados y pueden influir en el rendimiento de Hadoop, así como en el uso de los recursos. La carencia de modelos de comportamiento para la gestión de los recursos proporciona la oportunidad de considerar varias técnicas de optimizacion del uso de recursos asociados al Cloud. El objetivo de este trabajo es abordar este reto, determinando cómo se puede utilizar Hadoop sobre un entorno Cloud, así como realizar la monitorización del consumo energético para optimizarlo.

La principal motivación de este artículo es comprender el impacto de la computación de alto rendimiento (como Hadoop) en el consumo energético en entornos Cloud, y validarlo a través de un ejemplo de procesamiento de "big–data". Nuestro propósito en este trabajo es describir cómo el consumo energético: (i) puede ser monitorizado, comprendido (prestando especial atención en la variabilidad experimentada a través de múltiples ejecuciones de la carga de trabajo) y expuesto al usuario; (ii)

[1]Instituto de Investigación en Informática de Albacete (I^{3}A), Universidad de Castilla-La Mancha, e-mail: {Francisco.Conejero, Carmen.Carrion, MariaBlanca.Caminero}@uclm.es

puede estar relacionado con el número de máquinas virtuales (MVs) y la carga de trabajo, y (iii) obtener un modelo de comportamiento del mismo.

El presente artículo se estructura en cinco secciones. En la Sección II se muestra el trabajo relacionado. A continuación, en la Sección III se describe la infraestructura, carga de trabajo e instrumentación utilizadas. En la Sección IV se presentan los resultados obtenidos sobre el comportamiento, y en la Sección V se sintetiza el modelo de comportamiento observado. Finalmente, las conclusiones y trabajo futuro se presentan en la Sección VI.

II. Trabajo Relacionado

El consumo energético en entornos de computación es considerado un tema crucial hoy en día, y especialmente en computación Cloud, donde se han realizado esfuerzos significativos [6].

Compañías como APC (*Schneider electric*) [7] y VMware [8] han diseñado modelos de consumo energético estáticos para ayudar a los clientes a determinar el consumo energético de una determinada computadora (dependiendo de sus componentes internos) o evaluar las necesidades para un Sistema de Alimentación Ininterrumpida (SAI). Este tipo de propuestas se basan en bases de datos de consumo de componentes computacionales y carecen de información relacionada con la carga de trabajo y su comportamiento.

También hay varias propuestas que sugieren el desarrollo de una arquitectura Cloud [1], [9] para proveer y utilizar mecanismos de ahorro energético (tales como mejoras en la planificación o balanceo de carga en función del consumo monitorizado), garantizando el rendimiento desde el punto de vista del usuario.

Otras propuestas van más allá e intentan distribuir la carga de trabajo entre centros de trabajo geográficamente dispersos [10], de forma que se exploten diferentes fuentes de energía renovables dependiendo de las condiciones meteorológicas.

También hay trabajos centrados en la eficiencia energética de Hadoop en clusters. En [11], los autores resumen los principales problemas e ineficiencias inherentes al paradigma Map/Reduce, mientras que en [12] y [13] se proponen mecanismos para el ahorro energético en el sistema de ficheros utilizado en Hadoop (HDFS) y en su planificador de trabajos, conocidos como GreenHDFS y GreenHadoop respectivamente.

En [14], los autores exploran el impacto de los clusters haciendo uso de Hadoop en términos térmicos. Proponen un planificador para Hadoop centrado en la minimización de la disipación de calor mediante el balanceo de la carga entre clusters de forma que se mantengan por debajo de un umbral térmico definido. La identificación de un umbral de operación adecuado sigue siendo un gran reto.

También hay especial interés en la especificación de métricas sobre consumo energético en los SLAs. Por ejemplo, en [15] se identifican diversos parámetros que podrían ser utilizados en un SLA.

Una métrica ampliamente utilizada para medir la eficiencia energética es *Power Usage Effectiveness* (PUE), desarrollada por la asociación *Green Grid* [16]. El PUE representa el ratio de la cantidad de energía introducida en un centro de datos dividida entre la potencia utilizada para hacer funcionar la infraestructura, siendo 1 el valor ideal. Particularmente, el trabajo presentado en este artículo es mucho más detallado que calcular el PUE, dado que nuestra propuesta pretende determinar cómo el consumo energético puede estar asociado con una configuración de MVs y carga de trabajo particular. El resultado de este trabajo puede ser utilizado posteriormente para realizar un análisis de un PUE diferente teniendo en cuenta una carga de trabajo concreta.

III. Entorno Cloud

El objetivo de nuestro trabajo es caracterizar el consumo energético de Hadoop sobre un entorno IaaS (*Infrastructure as a Service*) Cloud. En la presente sección se describe la infraestructura sobre la que se ha realizado la validación. También se describen la carga de trabajo y la instrumentación utilizada.

A. Infraestructura Cloud

La infraestructura Cloud utilizada en este trabajo (Figura 1) está compuesta por un cluster de un único nodo de cómputo. Dicho nodo es un Viglen ix4600 y dispone de 2 procesadores Xeon e5620 (4 Cores + hyperthreading cada uno) [17], 24 GB de memoria principal y 4 TB de almacenamiento. El sistema operativo instalado es Linux CentOS 6.2 [18]. Para la gestión y coordinación del entorno Cloud, disponemos de OpenNebula [19]. OpenNebula es una aplicación abierta, flexible y extensible para la constitución de infraestructuras Cloud. Es capaz de proveer infraestructura como servicio (*Infrastructure as a Service* (IaaS)) mediante la gestión de diferentes sistemas de virtualización (como KVM o XEN, entre otros). Concretamente, hacemos uso de KVM [20], el cual proporciona virtualización completa (con soporte para extensiones de virtualización de hardware) en Linux y está soportado por RedHat. La decisión de utilizar un entorno privado es permitirnos un control absoluto sobre las tareas de medición de consumo energético.

En este entorno Cloud, desplegamos una aplicación para el analisis de datos sociales utilizando Hadoop a través de clusteres formados por máquinas virtuales, descrito en la siguiente Subsección.

B. Carga de Trabajo

Los medios (o redes) sociales pueden incluir una gran variedad de contenidos diferentes, como video (YouTube), sonido (Spotify), imágenes (Facebook, Flickr) o incluso texto (Twitter, Facebook). En este trabajo, hacemos uso de un analizador de textos a través de Hadoop [2], [3]. Hadoop implementa el paradigma Map/Reduce [4], en el cual, la in-

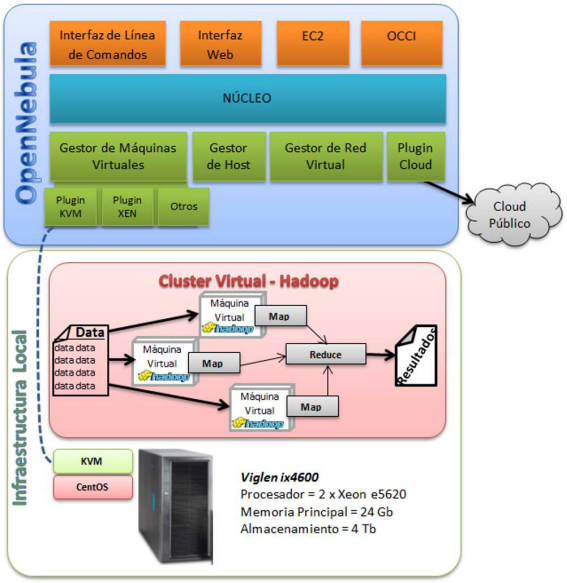


Fig. 1. Infraestructura Cloud.

formación de entrada es dividida en bloques y distribuida a través de multiples recursos de cómputo (fase *Map*), y los resultados son combinados en la siguiente fase (*Reduce*). Hadoop también provee transferencia transparente de datos, mecanismos de tolerancia a fallos (mediante replicación), y un sistema de ficheros distribuido para almacenar la información a través de los nodos de cómputo (HDFS). El despliegue de Hadoop requiere un cluster (Figura 1), el cual debe estar formado por un nodo principal (*master*) y *1...n* nodos de cómputo (*workers*). Por lo tanto, para desplegar Hadoop en un entorno Cloud, es necesario desplegar múltiples MVs y crear un cluster virtual para Hadoop siguiendo la estructura definida. La virtualización nos permite configurar el tamaño y las características de cada MV. Además, provee elasticidad, portabilidad y la capacidad para reemplazar el hardware subyacente de forma dinámica. Por contra, introduce una sobrecarga computacional.

El proyecto COSMOS (*Cardiff Social Media Observatory*) [21] ofrece soporte a científicos del área de las ciencias sociales para el análisis de información socialmente significativa (por ejemplo, tweets, blogs y noticias). El volumen de datos producido diariamente por la sociedad requiere una cantidad significativa de recursos computacionales. Por ejemplo, COSMOS recoge sobre 3.5 millones de tweets diariamente (∼1% del total). Para realizar un análisis longitudinal de una opinión pública o sentimiento, o sobre un evento social significativo (como una campaña política, cambio de legislación, evento deportivo de impacto, etc.) puede ser necesario recopilar datos durante varias semanas, lo cual representa un gran volumen de datos. Realizar este analisis en un ordenador común es inviable. En consecuencia, es necesario optimizar el analizador para mejorar o conseguir tiempos aceptables de respuesta. Para esto, se ha desarrollado una aplicación capaz de ser ejecutada mediante Hadoop para escalar los análisis y extraer el sentimiento de cada tweet. Concretamente, este proceso hace uso de la herramienta *SentiStrength* [22]. Se trata de un proceso muy pesado computacionalmente.

Esta aplicación representa una carga realista para estresar la infraestructura Cloud definida en la Sección III-A utilizando diferentes configuraciones para clusters virtuales.

Para las pruebas realizadas en este trabajo, se ha utilizado un volumen de datos correspondiente a 15 millones de tweets, el cual representa más de 2GB de datos en ficheros de texto plano.

C. Instrumentación y Monitorización

En este trabajo, nos centramos en el consumo energético del nodo de cómputo completo. Para obtener su consumo energético es necesario disponer de un dispositivo de monitorización externo (esta métrica no se puede obtener mediante software). Hay múltiples productos comerciales para esta finalidad y los más utilizados son Kill-A-Watt[1] y WattsUp[2].

A nuestra disposición tenemos el medidor WattsUp PRO para monitorizar y almacenar toda la información relacionada con el consumo energético. Debe estar situado entre el enchufe y la fuente de alimentación del nodo de cómputo (Figura 2), y por lo tanto, es necesario disponer de un segundo ordenador para obtener la información de monitorización almacenada en la memoria del WattsUp PRO. La razón por la que elegimos WattsUp PRO en vez de Kill-A-Watt es porque éste último carece de capacidad de almacenamiento (necesario para largos períodos de monitorización).

La frecuencia de monitorización ha sido establecida en una muestra por segundo para todos los experimentos.

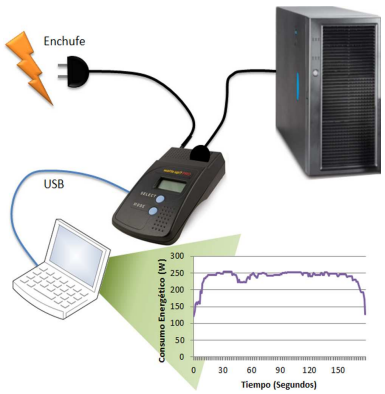


Fig. 2. Monitorización del consumo energético.

IV. Comportamiento del Sistema

Para comprender el comportamiento del consumo energético, diseñamos varios experimentos, cuyos resultados se muestran en las siguientes subsecciones. Nuestra propuesta está formada por tres partes:

1. *Consumo Energético Básico*: Se monitoriza el consumo energético del funcionamiento del cluster. Esto incluye el arranque y apagado del

[1]http://www.p3international.com/products/special/p4400/p4400-ce.html
[2]https://www.wattsupmeters.com/secure/index.php

sistema, así como el mantenimiento en funcionamiento sin carga de trabajo.
2. *Rango de Consumo Energético*: Esta etapa está centrada en obtener el rango de consumo energético, es decir, el consumo máximo y mínimo.
3. *Consumo Energético con Carga de Trabajo*: Esta etapa requiere la monitorización con diferentes cargas de trabajo.

Un desarrollo más completo y detallado se puede encontrar en [23].

A. Consumo Energético Básico

Esta etapa se centra en el análisis de la demanda energética del sistema descrito en la Sección III-A sin ningún tipo de carga de trabajo. El perfil de consumo energético, ilustrado en la Figura 3, se comporta según lo esperado: particularmente alto durante el arranque y apagado, y estable una vez ha arrancado (y estabilizado). Hay un incremento notable de consumo durante el arranque, que se estabiliza justo después de terminar de cargar el sistema operativo, y entonces el sistema alcanza un nivel estable (∼105 Vatios (W)). El apagado representa una carga a tener en cuenta dada la necesidad de detener todos los servicios del sistema operativo para un apagado controlado.

Además, mientras el servidor está apagado se observa un consumo de ∼10 W, que se corresponde al estado *standby*. Puede parecer un consumo muy bajo para un único servidor, pero debe tenerse en cuenta en grandes infraestructuras, dado que se incrementa linealmente con el número de servidores.

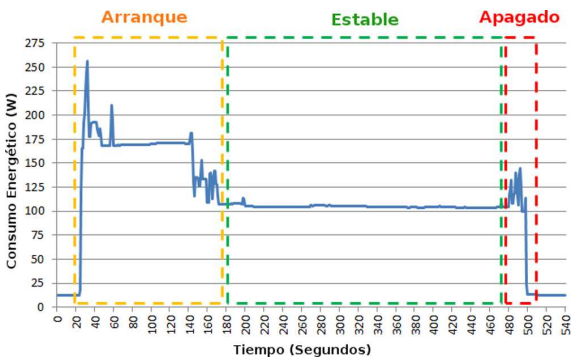


Fig. 3. Resultados de Simulación.

B. Rango de Consumo Energético

En esta etapa, medimos el rango de consumo energético del servidor. El consumo del servidor durante la fase estable determina el consumo mínimo requerido por el servidor (estando disponible y preparado para alojar MVs). Pero para obtener el nivel máximo de consumo es necesaria una carga de trabajo capaz de explotar los recursos del servidor al máximo. El algoritmo criptográfico MD5 (*The MD5 Message-Digest Algorithm*) [24] destaca en esta tarea, fundamentalmente ejecutado en múltiples hilos de forma simultánea.

Los resultados obtenidos por la ejecución de múltiples hilos MD5 muestran que el uso de CPU se incrementa conforme se aumenta el número de hilos, alcanzando la utilización máxima con 16 hilos, y manteniendo la utilización al 100% para valores mayores. A través de este experimento hemos encontrado el consumo energético máximo del servidor, establecido en 268 W.

Una vez medidos los límites máximo y mínimo de consumo, así como el coste de arranque y apagado del servidor, procedemos a medir el consumo bajo la carga de trabajo definida en la Sección III-B.

C. Consumo Energético con Carga de Trabajo

Esta etapa se centra en el análisis del comportamiento del sistema bajo una carga de trabajo realista basada en Hadoop, en diferentes clusters virtuales. En este escenario, se modifica el número de *workers* y sus caracteristicas, mientras que se mantiene constante la cantidad de recursos asignados para todos los clusters virtuales. La evaluación se realiza con 4 configuraciones diferentes (Tabla I).

TABLA I
CONFIGURACIONES DE LOS CLUSTERS VIRTUALES

Conf. MV	# MVs	CPU	RAM (GB)	HDD (GB)
Master	1	20%	6	100
1 Worker	1	70%	14	200
2 Workers	2	35%	7	100
4 Workers	4	17,5%	3,5	50
8 Workers	8	8,75%	1,75	25

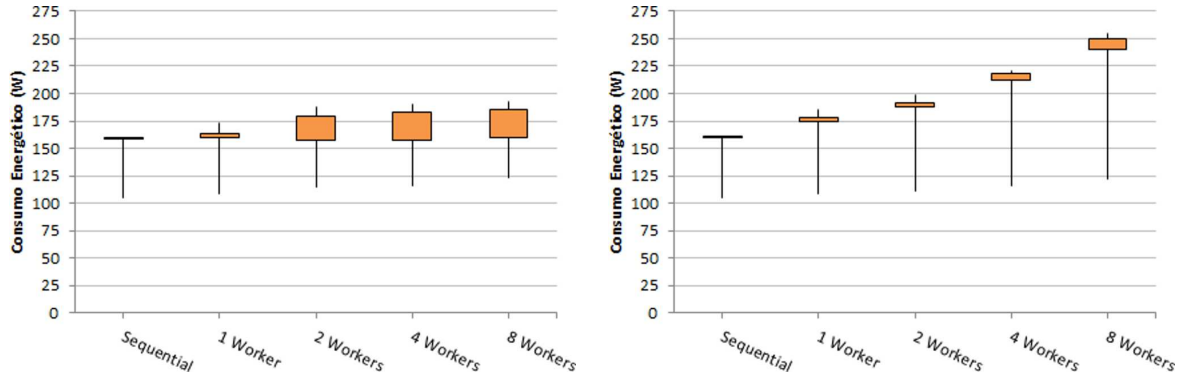
Estas configuraciones de clusters virtuales están diseñadas para maximizar el uso de la infraestructura Cloud, reservando un 10% de CPU y 4 GB de memoria principal para el sistema operativo y herramientas de gestión. Cada configuración es evaluada independientemente de las demás. Todas las MVs que forman los clusters virtuales disponen de Ubuntu 10.04.

Cualquier cluster virtual desplegado añade una sobrecarga al servidor. Es decir, para configuraciones con más máquinas virtuales, se observa un aumento en el consumo energético, incluso sin estar procesando ningún tipo de carga de trabajo (Figura 4). Los resultados obtenidos muestran el máximo y mínimo consumo experimentado (líneas verticales) y el 90% de los valores observados (caja). Por lo tanto, el tamaño del cluster virtual también tiene un impacto en el consumo energético, incluso sin procesar aplicaciones para Hadoop.

A continuación, se ejecuta la carga de trabajo sobre los clusters virtuales (Figura 5).

Se observa que se alcanza el máximo consumo energético con 8 MVs. Pero se observa una correlación entre el tamaño del cluster virtual y el consumo energético.

El tamaño del problema (cantidad de tweets), y en consecuencia la longitud de las ejecuciones, reducen la variabilidad en el consumo (evolución entre la Figura 5(a) y la Figura 5(b)). Esto facilita la predicción de consumo energético.



a) 1.000.000 Tweets b) 15.000.000 Tweets

Fig. 5. Consumo Energético del Análisis de Sentimientos mediante Hadoop en el Cloud.

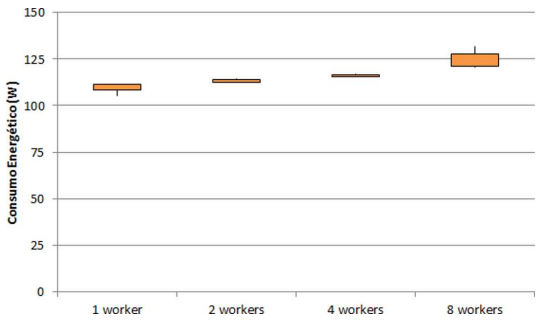


Fig. 4. Consumo de los Clusters Virtuales sin Carga.

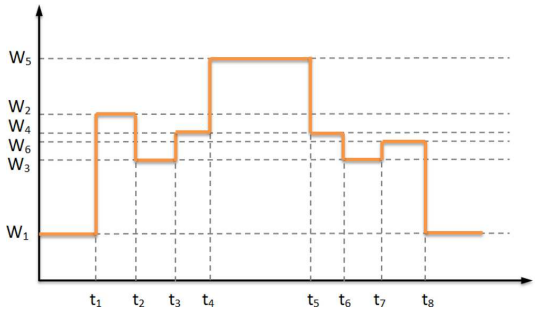


Fig. 6. Perfil de Consumo Energético.

Más específicamente, la variabilidad observada se comporta de dos maneras diferentes dependiendo de la cantidad de tweets o del número de MVs. Repitiendo el experimento para la misma cantidad de tweets, pero incrementando el n° de MVs, se observa que la variabilidad aumenta en todos los casos. Por otro lado, para la misma cantidad de MVs, el incremento de tweets resulta en una reducción en la variabilidad.

V. Caracterización del Consumo Energético

Sintetizando los resultados obtenidos en la Sección IV, podemos identificar un perfil de consumo energético (Figura 6). Se observan seis niveles de consumo claramente definidos y diferenciados. Estos niveles están condicionados por las características del sistema utilizado, así como por el sistema de virtualización elegidos. Dado que puede haber variaciones en el consumo energético para cada nivel, consideramos un valor medio para representar gráficamente el modelo.

El primero de los niveles observados (W_1) se corresponde con la máquina física conectada a la red eléctrica pero apagada. Se conoce como modo *Standby* y se observa un consumo en este estado. Aunque puede parecer pequeño comparado con el consumo durante el funcionamiento, es importante tenerlo en cuenta. A continuación, se observa un gran incremento en el consumo durante el arranque de la máquina física (W_2), pero está temporalmente acotado (y puede ser reducido haciendo uso de discos duros de almacenamiento sólido (SSDs) y memoria principal más rápida). Una vez ha terminado de arrancar y todos los servicios del sistema operativo están cargados, se alcanza el nivel *preparado* (W_3), en el cual la máquina está preparada para alojar MVs.

Cuando se despliegan las MVs, se identifican dos niveles de consumo: el primero como consecuencia del despliegue y estado ocioso de las MVs (W_4) (incrementando el consumo debido al aumento de procesos), y el segundo como consecuencia de las ejecuciones de la carga de trabajo mediante Hadoop (W_5). Como ha sido observado en los experimentos realizados en este trabajo, Hadoop estresa el sistema, y en consecuencia, el consumo energético aumenta significativamente (alcanzando incluso el máximo).

La finalización de la carga de trabajo hace que el sistema vuelva al estado W_4. El sistema se encuentra preparado para procesar más trabajos. Si las MVs se detienen, el sistema vuelve al estado ocioso (W_3) y el sistema puede alojar otras MVs (u otro cluster virtual) o ser detenido.

Finalmente, la máquina física puede ser detenida y vuelve al modo *Standby* (W_1), y como consecuencia de la necesidad de detener los servicios asociados al sistema operativo, se observa un incremento en el consumo (W_6), aunque al igual que el arranque también está acotado temporalmente.

La metodología propuesta y utilizada en este artículo puede ser extrapolada a otras infraestructuras Cloud, pero la caracterización de la energía (así como los niveles identificados) no puede ser generalizada para ellas. Sin embargo, observamos la misma tendencia cuando se despliega OpenNebula, KVM y Hadoop en otras máquinas físicas.

La información que este modelo provee puede ser utilizada para seleccionar y posteriormente optimizar las políticas de despliegue. Es decir, decidir cuál es el cluster virtual óptimo para satisfacer las necesidades de cada caso, teniendo en cuenta el consumo energético requerido. Además, ofrece una información importante para decidir cuándo detener la máquina física (dependiendo de restricciones temporales y requisitos de los usuarios) y estimar el beneficio entre el rendimiento y el correspondiente ahorro energético.

La caracterización expuesta en este trabajo permite tomar estas decisiones para diferentes cargas de trabajo basadas en Hadoop.

VI. Conclusiones y Trabajo Futuro

El objetivo de este trabajo es caracterizar el consumo energético de aplicaciones "big–data" mediante Hadoop en entornos Cloud. Y el resultado final obtenido ha sido la obtención de un modelo de consumo energético.

La principal conclusión obtenida en este estudio es que hay una relación no lineal entre el número de MVs, la carga de trabajo, y el consumo energético observado en la máquina física. Los resultados muestran la posibilidad de identificar cuántas MVs son necesarias para obtener un consumo determinado mediante el modelo obtenido. En consecuencia, desplegar y utilizar 8 o más MVs en la misma máquina física requiere el máximo consumo energético para la infraestructura Cloud particular sobre la que hemos realizado nuestro desarrollo, pero consigue el mejor rendimiento. Esta infraestructura hace uso de OpenNebula, Hadoop y KVM. La metodología utilizada para analizar y comparar el consumo energético (utilizando las tres etapas descritas en la Sección IV) puede ser adaptada para otras aplicaciones, o incluso otros entornos Cloud.

Además, hemos analizado la variabilidad en el consumo energético sobre múltiples ejecuciones de la misma carga de trabajo. Esta variablidad se ve reducida conforme la longitud de las ejecuciones aumenta. Así que, para trabajos cortos, la utilización de métricas de consumo puede ser restrictiva, incluso en Clouds privados. Sin embargo, la aproximación que proponemos en este trabajo puede ser utilizada para incluir métricas tales como consumo energético en los acuerdos de nivel de servicio, junto con métricas de rendimiento tradicionales.

Futuras líneas de trabajo son: (i) realizar un estudio similar sobre una infraestructura Cloud distribuida; (ii) analizar el mapeo entre métricas de consumo en acuerdos de nivel de servicio y el modelo propuesto; (iii) diseñar e implementar políticas de gestión eficiente para entornos Cloud a partir del modelo de consumo energético.

Agradecimientos

Este trabajo ha sido soportado por MINECO y la Comisión Europea (fondos FEDER) a través del proyecto TIN2012-38341-C04-04, así como a través de una beca FPI asociada al proyecto "TIN2009-14475-C04-03".

Referencias

[1] Saurabh Garg and Rajkumar Buyya, *Green Cloud Computing and Environmental Sustainability, Harnessing Green IT: Principles and Practices*, Wiley Press, UK, Noviembre 2012.

[2] Chuck Lam, *Hadoop in Action*, Manning Publications, Diciembre 2010.

[3] Tom White, *Hadoop: The Definitive Guide*, O'Reilly, Junio 2009.

[4] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] "CloudSuite 1.0," Página web: `http://parsa.epfl.ch/cloudsuite/cloudsuite.html`, Último acceso: 18 de Abril, 2013.

[6] Daman Dev Sood and Shiv Kumar, "Cloud Computing & Green IT," Tech. Rep., 2010.

[7] APC, "UPS Selector Sizing Application," Página web: `http://www.apc.com/template/size/apc/`, Último acceso: 22 de Marzo, 2013.

[8] VMWare, "Green IT Calculator," Página web: `http://www.vmware.com/solutions/green/calculator.html`, Último acceso: 22 Marzo, 2013.

[9] Liang Liu et al., "Greencloud: a new architecture for green data center," in *Proc. of the 6th international conference industry session on Autonomic computing and communications industry session*, New York, NY, USA, 2009, ICAC-INDST '09, pp. 29–38, ACM.

[10] Mahdi Ghamkhari and Hamed Mohsenian-Rad, "Optimal Integration of Renewable Energy Resources in Data Centers with Behind-the-Meter Renewable Generator," in *Proc. of the IEEE International Conference in Communications (ICC'2012)*, Ottawa, Canada, June 2012.

[11] Jacob Leverich and Christos Kozyrakis, "On the energy (in)efficiency of hadoop clusters," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 61–65, Mar. 2010.

[12] Íñigo Goiri et al., "Greenhadoop: leveraging green energy in data-processing frameworks," in *Proc. of the 7th ACM european conference on Computer Systems*, New York, NY, USA, 2012, EuroSys '12, pp. 57–70, ACM.

[13] Rini T. Kaushik and Milind Bhandarkar, "Greenhdfs: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster," in *Proc. of the 2010 international conference on Power aware computing and systems*, Berkeley, CA, USA, 2010, HotPower'10, pp. 1–9, USENIX Association.

[14] Bing Shi and Ankur Srivastava, "Thermal and power-aware task scheduling for hadoop based storage centric datacenters," in *Proc. of the International Conference on Green Computing*, Washington, DC, USA, 2010, GREENCOMP '10, pp. 73–83, IEEE Computer Society.

[15] Gregor Laszewski and Lizhe Wang, "GreenIT Service Level Agreements," in *Grids and Service-Oriented Architectures for Service Level Agreements*, pp. 77–88. Springer US, 2010.

[16] "Green Grid Association," Página web: `http://www.thegreengrid.org/`, Último acceso: 30 de Marzo, 2013.

[17] Intel, "Xeon Processor e5 Family," Página web: `http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-5000-sequence.html`, Último acceso: 12 de Marzo, 2013.

[18] "CentOS," Página web: `http://www.centos.org/`, Último acceso: 13 de Febrero, 2013.

[19] "OpenNebula," Página web: `http://opennebula.org/`, Último acceso: 13 de Febrero, 2013.

[20] "KVM," Página web: `http://www.linux-kvm.org/`, Último acceso: 13 de Febrero, 2013.

[21] COSMOS, "Cardiff On-line Social Media Observatory," Página web: `http://www.cs.cf.ac.uk/cosmos/`, Último acceso: 16 de Abril, 2013.

[22] "SentiStrength," Página web: `http://sentistrength.wlv.ac.uk/`, Último acceso: 16 de Abril, 2013.

[23] Javier Conejero, Omer Rana, Pete Burnap, Jeffrey Morgan, Carmen Carrión, and Blanca Caminero, "Characterising the Power Consumption of Hadoop Clouds: A Social Media Analysis Case Study," in *Proc. of the 2013 International Conference and Cloud Computing and Services Science (CLOSER 2013)*, Aquisgán, Alemania, Mayo 2013.

[24] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321 (Informacional), 1992, Actualizado por RFC 6151.

Effective On-Chip Traffic Compression in Embedded Systems

María Soler[1] and José Flich[2]

Abstract— **As technology advances, multiprocessor systems-on-chip (MPSoCs) increase with the number of components, relying on an efficient on-chip network (NoC). As the size of the system increases, NoC performance and power consumption become a central issue.**

In this paper we design compression strategies at the NoC level reducing the number of transmitted flits and consequently the energy consumed. The provided mechanism relies on the abundance of memory data blocks filled with zeros in the analysed applications. We provide a hardware implementation for both compression and decompression at a generic network interface (NI). Results show the effectiveness of the compression and decompression mechanisms and the low overhead they introduce. The percentage of traffic reduced by the compression strategy (it is reduced by a factor of 3) justifies the added resources.

Key words— **Compression; NoC; Network-on-Chip; Embedded systems;**

I. Introduction

NETWORKS-ON-CHIP (NoCs) [1] have been accepted as the way to efficiently implement communication inside a chip. In the embedded domain, multiprocessor systems-on-chip (MPSoCs) are being implemented, where multiple cores (different IPs) are integrated and interconnected with a specialized NoC.

As system size increases, allowed by the technology evolution, the network becomes a principal component, both in terms of performance and power consumption. In this paper we focus on the power consumption of the NoC subsystem.

In this sense, compression of traffic along the NoC becomes an interesting approach to save power. Data streams sent through the NoC can be compressed at injection and decompressed at ejection, saving power and even latency (shorter messages). This is specially beneficial for long messages.

We tackle the compression of NoC traffic assuming a coherence protocol running on top of the NoC. This is becoming a general trend since ARM-based systems may accept coherence systems as the baseline design [2]. So, in this paper block memory transfers between the memory controller (MC) and the cache hierarchy have been chosen as the target target for compression.

A basic network interface (NI) has been taken as a baseline, thus not adopting any current NI technology: it only takes into account the existence of a decoupling buffer at the NI for the injection of packets, and a reception buffer at the NI for the ejection of packets. Communication between the core and the NI is not affected and is completely orthogonal to the solutions provided.

Two different solutions may be applied: parallel and sequential compression. Parallel compression is more resource demanding, but it is also a better informed technique, allowing for a more effective compression. In parallel compression the end node transfers a whole memory block to the NI (512 bits in our case), so the NI needs to allocate a message-size buffer, with the corresponding buffering overhead. At ejection, however, the NI needs to reconstruct the message and thus, some extra latency is incurred before delivering the complete message to the end node.

The solution we propose is orthogonal to the use of virtual channels. Indeed, an NoC with different virtual channels can be built up by physically replicating one VC-less network. Also, the baseline flit format will be specified and customized to the compression solution provided. Compression information needs to be injected together with the packets so that it can be decompressed at destination. Also, information for the coherence protocol must be included (source node address and memory address). Flit size is assumed to be 32 bits.

With these assumptions, we achieve a high compression rate, thus reducing long-message traffic by a factor of 3, by introducing an overhead of 8.91% in area and 24% in power. However, the large compression factor achieved allows to cancel and even obtain positive power reductions.

The rest of the paper is organized as follows. In Section II the Related Work is summarized. Then, in Section III the compression opportunities performed in the frame of the vIrtical project [3] are presented. Subsequently in Section IV we introduce the NI taken as a baseline; in Sections V and we present our compression strategy; in section VI the results obtained are show, followed by a discussion in Section VII closing by briefly acknowledging our sponsors in Section VIII and giving our final conclusions in Section IX.

II. Related Work

Compression of transmitted information does not seem to be a largely explored field. There are many papers on test data compression for NoCs, but on these papers the data is compressed offline, off the network, and only decompression is applied in the NIs. Some examples can be found at [4][5] among others. Since compression is not performed in the NoC, the strategies explained in these papers are not applicable to our needs. Other papers [6] also deal with compression of the routing table, which again is not what we were looking for. Some papers however study NoC compression strategies, which require special attention. Some examples of these are

[1]GAP, Universitat Politècnica de València, e-mail: msoler@gap.upv.es
[2]GAP, Universitat Politècnica de València, e-mail: jflich@disca.upv.es

[7][8][9][10][11][12].

In [7], a real-time compression technique that reduces the amount of bits sent is presented. The USBR technique removes the bits which do not change, and sends some extra information (which bits change and how long the data block is). This technique aims at data streams where the most significant bits are less likely to change than least significant bits, and thus it is not applicable to the type of data involved in our research.

The authors in [8], integrate compression and decompression with a coherence protocol. Basically, a protocol processing core is included in multi-core nodes to perform both the coherence protocol, compression and decompression. Avoiding unnecessary modification of the existing components and independence from other components or protocols are goals that prevent us from chosing an option that adds an extra core.

In [9] two compression strategies are presented. The first one, named Cache Compression is performed between the L1 and L2 (inside the cache) and is meant for storage (squeezing more data in the same sized cache structure). This strategy is not transmission-oriented. The second one is called Compression in the NIs and it consists in compressing the information right before injecting. The pitfall of this second strategy is that it needs to integrate such modules in every NI, when it is not necessary for us, and that it uses a table-based frequent-pattern compression.

Adaptive data compression for on chip network performance optimization is presented in [10]. It uses a table-based strategy, improving it by using shared tables and pipelining compression and injection of the flits. It is interesting, but the nature of the data characteristic to the applications of interest makes using a table redundant.

A similar approach is shown in [11], except that instead of frequent patterns, it is based on frequent values. The authors claim that with a small codebook for end to end communications, that does not need to be the same among different cores, they reduce power consumption in the router up to a 16.7%. This proposal is again not applicable to our needs.

Finally, in [12] an interesting proposal is presented: Although it uses tables (in fact it needs not only compression tables but also candidate ones, increasing redundancy), it allows to completely eliminate some transmissions (in the best case), sending only a short message and using matching status bits in the directory, which simplifies and fastens miss handling.

The particular solution discussed here is not compliant with any of these found in previous literature. It is a new proposal based on the massive appearance of zeros in the applications analysed for the vIrtical project.

III. Compression Opportunities

In the frame of the vIrtical project, representative workloads for the targeted architecture were selected and analyzed, concluding that long strings of only zeros are very common in traffic between L2 and the memory controller. Three cases can be distinguished:

- Aligned Consecutively Repeating Patterns (ACRP), consecutive and aligned to byte size.
- non-Aligned Consecutively Repeating Patterns (nACRP), consecutive but not necessarily aligned to byte size.
- non-Aligned non-Consecutively Repeating Patterns (nAnCRP), not necessarily aligned to any data size nor necessarily consecutive.

Table I shows the percentage of flits transmitted with the nACRP pattern type, which is the most frequent one. The table shows the results for two encription applications: sha and aes. The Secure Hash Algorithm (SHA) is a family of cryptographic hash functions. In our case sha1 and sha2 are used as test cases: sha1 is a 160-bit hash function similar to MD5 and sha2 can be further divided into two functions sha256 and sha512 that differ in word size (32-bit and 64-bit respectively). Of the various implementation of the Advanced Encryption Standard (AES) we chose the mode Encription Code Book (ECB) that encrypts and decrypts each block separately (in our case with two block sices, 128-bit and 256-bit).

The table shows that 93.45% of the bits transmitted between L2 and the memory controller fall under the nACRP pattern . Due to flit alignment and length restrictions, not all these bits will be eliminated, but over 80% of the transmitted information can in fact be compressed with high resulting benefits.

TABLA I
Comparison nACRP zeros

sha			aes-ecb	
1	256	512	128	256
93,45	93,26	93,42	92,83	92,74

IV. Baseline Network Interface (NI)

Figure 1 shows the schematic of the NI used as a baseline. It implements two ports connected to the network, one in each direction (for injection and ejection). Injection logic is driven by the node connected to the network (through the NI) and is specific of the protocol used (AMBA, AXI, OCP, etc.) Likewise, the NI has an ejection buffer where messages received from the network are temporarily stored and delivered to the node. The logic to read from this buffer is also protocol dependent. The size of both buffers will be customized to different configurations, both in number of slots (1, 2, 4 & 8) and slot width (32 & 558).

The NI has three extra logic blocks, all of them implemented in the solutions provided. The header builder logic is in charge of preparing the header of the flits of the message to be injected. At both sides of injection/reception flow control logic is also implemented: the Stop&Go protocol. Finally, as shown in the figure, we add two logic blocks for the compression/decompression solution. These blocks are intimately linked to the buffers.

The NI evaluated in this work does not support virtual channels. Indeed, injection and ejection buffers consist of a single FIFO queue each. To fully support virtual channels (for correctness) and decouple request and reply messages, both the standard

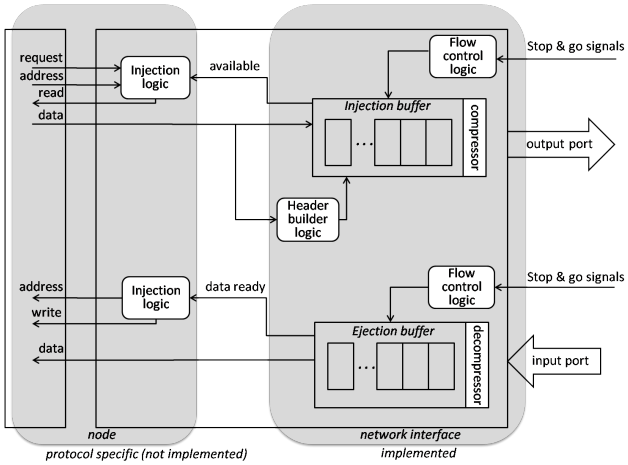


Fig. 1. Baseline NI

(not implemented) part of the NI and the network are replicated. The results obtained for this paper in terms of area must be multiplied by the number of virtual channels needed.

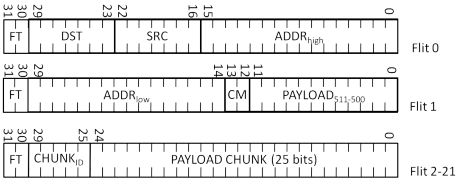
Compression logic needs to be applied only to the channel used by long messages carrying memory blocks. This motivates also for a design of the NI in isolation of the virtual channel requirements, since the final solution will require a NI for the request layer with no compression logic built-in and a NI for the reply layer with built-in compression logic.

V. Parallel Compression Strategy

In this section we develop the strategy to compress messages with a parallel approach, where the whole message is simultaneously accessed by the NI. Long packets carry a memory block, which is the target of our compression strategy. 64-byte memory blocks are divided in 20 chunks of 25 bits each and a remainder set of 12 bits (the highest order bits of the memory block). 25-bit chunks have been selected as it allows a perfect match with the flit and packet width. If all the bits of a chunk are set to zero, then the chunk is not transmitted over the network. In order to support chunks and align them to flits, we need to define the packet format for long messages. Since short messages are not compressible, their packet format has been omitted.

In Figure 2 we see the packet format for long messages: FT is a two bit flit-type field in every flit; DST and SRC (7-bit each) are respectively the destination and source of the message; $ADDR_{high}$ is made up of the most significant half of the memory address (16 of 32 bits), while $ADDR_{low}$ corresponds to the least significant half; CM is a command of the of the coherence protocol; $CHUNK_{ID}$ is the flit-number (2-21) and PAYLOAD is the part of the message the flit carries.

If a memory block with all its bits set to zero arrives, the strategy will only inject into the network flits 0 and 1. This is the maximum compression achievable with this strategy. Notice that these two flits carry extra fields (address, source, destination), which are typically non-zero.



FT	Description
00	Non valid
01	Tail
10	Payload
11	Header

Fig. 2. Long packet format (memory blocks) for parallel compression

A. Implementation

Figure 3 shows the parallel compression implementation. When a node requests to inject a new message, the injection logic copies the entire message into the injection buffer (if there is any free slot). Messages are injected together with control information (destination node, source node, and memory address) thus adding 558 bits in total (packet). The whole packet is treated as payload, although the first 60 bits (the first two flits) are never compressed. Those bits correspond to the source, destination, address, and the 12 most significant bits of the message.

The buffer slot is logically divided in chunks of 25 bits, excluding the aforementioned 60 bits (chunks 0 & 1). Chunks 2 to 21 are sent to the OR stage where all-zero chunks will be detected. The resulting output of the OR stage will be used to compute the flit type as well as to select the next flit to inject into the network. This selection drives the multiplexer shown in the figure. Let us describe each stage details.

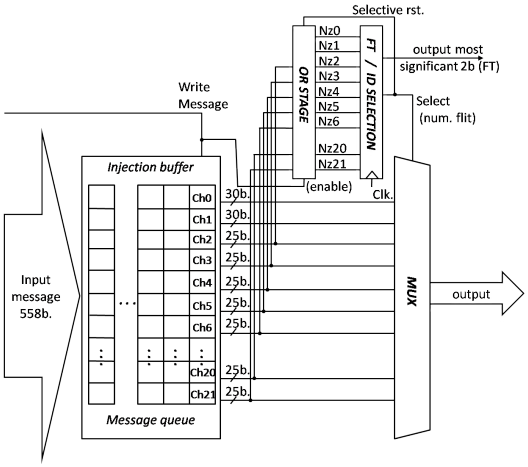


Fig. 3. Injection NI with compression (general view)

Figure 4 shows the OR and FT/ID selection stages. First, a 22-bit register stores the output of the OR gate for every chunk (notice that the first two chunks are not computed with an OR gate but are always set to one). The register (Nz) will keep account of the chunks that need to be injected (i.e. are non-zero). This register is written once per message injected when the message is exposed at the head of the queue (in the first slot), and can be driven by the write signal of the buffer. The Nz register output is now fed to the FT/ID selection logic. This logic selects the next flit to inject by means of a priority encoder with all the Nz bits as input. The encoder selects the most significant bit set to one. The Se-

lect signal is then used to drive the multiplexer at the injection port, as well as to reset the appropriate Nz bit so that, at the next cycle, a different chunk is selected (if any).

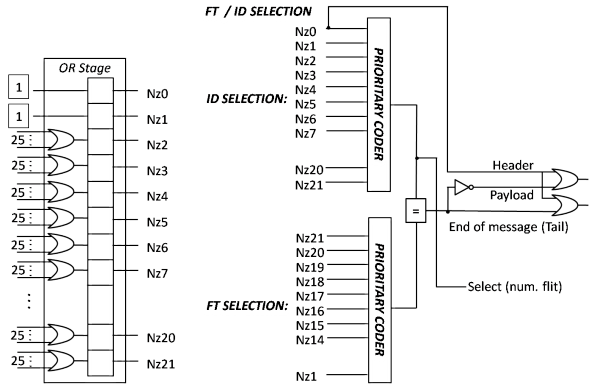


Fig. 4. OR and FT/ID selection stages of compression

In addition, the FT/ID selection logic computes the flit type (see the table in Figure 2). Therefore, the complexity lies only in computing which of the chunks is the tail (last flit to be sent). This is achieved by an additional priority encoder but this time the Nz bits are input in reverse order. Hence the last chunk with the Nz signal set (meaning the chunk is non-zero chunk) is identified: with a comparator of the two outputs of the priority encoders, upon a match, the last flit being injected is identified as the tail flit.

The decompression strategy, shown in Figure 5, is much simpler since we do not need to calculate anything. When the first flit arrives (with the FT field set to head) the 30 remaining bits are placed at the beginning of the reception buffer and the 25-bit chunks are reset. The second flit is also automatically copied right after. If the second flit is not the tail, all subsequent flits are placed according to their flit address (bits 29 to 25 of the flit). The tail flit activates the end of message signal to indicate that the message is complete and the buffer slot is full.

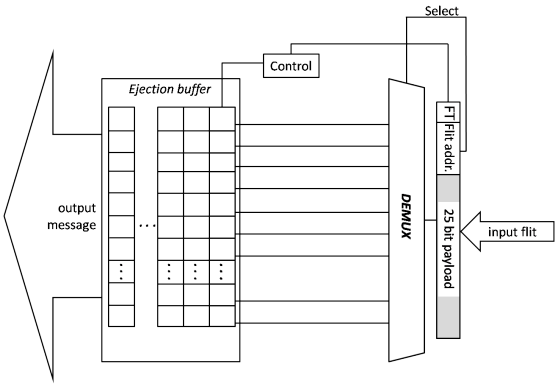


Fig. 5. NI ejection with compression mechanism

VI. Evaluation Results

In this section we provide the results when analysing both performance of the compression strategy, and the overheads of the implemented solutions. Therefore, we present the results in two separate parts. In the first part, we provide compression effectiveness by analysing the percentage of traffic being really compressed. For this, we use our simulation platform and inject the application memory access traces obtained in the vIrtical framework. In the second part, we focus our attention on the implementation overheads, showing results for area overheads, operating frequencies, and power consumption estimations of the implemented modules.

The combination of both parts indicates the effectiveness of the compression solution. After the evaluation, the discussion section is focused on the combination of both parts.

A. Compression Rate Achieved

One important aspect of the compression mechanism is its effectiveness. For this, the compression rate achieved has been analysed by simulation with gMemNoCsim. The scenario modelled is shown in Figure 6. Four cores are attached to the same router. The cores are modelled by private L1 caches. Also, an L2 cache is implemented and attached to the same router. The memory controller (MC) is attached to a neighbour router.

An invalidation directory-based coherence protocol with MOESI is implemented. Flit size is set to 32 bits, short messages sized to 64 bits (2 flits) and long messages sized to 512 bits (block size).

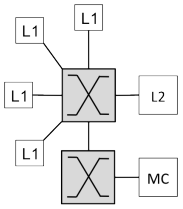


Fig. 6. Simulated scenario

Figure 7 shows the number of flits injected into the NoC (the traffic between the L2 cache and the memory controller). There is a significant traffic reduction in all the application cases analyzed. On average, traffic is reduced by a factor of 3.5.

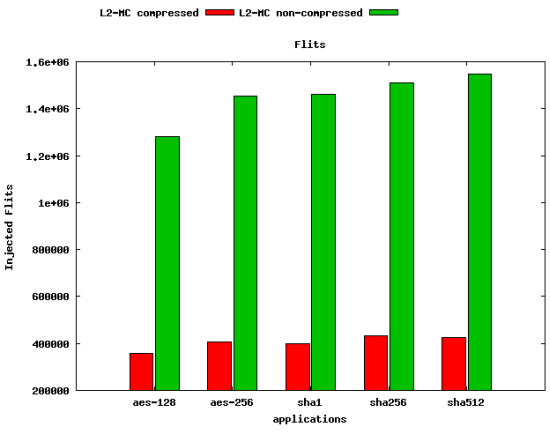


Fig. 7. Number of flits injected into the NoC

The impact in execution time is negligible as shown in Figure 8, since memory blocks accesses between the MC and the L2 bank are not in the critical path of the application.

B. Implementation Overheads

The compression design has been synthesized using the 45nm technology open source Nangate [45nm.

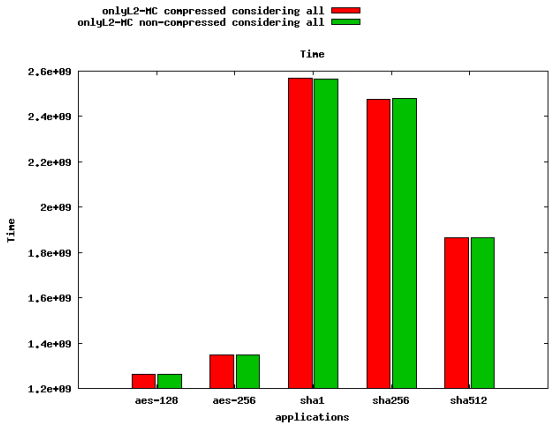


Fig. 8. Execution time of application cases

FreePDK] with Synopsys Design Compiler 7 [13]. Table II shows the area for two models of the injection part of the NI, one with no compression logic added (baseline) and one with the compression logic. We show results for different number of slots (ranging from one to eight). Notice that slot width is set to 558 bits (a whole large message and its header per slot).

TABLA II
AREA OVERHEADS. INJECTION. 558-BIT SLOTS

	No compression		Compression		Overhead
Slots	Area	/1 slot	Area	/1 slot	
1	6587.09	1	7768.6	1	17.94
2	12156.7	1.85	13313.9	1.71	9.52
4	23444.7	3.56	24789.3	3.19	5.74
8	45839.2	6.96	47911.5	6.17	4.52

As we increase the number of slots we achieve larger area overheads. Indeed, the buffer is the most demanding area resource. This trend is seen in both injection logic implementations.

For the ejection part of the NI, see Table III, we see different trends. As the number of receiving slots increases (each slot is 558 bits wide), the area overhead increases. When looking at the compression overheads, it ranges from no overhead for one slot to 14% for the 8-slot solution.

TABLA III
AREA OVERHEADS. EJECTION. 558-BIT SLOTS

	No compression		Compression		Overhead
Slots	Area	/1 slot	Area	/1 slot	
1	5710.4	1.00	5741.8	1.00	0%
2	11168.3	1.95	11788.5	2.05	6%
4	22334.2	3.91	25120.6	4.37	12%
8	45823.3	8.02	52134.6	9.08	14%

Now, let us turn our attention to power consumption. Power has been estimated by using Cadence Encounter 7 [14] (both for Place&Route and measurements). Table IV shows the power consumed by the injection part of the NI with and without compression capabilities (baseline vs. compression). As we can see, power consumption trends keep similar to the ones achieved when no compression is included. It has to be noted also that the power consumption achieved by injecting less flits onto the network is not accounted in these tables.

TABLA IV
POWER CONSUMPTION. INJECTION. 558-BIT SLOTS

Slots	No Compression	Compression	Overhead
1	3.104	5.455	75.74%
2	7.733	12.62	63.19%
4	16.14	25.72	59.36%
8	34.39	50.86	47.89%

Finally, for power consumption, Table V shows a comparison of power consumption for the ejection part of the NI, when no compression and compression is used. Trends are similar and power consumption overhead is negligible.

TABLA V
POWER CONSUMPTION. EJECTION. 558-BIT SLOTS

Slots	No Compression	Compression	Overhead
1	6.231	6.475	4%
2	13.32	12.59	-6%
4	24.93	28.03	12%
8	56.22	57.07	1%

VII. DISCUSSION

The previous results show the area and power overheads of different parts of the NIs, also for different configurations of number of slots. Now, we need to compound those results for a possible NI being used by the coherence protocol. This means, the NI will be made of different components, servicing different message classes of the coherence protocol. In detail, the protocol triggers short and long messages and only long messages, carrying memory blocks are subject to be compressed. This means, the NI needs to be built with different injector and with different ejector configurations.

Table VI shows the area overheads and power consumption of the NI with no compression mechanism included but for slot sizes of 32 bits. The table shows the results for both components, injection and ejection. These components will be used to handle short control messages of the protocol.

TABLA VI
AREA AND POWER. EJECTION. 32-BIT SLOTS

Slots	Area	Power
1	314.32	0.309
2	655.20	0.826
4	1321.25	1.791
8	2643.48	3.589

With all these results, now we can build the overheads of a final NI with the following characteristics:

- Injection of long messages with compression facility enabled (1 slot)
- Ejection of long messages with compression facility enabled (1 slot)
- Injection of short messages with no compression facility (2 slots)
- Ejection of short messages with no compression facility (2 slots)

For the number of slots at each component we need to consider the flit size. Indeed, wide slots will need different network cycles to inject the whole message, while thin slots will need fewer cycles. Therefore, is reasonable to use few slots for long messages while more slots for short messages. We select one slot for large messages and two slots for short messages. With all these considerations, we derive the

results by adding the different overheads previously presented. Table VII shows the results achieved.

TABLA VII
OVERHEAD OF THE WHOLE NI.

NI component	Area	Overhead	Power	Overhead
Inject long	7768.6	17.94	5.455	75.74
Eject long	5741.8	0	6.475	3.91
Inject short	655.20	0	0.826	0
Eject short	655.20	0	0.826	0
Total	14820,8	8.91%	13.582	24%

As we can see, the overhead is significant but the potential reduction of the number of flits injected in the network (Table I) suggests that the global power consumption will be as well reduced. To corroborate this, we have derived the approximate power consumption of the network per flit for both the presented baseline NI implementation and our compression-enhanced implementation. In the graph in Figure 9 we can see the difference in power consumption for different injection rates. In every graph, the X-axis corresponds to the compression rate [0:1] and the Y-axis corresponds to the difference in power consumption between our solution and the baseline (we have taken as a baseline a NI with one injector and one ejector of 2 32-bit slots each). We have only taken into account the long message network, since the other network is exactly the same in both cases and would be nullified.

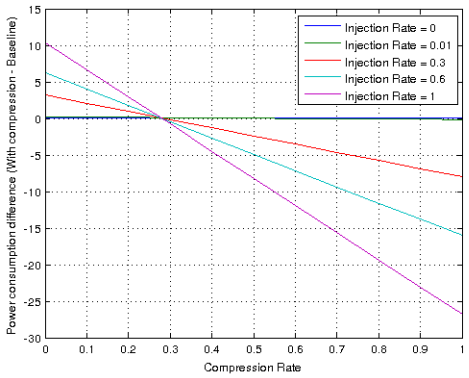


Fig. 9. Study of the difference in power consumption of our solution and the baseline depending on compression-rate for different injection rates

In the graph in Figure 9 we can see that depending on long message injection rate, we can save different amounts of power, but in any case the compression rate needed is 0.3 (we need to send at least 30% less flits in order to get some power benefit). Since our compression rate is much beyond the needed 30%, it is assured that with data that follows the pattern of the applications studied some power would be saved.

VIII. Acknowledgements

This work has been supported by the VIRTICAL project (grant agreement nº 288574) which is funded by the European Commission within the Research Programme FP7.

IX. Conclusions

In this paper we have provided the results obtained in the frame of the vIrtical project. Our goal was to design and analyse compression strategies at the NoC level, saving communication costs, by reducing the number of transmitted flits. The provided mechanism relies on the abundance of memory data blocks filled with zeros, thus easily compressible by using a detection strategy.

Our compression mechanism avoids sending flits without information (i.e. not sending flits that only consist of zeros). This technique is both easily applied and highly efficient. This compression scheme is only applied to long messages (a full memory block, 512 bits), not to short messages (of only 2 flits).

The results displayed in the previous section show the effectiveness of the compression and decompression mechanisms and the low overhead they introduce. The percentage of traffic reduced by the compression strategy (a factor of 3.5) justifies the overheads in resources for compression and decompression. As we can see in Table VII, the area overhead for the compression and decompression mechanisms required for a system with coherence support is 8.91% whereas the added power consumption is 24%.

In the graph in Figure 9 we can see that depending on long message injection rate, we can save different amounts of power, but in any case the compression rate needed is 0.3 (we need to send at least 30% less flits in order to get some power benefit). Since our compression rate is much beyond the needed 30%, it is assured that with data that follows the pattern of the applications studied some power would be saved.

Referencias

[1] F. Gilabert, F. Silla, M. E. Gomez, M. Lodde, A. Roca, J. Flich, J. Duato, C. Hernández, and S. Rodrigo, "*Designing network on-chip architectures in the nanoscale era*," 2010.
[2] ARM Limited, "*Amba axi and ace protocol specification*," 2013.
[3] "*Virtical: "sw/hw extensions for heterogenous multicore platforms"*," .
[4] V. Froese, R. Ibers, and S. Hellebrand, "*Reusing noc-infrastructure for test data compression*," in *VTS*, 2010.
[5] S. Chaki, C. Giri, and H. Rahaman, "*Binary difference based test data compression for noc based socs.*," in *ISVLSI*. 2012, pp. 114–119, IEEE.
[6] M. Palesi, S. Kumar, and R. Holsmark, ," in *A method for router table compression for application specific routing in mesh topology noc architectures*, 2006, SAMOS.
[7] S. Ogg and B. Al-Hashimi, ," in *Improved Data Compression for Serial Interconnected Network on Chip through Unused Significant Bit Removal*, 2006, VLSID.
[8] L. Vittanala and M. Chaudhuri, ," in *Integrating Memory Compression and Decompression with Coherence Protocols in Distributed Shared Memory Multiprocessors*, 2007, ICPP.
[9] R. Das, A. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M. Yousif, and C. Das, "*Performance and power optimization through data compression in network-on-chip architectures.*," in *HPCA*, 2008.
[10] J. Yuho, K. Yum, and E. Kim, ," in *Adaptive data compression for high-performance low-power on-chip networks*, 2008, MICRO 41.
[11] P. Zhou, Y. Zhang, J. Yang, and Li Zhao, "*Frequent value compression in packet-based noc architectures*," 2009.
[12] B. An, M. Lee, K. Yum, and E. Kim, ," in *Efficient Data Packet Compression for Cache Coherent Multiprocessor Systems*, 2012, DCC.
[13] *Design Compiler User Guide*, 2011.
[14] *Encounter User Guide*, 2011.

Detección Automática de Cuellos de Botella de Potencia en Aplicaciones Científicas Paralelas

María Barreda[1], Sandra Catalán[1], Manuel F. Dolz[1], Rafael Mayo[1] y Enrique S. Quintana-Ortí[1]

Resumen— **En este artículo presentamos una extensión del entorno `pmlib` para el análisis del balance entre potencia y rendimiento que permite una detección rápida y automática de sumideros de potencia durante la ejecución de cargas de trabajo científicas concurrentes. La extensión está formada por un módulo Python multithread que ofrece una alta fiabilidad y flexibilidad, produciendo un proceso de inspección general que introduce bajo sobrecoste.**

Palabras clave—**Eficiencia energética, computación de altas prestaciones, análisis y trazas de consumo de energía, aplicaciones científicas.**

I. Introducción

ADEMÁS de los grandes beneficios que se espera que ofrezca la computación Exaescala a las disciplinas científicas fundamentales, desde la biología a la ingeniería nuclear, también se prevé que ésta ejerza un impacto positivo en la competitividad industrial que compense sus costes con mucho [1], [2]. Sin embargo, un número de estudios recientes [1], [3], [4] han identificado el consumo de potencia como uno de los retos clave en el escalado de hardware eficiente más allá de la Petaescala. Mientras que en la pasada década, las infraestructuras de computación de altas prestaciones han disfrutado de mejoras considerables en el ratio potencia-rendimiento [5] —sobre todo debido a la implantación de plataformas heterogéneas equipadas con aceleradores hardware o a la adopción de procesadores de bajo consumo— queda mucho por hacer en términos de eficiencia energética para hacer viables los sistemas Exaescala en 2020.

En los últimos años, la tecnología y mecanismos de ahorro de potencia concebidos para aplicaciones móviles y empotradas se han adoptado cada vez más por los diseñadores de los sistemas de escritorio y servidores. Por otra parte, la conciencia sobre la eficiencia energética en el software está todavía muy por detrás [6], [7], a pesar de la pérdida de energía que una aplicación con un mal comportamiento puede provocar.

En este artículo extendemos nuestro entorno `pmlib` [8], [9] para el análisis de potencia-rendimiento de códigos científicos paralelos con una sencilla pero potente herramienta de inspección que identifica automáticamente los sumideros de energía como *discrepancias* entre la actividad de la aplicación y los C-estados de la CPU [10]. El análisis se realiza a posteriori, (esto es, después de la ejecución de la aplicación) y produce una traza visual y/o un informe analítico a partir de una comparación directa de las trazas de rendimiento de las aplicaciones (obtenidas utiliando `Extrae` [11] o, en principio, cualquier otro entorno de obtención de trazas) con una traza de los C-estados del core, recogidos en tiempo de ejecución por `pmlib` accediendo a los registros MSR. En caso de que la plataforma bajo estudio esté equipada con algún tipo de medidor (interno o externo) o sensores hardware para monitorizar la disipación de potencia (p.e., *"Running Average Power Limit" de Intel*, RAPL [12], [13]), estas muestras pueden además ser integradas/visualizadas con trazas de rendimiento, C-estados y sumideros de potencia, y esta información puede ser procesada para ofrecer una estimación aproximada de los costes de energía de los cuellos de botella de potencia detectados.

El resto del artículo se estructura como sigue. En la Sección II proporcionamos un breve resumen del entorno `pmlib` para el análisis de potencia-rendimiento. La principal aportación del trabajo, es decir, la herramienta para detectar los sumideros de energía se describe a continuación, en la sección III. Tras esto se discuten las conclusiones y trabajo futuro en la Sección IV.

II. Monitorización de Rendimiento-Potencia utilizando `pmlib`

Durante los dos últimos años, hemos desarrollado nuestro entorno `pmlib` como una suite portable para la caracterización y análisis del balance de potencia-rendimiento de aplicaciones científicas paralelas [8], [9].

El entorno está formado por dos módulos/tipos de demonios: Un demonio `pmlib` externo se ejecuta en un sistema separado (el servidor de trazas de potencia), para evitar interferir con la apliación mientras se recogen las muestras de potencia de los diferentes medidores utilizados en la plataforma objetivo donde la aplicación es ejecutada. Además, un demonio `pmlib` interno se ejecuta en cada nodo de la plataforma de destino, recogiendo información (por ejemplo, los C-estados del core) que sólo es acesible localmente.

La Figura 1 ilustra la interacción del entorno `pmlib`, una suite para obtener trazas de rendimiento y una herramienta de visualización gráfica con la aplicación objetivo. El punto de inicio es una aplicación científica concurrente, instrumentada con el software `pmlib`, que se ejecuta en la plataforma objetivo. Unidos a los nodos utilizados por la apliación hay distintos dispositivos de medición (ya sean inter-

[1]DICC, Universitat Jaume I (UJI), 12071 - Castellón (España), e-mails: {`mvaya, catalans,dolzm, mayo, quintana`}`@uji.es`

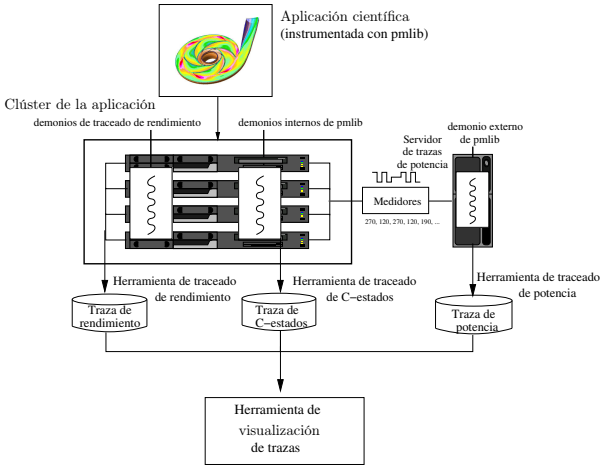


Fig. 1. Interacción de `pmlib` y de herramientas de obtención de trazas de rendimiento y de visualización con cargas de trabajo científicas paralelas, produciendo trazas del rendimiento y de la disipación de potencia de la aplicación que se convierten en la entrada de la herramienta de visualización.

nos o externos; ver [9] para una lista de dispositivos soportados) que el demonio `pmlib` externo muestrea contínuamente desde la plataforma de trazas de potencia. Las llamadas desde el código de usuario que se ejecuta en la plataforma objetivo, utilizando la API de `pmlib`, posibilitan la comunicación con el módulo externo `pmlib` [9] para indicar al servidor de trazas que debe empezar/parar de recoger los datos capturados por los medidores, volcar las trazas de potencia en ficheros de disco, etc. Al terminar la ejecución de la aplicación, la traza de potencia puede ser inspeccionada, opcionalmente junto a una traza de rendimiento, utilizando la herramienta de visualización apropiada.

Nuestro framework permite integrar de forma sencilla las trazas de potencia de `pmlib` y las trazas de rendimiento obtenidas con `Extrae`, mientras que el resultado puede ser visualizado con `Paraver` [14]; ver la Figura 2 por ejemplo. No obstante, el diseño modular del framework permite acomodar fácilmente otras suites de obtención de trazas, por ejemplo, TAU [15], Vampir [16], HDTrace [17], etc.

III. Detección Automática de Sumideros de Potencia

En trabajos previos, el uso de `pmlib` nos había permitido identificar comportamientos o configuraciones en las que se producían pérdidas de energía en aplicaciones [18], runtimes [19] y bibliotecas [20] relacionadas con la ejecución paralela de códigos de álgebra lineal. En todos estos casos, los cuellos de botella de potencia fueron detectados como desacuerdos entre la actividad de la aplicación y el consumo de energía del sistema y, en general, el origen podía ser asociado al uso de esperas activas que consumen gran cantidad de energía (p.e., bucles de polling) con escasos beneficios en el tiempo de ejecución. Sin embargo, durante estos estudios, la inspección visual simultánea de las trazas de rendimiento y de potencia se reveló como una tarea engorrosa, además de ser

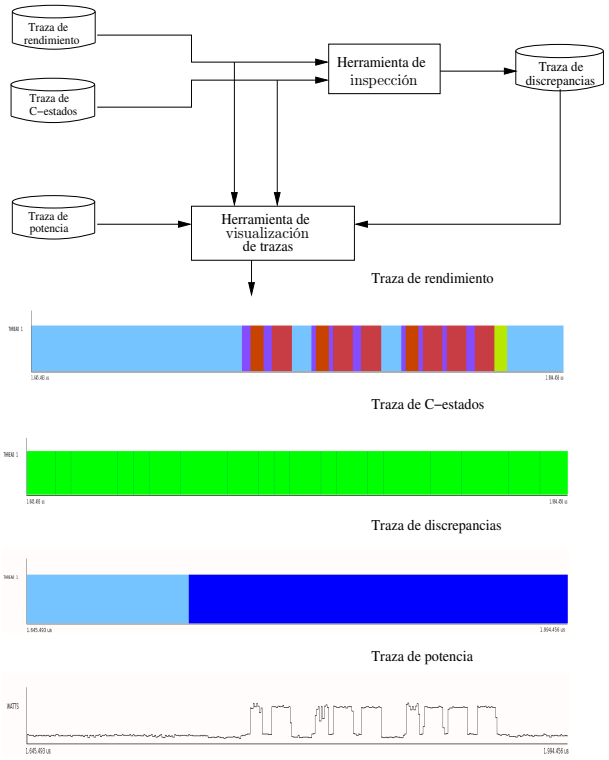


Fig. 3. Funcionamiento de la herramienta de inspección para detectar y reportar los sumideros de energía.

un proceso propenso a errores.

A. Resumen

En esta sección presentamos un analizador paralelo a posteriori para detectar cuellos de botella de potencia con varias ventajas importantes sobre el enfoque manual anterior:

- La herramienta automatiza y acelera el proceso de inspección.
- El proceso es más seguro, ya que la detección de sumideros de potencia se basa en la comparación entre la traza de rendimiento de la aplicación y la traza de C-estados por core (en vez de la traza de rendimento frente a la traza de potencia de todo el socket/plataforma que hemos explotado antes).
- El analizador es flexible, puesto que la longitud del intervalo de análisis y el umbral de divergencia (discrepancia) son paramétros, entre otros, que pueden ser ajustados por el usuario a los niveles deseados.
- La inspección introduce un bajo sobrecoste durante la recogida de muestras, con pocos efectos secundarios en el rendimiento de las aplicaciones y las trazas de energía.

La interacción de la herramienta de inspección con el resto del entorno se ilustra en la Figura 3.

B. Funcionamiento e implementación

A continuación describimos el analizador y sus propiedades con más detalle. La herramienta de inspección está escrita en Python y, después de la ejecución de la aplicación, puede ser empleada para analizar

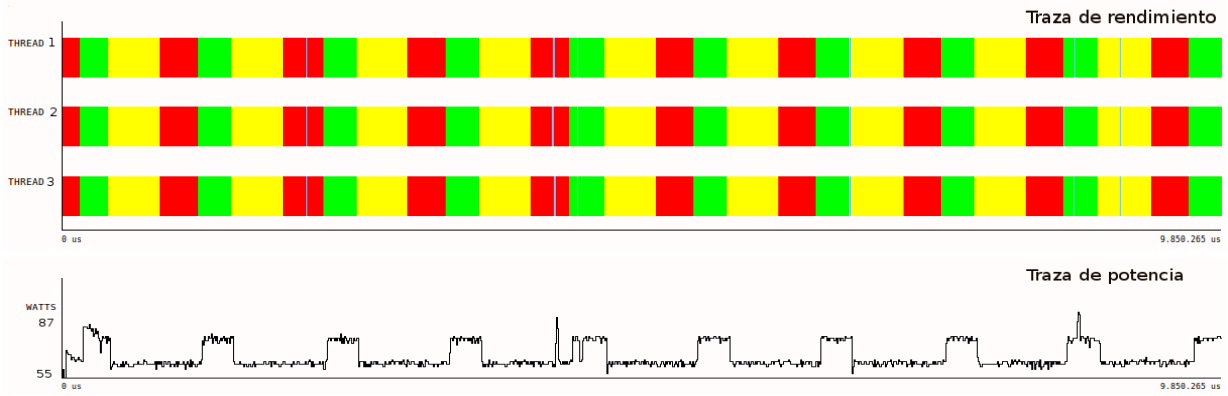


Fig. 2. Información capturada con el framework `pmlib` y visualizada con `Paraver`: trazas de rendimiento y de potencia de la aplicaicón (imagen superior e inferior, respectivamente).

una traza de rendimiento, en principio producida por `Extrae`, y una traza de C-estados por core compatible producida por el demonio interno `pmlib`.

El uso de `Extrae`, combinado con el entorno de visualización `Paraver` [14], es conveniente porque nos permite analizar interactivamente aplicaciones científicas concurrentes, paralelizadas con p-threads, OpenMP o MPI. Sin embargo, esta decisión no nos impide adoptar otras suites de rendimiento en nuestro entorno ya que la herramienta de inspección sólo procesa las trazas de rendimiento, y no interactúa de ninguna otra manera con la suite específica que las ha producido.

Por otra parte, la traza de C-estados se obtiene con el demonio interno `pmlib`, que monitoriza el estado de los cores del procesador en tiempo de ejecución. Mientras que en el trabajo anterior [9] ya extrajimos información sobre los C-estados de los ficheros en `/sys/devices/system/cpu/`, cuando están disponibles, la nueva aplicación recupera estos datos de los registros MSR del sistema de destino en su lugar, lo que se traduce en un menor sobrecoste. Además, la frecuencia de lectura del demonio es también configurable, lo que permite al usuario ajustar el nivel de ruido introducido en las trazas de potencia y rendimiento.

Para procesar la información contenida en las trazas, la herramienta de inspección las divide en intervalos de cierta longitud (configurable por el usuario), empezando en los instantes de tiempo 0 y $0.5t$ de las trazas. (La razón para hacer una doble división de las trazas, empezando en 0 y en $0.5t$, es para evitar que cuellos de botella de potencia pasen el análisis sin ser detectados si están divididos entre dos intervalos consecutivos.) Todos los intervalos son entonces insertados (encolados) en un pool de tareas para ser analizados por el analizador multithread, empleando la clase `Pool` de Python [21]. Para cada core de la CPU e intervalo, el analizador compara los ratios de tiempo que la aplicación ha estado inactiva (p.e., sin realizar ningún cálculo útil desde el punto de vista de la aplicación) mientras el core permanece en estado C0 (activo), detectando un potencial sumidero de potencia siempre que la diferencia entre estos dos valores sea superior a un umbral dado (configurable por el usuario). Mientras que la versión de la herramienta de inspección que se acaba de describir realiza un análisis lineal de las trazas, también hemos desarrollado una implementación alternativa que realiza una búsqueda binaria de los cuellos de botella de potencia y, en general, es más rápida cuando el número de discrepancias es pequeño.

El resultado de la búsqueda de cuellos de botella es doble: analítico y gráfico. Por un lado, produce una salida de texto simple con una tupla (c, t_i, t_f, $\%divergencia$) por cada sumidero de potencia detectado, donde es c el identificador de core, $[t_i, t_f]$ es el intervalo de tiempo donde ocurre el cuello de botella en la línea de tiempo de la traza, y el último parámetro cuantifica la divergencia entre la inactividad de la aplicación y los ratios de los C-estados del core en este periodo. Por otro lado, el analizador también produce una nueva traza que permite una rápida identificación de las fuentes de las discrepancias utilizando una herramienta de visualización, en nuestro caso, `Paraver`.

C. Ejemplos

A continuación ilustramos las posibilidades de la herramienta de inspección utilizando dos aplicaciones. Nuestro primer ejemplo se corresponde con la ejecución de un benchmark sintético, en el que participan dos procesos MPI, que hacen uso de las rutinas `MPI_Send` y `MPI_Recv`. En concreto, el proceso 1 realiza un cómputo intensivo durante unos 3 segundos y después hace una llamada a la rutina `MPI_Recv` a la espera de recibir un vector de números reales del proceso 0. Este último, a su vez, permanece suspendido (función `sleep`) durante 10 segundos, y después llama a la rutina `MPI_Send` para enviar el vector al proceso 1. Este comportamiento se repite varias veces.

La Figura 4 muestra fragmentos de las trazas de rendimiento y C-estados obtenidas, respectivamente, con `Extrae` y `pmlib` para la ejecución del benchmark sintético de MPI en una plataforma híbrida equipada con un procesador Intel Xeon i7-3770, 16GB de RAM y una NVIDIA Tesla C2050 ("Fermi"). En principio, en este caso se puede esperar que, como la rutina `MPI_Recv` es bloqueante, durante el tiempo que el

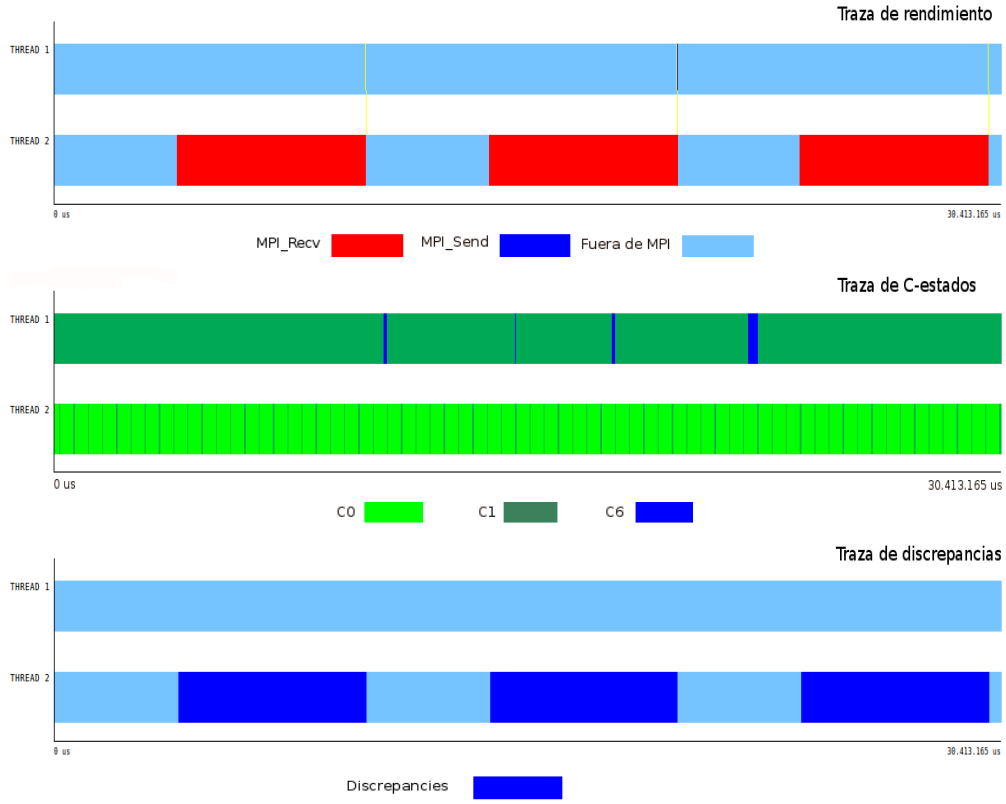


Fig. 4. Trazas de rendimiento (arriba), C-estados (enmedio) y discrepancias (abajo), visualizadas con `Paraver`, para la ejecución del benchmark sintético de MPI.

proceso 1 está esperando a recibir el vector por parte del proceso 0, el core pase a un C-estado "sleep" de ahorro de energía (C1 o superior). Sin embargo, las trazas en la parte superior y media de la Figura 4 muestran que éste no es el caso, y esto es claramente identificado en la traza de discrepancias en la parte inferior de la misma figura. Por tanto, este análisis nos permite descubrir que, en este caso, en el bloqueo producido por la rutina `MPI_Recv` el proceso se encuentra haciendo una espera activa y por tanto la CPU se mantiene en un C-estado C0, consumiendo una gran cantidad de energía.

El segundo ejemplo corresponde a la resolución de un sistema lineal disperso $Ax = b$, utilizando el método CG [28], en un sistema equipado con un procesador gráfico (GPU). El coste computacional del método CG está dominado por el producto matriz-vector disperso (`spmv`), realizado en este ejemplo en la GPU. Además, se emplean otras operaciones sobre vectores, también en la GPU, aunque con menor coste computacional que se realizan mediante llamadas a la librería CUBLAS de NVIDIA (`cublasSdot`, `cublasSaxpy` y `cublasSscal`).

La operación `spmv` es ubicua en computación científica, siendo clave para la solución iterativa de sistemas lineales. En la implementación utilizada de esta operación, la matriz A se organiza en formato ELLPACK que, si bien, produce cierta sobrecarga de almacenamiento, a cambio permite un uso más eficiente del hardware cuando se emplea sobre aceleradores como las GPUs.

La Figura 5 muestra las trazas de rendimiento y C-estados obtenidas, respectivamente, con `Extrae` y `pmlib`, para la ejecución de la aplicación que resuelve un sistema lineal usando el método CG, con la matriz de coeficientes del sistema siendo el benchmark `audikw` de la colección de matrices dispersas UFMC. Para dicha ejecución se ha usado una plataforma híbrida equipada con un procesador Intel Xeon i7-3930K, 24GB de RAM y una NVIDIA Tesla K20. En principio, en este caso se puede esperar que, mientras la aplicación se está ejecutando en la GPU, la CPU esté en un C-estado "sleep" de ahorro de energía. De nuevo, como podemos observar, en este caso, en las trazas de la parte superior y media de la Figura 5, éste no es el comportamiento, lo cual se identifica rápidamente en la traza de discrepancias de la parte inferior de la figura.

D. Impacto de los sumideros de potencia

Los dos ejemplos previos demuestran el potencial de la herramienta de inspección para detectar fácilmente las discrepancias entre las trazas de rendimiento y de C-estados que identifican posibles sumideros de potencia. Además, ilustran la interfaz de la herramienta de inspección, mostrando cómo se suministra esta información en forma de traza, que puede ser visualizada utilizando `Paraver`.

Por otro lado, el analizador también proporciona información estadística, como un informe analítico de la ejecución global o de un cierto periodo de ésta; ver Tabla I. Este tipo de información puede ser procesada a continuación para obtener una estimación de los costes de los sumideros de potencia. Asuma-

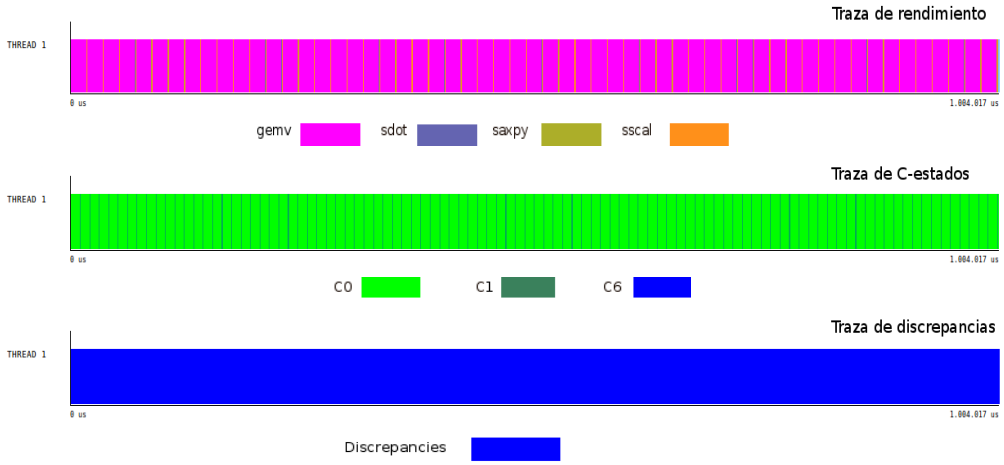


Fig. 5. Trazas de rendimiento (arriba), C-estados (enmedio) y discrepancias (abajo), visualizadas con `Paraver`, para la ejecución de la resolución del sistema lineal disperso usando el método CG.

TABLA I
EJEMPLO DEL RESUMEN ANALÍTICO DE LAS TRAZAS DE RENDIMIENTO, C-ESTADOS Y DISCREPANCIA DEVULETAS POR LA HERRAMIENTA DE INSPECCIÓN.

	Computation	Polling	C0	C1	C6	Discrepancies
THREAD 1	72.00 %	25.56 %	99.33 %	0.29 %	0.39 %	27.49 %
THREAD 2	96.45 %	2.50 %	99.25 %	0.26 %	0.50 %	4.77 %
THREAD 3	59.90 %	39.14 %	99.53 %	0.10 %	0.37 %	40.59 %
THREAD 4	70.81 %	28.13 %	99.48 %	0.10 %	0.42 %	30.11 %
THREAD 5	74.00 %	25.14 %	99.29 %	0.90 %	0.61 %	26.61 %
THREAD 6	99.18 %	0.00 %	99.34 %	0.22 %	0.45 %	0.00 %
THREAD 7	61.52 %	37.17 %	99.53 %	0.12 %	0.35 %	38.84 %
THREAD 8	75.03 %	23.69 %	99.27 %	0.10 %	0.64 %	25.74 %

mos por simplicidad que sólo hay un tipo de sumidero durante la ejecución de la aplicación inspeccionada. El informe analítico del analizador ofrece una estimación de cuánto tiempo los threads/cores no están realizando trabajo útil y, por lo tanto, desperdiciando potencia (por ejemplo, hasta un 40.59 % para el thread 3 en la Tabla I, que hace referencia a una aplicación diferente a las de los ejemplos). Por supuesto, para obtener una medida del coste energético total, se necesita tener en cuenta cuál es el ratio de potencia de un core que permanece en un estado "sleep" de ahorro de energía. De esta forma, la estimación es fácil de obtener como parte de un experimento simple e independiente. La parte difícil, sin embargo, es calibrar la situación opuesta, por ejemplo, qué parte del consumo total de la potencia que aparece en la traza corresponde a los cores culpables. Desafortunadamente, nuestro muestreo actual sólo ofrece el total de potencia consumida (utilizando un medidor) o, algo mejor aún, la potencia consumida por "socket" (utilizando sensores de potencia) mientras que, para obtener una estimación más precisa, sería deseable disponer de una estimación de consumo de energía por core. Para superar esta dificultad, se puede diseñar un test especial que imite el sumidero de potencia que se ha detectado en un determinado experimento, y que nos puede dar la estimación deseada. Para la aplicación de MPI, por ejemplo, ésta sería una llamada a la rutina `MPI_Recv`, realizada por un único core, lo cual permitiría evaluar el coste de potencia de este proceso. Para la resolución del sistema lineal disperso usando el método CG, por otra parte, el test debería considerar una larga secuencia de llamadas CUDA y la configuración apropiada del runtime de CUDA para reproducir el sumidero de potencia observado para esta aplicación.

Con estas estimaciones, calculando simplemente las diferencias entre la potencia debida a estos comportamientos/configuraciones que demandan una gran cantidad de potencia y la de los estados de ahorro de energía, así como la duración total de los sumideros de potencia, se puede obtener una aproximación de los costes energéticos debidos a los "puntos calientes" y el ahorro potencial que una implementación que tenga en cuenta el consumo energético y/o configuración del software puede ofrecer. Cabe destacar que esta elaboración sólo necesita un informe fiable de la duración de los cuellos de botella energéticos y los datos de los experimentos de calibración. Una vez completada la calibración, en realidad no existe necesidad de la traza/medidas de potencia.

IV. Conclusiones y Trabajo Futuro

Hemos presentado una extensión de nuestra herramienta de obtención de trazas de potencia `pmlib` que puede ser aprovechada para detectar automáticamente cuellos de botella de potencia durante la ejecución de una aplicación comparando las trazas de rendimiento y de los C-estados de los cores. El análisis se realiza rápidamente en paralelo, utilizando la clase `Pool` de Python, y el usuario puede configurar el proceso, por ejemplo, ajustando parte de la traza a inspeccionar, la longitud del intervalo de análisis y el umbral de detección, entre otros.

Como parte del trabajo futuro, planeamos *i)* extender `pmlib` para recuperar información de una variedad de sensores de potencia/temperatura, como los ofrecidos por los procesadores modernos de AMD, ARM, IBM, NVIDIA y Texas Instruments; *ii)* acomodar otras herramientas para la obtención de trazas de rendimiento (TAU, Vampir y HDTrace); y *iii)*

analizar las fuentes de sobrecostes del demonio interno `pmlib`, para obtener un sistema de obtención de trazas con menor sobrecoste.

Agradecimientos

Este trabajo ha sido financiado por el proyecto CICYT TIN2011-23283 del *Ministerio de Economía y Competitividad* y FEDER, el Proyecto Europeo FP7 318793 "EXA2GREEN", y el Programa de FPU del *Ministerio de Educación, Cultura y Deporte*.

Referencias

[1] S. Ashby and *et al*, "The opportunities and challenges of Exascale computing," Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, November 2010.

[2] K. Bergman and *et al*, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA IPTO ExaScale Computing Study, 2008.

[3] J. Dongarra and *et al*, "The international ExaScale software project roadmap," *Int. J. of High Performance Computing & Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[4] M. Duranton and *et al*, "The HiPEAC vision for advanced computing in horizon 2020," 2013.

[5] "The Green500 list," 2012.

[6] Susanne Albers, "Energy-efficient algorithms," *Commun. ACM*, vol. 53, pp. 86–96, May 2010.

[7] E. Saxe, "Power-efficient software," *ACM Queue*, 2010.

[8] P. Alonso, R. M. Badia, J. Labarta, M. Barreda, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí, and Ruymán Reyes, "Tools for power-energy modelling and analysis of parallel scientific applications," in *41st Int. Conf. on Parallel Processing – ICPP*, 2012, pp. 420–429.

[9] M. Barreda, S. Barrachina, S. Catalán, M. F. Dolz, G. Fabregat, R. Mayo, and E. S. Quintana, "A framework for power-performance analysis of parallel scientific applications," in *Third Int. Conference on Smart Grids, Green Communications and IT Energy-aware Technologies – Energy 2013*, 2013, pp. 114–119.

[10] HP Corp., Intel Corp., Microsoft Corp., Phoenix Tech. Ltd., and Toshiba Corp., "Advanced configuration and power interface specification, revision 5.0," 2011.

[11] H. Servat and G. Llort, *Extrae user guide manual for version 2.1.1*.

[12] Intel Corp., *Intel 64 and IA-32 Architectures Software Developer Manual*, 2012.

[13] Intel Corp., "Intel Xeon processor," `http://www.intel.com/xeon`, 2012.

[14] "Paraver: the flexible analysis tool," `http://www.cepba.upc.es/paraver`.

[15] Performance Research Lab., Dept. of Computer Sciences, University of Oregon, "TAU performance system," `http://www.cs.uoregon.edu/research/tau/home.php`.

[16] GWT-TUD GmbH, "Vampir performance optimization," `http://www.vampir.eu/`.

[17] Julian Kunkel, "HDTrace - a tracing and simulation environment of application and system interaction," Tech. Rep. 2, Department of Informatics, Scientific Computing. Universität Hamburg, 2011.

[18] José I. Aliaga, Manuel F. Dolz, Alberto F. Martín, Rafael Mayo, and Enrique S. Quintana-Ortí, "Leveraging task-parallelism in energy-efficient ILU preconditioners," in *2nd Int. Con. on ICT as Key Technology against Global Warming – ICT-GLOW*, number 7453 in Lecture Notes in Computer Science, pp. 55–63. 2012.

[19] P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Ortí, "Reducing energy consumption of dense linear algebra operations on hybrid CPU-GPU platforms," in *Proc. 10th IEEE Int. Symp. on Parallel and Distributed Processing with Applications– ISPA 2012*, 2012, pp. 56–62.

[20] M. Castillo, J. C. Fernández, R. Mayo, E. S. Quintana-Ortí, and V. Roca, "Analysis of strategies to save energy for message-passing dense linear algebra kernels," in *Proc. 20th Euromicro Conference on Parallel, Distributed and Network based Processing*, 2012, pp. 346–352.

[21] *Official Website, Python Programming Language*, http://www.python.org/.

[22] Jose I. Aliaga, Matthias Bollhöfer, Alberto F. Martín, and Enrique S. Quintana-Ortí, "Exploiting thread-level parallelism in the iterative solution of sparse linear systems," *Parallel Computing*, vol. 37, no. 3, pp. 183–202, 2011.

[23] G. Quintana-Ortí, E.S. Quintana-Ortí, R.A. van de Geijn, F.G. Van Zee, and E. Chan, "Programming matrix algorithms-by-blocks for thread-level parallelism," *ACM Trans. Math. Softw.*, vol. 36, no. 3, pp. 14:1–14:26, 2009.

[24] Gregorio Quintana-Ortí, Francisco D. Igual, Enrique S. Quintana-Ortí, and Robert van de Geijn, "Solving dense linear systems on platforms with multiple hardware accelerators," in *ACM SIGPLAN 2009 symposium on Principles and practices of parallel programming (PPoPP'09)*, 2009, To appear.

[25] Intel, "Intel math kernel library (mkl) 11.0," `http://software.intel.com/en-us/intel-mkl`.

[26] NVIDIA Corporation, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, 2.3.1 edition, August 2009.

[27] P. Alonso, M. F. Dolz, F. D. Igual, E. S. Quintana-Ortí, and R. Mayo, "Runtime scheduling of the LU factorization: Performance and energy," in *Proc. Energy Efficiency in Large Scale Distributed Systems conference – EE-LSDS 2013*, 2013,
To appear.

[28] Y.Saad, "Iterative Methods for Sparse Linear Systems" *Society for Industrial and Applied Mathematics*, Philadelphia, USA, 2003

On the Leakage-Power Modeling for Optimal Server Operation

Patricia Arroba [1], Marina Zapater [2], José L. Ayala[3], José M. Moya[1], Katzalin Olcoz[3] and Román Hermida[3]

Resumen— **Leakage power consumption is a component of the total power consumption in data centers that is not traditionally considered in the setpoint temperature of the room. However, the effect of this power component, increased with temperature, can determine the savings associated with the careful management of the cooling system, as well as the reliability of the system. The work presented in this paper detects the need of addressing leakage power in order to achieve substantial savings in the energy consumption of servers. In particular, our work shows that, by a careful detection and management of two working regions (low and high impact of thermal-dependent leakage), energy consumption of the data-center can be optimized by a reduction of the cooling budget.**

Palabras clave— **Power consumption, Leakage, cooling, efficiency.**

I. Introduction

ONE of the big challenges in data centers is to manage system resources in a power-efficient way. Data centers consume from 10 to 100 times more power per square meter than typical office buildings [1]. They can even consume as much electricity as a city [2]. The power consumption budget in data centers comes from computation processing, disk storage, network, and cooling systems.

It must be said that greening the computer industry is touching off an unprecedented level of cooperation and information-sharing among companies, government, and laboratories. In the USA, *Green Grid*, a consortium of industry leaders (like AMD, Intel, Dell, HP, IBM, Sun, and others who are normally competitors) to share data and strategies for greener data centers. Green Grid's membership also includes the Pacific Gas and Electric Company (better known as PGE), and it recently announced a collaboration agreement with the U.S. Department of Energy.

However, data center designers have collided with the lack of accurate power models for the energy-efficient provisioning of their devised infrastructures, and the real-time management of the computing facilities. The work proposed in this paper makes substantial contributions in the area of power modeling of high-performance servers for data center-operated services.

Interestingly, the key issue of how to control the setpoint temperature at which to run the cooling system of a data center, is still to be clearly defined [3]. Data centers typically operate in a temperature range between 20° C and 22° C, but we can find some of them as cold as 13° C degrees [4,5]. Due to lack of scientific data in the literature, these values are often chosen based on conservative suggestions provided by the manufacturers of the equipment. Some authors estimate that increasing the setpoint temperature by just one degree can reduce energy consumption by 2 to 5 percent [4,6]. Microsoft reports that raising the temperature from two to four degrees in one of its Silicon Valley data centers saved $250,000 in annual energy costs [5]. Google and Facebook have also been considering increasing the temperature in their data centers [5].

Power consumption in servers can be estimated by the summation of the dynamic power consumption of every active module, dependent on the activity, and the leakage power consumption, that is strongly correlated with the integration technology. In particular, leakage power consumption is a component of the total power consumption in data centers that is not traditionally considered in the setpoint temperature of the room. However, the effect of this power component, increased with temperature, can determine the savings associated with the careful management of the cooling system, as well as the reliability of the system itself.

The work presented in this paper detects the need of addressing leakage power in order to achieve substantial savings in the energy consumption of servers and makes the following contributions:

- we establish the need of considering leakage power consumption and its dependency with temperature for modern data centers;
- we detect and define two working regions depending on the impact of leakage power in the total power consumption of high-performance servers;
- we observe that substantial energy savings can be achieved by the careful management of the cooling system once the previous contribution has been verified;
- we validate the previous hypothesis with a deep experimental work that resembles the infrastructure of current enterprises.

II. Related work

In [7] a statistical model that provides run-time system-wide prediction of energy consumption on server blades is proposed. The authors develop a

[1]Electronic Engineering Dept., ETSI. Telecomunicación, Universidad Politécnica de Madrid, e-mail: {parroba,josem}@die.upm.es
[2]CEI Campus Moncloa UCM-UPM, e-mail: marina@die.upm.es
[3]DACYA, Universidad Complutense de Madrid, e-mail: {jlayalar,katzalin,rhermida}@ucm.es

linear regression model that relates processor power, bus activity, and system ambient temperatures into real-time predictions of the power consumption. Other works such [8–10] also present the power consumption of a server as a linear function of the CPU usage of that server.

Some other linear models can be found in [9], where server's power is formulated as a quadratic function of the CPU usage, or in [11], where the transition between server functionality state (Idle - ON) is taken into consideration. The work in [12] follows a similar approach but, in this case, the CPU power consumption percentage is separated in two parts: the one due to the applications, and the second one due to management services turning on and off the server. The power modeling technique vMeter, proposed by Bohra et al. [13] observes a correlation between the total system's power consumption and component utilization. They created a four-dimensional linear weighted power model for the total power consumed $P(total)$ by separating the contribution of each active domain in a node. However, none of these works have considered the effect of leakage power and temperature in the total power consumption of the servers.

The work presented in [14] is most relevant for us. In this paper, the authors compare the impact of increasing the air temperature entering the rack on the complete cooling infrastructure, with the alternative approach of allowing greater air temperature rise across the rack. Even though in their approach they develop a power model from chip to cooling tower, they still ignore the leakage as a key factor in data center power consumption and energy saving opportunities.

III. Background on data center power modeling

The main contributors to the energy consumption in a data center are the computing power (also known as IT power), i.e. the power drawn by servers in order to run a certain workload, and the cooling power needed to keep the servers within a certain temperature range that ensures safe operation. Traditional approaches have tried to reduce the cooling power of data center infrastructures by increasing the supply temperature of Computer Room Air Conditioning Units (CRAC units). However, because of the direct dependency of leakage current with temperature, the leakage-temperature tradeoffs at the server level must be taken into account when optimizing energy consumption.

In this section we show the impact of these tradeoffs on the total energy consumption of the data center, as well as how the ambient room temperature influences the cooling power of data centers. This fact, as will be shown later, can be exploited to optimize the power consumption of the data center.

A. Computing power

Current state-of-the-art resource management and selection techniques were contemplating only the dynamic power consumption of servers when allocating tasks or selecting machines. Moreover, the devised power models have not traditionally included the impact of leakage power consumption and its thermal dependency, driving to non-optimal solutions in their energy optimization plans.

Theoretically, no electrical current should circulate through the substrate of a MOS transistor between drain and source when it is powered off due to an infinite gate resistance. However, in practice this is not true, and leakage currents flow through the reverse-biased source and drain-bulk pn junctions in dynamic logic. Also due to the continuous technology scaling, the influence of leakage effects is rising, increasing junction leakage currents by 5 orders of magnitude compared to previous feature sizes according to Rabaey [15].

Dynamic consumption has historically dominated the power budget. But when scaling technology below the $100nm$ boundary, static consumption becomes much more significant, being around 30-50% [16] of the total power under nominal conditions. This issue is intensified by the influence of temperature on the leakage current behavior. With increasing temperature the on-current of a transistor is reduced slightly. However, the reduction of the threshold voltage is not sufficient to compensate for the decreased carrier mobility that has a strong exponential impact on leakage current.

Therefore, it is important to consider the strong impact of static power consumed by devices as well as its temperature dependence and the additional effects influencing their performance. In this section, we derive a leakage model for the static energy consumption of servers and we validate it with real measurements taken in an AMD Opteron machine of our case study.

The current that is generated in a MOS device due to leakage is the one shown in equation 1.

$$I_{leak} = I_s \cdot e^{\frac{V_{GS}-V_{TH}}{nkT/q}} \cdot (1 - e^{\frac{Vds}{kT/q}}) \quad (1)$$

Research by Rabaey [15] shows that if $V_{DS} > 100mV$ the contribution of the second exponential is negligible, so the previous formula can be rewritten as in Equation 2:

$$I_{leak} = I_s \cdot e^{\frac{V_{GS}-V_{TH}}{nkT/q}} \quad (2)$$

where technology-dependent parameters can be grouped together to obtain the formula in Equation 3:

$$I_{leak} = B \cdot T^2 \cdot e^{\frac{V_{GS}-V_{TH}}{nkT/q}} \quad (3)$$

where B defines a constant that depends on the manufacturing parameters of the server.

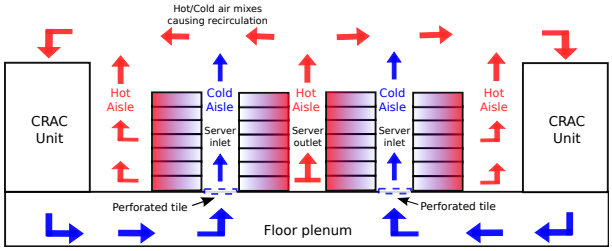


Fig. 1. Data Center cooling scheme

B. Cooling power

The cooling power is one of the major contributors to the overall data center power budget, consuming over 30% of the overall electricity bill in typical data centers [14]. In a typical air-cooled data center room, servers are mounted in racks, arranged in alternating cold/hot aisles, with the server inlets facing cold air and the outlets creating hot aisles. The CRAC units pump cold air into the data room and extract the generated heat (see Figure 1). The efficiency of this cycle is generally measured by the *Coefficient of Performance* (COP). The COP is a dimensionless value defined as the ratio between the cooling energy produced by the air-conditioning units (i.e. the amount of heat removed) and the energy consumed by the cooling units (i.e. the amount of work to remove that heat), as shown in Equation 4.

$$COP_{MAX} = \frac{\text{output cooling energy}}{\text{input electrical energy}} \quad (4)$$

Higher values of the COP indicate a higher efficiency. The maximum theoretical COP for an air conditioning system is described by Carnot's theorem as in Equation 5:

$$COP_{MAX} = \frac{T_C}{T_H - T_C} \quad (5)$$

where T_C is the cold temperature, i.e. the temperature of the indoor space to be cooled and T_H is the hot temperature, i.e. the outdoor temperature (both temperatures in Celsius). As the difference between hot and cold air increases, the COP decreases, meaning that the air-conditioning is more efficient (consumes less power) when the temperature difference between the room and the outside is smaller.

According to this, one of the techniques to reduce the cooling power is to increase the COP by increasing the data room temperature. We will follow this approach to decrease the power wasted on the cooling system to a minimum, while still satisfying the safety requirements of the data center operation.

IV. Experimental methodology

The experimental methodology in this paper pursues two goals: (i) to describe the leakage-temperature tradeoffs at the server level, by means of measuring the power consumption of an enterprise server at different temperatures and under a controllable workload; and (ii) to validate the model in a real data room environment where the air-conditioning can be controlled. After this, we will be able to evaluate the energy savings that could be obtained in a data center when our modeling strategy is applied.

A. Server-level setup

As temperature-dependent leakage cannot be measured separately from the dynamic power in a server by the power measurement devices, we use a controllable workload, *lookbusy*[1], in order to explore the leakage at the server level. *Lookbusy* can stress all the hardware threads to a fixed CPU utilization percentage without memory or disk usage, for a paricular period of time. The usage of a synthetic workload to derive the leakage model has many advantages, the most important of which is that dynamic power can be described as linearly dependent with CPU utilization and Instructions Per Cycle (IPC), or kept constant. In our case, we stress the system at the maximum CPU utilization, in order to isolate the dynamic power. The platform under test is a SunFire V20z server with 2 Dual-Core AMD Opteron processors and 4GB of RAM. Keeping the workload constant, we slowly vary the inlet temperature of the server, obtaining CPU temperatures ranging from 45°C to 70°C, while monitoring the following server parameters: (i) CPU temperature (1 sensor per CPU), (i) memory temperature (1 sensor per each of the 2 memory banks), (iii) fan speed and (iv) overall power consumption.

All temperatures and fan speed values are obtained via the server internal sensors, collected through the Intelligent Platform Management Interface (IPMI) tool [2]. IPMI allows to poll the internal sensors of the enterprise server with negligible overhead. Because the server is not shipped with power consumption sensors, we use non-intrusive current clamps. The current clamp is connected to the power cord of the server and wirelessly transmits the monitored data to a base station connected to a desktop computer. Data gathered via the current clamp and the server internal sensors are aligned to ensure they have a common timestamp.

Our hypothesis is that we can find and define two different working regions depending on the impact of leakage power in the total power consumption of high-performance servers. In this sense, we aim to prove that for the lower range of temperatures, the impact of the temperature-dependant leakage is negligible, whereas for a higher temperature range leakage needs to be considered. This hypothesis will be verified throughout the extensive experimental work and the methodology just described.

B. Data room setup

In order to validate the server-level model in an infrastructure resembling a real data center scenario, we install eight Sunfire V20z servers in a rack inside

[1] http://www.devin.com/lookbusy/
[2] http://ipmitool.sourceforge.net/

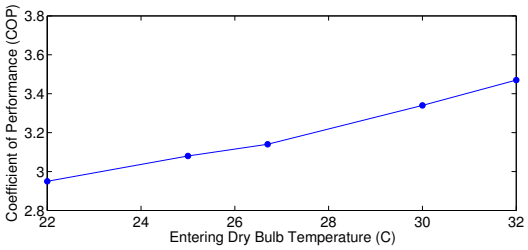


Fig. 2. Evolution of the air-conditioning COP with room temperature

an air-cooled data room, with the rack inlet facing the cold air supply and the outlet facing the heat exhaust. The air conditioning unit mounted in the data room is a Daikin FTXS30 unit, with a nominal cooling capacity of 8.8kW and a nominal power consumption of 2.8KW. We assume an outdoor temperature of 35°C and use the manufacturers technical data to obtain the COP curve depending on the room temperature [17]. This temperature is only used to estimate the energy savings based on the curve provided by the manufacturer and does not affect the experimental results.

As can be seen in Figure 2, as the room temperature and the heat exhaust temperature raises, approaching the outdoor temperature, the COP increases and, thus the cooling efficiency improves.

We monitor all the servers by means of IPMI tool to gather the server internal sensors and the current clamps to obtain power consumption. We set the air supply temperature at various values ranging from 18°C to 24°C, and run from 1 to 4 simultaneous instances of the different tasks of the SPEC CPU 2006 benchmark suite [18] in the servers of the data room. Our goal is to verify the leakage-temperature model, finding the maximum air-supply temperature that makes the servers work in the temperature region where leakage is negligible.

V. Results

Characterizing the power with respect to temperature under a constant synthetic workload allows us to define different working regions depending on the impact of leakage power. In region I, for CPU temperatures ranging from 44° C to 48° C, we find that the contribution of the leakage to the total consumption of the server is negligible (see Figure 3). As can be seen, the obtained data follows a linear trend, as expected. Once the regression model is built, we obtain Equation 6 that fits the experimental data.

$$P_I = 0.0288 \cdot T_{CPUaverage} + 182.15 \qquad (6)$$

On the other hand, region II is defined for those CPU temperatures higher than 48° C. In this region, the impact of power consumption due to leakage needs to be considered, as can be seen in Figure 4. The data fitting to a linear curve in this region is shown in Equation 7, where an increase of about one order of magnitude in the slope of the curve can be appreciated if compared with region I.

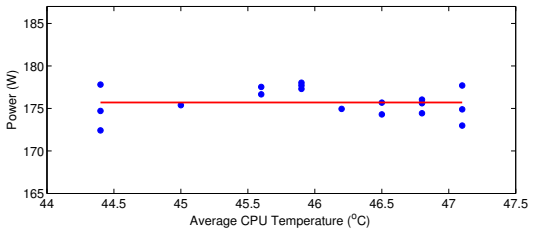


Fig. 3. Power consumption of SunFire V20z for temperature region I

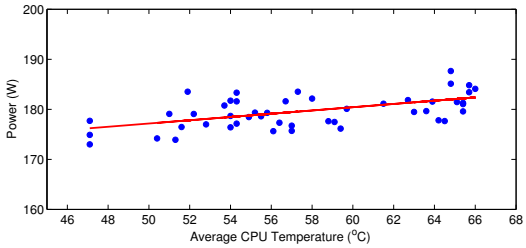


Fig. 4. Power consumption of SunFire V20z for temperature region II

$$P_{II} = 0.3255 \cdot T_{CPUaverage} + 160.894 \qquad (7)$$

In both cases, the dispersion of the samples are due to the inaccuracy of the clamp, whose error in performing the measurements is close to ±5W.

After obtaining the two working regions for the leakage power, we move to the data room setup. We run the tasks of the SPEC CPU 2006 benchmark suite in the AMD servers under different data room conditions. In our experiments, we run from 1 to 4 instances of SPEC CPU in the AMD servers at different room temperatures of 18° C, 20° C, 22° C and 24° C. Figure 5a shows the power consumption values for two simultaneous instances of the SPEC CPU 2006 benchmark at an air supply setpoint temperature of 18° C, 20° C and 24° C, respectively. Figure 5b shows the CPU temperature for each of these tests under the same conditions.

Because all other variables are constant, and as

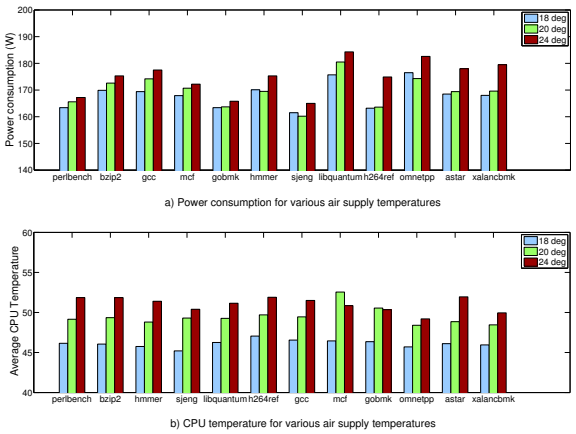


Fig. 5. Power consumption of SPEC CPU 2006 at different air supply temperatures

the measurement error with the current clamp is already controlled, the changes in the power consumption for each test can be due to the differences in ambient temperature. As can be seen in the plots, even though there are differences in the average CPU temperature between the 18° C and the 20° C case, for most of the benchmarks CPU temperature does not go above the 50° C, staying in the negligible leakage area. In fact, the power consumption differences between the 18° C and the 20° C case are in the range of ±5W, so we cannot consider them to be due to leakage, but to the inaccuracy of our current clamp. However, for the 24° C case, CPU temperatures raise above 50° C and power consumption for most of the benchmarks is considerably higher than in the 18° C scenario, achiving differences higher than 8W for gcc, libquantum, astar and xalancbmk benchmarks. Thus, in this region we begin to observe temperature-dependant leakage.

The experimental results for our data room scenario show that if we allow temperature to raise above this 24° C barrier, the contribution of the leakage increases, increasing the computing power drawn by our infrastructure. However, for our data room configuration and under our workload, leakage is negligible in the 18° C-24° C range and, thus, we can raise the ambient temperature in order to reduce cooling power.

If we increase the air supply temperature from 18° C to 24° C, the room temperature increases and the COP varies (see Figure 2) from 2.95 to 3.47, increasing the energy efficiency of the cooling equipment and reducing the cooling power. This increase has a proportional impact on the energy savings of the infrastructure, leading to a decrease of 11.7% in cooling power as predicted by the curve.

VI. Conclusions

Power consumption in servers can be estimated by the summation of the dynamic power consumption of every active module, dependent on the activity, and the leakage power consumption, that is strongly correlated with the integration technology. However, traditional approaches have never incorporated the impact of leakage power consumption in these models, and the noticeable values of leakage power consumption that appear at higher CPU temperatures.

The work presented in this paper detects the need of addressing leakage power in order to achieve substantial savings in the energy consumption of servers. In particular, our work shows that, by a careful detection and management of two working regions (low and high impact of thermal-dependent leakage), energy consumption of the data-center can be optimized by a reduction of the cooling budget. Finally, we validate these facts with a deep experimental work that resembles the infrastructure of current enterprises, where an 11 % of the cooling budget can be reduced.

Acknowledgement

Research by Marina Zapater has been partly supported by a PICATA predoctoral fellowship of the Moncloa Campus of International Excellence (UCM-UPM). This work has been partially supported by the Spanish Ministry of Economy and Competitiveness, under contracts TIN2008-00508, TEC2012-33892 and IPT-2012-1041-430000, and INCOTEC. The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Centro de Supercomputación y Visualización de Madrid (CeSViMa).

Referencias

[1] P. Scheihing, "Creating energy efficient data center," in *Data Center Facilities and Engineering Conference*, Washington DC, USA, May 2007.

[2] J. Markoff and S. Lohr, "Intel's huge bet turns iffy," *New York Times Technology Section*, September 2002.

[3] Nosayba El-Sayed, Ioan A. Stefanovici, George Amvrosiadis, Andy A. Hwang, and Bianca Schroeder, "Temperature management in data centers: why some (might) like it hot," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, 2012, SIGMETRICS '12, pp. 163–174, ACM.

[4] J. Brandon, "Going green in the data center: Practical steps for your SME to become more environmentally friendly," *Processor*, , no. 29, 2007.

[5] Rich Miller, "Google: Raise your data center temperature.," October 2008.

[6] "Summer time energy-saving tips," .

[7] Adam Lewis and et al., "Run-time energy consumption estimation based on workload in server systems," in *HotPower*, Berkeley, CA, USA, 2008, pp. 4–4.

[8] Steven Pelley and et al., "Understanding and abstracting total data center power," in *WEED*, June 2009.

[9] Xiaobo Fan and et al., "Power provisioning for a warehouse-sized computer," in *ISCA*, New York, NY, USA, 2007, pp. 13–23.

[10] Frank Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *ACM SIGOPS*, New York, NY, USA, 2000, pp. 37–42.

[11] David Meisner and et al., "Peak power modeling for data center servers with switched-mode power supplies," in *ISLPED*, New York, NY, USA, 2010, pp. 319–324.

[12] G. Warkozek and et al., "A new approach to model energy consumption of servers in data centers," in *ICIT*, 2012, pp. 211–216.

[13] A.E.H. Bohra and V. Chaudhary, "Vmeter: Power modelling for virtualized clouds," in *IPDPSW*, 2010, pp. 1–8.

[14] T.J. Breen, E.J. Walsh, J. Punch, A.J. Shah, and C.E. Bash, "From chip to cooling tower data center modeling: Part i influence of server inlet temperature and temperature rise across cabinet," in *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on*, 2010, pp. 1–10.

[15] J. Rabaey, *Low Power Design Essentials*, Engineering (Springer-11647). Springer, 2009.

[16] S.G. Narendra and A.P. Chandrakasan, *Leakage in Nanometer CMOS Technologies*, Integrated Circuits and Systems. Springer, 2010.

[17] Daikin AC (Americas), Inc., "Engineering data split, ftxs-l series," 2010.

[18] SPEC CPU Subcommittee and John L. Henning, "SPEC CPU 2006 benchmark descriptions," http://www.spec.org/cpu2006/.

Algoritmos y Técnicas de Programación Paralelas

Biblioteca de clases de Objetos Paralelos para implementar Patrones de Comunicación usando CPANs

Mario Rossainz López[1] y Manuel I. Capel Tuñón[2]

Resumen— **Se propone una biblioteca de clases de Objetos Paralelos [10] (utilizando los modos de comunicación síncrono, asíncrono y futuro asíncrono) para implementar los patrones de comunicación/interacción más comúnmente utilizados, en particular las granjas de procesos o Farms, los cauces de procesos o PileLine y los árboles binarios de procesos usando Divide y Vencerás o TreeDV; a través de un enfoque de Programación Paralela Estructurada junto con un método de programación basado en Composiciones Paralelas de Alto Nivel o CPANs [14], bajo el paradigma de la orientación a objetos para potencializar las propiedades de la encapsulación y la abstracción y así poder proporcionar al programador la posibilidad de la reusabilidad de los patrones antes mencionados para la construcción de otros más complejos en la solución de problemas [15], junto con un conjunto de restricciones predefinidas de sincronización entre procesos (maxpar o paralelismo máximo, mutex o exclusión mutua, sync o sincronización entre procesos usando el modelo productor-consumidor).**

Palabras clave—**CPANs, Programación Paralela Estructurada, Programación Orientada a Objetos, Objetos Paralelos.**

I. Introducción

Como es sabido, existen infinidad de aplicaciones que utilizando máquinas con un solo procesador tratan de obtener el máximo rendimiento de un sistema al resolver un problema; sin embargo, cuando tal sistema no puede proporcionar el rendimiento esperado, una posible solución consiste en optar por aplicaciones, arquitecturas y estructuras de procesamiento paralelo o concurrente. Hoy en día el avance reciente en sistemas masivamente paralelos, comunicaciones de gran ancho de banda, procesadores rápidos para el tratamiento de señales, etc., así lo permiten. Parte importante de estas investigaciones son los algoritmos paralelos, metodologías y modelos de programación paralela que actualmente se están desarrollando.

El presente trabajo propone una biblioteca de clases que proporciona al programador los patrones de comunicación más comúnmente utilizados en la programación paralela [8],[12], en particular, los patrones pipeline, farm y tree de la técnica de diseño de algoritmos divide y vencerás y que han sido implementados usando Composiciones Paralelas de Alto Nivel (CPANs) [14],[16], las cuales tienen las propiedades siguientes:

1. Modos asíncrono y futuro asíncrono de comunicación entre los objetos paralelos del CPAN [10].
2. Objetos con paralelismo interno.
3. Disponibilidad de mecanismos de sincronización (MaxPar, MuTex y Sync Productor-Consumidor).
4. Disponibilidad de control de tipos genéricos.
5. Transparencia en la distribución de aplicaciones paralelas.
6. Rendimiento satisfactorio: Programabilidad, Portabilidad y Performance

II. Composiciones Paralelas de Alto Nivel

Un CPAN es la composición de un conjunto de objetos de tres tipos: Un objeto manager que representa al CPAN en sí mismo y hace de él una abstracción encapsulada que oculta su estructura interna. El manager controla las referencias de un conjunto de objetos (un objeto denominado Collector y varios objetos denominados Stage), que representan los componentes del CPAN y cuya ejecución se lleva a cabo en paralelo y debe ser coordinada por el propio manager. Los objetos Stage son los encargados de encapsular una interfaz tipo cliente-servidor que se establece entre el manager y los objetos esclavos (objetos pasivos que contienen el algoritmo secuencial de la solución de un problema). Y un objeto Collector que es un objeto encargado de almacenar en paralelo los resultados que le lleguen de los objetos stage que tenga conectados (ver fig 1).

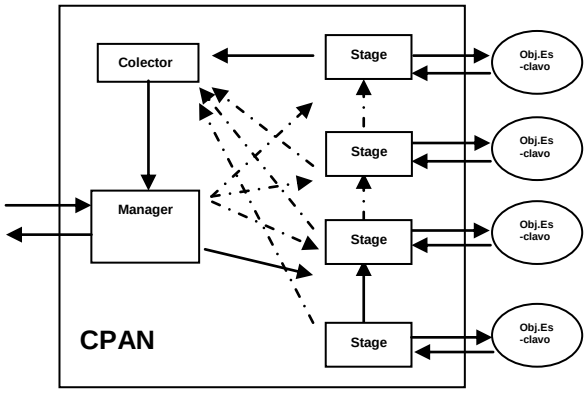


Fig. 1. Estructura Interna de un CPAN. Composición de sus Componentes.

[1]Universidad Autónoma de Puebla, Avenida. San Claudio y 14 Sur, San Manuel, Puebla, Puebla, 72000, México. E-mail: rossainz@cs.buap.mx
[2]ETS Ingeniería Informática y de Telecomunicación, Universidad de Granada, Periodista Daniel Saucedo Aranda s/n, 18071 Granada, España. E-mail: manuelcapel@ugr.es

A. *La clase abstracta ComponentManager de un CPAN*

Define la estructura genérica del componente manager de un CPAN

```
CLASS ABSTRACT ComponentManager
{
 ComponentStage[ ] stages;

 PUBLIC VOID init (ASOCIACION[] list)
   {ABSTRACT;}

 PUBLIC ANYTYPE execution(ANYTYPE datain)
   VAR
     ComponentCollector res;
   {
    res = ComponentCollector CREATE();
    commandStages(datain,res);
    RETURN res.get();
   }

 PRIVATE VOID commandStages(ANYTYPE
 datain,ComponentCollector res)
   {ABSTRACT;}

 MAXPAR (execution);
};
```

B. *La clase Abstracta ComponentStage de un CPAN*

Define la estructura genérica del componente stage de un CPAN, así como de sus interconexiones.

```
CLASS ABSTRACT ComponentStage
{
 ComponentStage[] otherstages;
 BOOL am_i_last;
 METHOD meth;
 OBJECT obj;

 PUBLIC VOID init (ASOCIACION[ ] list)
   VAR
     ASOCIACION item;
   {
    item = HEAD(list);
    obj = item.obj;
    meth= item.meth;
    if (TAIL(list) == NULL)
    am_i_last = true;
   }

 PUBLIC VOID request (ANYTYPE datain,
 ComponentCollector res)
   VAR
     ANYTYPE dataout;
   {
    dataout = EVAL (obj,meth, datain);
    IF (am_i_last)
        TREAD res.put(dataout)
    ELSE commandOtherStages (dataout,
                             res);
   }

 PRIVATE VOID commandOtherStages(ANYTYPE
 dataout, ComponentCollector res)
   { ABSTRACT;}

 MAXPAR (request);
};
```

C. *La clase Concreta ComponentCollector de un CPAN*

Define la estructura concreta del componente colector de cualquier CPAN. Éste componente implementa un buffer multi-item, donde se irán almacenando los resultados de los stages que hagan referencia a este colector.

```
CLASS CONCRETE ComponentCollector
{
  VAR
    ANYTYPE[] content;

  PUBLIC VOID put (ANYTYPE item)
     {CONS(content, item);}

  PUBLIC ANYTYPE get()
    VAR
      ANYTYPE result;
    {
     result = HEAD(content[]);
     content = TAIL(content[]);
     RETURN result;
    }

   SYNC(put,get);
   MUTEX(put);
   MUTEX(get);
 };
```

III. DISEÑO Y CONSTRUCCIÓN DEL CPAN PIPELINE

La Fig.2 representa el patrón paralelo de comunicación PipeLine como un CPAN.

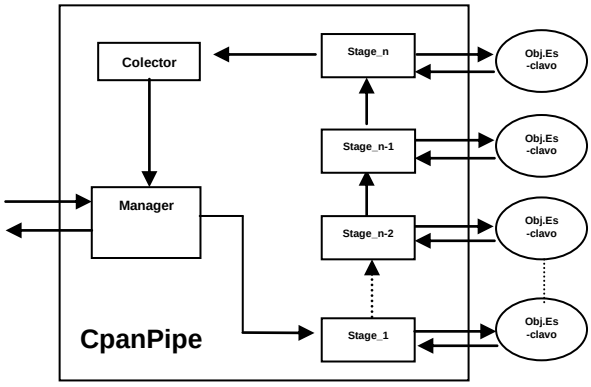


Fig. 2. El CPAN de un PipeLine

Los objetos stage_i y Manager del modelo gráfico del CpanPipe son instancias de clases concretas que heredan las características de las clases ComponentManager y ComponentStage.

A. *Definición Sintáctica y Semántica del Cpan Pipe*

Cualquier objeto PipeManager se encarga sólo del primer stage del pipeline en su inicialización. Durante la ejecución de una petición de servicio, sólo es comandado el primer stage .

```
CLASS CONCRETE PipeManager EXTENDS OF
ComponentManager
{
 PUBLIC VOID init(ASOCIACION[ ] list)
    {
     stages[0] = PipeStage CREATE( list );
    }

 PRIVATE VOID commandStages (ANYTYPE
 datain,ComponentCollector res)
    {
     THREAD stages[0].request(datain,res);
    }
};
```

Los objetos de la clase PipeStage crean el siguiente stage del pipeline durante su fase de inicialización. En la ejecución de su operación request(), un objeto stage comanda directamente al siguiente y es el último el que envía el resultado al objeto Colector cuya referencia se transmite dinámicamente stage por stage.

```
CLASS CONCRETE  PipeStage EXTENDS OF
ComponentStage
{
 PUBLIC VOID init (ASOCIACION[] list)
    {
     stage.init(list);
     IF (!am_i_last)
        {
         otherstages[0] = PipeStage
                  CREATE(TAIL(list));
        }
     }

  PRIVATE VOID commandOtherStages (ANYTYPE
  datain,ComponentCollector res)
    {
     THREAD otherstages[0].request
                          (datain, res);
    }
};
```

IV. Diseño y Construcción del CPAN Farm

La representación del patrón paralelo FARM como un CPAN se muestra en la fig.3.

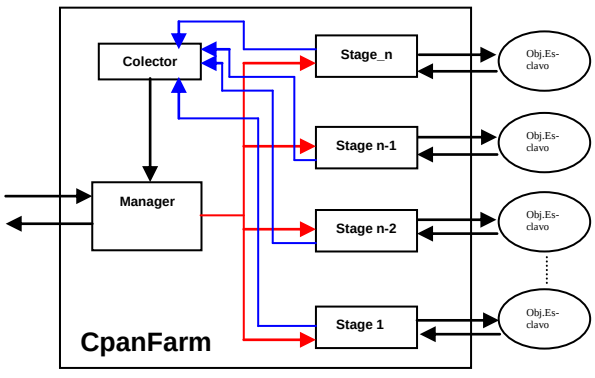


Fig. 3. El Cpan de un Farm

A. *Definición sintáctica y semántica del Cpan Farm*

Una primera política para la composición del FARM es que el manager espere sólo el primer resultado disponible dado por cualquiera de los stages, los cuales responden a una petición de servicio de forma asíncrona.

```
CLASS CONCRETE FarmManager EXTENDS OF
ComponentManager
  {
   VAR
      INT nWorker;

   PUBLIC VOID init (ASOCIACION[] list)
     VAR
       asociacion[] newlist,
       INT i=0;
     {
      WHILE (!(newlist = TAIL(list)))
        {
         stages[i++] = FarmStage CREATE
         (CONS(HEAD(list),NULL));
         list = newlist;
        }
      nWorker = i;
     }

    PRIVATE VOID  commandStages(ANYTYPE
    datain,ComponentCollector res)
      VAR
        INT i;
      {
       FOR i =(0,nWorker)
        {
         THREAD stages[i].request(datain,
         res);
        }
      }
  };
```

La clase concreta FarmManager hereda de ComponentManager. La operación init() crea todos los stages necesarios, mientras que la operación execution() es lanzada en paralelo de forma asíncrona, distribuyendo datos a todos los stages, esperando el primer resultado disponible en el objeto colector. Como en el caso del pipeline, las restricciones de sincronización son heredadas de la clase abstracta ComponentManager.
Los stages del farm son objetos de FarmStage, que heredan de ComponentStage:

```
CLASS CONCRETE FarmStage EXTENDS OF
ComponentStage
    { };
```

Los stages del Farm no se conectan unos con otros. El manager los comanda a todos en su ejecución y el resultado de cada uno es enviado al objeto colector y puesto a disposición del manager. Como los stages son ejecutados en paralelo de forma asíncrona, la política de planificación heredada de la superclase garantiza que el acceso al colector, necesario para retornar los resultados, se hará de una forma sincronizada por parte de dichos objetos.

V. Diseño y Construcción del CPAN TreeDV

La representación del patrón tree que define la técnica de Divide y Vencerás como CPAN tiene su modelo representado en la fig.4.

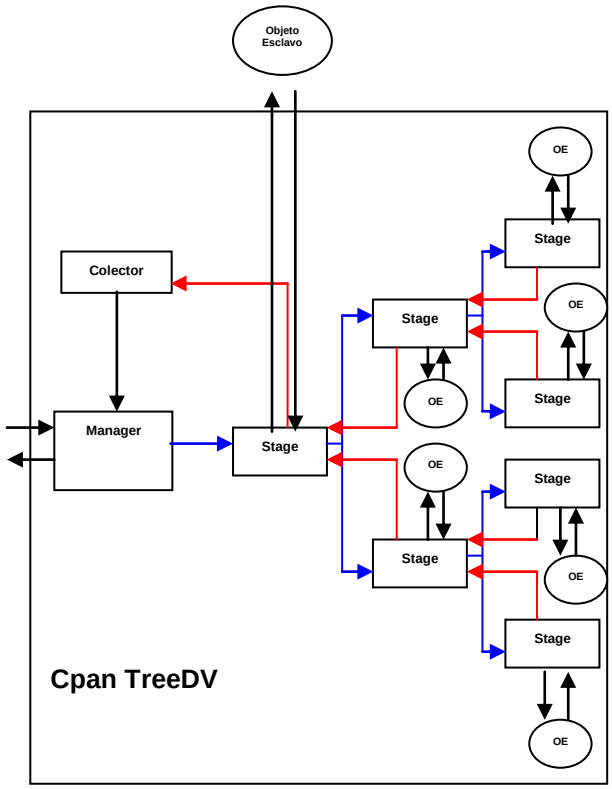


Fig. 4. El Cpan de un TreeDV

A diferencia de los modelos anteriores, donde los objetos esclavos eran predeterminados fuera del modelo CPAN, en este modelo sólo un objeto esclavo es predefinido estáticamente y asociado al primer stage del árbol. Los siguientes objetos esclavos serán creados internamente por los propios stages de forma dinámica.

A. Definición sintáctica y semántica del Cpan TreeDV

La clase TreeDVManager implementa un patrón de comunicación de un árbol binario bajo la técnica de programación Divide y Venceras.

```
CLASS CONCRETE TreeDVManager EXTENDS OF
ComponentManager
{
 PUBLIC VOID init (ASOCIACION[ ] list)
   {
    stages[0] = TreeDVStage CREATE (list);
   }

 PRIVATE VOID  commandStages(ANYTYPE
 datain,ComponentCollector res)
   {
    THREAD stages[0].request(datain,res);
    THREAD res.put(datain);
   }
};
```

Cualquier objeto TreeDVStage se encargará de crear un nodo del árbol binario (izquierdo o derecho). Cuando el nodo raíz o stage inicial ejecuta la operación request() en paralelo, se evalúa el problema con el método del objeto esclavo asociado, retornando la división del problema en dos partes. Posteriormente, se llamará al método commandOtherStages(), quien tomará dichos subproblemas, creará dos nodos stages asociados a su nodo padre, asociándoles a éstos últimos sus respectivos objetos esclavos, que serán creados dinámicamente conforme se vayan creando los nodos del árbol binario, y les enviará a cada uno una parte del problema a resolver, todo ello de forma recursiva.

```
CLASS CONCRETE  TreeDVStage EXTENDS OF
ComponentStage
{
 VAR
    ASOCIACION list;

 PUBLIC VOID init(ASOCIACION[] *list)
   {
    this.list=list;
    stage.init(list);
    am_i_last= FALSE;
   }

 PRIVATE VOID commandOtherStages (ANYTYPE
 datain,ComponentCollector res)
    VAR
      ANYTYPE data_izq, data_der;
   {
    IF(datain.inicio<datain.fin)
      {
       otherStages[0]= TreeDVStage
       CREATE(list);
       otherStages[1]=TreeStage
       CREATE(list);
       THREAD otherStages[0].request(dv(
              datain,datain.inicio,
              datain.medio),res);
       THREAD otherStages[1].request(dv(
              datain,datain.medio+1,
              datain.tam-1),res);
      }
   }

 PRIVATE ANYTYPE dv(ANYTYPE dataout, int
 inicio, int fin)
   {
    datain.inicio=inicio;
    datain.fin=fin;
    RETURN datain;
   }
};
```

VI. RENDIMIENTO DE LOS CPANS

El análisis de speedup de los CPANs Farm, PipeLine y TreeDV aparece en [17] donde se muestra la escalabilidad del rendimiento de los CPANS con 2, 4, 8, 16 y 32 procesadores resolviendo problemas de ordenación y problemas NP-Completos como el del Agente Viajero [5],[15].

VII. DESCRIPCIÓN DE LA BIBLIOTECA DE CLASES DE OBJETOS PARALELOS

La implementación paralela del modelo CPAN se concretiza con la biblioteca de clases de objetos paralelos que se propone y que define los patrones paralelos de comunicación de procesos con los que se han venido trabajando, vistos como Composiciones Paralelas de Alto Nivel: El patrón Farm [13] representado por el Cpan Farm, el patrón Pipeline [13] representado por el Cpan Pipe, el patrón Binary-tree junto con la técnica de Divide y Vencerás[3],

representado por el Cpan TreeDV y el patrón FarmB&B que implementa la técnica de Ramificación y poda representado por el Cpan FarmBB [5],[15],[16].
La biblioteca ha sido implementada en el lenguaje C++ bajo el paradigma de la orientación a objetos en su parte secuencial y para la contraparte paralela se ha utilizado el estándar PThread, para el manejo de hilos y compartición de recursos.
La adopción del enfoque de la orientación a objetos en la implementación de la biblioteca ha hecho posible solucionar el problema de encontrar un conjunto completo de patrones paralelos de comunicación, ya que el paradigma permite añadir nuevos patrones a un conjunto inicial incompleto mediante la definición de subclases. La línea de investigación que se siguió fue el encontrar representaciones de patrones paralelos como clases, a partir de las cuales se pueden instanciar objetos paralelos o CPANS que son, a su vez, ejecutados como consecuencia de una petición de servicio externa a dichos objetos y procedentes de la aplicación de usuario.
La conveniencia de utilizar los PThreads [1] en la implementación de los CPANS radica en el hecho de que los PThreads como paquete de hilos, permiten escribir programas con varios puntos simultáneos de ejecución, sincronizados a través de memoria compartida. La biblioteca de clases de los CPANS se forma principalmente de tres grupos de clases:

El grupo de las clases base necesarias para construir un CPAN, o dicho de otra forma, las clases que implementan los Objetos Paralelos de éste.

TABLA I

CLASES BASE QUE CONFORMAN LOS OBJETOS PARALELOS DE UN CPAN

Object	Define objetos esclavos genéricos y debe ser la superclase de las clases que definen objetos esclavos específicos.
CHilo	Proporciona métodos para iniciar un hilo de ejecución. Es una clase abstracta con un método virtual puro fnHilo( ) que se corresponderá con el hilo de ejecución. Será la clase padre de las clases ComponentCollector, ComponentStage y ComponentManager de un CPAN, las cuales tendrán que redefinir el método fnHilo().
ComponentCollector	Es la clase Concreta que define el componente "collector" de un CPAN y genera objetos colectores que recopilan las soluciones de los objetos "stage" conectados a él.
ComponentStage	Es la clase Abstracta que define el componente "stage" de un CPAN que debe ser especializado y definido en aquellas clases que hereden de ésta.
ComponentManager	Es la clase Abstracta que define el componente "manager" de un CPAN que debe ser especializado y definido en aquellas clases que hereden de ésta.
Util	Fichero de encabezado que contiene la definición de diversas "funciones primitivas" y de varios tipos de datos abstractos necesarios en la implementación de los CPANS

El grupo de las clases que definen los CPANS concretos que constituyen la biblioteca: Cpan Farm, Cpan Pipe, Cpan TreeDV y Cpan FarmBB.

TABLA II

CLASES QUE DEFINEN LOS CPANS FARM, PIPE,TREEDV Y FARMBB

FarmManager	Definen la construcción de un objeto *"manager"* concreto para los patrones *FARM, PIPE, Tree-Divide y Venceras y Farm-Ramificación y poda*, respectivamente. Heredan de la clase base *"ComponentManager"*
PipeManager	
TreeDVManager	
FarmBBManager	
FarmStage,	Definen la construcción de un objeto *"stage"* concreto para los patrones *FARM, PIPE, Tree-Divide y Venceras y Farm-Ramificación y poda*, respectivamente. Heredan de la clase base *"ComponentStage"*
PipeStage	
TreeDVStage	
FarmBBStage	

El grupo de clases definidas por el usuario para la creación de los objetos esclavos a ser trabajados por los CPANS y tipos de datos abstractos utilizados en ellos.

TABLA III

CLASES DE TIPOS DE DATOS ABSTRACTOS PARA LOS CPANS

MyType	Clase definida por el usuario donde se establece el tipo de datos enteros iniciales a procesar así como la estructura de almacenamiento de estos. Esta clase es utilizada para proporcionar el tipo abstracto de datos así como la su estructura de almacenamiento. Elementos que serán utilizados por los *CPAN Farm y Pipe* en su prueba de ejecución .
MyTypeDV	Clase definida por el usuario donde se establece el tipo de datos enteros iniciales a procesar así como el almacenamiento de estos dentro del *CPAN TreeDV* en su prueba de ejecución .
Nodo	Clase definida por el usuario donde se establece el tipo de datos a procesar por un *Cpan FarmBB*: En este caso el tipo *Nodo*, formado de variables que almacenan datos como por ejemplo, el costo acumulado de un nodo, su matriz de costos reducida, el nivel donde se encuentra en el árbol de expansión y el vector de solución que muestra el camino de costo mínimo calculado en ese nodo. Además cuenta con funciones miembro que calculan: la reducción de una matriz de costos, y funciones que imprimen los datos anteriormente citados.

En este caso se han incorporado a la biblioteca a manera de ejemplo, clases que constituyen los objetos esclavos para los CPANS farm, pipe y treeDV, que implementan la solución de problemas de ordenamiento de un conjunto de valores. Para más detalles consultar [14].
Finalmente, la jerarquía de clases que se presenta en la Fig. 5, constituye la biblioteca de los *CPANS* propuestos, proporcionando una gran ventaja al programador al tener la posibilidad de, mediante la herencia, simplificar la definición de nuevos CPANs haciendo uso de la jerarquía inicial definida en biblioteca de clases[7],[14].

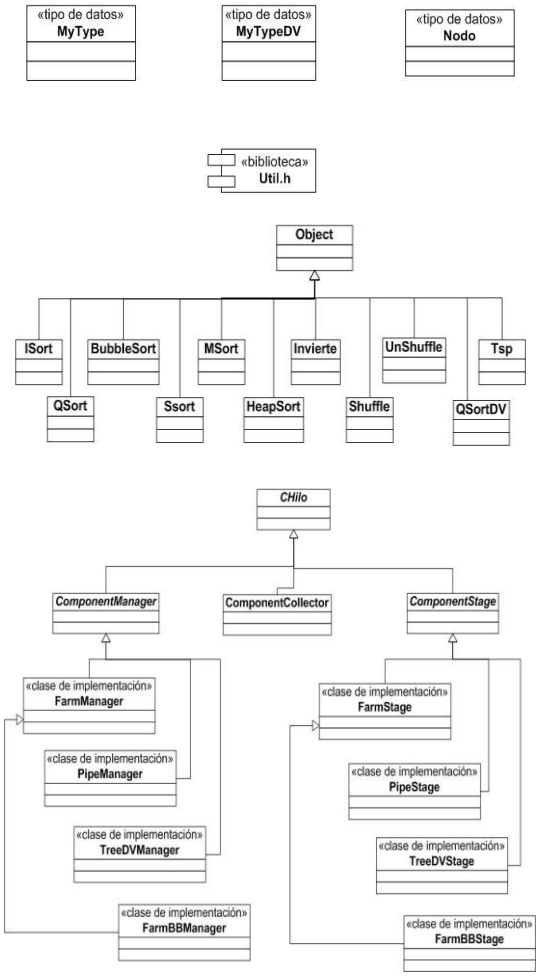


Fig. 5. Jerarquía de clases de la biblioteca de los CPANs

VIII. CONCLUSIONES

1. Se ha desarrollado un método de programación basado en Composiciones Paralelas de Alto Nivel o CPANs [16].
2. Se han implementado los CPANS PipeLine, Farm y TreeDV cuyos detalles están publicados en [17].
3. Los CPANs implementados son reusables gracias a la adopción del enfoque orientado a objetos para definir nuevos patrones utilizando los ya construidos. Para más detalle ver las referencias [15] y [17].
4. Se han transformado algoritmos conocidos que resuelven problemas secuenciales en algoritmos paralelizables.
5. Los CPANs Pipe, Farm y TreeDV conforman una biblioteca de clases.
6. Se han programado las restricciones de sincronización sugeridas por el modelo del Cpan: el paralelismo máximo (MaxPar), la exclusión mutua (Mutex) y la sincronización de comunicación de procesos lectores/escritores (Sync).
7. La programación del modo de comunicación futuro asíncrono para resultados "futuros" dentro de los Cpans se ha llevado a cabo de manera original mediante clases.

REFERENCIAS

[1] Birrell, Andrew, 1989. *An Introduction to programming with threads*. Digital Equipment Corporation, Systems Research Center.
[2] Blelloch, Guy E. 1996. *Programming Parallel Algorithms*. Comunications of the ACM. Volume 39, Number 3.
[3] Brinch Hansen, 1993. *Model Programs for Computational Science: A programming methodology for multicomputers, Concurrency: Practice and Experience*, Volume 5, Number 5.
[4] Brinch Hansen, 1994. SuperPascal- a publication language for parallel scientific computing, Concurrency: Practice and Experience, Volume 6, Number 5.
[5] Capel M.I., Palma A., 1992. *A Programming tool for Distributed Implementation of Branch-and-Bound Algorithms*. Parallel Computing and Transputer Applications. IOS Press/CIMNE. Barcelona.
[6] Capel, M.; Troya J. M., 1994. *An Object-Based Tool and Methodological Approach for Distributed Programming. Software Concepts and Tools*.
[7] Capel M., Rossainz, M., 2004. *A parallel programming methodology based on high level parallel compositions*. Proceedings of the 14th International Conference on Electronics, Communications and Computers, IEEE CS press. 0-7695-2074-X.
[8] Cole, M., 1989. *Algorithmic Skeletons: Structured Managment of Parallel Computation*. The MIT Press.
[9] Corradi A., Leonardi L., 1991. *PO Constraints as tools to synchronize active objects*. Journal Object Oriented Programming 10, pp. 42-53.
[10] Corradi A, Leonardo L, Zambonelli F., 1995. *Experiences toward an Object-Oriented Approach to Structured Parallel Programming*. DEIS technical report no. DEIS-LIA-95-007.
[11] Danelutto, M.; Orlando, S; et al., 1995. *Parallel Programming Models Based on Restricted Computation Structure Approach*. Technical Report-Dpt. Informatica. Universitá de Pisa.
[12] Darlington et al., 1993, *Parallel Programming Using Skeleton Functions*. Proceedings PARLE'93, Munich (D).
[13] Roosta, Séller, 1999. *Parallel Processing and Parallel Algorithms*. Theory and Computation. Springer.
[14] Rossainz, M., 2005. *Una Metodología de Programación Basada en Composiciones Paralelas de Alto Nivel (CPANs)*. Universidad de Granada, PhD dissertation, 02/25/2005.
[15] Rossainz, M., Capel M., 2006. *Design and Implementation of the Branch & Bound Algorithmic Design Technique as an High Level Parallel Composition*. International Mediterranean Modelling Multiconference.Barcelona, Spain.
[16] Rossainz, M., Capel M., 2008. *A Parallel Programming Methodology using Communication Patterns named CPANS or Composition of Parallel Object*. 20TH European Modeling & Simulation Symposium.Campora S. Giovanni. Italy.
[17] Rossainz, M., Capel M., 2012. *Compositions of Parallel Objects to Implement Communication Patterns*. XXIII Jornadas de Paralelismo. SARTECO 2012. Septiembre de 2012. Elche, España.

Paralelizando una Aproximación Multiobjetivo y Bioinspirada para Filogenética Usando Esquemas Híbridos MPI/OpenMP

Sergio Santander-Jiménez[1] y Miguel A. Vega-Rodríguez[1].

Resumen— **La inferencia filogenética es uno de los problemas más importantes en Bioinformática. Recientes estudios han planteado su resolución mediante técnicas de optimización multiobjetivo, con objeto de solventar los problemas que surgen cuando distintas técnicas filogenéticas dan lugar a relaciones evolutivas conflictivas entre sí. En este sentido, resulta esencial la combinación de computación bioinspirada y paralela para afrontar la complejidad computacional de esta nueva formulación del problema. En este artículo proponemos el empleo de esquemas híbridos basados en MPI y OpenMP para paralelizar en arquitecturas tipo clúster un algoritmo multiobjetivo inspirado en el comportamiento de las luciérnagas aplicado a la inferencia de historias evolutivas. La experimentación realizada sobre cuatro bases de datos reales muestra que el algoritmo puede lograr significativos factores de aceleración y eficiencias mediante el empleo del modelo más apropiado para explotar el paralelismo en los niveles de inferencia y evaluación de soluciones.**

Palabras clave— **Computación con Clusters, Inteligencia de Enjambre, Optimización Multiobjetivo, Inferencia Filogenética.**

I. Introducción

La creciente necesidad de afrontar complejos retos computacionales en una amplia variedad de ámbitos científicos ha dado lugar a avances significativos en el desarrollo de arquitecturas clúster multicore. Mediante el empleo de técnicas de computación con clusters, es posible desarrollar nuevas estrategias paralelas para resolver problemas NP-completos en los que el espacio de búsqueda crece exponencialmente con el tamaño de los datos de entrada. Estos diseños algorítmicos tratan de maximizar el rendimiento paralelo explotando las características de los paradigmas de memoria compartida y memoria distribuida, siendo especialmente populares los esquemas híbridos basados en OpenMP y MPI para afrontar satisfactoriamente este tipo de problemas [1].

Múltiples ámbitos de investigación en bioinformática pueden beneficiarse de la aplicación de técnicas de computación paralela. La inferencia de relaciones evolutivas entre especies representa uno de los problemas de optimización más complejos en este campo. Su complejidad viene dada por el hecho de que el espacio de búsqueda de posibles topologías filogenéticas crece exponencialmente con el número de secuencias moleculares a estudiar [2]. En este contexto, resulta inviable la aplicación de técnicas de búsqueda exhaustiva por sus prohibitivos tiempos de ejecución. Para llevar a cabo el análisis de bases de datos biológicas cada vez más complejas, es necesario desarrollar nuevas aproximaciones basadas en computación de altas prestaciones [3]. Bader et al. resumieron en [4] diversas estrategias para afrontar el problema, definiendo un modelo jerárquico con distintos niveles de granularidad para el desarrollo de procedimientos filogenéticos paralelos.

Por otro lado, al formular la inferencia filogenética como un problema de optimización, se pueden utilizar numerosos criterios de optimalidad [5]. Sin embargo, criterios diferentes pueden dar lugar a hipótesis evolutivas conflictivas entre sí. La resolución de este problema exige el empleo de técnicas de optimización multiobjetivo [6], las cuales pueden beneficiarse del paralelismo a fin de disminuir los tiempos requeridos para realizar búsquedas filogenéticas conforme a múltiples criterios simultáneamente [7].

En este artículo, afrontamos el problema de la inferencia filogenética conforme a dos criterios: parsimonia y verosimilitud. Para ello, proponemos distintos diseños híbridos basados en OpenMP y MPI para paralelizar un algoritmo multiobjetivo de inteligencia de enjambre, el Multiobjective Firefly Algorithm (MO-FA). Cada propuesta planteada pretende paralelizar el algoritmo en los niveles de inferencia y evaluación de soluciones [3], usando para ello distintas estructuras de datos para comunicar tareas entre procesos bajo un esquema *master-worker*. El objetivo principal es seleccionar el diseño paralelo más eficiente, identificando los factores que puedan tener un impacto negativo en la escalabilidad. Estos diseños paralelos han sido evaluados utilizando las métricas de aceleración y eficiencia, efectuando experimentos sobre cuatro bases de datos reales de nucleótidos.

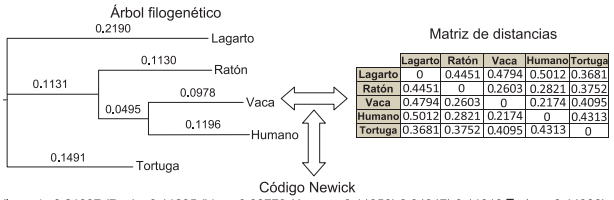
Este artículo está organizado como sigue. En la siguiente sección, resumimos los fundamentos de la inferencia filogenética. La Sección III detalla diversos esquemas híbridos para paralelizar el algoritmo MO-FA. En la Sección IV, explicamos nuestra metodología experimental y presentamos resultados paralelos y biológicos. Finalmente, la Sección V proporciona conclusiones y define líneas de trabajo futuro.

II. Reconstrucción Filogenética

Dado un conjunto de N secuencias moleculares procedentes de distintos organismos en la naturaleza, los procedimientos filogenéticos tratan de inferir una estructura tipo árbol $T = (V, E)$ que describe las relaciones ancestro-descendiente entre las espe-

[1]Universidad de Extremadura, Dept. Tecnología de los Computadores y de las Comunicaciones, Escuela Politécnica. Campus Universitario s/n, 10003, Cáceres, España. {sesaji, mavega}@unex.es

cies estudiadas. Cada organismo del conjunto de entrada viene caracterizado por una cadena de S caracteres de un alfabeto Σ que representan nucleótidos, aminoácidos u otros datos biológicos. En un árbol filogenético, las especies de entrada se localizan en las hojas de T, sus posibles ancestros se representan como nodos internos en V, y las relaciones ancestrales se modelan usando las ramas del conjunto E, que definen tiempos evolutivos por medio de valores de longitud de rama. La representación de árboles filogenéticos puede llevarse a cabo de diversas maneras. La Figura 1 muestra una topología filogenética y sus representaciones correspondientes usando códigos Newick [2] y matrices de distancias genéticas [5].



	Lagarto	Ratón	Vaca	Humano	Tortuga
Lagarto	0	0.4451	0.4794	0.5012	0.3681
Ratón	0.4451	0	0.2603	0.2821	0.3752
Vaca	0.4794	0.2603	0	0.2174	0.4095
Humano	0.5012	0.2821	0.2174	0	0.4313
Tortuga	0.3681	0.3752	0.4095	0.4313	0

Fig. 1: Representando árboles filogenéticos

Múltiples propuestas han permitido definir la inferencia de árboles filogenéticos conforme a criterios de optimalidad. Las aproximaciones multiobjetivo tratan de inferir un conjunto de soluciones *Pareto* que representan un compromiso entre criterios dispares. Nos centraremos en dos de los principios más populares: máxima parsimonia y máxima verosimilitud.

Las aproximaciones basadas en parsimonia tratan de inferir la explicación más sencilla a la evolución de las especies de entrada. Dado un conjunto de datos compuesto por N secuencias de S caracteres, el criterio de máxima parsimonia busca aquel árbol filogenético $T = (V, E)$ que minimice el número de cambios a nivel molecular a lo largo de su topología:

$$P(T) = \sum_{i=1}^{S} \sum_{(a,b)\in E} C(a_i, b_i), \qquad (1)$$

donde $(a, b) \in E$ representa la rama que relaciona a las especies $a, b \in V$, a_i and b_i son los valores del i-ésimo carácter en las secuencias correspondientes a a y b, y $C(a_i, b_i)$ es el coste evolutivo entre a_i y b_i.

Por su parte, los métodos de máxima verosimilitud fueron propuestos con la idea de reconstruir la hipótesis evolutiva más probable que explique los datos observados. Dado un conjunto de N cadenas moleculares de longitud S y un modelo evolutivo m que define las probabilidades de eventos de mutación a nivel molecular, estos métodos buscan aquel árbol filogenético $T = (V, E)$ que maximice la función de verosimilitud, definida del siguiente modo:

$$L[D, T, m] = \Pr[D|T, m] = \prod_{i=1}^{S} \prod_{j=1}^{E} (r_i t_j)^{n_{ij}} \qquad (2)$$

siendo r_i la probabilidad de mutación para el carácter i-ésimo, t_j el tiempo evolutivo entre los nodos emparentados por la rama $j \in E$, y n_{ij} el número de eventos de mutación observados entre los organismos conectados por j en el carácter i-ésimo. Dado que las búsquedas de topologías óptimas según los criterios de parsimonia y verosimilitud son consideradas como problemas NP-completos ([8], [9]), la inferencia y evaluación de árboles filogenéticos desde una perspectiva multiobjetivo debe ser afrontada mediante la aplicación de metaheurísticas paralelas.

III. Una Propuesta Multiobjetivo y Paralela Inspirada en las Luciérnagas

En esta sección, explicamos los conceptos básicos del algoritmo propuesto, Multiobjective Firefly Algorithm (MO-FA), e introducimos varios esquemas para su paralelización sobre clusters multicore. El Firefly Algorithm (FA) [10] es un algoritmo de inteligencia de enjambre inspirado en la bioluminiscencia de las luciérnagas para la resolución de problemas de optimización. Las luciérnagas hacen uso de un sistema de comunicación basado en la emisión de luz para atraer otras luciérnagas a su posición. Aquellas luciérnagas con los patrones más intensos de emisión de luz atraerán a un mayor número de luciérnagas, teniendo en cuenta factores como la distancia entre luciérnagas y la absorción de luz por parte del entorno. Este comportamiento se modela identificando la intensidad de la luz emitida con la calidad de una posible solución a un problema. MO-FA extiende el diseño del FA para resolver problemas de optimización multiobjetivo, utilizando conceptos como el de dominancia [6] para decidir qué luciérnagas muestran los patrones de luz más atractivos, esto es, las mejores soluciones desde una perspectiva multiobjetivo.

Con objeto de adaptar MO-FA al problema de la inferencia filogenética, emplearemos matrices de distancias para representar los individuos en la población. El algoritmo moverá las luciérnagas hacia las soluciones más prometedoras operando sobre las distancias genéticas y aplicando el método BIONJ para reconstruir las correspondientes topologías filogenéticas [11], las cuales serán optimizadas utilizando para ello métodos de búsqueda topológica [2].

A lo largo de la ejecución, las matrices de distancias serán actualizadas para mover el enjambre hacia las mejores soluciones. Dada una luciérnaga X_r en un enjambre X, dominada por otra luciérnaga X_s, con matrices de distancias $X_r.M$ y $X_s.M$, el algoritmo moverá X_r hacia X_s utilizando la siguiente ecuación:

$$\begin{aligned} X_r.M[i,j] =& X_r.M[i,j] + \beta_0 e^{-\gamma \delta_{rs}^2} (X_s.M[i,j] \\ & - X_r.M[i,j]) + \alpha(rand[0,1] - \frac{1}{2}) \end{aligned} \qquad (3)$$

donde $M[i, j]$ representa la distancia genética entre dos especies i y j, δ_{rs} la distancia global entre X_r y X_s, β_0 un factor de atracción, γ el coeficiente de absorción de luz y α un factor aleatorio que permite introducir aleatoriedad al movimiento de las luciérnagas. Estos tres últimos parámetros representan parámetros de entrada del algoritmo. Por su

parte, δ_{rs} se calcula a partir de $X_r.M$ y $X_s.M$ como $\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{i}(X_r.M[i,j]-X_s.M[i,j])^2}$, siendo N el número de especies en el conjunto de datos de entrada. La Ecuación 3 es aplicada sobre todas las entradas i, j en $X_r.M$, generando a partir de la matriz resultado un nuevo árbol $X_r.T$ que será evaluado según los criterios de parsimonia y verosimilitud. En cada generación, se actualiza el conjunto de Pareto con las mejores soluciones encontradas.

Dado que los algoritmos de inteligencia de enjambre presentan características que los hacen idóneos para su paralelización [10], es posible explotar las características de las arquitecturas clúster para paralelizar MO-FA en los niveles de inferencia y evaluación utilizando aproximaciones de grano grueso [4]. Varias operaciones, tales como las llamadas a los métodos de reconstrucción y evaluación, requieren un mayor esfuerzo computacional según aumenta el número y la longitud de las secuencias moleculares a procesar. Para afrontar dicha complejidad, proponemos el uso de modelos *master-worker* con paralelización a nivel inter-nodo e intra-nodo basada en MPI y OpenMP.

Dada una arquitectura compuesta por varios nodos multicore interconectados mediante una red de comunicaciones, un modelo híbrido *master-worker* inicializa *nproc* procesos MPI, de los cuales $nproc-1$ procesos tomarán el rol de trabajadores, esperando por las tareas que les asigne un proceso maestro. Cada proceso usará hasta *num_threads* hilos OpenMP para paralelizar las operaciones más costosas en las tareas asignadas. En nuestro modelo, el maestro no permanecerá ocioso y llevará a cabo la parte proporcional de la carga de trabajo que le corresponda, evitando recursos desaprovechados. Para aplicar este modelo a MO-FA, es preciso diseñar tareas apropiadas y decidir cómo comunicarlas. Por ello, proponemos dos esquemas paralelos con diferentes cargas de trabajo y dos variantes por cada esquema según la estructura de datos usada para comunicar tareas.

A. Primera Aproximación Híbrida

Se pueden distinguir tres pasos principales en MO-FA: la actualización de matrices de distancia para modelar el movimiento de las luciérnagas, la reconstrucción de árboles filogenéticos a partir de las matrices procesadas y la evaluación de las topologías generadas. En esta primera propuesta, dada por el Algoritmo 1, cada proceso realizará el paso de movimiento de luciérnagas y las llamadas a los procedimientos de construcción y evaluación de árboles para *indexDom.size/nproc* luciérnagas, siendo *indexDom* un array de enteros que contiene los índices a las luciérnagas dominadas, identificadas por el proceso maestro (línea 4 en el Algoritmo 1). De este modo, el maestro debe enviar a cada trabajador no solo la estructura *indexDom*, sino también la población completa de luciérnagas (*tamañoEnjambre* individuos), con objeto de actualizar las luciérnagas dominadas según la información proporcionada por el enjambre. Cada proceso paralelizará su carga de trabajo (líneas 13-19) usando OpenMP para aprovechar los recursos multicore, definiéndose una estructura auxiliar que almacene el estado actual del enjambre para evitar posibles riesgos de lectura/escritura.

Algorithm 1 Esquema Paralelo A

```
1:  #pragma omp parallel num_threads(num_threads)
2:  for numGen = 1 to maxGeneraciones do
3:    if proceso maestro then
4:      Comprobar luciérnagas dominadas (indexDom, X)
5:      /* ∀i : i = 2 to nproc */
6:      MPI_Send (X, tamañoEnjambre, i)
7:      MPI_Send (indexDom, indexDom.size/nproc, i)
8:    else
9:      MPI_Recv (X, tamañoEnjambre, maestro)
10:     MPI_Recv (indexDom, indexDom.size/nproc, maestro)
11:   end if
12:   [Inicializar matrices de distancias (X)] /*solo en A.2*/
13:   #pragma omp for
14:   for i = 1 to indexDom.size/nproc do
15:     j ← indexDom.getIndex (i)
16:     Actualizar luciérnaga dominada (X_j, X, β_0, γ, α)
17:     Reconstruir y optimizar X_j.T (X_j.M)
18:     Evaluar árbol (X_j.T)
19:   end for
20:   if proceso maestro then
21:     /* ∀i : i = 2 to nproc */
22:     MPI_Recv (X, indexDom.size/nproc, i)
23:     MPI_Recv (X.Scores, indexDom.size/nproc, i)
24:     Actualizar conjunto de Pareto (X)
25:   else
26:     MPI_Send (X, indexDom.size/nproc, maestro)
27:     MPI_Send (X.Scores, indexDom.size/nproc, maestro)
28:   end if
29: end for
```

Al asociar las luciérnagas con topologías filogenéticas, podemos comunicar los individuos a procesar usando dos representaciones: matrices de distancia o códigos Newick. Denotamos como Esquema A.1 la variante del Algoritmo 1 que hace uso de matrices de distancias para comunicar individuos, y como Esquema A.2 la variante basada en Newick. Ambas representaciones presentan ventajas e inconvenientes. Por ejemplo, dado que los tiempos para comunicar matrices de tipo double de tamaño NxN aumentan significativamente en instancias con un elevado número de especies, puede ser útil emplear cadenas Newick para reducir estos tiempos. Por otra parte, ya que MO-FA opera sobre distancias, el Esquema A.2 debe introducir carga de trabajo adicional para generar matrices a partir de los códigos Newick (línea 12).

B. Segunda Aproximación Híbrida

El segundo esquema considerado se define con el ánimo de reducir los tiempos por comunicaciones. Una de las principales fuentes de comunicación en el primer esquema venía dado por el paso de actualización, que requería una copia del estado actual del enjambre para cada proceso. En este segundo diseño, dado por el Algoritmo 2, el proceso maestro se encargará de realizar este paso, usando *num_threads* hilos OpenMP (líneas 6-11 en Algoritmo 2). Tras completar esta tarea, la evaluación de luciérnagas será distribuida entre los procesos MPI, asignando *indexDom.size/nproc* individuos a cada una. A su vez, se paraleliza el bucle de optimización y evaluación (líneas 17-21) usando la directiva *#pragma omp for*, devolviendo los resultados obtenidos al maestro.

Podemos emplear dos variantes de este esquema para distribuir entre los procesos las soluciones asociadas a las luciérnagas actualizadas: ma-

Algorithm 2 Esquema Paralelo B

```
1:  #pragma omp parallel num_threads(num_threads)
2:  for numGen = 1 to maxGeneraciones do
3:    if proceso maestro then
4:      Comprobar luciérnagas dominadas (indexDom, X)
5:      [Inicializar matrices de distancias (X)] /*solo en B.2*/
6:      #pragma omp for
7:      for i = 1 to indexDom.size do
8:        j ← indexDom.getIndex (i)
9:        Actualizar luciérnaga dominada (X_j, X, β_0, γ, α)
10:       [Reconstruir X_j.T (X_j.M)] /*solo en B.2*/
11:     end for
12:     /* ∀i : i = 2 to nproc */
13:     MPI_Send (X, indexDom.size/nproc, i)
14:   else
15:     MPI_Recv (X, indexDom.size/nproc, maestro)
16:   end if
17:   #pragma omp for
18:   for i = 1 to indexDom.size/nproc do
19:     [Reconstruir X_i.T (X_i.M)] /*solo en B.1*/
20:     Optimizar y evaluar árbol (X_i.T)
21:   end for
22:   if proceso maestro then
23:     /* ∀i : i = 2 to nproc */
24:     MPI_Recv (X, indexDom.size/nproc, i)
25:     MPI_Recv (X.Scores, indexDom.size/nproc, i)
26:     Actualizar conjunto de Pareto (X)
27:   else
28:     MPI_Send (X, indexDom.size/nproc, maestro)
29:     MPI_Send (X.Scores, indexDom.size/nproc, maestro)
30:   end if
31: end for
```

trices de distancias (Esquema B.1) y cadenas Newick (Esquema B.2). En el Esquema B.1, un conjunto de *indexDom.size/nproc* matrices de distancias será enviado a cada trabajador, reconstruyendo las correspondientes topologías filogenéticas en el bucle de evaluación (línea 19). Dado que el paso de optimización puede dar lugar a cambios en las distancias, las matrices actualizadas son devueltas al maestro junto con los valores de las funciones objetivo.

En cuanto a la segunda variante, dado que los códigos Newick deben calcularse a partir de topologías filogenéticas, el Esquema B.2 invoca el método de reconstrucción de árboles antes de realizar el reparto de cadenas Newick entre procesos, siendo el proceso maestro responsable de efectuar estas llamadas en esta variante (línea 10). Los trabajadores recibirán los códigos Newick resultantes, realizando posteriormente los pasos de optimización y evaluación.

IV. Resultados Experimentales

Introducimos en esta sección nuestra metodología experimental y mostramos los resultados obtenidos por cada uno de los esquemas propuestos de acuerdo a las métricas de aceleración y eficiencia. De manera adicional, comparamos nuestros resultados paralelos y biológicos con otros métodos multiobjetivo y biológicos de la literatura. Los experimentos dirigidos en este estudio se han realizado sobre una arquitectura clúster compuesta por seis nodos de cómputo, cada uno formado por dos procesadores Intel Quad Xeon CPU E5410 a 2.33Ghz (8 cores por nodo, un total de 48 cores disponibles) con 8GB DDR2 RAM, bajo Scientific Linux 6.1, e interconectados mediante una red Giga-Ethernet. Nuestro software fue compilado usando GCC 4.4.5 y las librerías MPICH2 1.4.1.

Los experimentos han sido realizados sobre cuatro bases de datos reales de nucleótidos [7]: *rbcL_55*, 55 secuencias (1314 nucleótidos por secuencia) del gen rbcL de plantas verdes, *mtDNA_186*, 186 secuencias (16608 nucleótidos) de ADN mitocondrial humano, *RDPII_218*, 218 secuencias (4182 nucleótidos) de ARN procariota, y *ZILLA_500*, 500 secuencias (759 nucleótidos) del gen cloroplástico rbcL. Los parámetros de entrada de MO-FA fueron configurados mediante experimentación, dando como resultado los siguientes valores: tamañoEnjambre=100, maxGeneraciones=100, β_0=1, γ=0,5, and α=0,05.

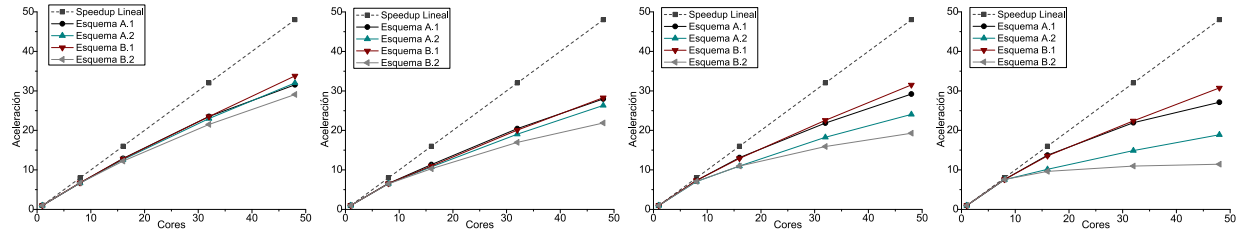
A. Resultados Paralelos

Para evaluar la escalabilidad de cada diseño, hemos realizado 10 ejecuciones completas por esquema e instancia usando las siguientes configuraciones: 1 proceso MPI - 8 hilos OpenMP (8 cores), 2 procesos MPI - 8 hilos OpenMP (16 cores), 4 procesos MPI - 8 hilos OpenMP (32 cores), y 6 procesos MPI - 8 hilos OpenMP (48 cores). La Tabla I presenta los factores de aceleración medios (columnas 2, 4, 6 y 8) y valores de eficiencia (columnas 3, 5, 7 y 9) para cada aproximación paralela, según la base de datos analizada y el número de cores, así como los tiempos medios secuenciales requeridos. Estos resultados sugieren que los esquemas basados en la comunicación de matrices de distancias (esquemas A.1 y B.1) superan a las propuestas basadas en Newick (A.2 y B.2), especialmente al aumentar el número de especies.

TABLA I: Factores de aceleración y eficiencias.

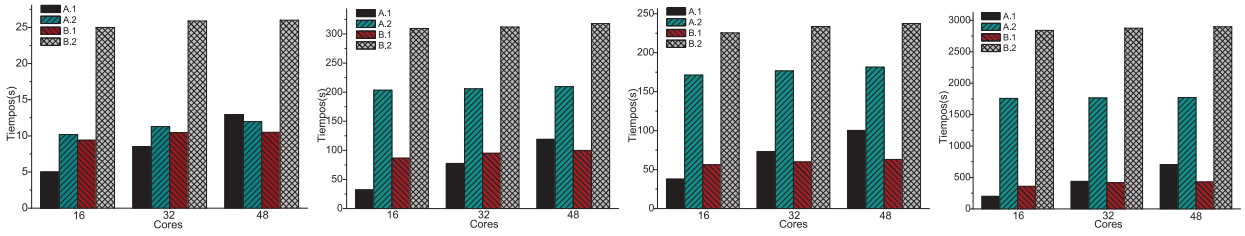
rbcL_55 (tiempo secuencial medio = 4910,63s)								
	8 cores		16 cores		32 cores		48 cores	
Esq.	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)
A.1	6,705	83,818	**12,899**	**80,619**	**23,509**	**73,466**	31,550	65,729
A.2	6,682	83,525	12,604	78,776	23,004	71,888	31,997	66,661
B.1	**6,750**	**84,374**	12,819	80,120	23,489	73,402	**33,780**	**70,374**
B.2	6,701	83,761	12,228	76,4226	21,498	67,181	29,087	60,598
mtDNA_186 (tiempo secuencial medio = 35801,83s)								
	8 cores		16 cores		32 cores		48 cores	
Esq.	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)
A.1	**6,569**	**82,110**	**11,352**	**70,952**	**20,464**	**63,948**	27,968	58,266
A.2	6,521	81,509	10,753	67,203	19,043	59,509	26,327	54,848
B.1	6,564	82,052	11,001	68,757	20,081	62,754	**28,303**	**58,964**
B.2	6,481	81,015	10,291	64,320	16,975	53,046	21,897	45,620
RDPII_218 (tiempo secuencial medio = 23375,27s)								
	8 cores		16 cores		32 cores		48 cores	
Esq.	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)
A.1	7,323	91,536	**13,107**	**81,918**	21,855	68,298	29,195	60,822
A.2	7,062	88,274	10,969	68,556	18,263	57,071	24,055	50,115
B.1	**7,343**	**91,785**	12,906	80,659	**22,561**	**70,503**	**31,508**	**65,642**
B.2	7,082	88,522	10,962	68,515	15,929	49,777	19,269	40,143
ZILLA_500 (tiempo secuencial medio = 64244,58s)								
	8 cores		16 cores		32 cores		48 cores	
Esq.	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)	Acel.	Ef.(%)
A.1	**7,590**	**94,870**	**13,725**	**85,783**	21,938	68,558	27,110	56,479
A.2	7,505	93,809	10,153	63,459	14,897	46,551	18,889	39,352
B.1	7,587	94,832	13,545	84,654	**22,412**	**70,039**	**30,753**	**64,069**
B.2	7,511	93,883	9,635	60,220	10,982	34,317	11,426	23,804

Mientras que el Esquema A.1 obtiene eficiencias prometedoras considerando un bajo número de procesos MPI, el Esquema B.1 muestra los mejores resultados paralelos cuando se utilizan todos los recursos hardware disponibles. Como muestra la Tabla I, el Esquema B.1 motiva un aumento medio de la eficiencia en un 4,43825 % para 48 cores con respecto al Esquema A.1. Con respecto a las aproximaciones basadas en Newick, aún pudiendo dar lugar a resultados significativos para *rbcL_55*, los valores de eficiencia resultantes caen de manera drástica a medida que aumenta la complejidad de la instancia analizada. La Figura 2 presenta una comparativa gráfica entre



(a) $rbcL\_55$ (b) $mtDNA\_186$ (c) $RDPII\_218$ (d) $ZILLA\_500$

Fig. 2: Comparativa de factores de aceleración



(a) $rbcL\_55$ (b) $mtDNA\_186$ (c) $RDPII\_218$ (d) $ZILLA\_500$

Fig. 3: Overheads y tiempos no paralelizables para cada propuesta

los factores de aceleración obtenidos por estos cuatro esquemas según aumenta el número de cores.

Para justificar estos resultados, es necesario considerar una serie de factores que pueden afectar el rendimiento de los sistemas híbridos [12], como los patrones de comunicación entre procesos MPI, los tiempos requeridos para realizar tareas en el proceso maestro y los tiempos no paralelizables debido a la gestión de secciones críticas e hilos OpenMP. La Figura 3 muestra cómo estos factores evolucionan con el número de cores para cada esquema.

El Esquema A.1 reduce el impacto de estos factores en los tiempos de ejecución cuando se consideran 2 procesos MPI sobre $RDPII\_218$ y $ZILLA\_500$, y 2 y 4 procesos MPI sobre $rbcL\_55$ y $mtDNA\_186$. Sin embargo, a medida que aumentamos el número de procesos MPI, los tiempos necesarios para comunicar a cada trabajador las matrices de distancias de toda la población de luciérnagas crece de manera significativa, disminuyendo el rendimiento al usar toda la estructura hardware. El Esquema B.1 resuelve este problema, mostrando una escalabilidad prometedora en cada una de las bases de datos. Este comportamiento viene motivado por los tiempos de actualización de luciérnagas en el maestro (que permanecen constantes) y el reducido overhead por comunicaciones mostrado en comparación con el Esquema A.1. Por su parte, los Esquemas A.2 y B.2 son los que sufren penalizaciones más notables en el rendimiento, debido a los tiempos requeridos para inicializar las matrices de distancia a partir de los códigos Newick (A.2) y la complejidad computacional que conlleva el método de construcción de árboles BIONJ [11] ejecutado en el proceso maestro (B.2).

Por lo tanto, aunque el envío de códigos Newick precisa menores tiempos de comunicación, los elevados tiempos de procesamiento necesarios para realizar la conversión de Newick a distancias (y viceversa) explican el pobre rendimiento de los esquemas basados en Newick en instancias con un elevado número de especies. Con respecto a los esquemas basados en matrices, los factores críticos presentados en la Figura 3 explican la mejora en la escalabilidad que supone el Esquema B.1. Esta propuesta permite a MO-FA alcanzar factores de aceleración notables, reduciendo los tiempos requeridos para realizar análisis filogenéticos multiobjetivo en bases de datos reales.

Con ánimo de evaluar la calidad del rendimiento paralelo de la propuesta, presentamos en la Tabla II una comparativa con dos aproximaciones desarrolladas por Cancino et al. para paralelizar su software multiobjetivo para reconstrucción filogenética, PhyloMOEA. Estos autores propusieron dos diseños diferentes [7]: un modelo *master-worker* basado en MPI y una propuesta híbrida OpenMP/MPI basada en paralelismo de grano fino para reducir los tiempos de cómputo de la función de verosimilitud. De acuerdo a la Tabla II, los factores de aceleración obtenidos por nuestro diseño paralelo según el Esquema B.1 mejoran significativamente los valores publicados por Cancino et al. usando 16 cores en [7], especialmente para la instancia con mayor número de especies.

TABLA II: Comparativa paralela MO-FA - PhyloMOEA (usando 16 cores ambas aproximaciones)

	MO-FA	PhyloMOEA	
Instancia	MPI-OpenMP	MPI	MPI-OpenMP
$rbcL\_55$	**12,82**	7,30	8,30
$mtDNA\_186$	**11,00**	7,40	8,50
$RDPII\_218$	**12,91**	9,80	10,20
$ZILLA\_500$	**13,55**	6,70	6,30

B. Resultados Multiobjetivo y Biológicos

En esta subsección, mostramos los resultados multiobjetivo y biológicos de MO-FA, estableciendo comparaciones con PhyloMOEA [7] y dos métodos biológicos del estado del arte: TNT (máxima parsimonia) [13] y RAxML (máxima verosimilitud) [14].

El rendimiento multiobjetivo bajo el modelo evolutivo $HKY85+\Gamma$ ha sido evaluado utilizando la conocida métrica de hipervolumen [6]. La Tabla III muestran los valores de hipervolumen medios obtenidos (columnas 2-3), así como los puntos de referencia usados para su cálculo. De acuerdo a esta tabla, MO-FA obtiene porcentajes notables de hipervolumen (por encima del 69 %) para cada base de datos.

TABLA III: Valores de hipervolumen

	Hipervolumen		Ref. ideal		Ref. nadir	
Instancia	Media	Desv. Est.	Pars.	Veros.	Pars.	Veros.
rbcL_55	70,14 %	0,06	4774	-21569,69	5279	-23551,42
mtDNA_186	69,66 %	0,01	2376	-39272,20	2656	-43923,99
RDPII_218	74,09 %	0,18	40658	-132739,90	45841	-147224,59
ZILLA_500	69,07 %	0,03	15893	-79798,03	17588	-87876,39

En la Tabla IV, resumimos los resultados filogenéticos de la ejecución que obtuvo el valor de hipervolumen más cercano a la media global. Por un lado, MO-FA alcanza los valores de máxima parsimonia proporcionados por la herramienta de referencia TNT. A su vez, las comparativas bajo el modelo $HKY85+\Gamma$ muestran una mejora significativa con respecto a los mejores valores de parsimonia y verosimilitud alcanzados por PhyloMOEA. Finalmente, la comparativa de verosimilitud con RAxML usando el modelo $GTR+\Gamma$ indica que los resultados de MO-FA pueden dominar a las topologías inferidas por RAxML en gran parte de las bases de datos analizadas. En conclusión, este estudio da cuenta de la bondad de aplicar una aproximación multiobjetivo, bioinspirada y paralela a filogenética.

TABLA IV: Resultados biológicos

	MO-FA					
	Mejor árbol parsimonia HKY85		Mejor árbol verosimilitud HKY85		Mejor árbol verosimilitud GTR	
Instancia	Pars.	Veros.	Pars.	Veros.	Pars.	Veros.
rbcL_55	**4874**	-21843,19	4889	**-21819,09**	**4890**	**-21788,34**
mtDNA_186	**2431**	-39954,29	2445	**-39889,14**	**2452**	**-39868,96**
RDPII_218	**41488**	-136324,43	42813	**-134167,67**	**42827**	-134097,86
ZILLA_500	**16218**	-81391,40	16309	**-80966,51**	**16304**	**-80607,16**

	Mejor parsimonia		Mejor verosimilitud		
	TNT	PhyloMOEA	PhyloMOEA-HKY85	RAxML-GTR	
Instancia	Pars.	Pars.	Veros.	Pars.	Veros.
rbcL_55	**4874**	**4874**	-21889,84	4893	-21791,98
mtDNA_186	**2431**	2437	-39896,44	2453	-39869,63
RDPII_218	**41488**	41534	-134696,53	42894	**-134079,42**
ZILLA_500	**16218**	16219	-81018,06	16305	-80623,50

V. Conclusiones

Hemos propuesto en este artículo distintos diseños híbridos OpenMP/MPI para paralelizar un algoritmo multiobjetivo de inteligencia de enjambre para la inferencia filogenética. Estas aproximaciones fueron definidas en base a distintas cargas de trabajo y patrones de comunicación utilizando dos conocidas representaciones filogenéticas: matrices de distancias y códigos Newick. Los experimentos realizados sobre bases de datos biológicas reales han mostrado diferencias significativas a nivel de rendimiento paralelo entre los distintos esquemas. Considerando los factores críticos que influyen en el rendimiento de los sistemas híbridos, hemos desarrollado un diseño paralelo que permite obtener factores de aceleración significativos sobre una arquitectura clúster multicore. La bondad de la aproximación ha sido evaluada mediante comparativas con otros algoritmos paralelos híbridos y métodos biológicos del estado del arte.

Como trabajo futuro, proponemos el desarrollo de otras metaheurísticas multiobjetivo paralelas para inferencia filogenética. Con ello, se pretende encontrar aquella propuesta que permita maximizar el rendimiento desde un punto de vista computacional y biológico haciendo un uso eficiente de los recursos hardware disponibles. Otra posible línea de investigación es el desarrollo de aproximaciones basadas en modelo de islas y equipos paralelos de algoritmos bioinspirados para mejorar los resultados filogenéticos.

Agradecimientos

Este trabajo está parcialmente financiado por el Ministerio de Economía y Competitividad y el FEDER (Fondo Europeo de Desarrollo Regional), bajo el proyecto TIN2012-30685 (proyecto BIO). Sergio Santander-Jiménez es becario del programa FPU del Gobierno de España (FPU12/04101).

Referencias

[1] L. Adhianto and B. Chapman, "Performance modeling of communication and computation in hybrid MPI and OpenMP applications," *Simulation Modelling Practice and Theory*, vol. 15, no. 4, pp. 481–491, 2007.

[2] J. Felsenstein, *Inferring Phylogenies*, Sinauer Associates, Sunderland, Massachusetts, 2004.

[3] M. Ott, J. Zola, S. Aluru, A. D. Johnson, D. Janies, and A. Stamatakis, "Large-scale phylogenetic analysis on current HPC architectures," *Scientific Programming*, vol. 16, no. 2-3, pp. 255–270, 2008.

[4] D. A. Bader, U. Roshan, A. Stamatakis, and C. W. Tseng, "Computational Grand Challenges in Assembling the Tree of Life: Problems and Solutions," in *Advances in Computers*, vol. 68, pp. 127–176. Elsevier, 2006.

[5] P. Lemey, M. Salemi, and A.-M. Vandamme, *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing*, Cambridge University Press, Cambridge, 2009.

[6] C. Coello, D. Van Veldhuizen, and G. Lamont, "Evolutionary Algorithms for Solving Multi-Objective Problems," in *Genetic Algorithms and Evolutionary Computation*, vol. 5. Kluwer Academic Publishers, 2002.

[7] W. Cancino, L. Jourdan, E. G. Talbi, and A. C. B. Delbem, "Parallel Multi-Objective Approaches for Inferring Phylogenies," in *EvoBIO 2010*. 2010, vol. 6023 of *LNCS*, pp. 26–37, Springer Verlag.

[8] W. H. E. Day, D. S. Johnson, and D. Sankoff, "The Computational Complexity of Inferring Rooted Phylogenies by Parsimony," *Mathematical Biosciences*, vol. 81, no. 1, pp. 33–42, 1986.

[9] B. Chor and T. Tuller, "Maximum Likelihood of Evolutionary Trees is Hard," in *Research in Computational Molecular Biology*. 2005, vol. 3500 of *LNBI*, pp. 296–310, Springer Verlag.

[10] X.-S. Yang, "Firefly Algorithm, Stochastic Test Functions and Design Optimisation," *Int. J. Bio-Inspired Computation*, vol. 2, no. 2, pp. 78–84, 2010.

[11] O. Gascuel, "BIONJ: An Improved Version of the NJ Algorithm Based on a Simple Model of Sequence Data," *Mol. Biol. Evol.*, vol. 14, no. 7, pp. 685–695, 1997.

[12] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes," in *PDP 2009*. 2009, pp. 427–436, IEEE Computer Society.

[13] P. A. Goloboff, J. S. Farris, and K. C. Nixon, "TNT, a free program for phylogenetic analysis," *Cladistics*, vol. 24, no. 5, pp. 774–786, 2008.

[14] A. Stamatakis, "RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

Paralelización del cálculo del campo de vientos para predicción de la propagación de incendios forestales[1]

Gemma Sanjuan[2], Carlos Brun[3], Tomàs Margalef[4], Ana Cortés[5]
Departamento de Arquitectura de Computadores y Sistemas Operativos
Universitat Autònoma de Barcelona

Resumen— **Los incendios forestales representan un problema muy significativo, especialmente en los países de la cuenca del Mediterráneo. Para luchar contra estos desastres, es necesario disponer de una predicción precisa de su evolución. Los modelos de propagación requieren unos parámetros de entrada precisos para poder alcanzar los resultados esperados. Sin embargo, hay parámetros que presentan una distribución espacial que hace necesario el uso de modelos complementarios como es el caso de los modelos de campo de vientos. A partir del valor meteorológico del viento, estos modelos proporcionan la dirección y la velocidad del viento en cada punto del terreno. Uno de los simuladores de viento disponibles es WindNinja. Este simulador de campo de vientos requiere un elevado tiempo de ejecución que penaliza la predicción de la propagación de los incendios. Para resolver este problema se propone particionar el mapa, y calcular de forma independiente el campo de viento en cada parte. Sin embargo, el modelo presenta ciertos efectos en los bordes de los mapas que hacen necesario un cierto grado de solapamiento entre las partes para poder alcanzar unos valores estables en el mapa completo.**

Palabras clave— **Incendios forestales, campo de vientos, simulación, partición de mapas, solapamiento.**

I. Introducción

LOS incendios forestales son una de las peores catástrofes que sufren los bosques en todo el mundo. El problema no son sólo las hectáreas destruidas por el incendio sino como afecta a los ecosistemas presentes en la zona, ya que son los causantes de la muerte de animales y vegetales, disminución de las zonas prolíferas, contaminación del aire y del agua, incluso de pérdida de vidas humanas.

La mayoría de los estudios actuales se centran tanto en la prevención de los incendios, como en la reducción de sus efectos una vez que el fuego ya ha comenzado.

Los simuladores de fuego son sistemas que mediante ecuaciones matemáticas intentan predecir de manera más o menos exacta como va evolucionar el fuego y ayudar a tomar las decisiones necesarias para reducir los efectos del fuego. La mayoría de simuladores utilizan un gran número de parámetros de entrada para describir el ambiente donde el fuego se desarrolla [1][2][3]. Sin embargo, en muchas ocasiones es difícil disponer del valor de los parámetros en tiempo real, ya que presentan una variación espacial y temporal.

La predicción clásica se basa exclusivamente en una única simulación de la propagación del incendio. La principal ventaja que presenta este sistema de predicción es la simplicidad, aunque las predicciones obtenidas generalmente difieren del fuego real. El principal problema es la falta de precisión de los parámetros de entrada ya que en muchos casos se utilizan valores uniformes en todo el terreno y constantes en el tiempo. Con la finalidad de superar este inconveniente, se propuso un método de predicción de la propagación de incendios forestales basados en dos etapas que reciben el nombre de etapa de calibración y etapa de predicción [4].

Inicialmente se analiza la evolución del incendio en un intervalo de tiempo (de t_0 a t_1) y se lleva a cabo la etapa de calibración de parámetros en la que se aplica un algoritmo genético[5]. Este algoritmo genera una población inicial en la que cada individuo está formado por un conjunto de valores de los parámetros que representan un escenario de propagación del incendio. Para cada individuo se lleva a cabo una predicción del comportamiento del incendio en el intervalo t_0 - t_1 y se compara el resultado con la propagación del frente real de fuego en el instante de tiempo t_1.

Los individuos son ordenados en función del error entre el resultado de la simulación y la evolución real del incendio. La función de error que se utiliza es la diferencia simétrica entre la predicción y el comportamiento real[6]. Con la población ordenada se aplican los operadores genéticos de elitismo, selección, mutación y cruzamiento para obtener una nueva generación de individuos.

Este proceso se repite un cierto número de iteraciones y al final de este proceso de calibración de parámetros, el individuo con el mejor resultado se utiliza para llevar a cabo la predicción de la propagación del incendio entre los instantes t_1 y t_2[7].

El esquema del algoritmo genético encaja de forma natural en el paradigma Master/Worker [8][9]. El Master genera la población inicial y envia los individuos a los workers. Los workers realizan la evaluación de cada uno de los individuos y devuelven el error obtenido al Master que, a su vez, aplica los operadores genéticos para alcanzar la siguiente población y, así repetir el proceso durante un cierto número de

[1]Este trabajo ha sido financiado por el MICINN bajo contrato TIN2011-28689-C02-01.
[2]e-mail: gemma.sanjuan@caos.uab.es.
[3]e-mail: carlos.brun@caos.uab.es.
[4]e-mail: tomas.margalef@uab.es.
[5]e-mail: ana.cortes@uab.es.

iteraciones.

En este esquema se ha incluido un modelo de campo de vientos que, a partir del viento meteorológico, permite calcular el viento en cada punto del mapa considerado para obtener una mayor precisión en la predicción de la propagación. Sin embargo, este modelo es computacionalmente costoso y se ha llevado a cabo una paralelización de dicho cálculo mediante la partición del mapa en un conjunto de partes con un cierto grado de solapamiento para poder calcular el viento en cada parte de forma independiente.

En la segunda sección se describe la utilización de un modelo de campos de vientos. A continuación, en la tercera sección, se describe el método de partición de los mapas para la paralelización del cálculo del campo de vientos. En la cuarta sección se analiza el estudio experimental que se ha llevado a cabo, mostrando tanto la calidad de los campos de vientos obtenidos con el mapa particionado como la ganancia en cuanto al tiempo de ejecución. Por último se presentan las conclusiones del trabajo.

II. Acoplamiento del modelo de campo de vientos en el esquema de predicción

Cuando aplicamos la metodología a terrenos complejos con condiciones no controladas e incendios que duran un tiempo elevado, las predicciones se ven penalizadas por la variación espacial y temporal de los parámetros como la dirección y velocidad del viento[10][11].

Hasta el momento las componentes del viento (velocidad y dirección) se han tratado como uniformes en todo el terreno estudiado, sin tener en cuenta las inevitables distorsiones que sufre cuando se encuentra con una superficie irregular. Por ello se plantea la necesidad de introducir un modelo que calcule las componentes de viento de forma individual para cada celda del terreno, en función de la pendiente y orientación de dicha celda.

Por tanto los valores de la dirección y velocidad del viento se introducen en un simulador de campo de vientos (WindNinja) [12] para obtener el viento en cada celda del terreno. Este esquema se muestra en la figura 1.

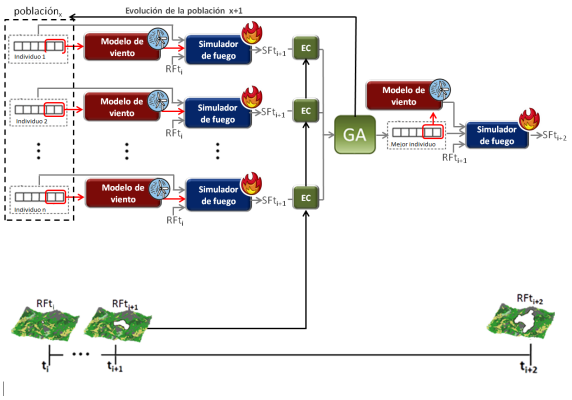


Fig. 1. Método de dos estapas con WindNinja

WindNinja es una aplicación que, a partir de un viento meteorológico formado por una velocidad, una dirección y un sentido, calcula las direcciones, sentidos y velocidades del viento para cada una de las celdas en que se ha dividido un terreno concreto. Esta aplicación se basa en resolver las ecuaciones diferenciales que describen el movimiento del aire en la atmósfera, concretamente el modelo de la conservación de la masa, introducidas por unas ecuaciones de contorno. Esto implica que la variación de pendiente del terreno genera cambios en los vientos y debido a las condiciones de contorno los resultados obtenidos en las fronteras del mapa no serán correctas hasta haber transcurrido un determinado número de celdas [12]. Los modelos de conservación de la masa plantean un problema de mínimos cuadrados en el dominio Ω con las velocidades a ajustar $\vec{u}(u,v,w)$ a partir de las observadas $\vec{v_0}(u_0,v_0,w_0)$.

La ecuación que resuelve el sistema es:

$$E(u,v,w) = \int [(\alpha_1)^2(u-u^0)^2 - (\alpha_1)^2(v-v^0)^2 - \\ -(\alpha_2)^2(w-w^0)^2 + \lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z})]d\omega \quad (1)$$

siendo α_1 y α_2 los módulos de precisión de Gauss, considerados idénticos para las direcciones horizontales. Entonces el campo buscado $\vec{v}(u,v,w)$ será la solución del problema.

El problema que presenta WindNinja es que a medida que aumentamos el tamaño del mapa, el tiempo de ejecución aumenta considerablemente. Además, la memoria necesaria para solucionar el mapa aumenta significativamente provocando que no se pueda resolver en un ordenador personal. Por ello, planteamos una paralelización del cálculo del campo de vientos basada en la partición del mapa del terreno en cuestión, de modo que el campo de vientos en cada parte pueda calcularse de forma independiente y paralela.

III. Partición del mapa para cálculo paralelo

El objetivo que se plantea consiste en paralelizar el cálculo del campo de vientos mediante la partición del mapa y llevando a cabo el cálculo del campo de vientos de forma independiente en cada partición, sin realizar modificaciones en el solver. Sin embargo, hay que tener en cuenta que, como se ha comentado, el campo de vientos proporcionado por WindNinja presenta una zona de incertidumbre cerca de las fronteras o bordes del mapa. Por tanto, es necesario extender cada partición del mapa y realizar un cierto solapamiento entre las particiones, de modo que la parte calculada correspondiente a cada partición se le añade un conjunto de celdas de las particiones vecinas (Figura 2). Para obtener el campo de vientos total se unen los campos de viento obtenidos en cada partición descartando los vientos calculados para la zona solapada con la celda vecina, que será la que aportará los valores del campo de vientos para sus celdas.

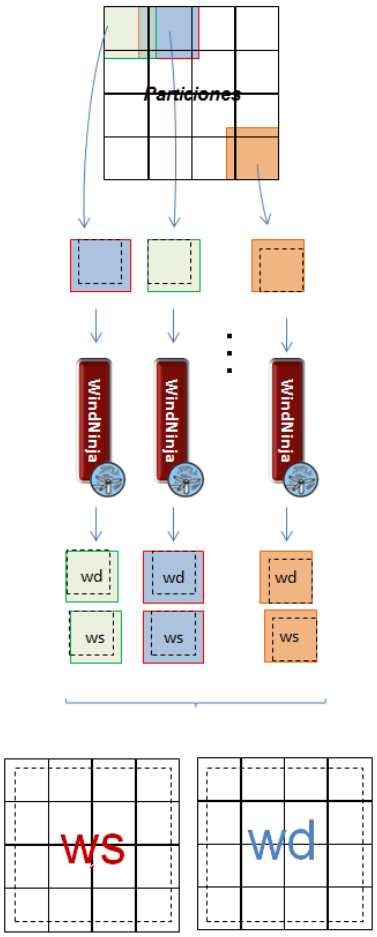


Fig. 2. Partición de mapas con solapamiento

Este solapamiento introduce un incremento en el tiempo de cómputo de cada parte, pero debería permitir obtener unos resultados más fiables. La cantidad de celdas que deben solaparse se estima sobre un 10% del mapa que se quiere tratar [12], pero debe ser evaluado de forma precisa. Por tanto, a cada uno de las partes se le debe añadir un número de celdas, que será deshechado al realizar la unión del mapa.

Para determinar la diferencia entre el campo de vientos obtenido con un solo mapa global y el campo de vientos obtenido mediante la agregación de mapas particionados se utiliza el llamado RMSE (Root Mean Square Error), ya que permite cuantificar las diferencia a partir de la expresión 2:

$$RMSE_{AxB}(ws) = \sqrt{\frac{\sum_{i=0}^{N}(NP(ws)_i - SC_{AxB}(ws)_i)^2}{N}} \quad (2)$$

donde AxB hace referencia al tipo de partición que presenta el mapa, N representa el numero de celdas totales e i hace referencia a cada una de las celdas del mapa. Los términos del sumatorio $NP(ws)_i$ son los valores obtenidos de la velocidad del viento en el mapa sin particionar para la celda i y $SC_{AxB}(ws)_i$ son los valores obtenidos de la velocidad del viento para cada celda i de las partición AxB. El objetivo es realizar una comparación entre los valores de la velocidad del viento obtenido con el mapa global o con el mapa particionado. Cuando se quiere evaluar el error para la dirección simplemete hay que cambiar los términos ws por wd.

IV. Estudio experimental

Para poder realizar el estudio se ha seleccionado un mapa correspondiente a una zona de la provincia de Valencia (España). Esta región se considera de interés ya que en el último año ardieron 50000 ha (verano 2012). El mapa se compone de celdas de 30 metros por 30 metros con una dimensión total de 1406 filas por 1802 columnas, lo que significa un área de $42kmx54km$.

El método experimental consisten en realizar diferentes tipos de partición, aplicar distintos rangos de solapamiento entre las partes y considerar distintos vientos de entrada. Se consideran particiones del mapa de 1x3, 2x2, 3x1, 3x3, 1x4, 4x1 y 4x4. Los niveles de solapamiento aplicados son del 0%, 2%, 4%, 6%, 8%, 10%, 12% y 14%. Se ha considerado un rango de vientos que van desde vientos suaves (5mph) a vientos de cierta intensidad (15mph). Con este estudio se pretende:

- Determinar la preferencia del tipo de partición (horizontal, vertical, cuadrado).
- Determinar el solapamiento adecuado para cada una de las particiones.
- Observar el comportamiento del error.
- Determinar los problemas que presentan cada una de las velocidades del viento.
- Determinar como se modifica el tiempo de cómputo.

Los experimentos se han realizado en dos plataformas diferentes:

- Cluster IBM x3650 compuesto de 32 nodos con 2 Dual-Core Intel(R) Xeon(R) CPU 5150 2.66GHz y 12 GB Fully Buffered DIMM 667 MHz por nodo.
- Cluster Dell compuesto por 2 x Six-Core Intel(R) Xeon(R) E5645 2.4 GHz con Hyper-Threading, 32KB y 256KB de cache L1 y L2 dedicada y 12MB de memoria compartida L3. Así, el sistema dispone de 12 cores (24 threads) y 96GB de memoria principal.

Para evaluar la diferencia entre el cálculo del campo de vientos con un mapa global y el mapa particionado se calcula el RMSE y se contabilizan el número de celdas con un error superior al 10%.

Se debe tener en cuenta que conseguir que la diferencia sea 0 es casi imposible, ya que el método de resolución del Gradiente Conjugado con precondicionador utilizado por WindNinja, provoca que a medida que el tamaño de la matriz aumenta el error se propague aumentando el número de celdas que pueden tener un valor diferente.

Las características del modelo que permite calcular el viento en cada punto del terreno a partir de la velocidad y dirección del viento meteorológico determinan que para ángulos suplementarios los errores

obtenidos son iguales. Esto es consecuencia de que las ecuaciones que definen en el sistema buscan encontrar los puntos de ensilladura del mismo. Estos puntos son aquellos puntos sobre una superficie en los que la función es máxima en una dirección y mínima en la dirección perpendicular. Matemáticamente se define como un punto de una función en el que la primera derivada es nula, mientras que el signo de la segunda derivada (curvatura) depende de la dirección en que se calcule. Además, como la ecuación es continua y diferenciable para todo su dominio, los resultados sólo dependen de la dirección elegida, pero no del sentido.

A. *Evaluación de la calidad del campo de vientos particionado*

La gráfica de la Figura 3 muestra la diferencia (RMSE) obtenida con una partición de 3x1 para distintos valores de la dirección y velocidad del viento a medida que va aumentado el grado de solapamiento. Se observa que la diferencia aumenta con la velocidad del viento y también depende de la dirección del viento, pero con un solapamiento del 10% la diferencia es inferior a 1mph%.

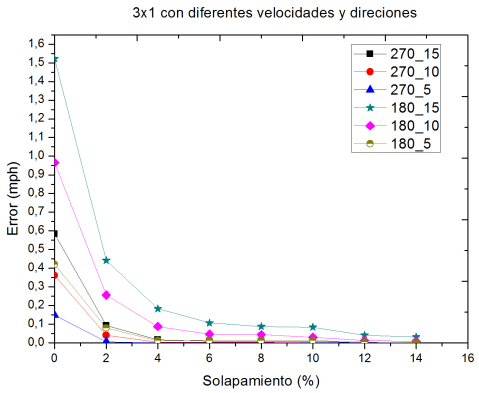


Fig. 3. RMSE para distintos vientos meteorológicos

La combinación más desfavorable se da para una dirección y sentido de 180º o 0º, con un módulo de velocidad del viento meteorológico de 15 mph. Esto es debido a que la variación de las pendientes en esta dirección es superior a la variación en otras direcciones. Esto conlleva que los errores sean superiores en comparación con los otros casos con la misma velocidad. Como consecuencia, el solapamiento propuesto no es suficiente para que se dé el caso de que ninguna celda presente un error inferior al 10%.

La gráfica de la figura 4 muestra el RMSE de la velocidad del viento para distintos tipos de partición longitudinal del terreno. Como se puede observar, la partición determina la diferencia del campo de vientos respecto al campo de vientos obtenido con el mapa original. Esto se debe a que el viento meteorológico se sitúa en el extremo superior izquierdo, debido a la condición de Dirichlet nula en las fronteras permeables y la condición de Neumann en las impermeables. Por tanto, resultan más favorables las particiones 3x1 y 4x1, ya que se recorre la matriz de izquierda a derecha y de arriba abajo. Asimismo, los peores resultados tal y como se comprueba en la gráfica se dan para los casos 1x3 y 1x4. También puede observarse que a medida que aumenta el porcentaje de solapamiento no se aprecia una mejora significativa del error sino que se estabiliza aproximadamente a partir de un solapamiento del 4%. A medida que aumenta el número de partes y, en consecuencia el número de celdas por parte disminuye, es necesario un porcentaje de solapamineto mayor ya que es necesario garantizar un número mínimo de celdas en el solapamiento.

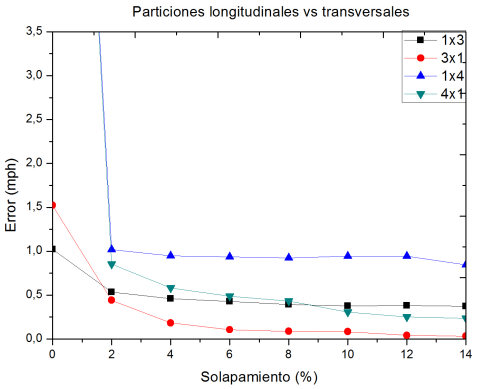


Fig. 4. RMSE para particiones longitudinales y transversales

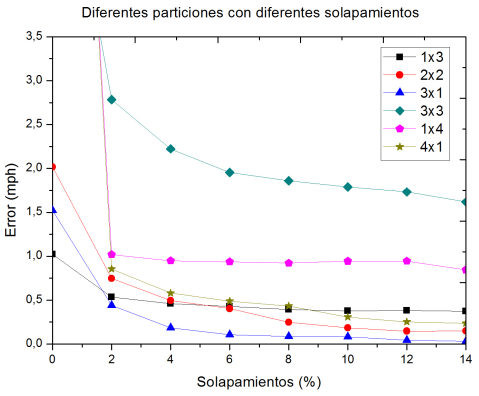


Fig. 5. RMSE para distintas particiones

En la gráfica de la figura 5 se inlcuye el RMSE para particiones en cuadricula (2x2 y 3x3). Se puede observar que la partición de 3x3 presenta un RMSE muy elevado que mejora con el grado de solapamiento, pero no consigue rebajar un RMSE de 1,5mph en la velocidad del viento. Esto se debe a que cada partición resulta muy pequeña y el porcentaje de solapamiento no tiene un número de celdas suficientemente significativo. Por el contrario, la partición 2x2 que tiene el mismo número de partes que la 1x4 y 4x1 presenta una mejor evolución del RMSE con el aumento del solapamiento. Esto se debe a que el tamaño de una parte en cada una de las direcciones es suficientemente grande para alcanzar buenos re-

sultados. En cambio, una partición 4x1 se compone de franjas más estrechas, para las cuales un cierto porcentaje de solapamiento contiene un número de celdas mucho menor.

Otra forma de presentar estos resultados son los mapas de diferencia que se muestran en la figura 6 y 7. En estos mapas se muestra en color los puntos del mismo en el que se ha producido una diferencia entre el valor del viento local del mapa global y del mapa particionado. El color indica la magnitud de la diferencia. En el mapa generado con un solapamiento del 0% se aprecian de forma clara las líneas correspondientes a la partición del mapa en tres franjas verticales. En cambio, en el mapa de diferencias generado con un solapamiento del 14% se observan unas diferencias puntuales que no permiten identificar la partición realizada y no serán significativas para el cálculo de la propagación del incendio.

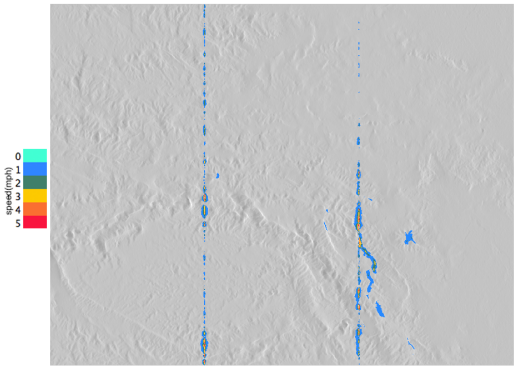


Fig. 6. Error para una partición 1x3 y solapamiento 0%

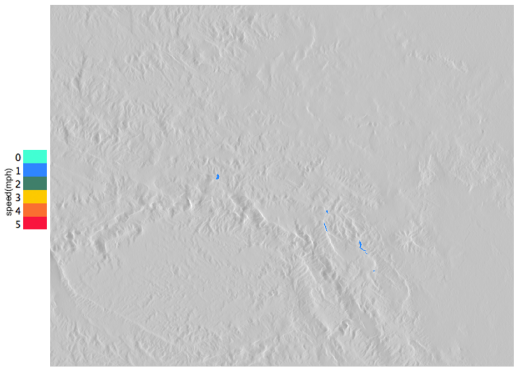


Fig. 7. Error para una partición 1x3 y solapamiento 14%

B. *Tiempo de ejecución y eficiencia de la paralelización*

En la subsección anterior se ha analizado la calidad del campo de vientos obtenido particionando el terreno con un cierto grado de solapamiento y se ha comparado con el campo de vientos obtenido aplicando WindNinja a un mapa global. Los resultados son muy prometedores ya que con un grado de solapamiento razonable se consigue una campo de vientos muy próximo al que se obtiene con un mapa global. Los valores obtenidos muestran que para los tamaños de los mapas considerados no parece adecuado particionar el mapa en más de 9 o 16 particiones, aunque si el tamaño del mapa es mayor este límite podría superarse.

El punto siguiente que debe considerarse es la reducción en el tiempo de cómputo que se alcanza y el speedup y eficiencia que se consiguen. De este modo, se pondrá de manifiesto la utilidad real y el coste que representa.

La gráfica 8 muestra el tiempo de ejecución alcanzado realizando distintas particiones del terreno. Se observa que la reducción de tiempo es significativa y es prácticamente idéntica para particiones con el mismo número de partes. Así, el cálculo de un campo de vientos, que en un único procesador tarda cerca de 1300 segundos, tarda menos de 100 segundos cuando se ejecuta en 16 nodos. Estos resultados son refrendados por los valores de speedup obtenidos. La gràfica 9 muestra el speedup y se observa que para particiones en 3, 4 y 9 partes, el speedup que se alcanza es prácticamente lineal, y para el caso de 16 partes el speedup no llega a 13, pero aún así es un resultado muy satisfactorio. Se debe notar que el speedup obtenido es independiente de la forma de la partición y depende exclusivamente del número de particiones.

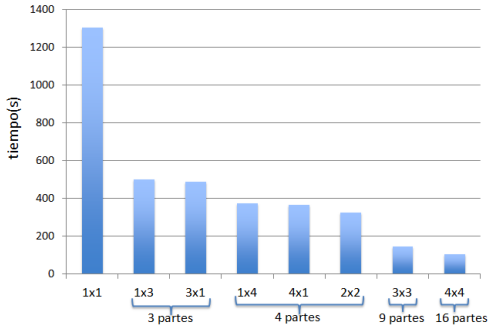


Fig. 8. Tiempo de ejecución en función del número de partes

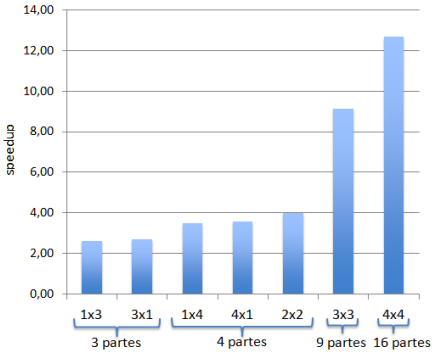


Fig. 9. Speed up dependiendo del número de partes

Otra medida significativa es la eficiencia conseguida, que se define como el speedup alcanzado

dividido por los recursos empleados. La gráfica 10 muestra que a medida que aumentamos el número de partes, la eficiencia se mantiene constante, a excepción de las particiones 2x2 y 3x3 que experimentan una mejora en la eficiencia. En cambio, en la partición 4x4 se observa que la eficiencia disminuye. Esto se debe a que al particionar el mapa en 4x4 partes, éstas son demasiado pequeñas y las comunicaciones y el cálculo del campo correspondiente al solapamiento introducen un coste adicional.

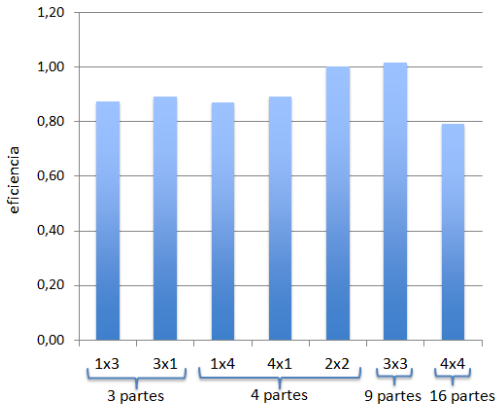


Fig. 10. Eficiencia dependiendo del número de partes

V. Conclusiones

La precisión de los parámetros de entrada de los modelos de propagación de incendios forestales es un factor clave para obtener unas predicciones satisfactorias. Sin embargo, determinados parámetros como el viento presentan una variación espacial y temporal que hacen imposible su medición precisa en todos los puntos del terreno, de modo que es necesario introducir modelos complementarios para el cálculo del campo de vientos. Estos modelos de campo de vientos son computacionalmente costosos y pueden llegar a tardar decenas de minutos en un computador personal. Estos tiempos no son asumibles y es necesario aplicar técnicas de paralelización para acelerar este cálculo. En este sentido se ha propuesto un método consistente en la partición del mapa del terreno en un cierto número de partes de modo que se puede calcular el campo en cada parte de forma independiente. Sin embargo, para minimizar los efectos debidos a los bordes es necesario añadir un cierto grado de solapamiento entre una parte y sus vecinas. Si el grado de solapamiento es adecuado, los campos de vientos generados a partir de un mapa global y los generados a partir de un mapa particionado difieren muy poco y pueden considerarse idénticos a efectos de la propagación del incendio. Por lo que se refiere a los tiempos de ejecución, los resultados muestran que el speedup que se obtiene es prácticamente lineal hasta particiones de 16 partes, en las que empieza a degradarse el rendimiento. Aún así, los resultados permiten reducir el tiempo de cómputo de unos 1300 segundos a cerca de 100 segundos, lo cual es una mejora significativa que puede resultar determinante en caso de una emergencia real.

Referencias

[1] R.C. Rothermel, *How to predict the spread and intensity of forest and range fires*, Intermountain Forest and Range Experiment Station Ogden, 1983.

[2] F.A. Albini, Intermountain Forest, Utah) Range Experiment Station (Ogden, and United States. Forest Service, *Estimating wildfire behavior and effects*, General technical report INT. Dept. of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1976.

[3] Josep Jorba, Tomas Margalef, Emilio Luque, J Campos da Silva Andre, and Domingos Xavier Viegas, "Parallel approach to the simulation of forest fire propagation," in *Proceedings of the 13 Internationales Symposium" Informatik fur den Umweltshutz" der Gesellshaft Fur Informatik*, 1999, pp. 69–81.

[4] Baker Abdalhaq, Ana Cortés, Tomàs Margalef, and Emilio Luque, "Enhancing wildland fire prediction on cluster systems applying evolutionary optimization techniques," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 61–67, 2005.

[5] M. Denham, A. Cortés, and T. Margalef, "Computational steering strategy to calibrate input variables in a dynamic data driven genetic algorithm for forest fire spread prediction," *Computational Science–ICCS 2009*, pp. 479–488, 2009.

[6] Andrés Cencerrado, Ana Cortés, and Tomàs Margalef, "Genetic algorithm characterization for the quality assessment of forest fire spread prediction," *Procedia Computer Science*, vol. 9, pp. 312–320, 2012.

[7] Andŕes Cencerrado, Ana Cortes, and Tom'as Margalef, "Prediction time assessment in a dddas for natural hazard management: Forest fire study casei," *Procedia Computer Science*, vol. 4, pp. 1761–1770, 2011.

[8] William Gropp, "Mpich2: A new start for mpi implementations," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Dieter Kranzlmüller, Jens Volkert, Peter Kacsuk, and Jack Dongarra, Eds., vol. 2474 of *Lecture Notes in Computer Science*, pp. 37–42. Springer Berlin / Heidelberg, 2002.

[9] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: portable shared memory parallel programming*, vol. 10, The MIT Press, 2007.

[10] Mónica Denham, Ana Cortés, Tomàs Margalef, and Emilio Luque, "Applying a dynamic data driven genetic algorithm to improve forest fire spread prediction," in *Computational Science – ICCS 2008*, Marian Bubak, Geert van Albada, Jack Dongarra, and Peter Sloot, Eds., vol. 5103 of *Lecture Notes in Computer Science*, pp. 36–45. Springer Berlin / Heidelberg, 2008, 10.1007/978-3-540-69389-5_6.

[11] Carlos Brun, Tomàs Artées, Tomàs Margalef, and Ana Cortées, "Coupling wind dynamics into a dddas forest fire propagation prediction system," *Procedia Computer Science*, vol. 9, pp. 1110–1118, 2012.

[12] Jason Forthofer, Kyle Shannon, and Bret Butler, "4.4 simulating diurnally driven slope winds with windninja," 2009.

Paralelización de Meta-heurísticas haciendo uso de MPI y OpenMP: Aplicación al Enrutado de Vehículos

Raúl Baños[1], Julio Ortega [1], Consolación Gil [2]

Resumen— **La paralelización de meta-heurísticas permite explorar el espacio de soluciones de forma más exhaustiva y acelerar el proceso de búsqueda. En la actualidad existe un creciente interés en el desarrollo de algoritmos paralelos utilizando componentes software estándar aprovechando el rendimiento de los modernos microprocesadores que incluyen varios núcleos de procesamiento con memorias caché locales y compartidas. En este trabajo analizamos los beneficios proporcionados por paralelizaciones de meta-heurísticas poblacionales basadas en el modelo maestro-trabajador y en el modelo de islas, haciendo uso de MPI y OpenMP. Dichas estrategias son evaluadas en la paralelización de un algoritmo secuencial que utiliza enfriamiento simulado con múltiples temperaturas (MT-SA) para resolver el problema de enrutado de vehículos con ventanas temporales (VRPTW). Los resultados empíricos obtenidos en un procesador multi-núcleo a la hora de resolver un conjunto de benchmarks ha permitido, no sólo comparar el rendimiento de dichas implementaciones paralelas, sino también mejorar claramente la calidad de los resultados obtenidos por la versión secuencial.**

Palabras clave— **Optimización computacional, meta-heurísticas paralelas, MPI, OpenMP, enrutado de vehículos.**

I. Introducción

LAS principales ventajas del procesamiento paralelo son la reducción del tiempo de ejecución de las versiones secuenciales y/o la mejora de dichas soluciones sin provocar un incremento del tiempo de cómputo. Los programas paralelos han sido evaluados tradicionalmente en equipos de altas prestaciones, como clusters de computadores o máquinas de memoria compartida. Sin embargo, la introducción de procesadores multi-núcleo, que constan de varios núcleos de procesamiento y memorias caché locales y compartidas en un solo microchip, permite también aplicar el paralelismo en computadores personales utilizando liberías estándar.

En el presente estudio analizamos el rendimiento de diferentes implementaciones de modelos evolutivos paralelos, incluyendo el modelo basado en islas y el paradigma maestro-trabajador, haciendo uso las librerías paralelas MPI y OpenMP, respectivamente, con el objetivo de paralelizar meta-heurísticas poblacionales en sistemas multi-núcleo de memoria compartida. El hecho de comparar MPI y OpenMP en un problema real resulta interesante dado que es importante determinar si resulta más eficaz utilizar MPI para enviar mensajes en sistemas de varios núcleos con memoria compartida, es decir, copiar los datos, que utilizar hebras con OpenMP que comparten los accesos a toda la memoria, incluyendo las secciones críticas. Ambas implementaciones paralelas son evaluadas en una estación de trabajo de memoria compartida que consta de un chip multi-núcleo para resolver un problema complejo como es el de enrutado de vehículos con ventanas temporales.

La Sección II describe el problema de enrutado de vehículos con restricciones de capacidad en dichos vehículos y ventanas temporales en los servicios a clientes. La Sección III resume las aproximaciones paralelas propuestas en la literatura para resolver este problema, a la vez que se presentan las implementaciones paralelas MPI y OpenMP de un algoritmo basado en enfriamiento simulado. La Sección IV muestra los resultados obtenidos en la resolución de un conjunto de benchmarks, y la Sección V ofrece las conclusiones y el trabajo futuro.

II. El Problema de Enrutado de Vehículos con Ventanas Temporales

El problema de enrutado de vehículos con ventanas temporales (Vehicle Routing Problem with Time Windows, VRPTW) [1] es una variante de la familia de problemas de enrutado de vehículos (VRP) que consiste en determinar las rutas óptimas que debe seguir un determinado número de vehículos, con idéntica capacidad, estacionados en un almacén central (centro logístico), que operan dentro de una franja temporal y que son utilizados para visitar y suministrar las mercancías a un determinado número de clientes. Los vehículos siguen una ruta que parte del centro logístico y vuelven a el tras realizar el servicio. La cantidad de mercancías suministradas por cada vehículo no puede exceder su capacidad. Los clientes, cuyas demandas pueden ser atendidas únicamente por un solo vehículo, están localizados geográficamente de forma dispersa, y tienen requisitos pre-establecidos en lo que respecta a las mercancías requeridas y al tiempo de servicio necesario para la descarga. Cada cliente tiene asociada una ventana temporal, de forma que el vehículo que le suministra puede llegar antes de que la ventana temporal abra (esperando a su apertura), pero no después de que esta haya cerrado. La distancia total recorrida por todos los vehículos determina el tiempo total del suministro. Existen formulaciones en las que el proceso de optimización consta de dos fases: una primera fase en la que se

[1]Dpto. de Arquitectura y Tecnología de Computadores, Centro de Investigación en Tecnologías de la Información y las Comunicaciones (CITIC-UGR), Universidad de Granada, e-mail:{rbanos,jortega}@ugr.es

[2]Dpto. de Informática, ceiA3, Universidad de Almería, e-mail:cgilm@ual.es

minimiza el número de vehículos utilizados y una segunda fase en la que, a igualdad de vehículos, se reduce la distancia total recorrida [2]. Sin embargo, en el presente trabajo consideramos directamente la distancia total recorrida como el objetivo a minimizar ya que, al ser el combustible el coste variable más importante del proceso de transporte, la reducción de la distancia total recorrida implica reducir su consumo. Además, en la gran mayoría de aplicaciones prácticas, el uso de más o menos vehículos y conductores no afecta al coste operativo real, ya que los vehículos suelen ser propiedad de la empresa y los conductores son empleados de la misma. Se han propuesto diferentes métodos exactos para resolver el problema del enrutado de vehículos, incluyendo métodos basados en relajación, programación dinámica, etc. [1]. Sin embargo, dada la complejidad del problema, los métodos aproximados, incluyendo las meta-heurísticas, resultan más convenientes ya que obtienen resultados aceptables en tiempos reducidos [3].

La definición matemática de los objetivos y restricciones del VRPTW, incluyendo la notación frecuentemente utilizada para definir las rutas, el almacén central, los clientes, y los vehículos, puede ser modelada como un problema de teoría de grafos [1]. Sea $G = (V, E)$ un grafo completo no dirigido, donde los vértices $V = \{1, ..., N\}$ corresponden al almacén central (centro logístico donde se encuentran los vehículos y mercancías) y a los clientes, y las aristas $e \in E\{(i, j) : i, j \in V\}$ a los enlaces entre ellos.

Variables de decisión

$$X_{ij}^k = \begin{cases} 1 & \text{si el vehículo } k \text{ viaja desde } i \text{ hasta } j \\ 0 & \text{en otro caso} \end{cases}$$

Parametros

a_j es el instante inicial en el que el cliente j permite el servicio,
b_j es el último instante de tiempo que el cliente j permite el servicio,
C_{ij} es el coste de viajar desde el nodo i al nodo j (C_{ij} se puede interpretar como distancia o tiempo),
d_j es la demanda en el cliente j,
K es el máximo número de vehículos disponibles,
N es el número de clientes (denotados por 2,...N), más el almacén (denotado por 1),
Q^k es la capacidad de carga del vehículo k (todos los vehículos tienen la misma capacidad, Q).

El VRPTW se puede definir como sigue:

$$\textit{minimizar} \quad TD = \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{N} X_{ij}^k C_{ij} \tag{1}$$

sujeto a:

$$X_{ii}^k = 0 \qquad (\forall i \in \{1, ..., N\},\ \forall k \in \{1, ..., K\}) \tag{2}$$

$$X_{ij}^k \in \{0, 1\} \quad (\forall i,j \in \{1, ..., N\}, \forall k \in \{1, ..., K\}) \tag{3}$$

$$\sum_{k=1}^{K} \sum_{i=1}^{N} X_{ij}^k = 1 \qquad (\forall j \in \{2, .., N\}) \tag{4}$$

$$\sum_{i=1}^{N} \sum_{j=2}^{N} X_{ij}^k d_j \leq Q^k \qquad (\forall k \in \{1, .., K\}) \tag{5}$$

$$\sum_{k=1}^{K} \sum_{j=2}^{N} X_{1j}^k \leq K \tag{6}$$

$$\sum_{j=2}^{N} X_{1j}^k - \sum_{j=2}^{N} X_{j1}^k = 0 \quad (\forall k \in \{1, ..., K\}) \tag{7}$$

$$a_j \leq s_{kj} \leq b_j \quad (\forall i,j \in \{1, .., N\}, \forall k \in \{1, ..., K\}) \tag{8}$$

$$s_{ki} + C_{ij} - L(1 - X_{ij}^k) \leq s_{kj} \qquad (\forall i,j \in \{1, ..., N\}, \forall k \in \{1, ..., K\}) \tag{9}$$

La ecuación (1) es la función objetivo del problema. La ecuación (2) denota que un vehículo debe viajar desde un nodo a otro diferente. La ecuación (3) indica que X_{ij}^k es igual a 1 si el vehículo k va desde el nodo i al nodo j, y es igual a 0 en otro caso, esto es, una ruta entre dos clientes puede o no ser cubierta por un vehículo. La ecuación (4) señala que un cliente es visitado una y sólo una vez por un solo vehículo. Especificando la restricción de la ecuación (5) se tiene en cuenta que para un vehículo dado, k, la carga que es transportada por un vehículo no puede exceder su capacidad Q^k. La ecuación (6) especifica que hay como máximo K rutas saliendo del almacén central. La ecuación (7) garantiza que los vehículos salen del almacen central y retornan a el tras el servicio. Sea s_{kj} la suma de las distancias viajadas por el vehículo k antes de llegar al cliente j, la ecuación (8) asegura que las ventanas temporales son satisfechas. Dado un escalar elevado, L, la desigualdad representada en la ecuación (9) especifica que, si el vehículo k viaja desde el cliente i al cliente j, el vehículo no puede llegar al cliente j antes del instante $s_{ki} + C_{ij}$. Como se especifica en [1], la variable s_{kj} corresponde al tiempo en el cual el vehículo k empieza a servir al cliente j, siempre que k visite a j.

III. Paralelización de Meta-heurísticas para Resolver el VRPTW

Dada la complejidad de la familia de problemas de enrutado de vehículos, el simple uso de procedimientos meta-heurísticos no siempre garantiza la obtención de soluciones de calidad en tiempos acotados. Es por ello que la computación paralela permite reducir el tiempo de ejecución de las versiones secuenciales o la mejora de las mismas sin provocar un incremento del tiempo de cómputo. Los programas paralelos han venido siendo aplicados habitualmente en equipos de altas prestaciones, como clusters de computadores o máquinas de memoria compartida. Sin embargo, la introducción de procesadores multi-núcleo, con varios núcleos de procesamiento y

cachés en un solo microchip, permite aplicar procesamiento paralelo incluso en computadores personales y otros dispositivos utilizando componentes software estándar. Algorítmicamente, el paralelismo puede ser explotado utilizando diferentes paradigmas: maestro-trabajador, islas, y difusión [4].

A. Implementaciones Paralelas en la Literatura

Diferentes algoritmos paralelos han sido diseñados para resolver VRPs. Uno de los trabajos pioneros en este campo fue el de Taillard [5], quién presentó un método paralelo que descompone el VRP en regiones polares incluyendo un gran número de vehículos en sub-problemas que se reselven de forma independiente. Otros autores propusieron una heurística paralela basada en búsqueda tabú para el VRPTW que utilizaba un esquema maestro-trabajador, donde cada trabajador lleva a cabo una búsqueda tabú y el maestro coordina el trabajo y distribuye a los trabajadores las soluciones iniciales [6]. La paralelización de la búsqueda tabú también fue aplicada en [7], y los resultados obtenidos mostraron que el algoritmo asíncrono maestro-trabajador mejoraba a la versión síncrona en términos de speedup, mientras que también permitía obtener resultados comparables a esos obtenidos por la versión síncrona y secuencial de la búsqueda tabú. Otras aproximaciones que también aplican búsqueda tabú paralela para resolver problemas de enrutado de vehículos sin ventanas temporales (es decir, CVRP) mediante el uso de un modelo maestro-trabajador en el que cada trabajador ejecuta un procedimiento de búsqueda tabú con diferentes parámetros, mientras que el maestro es responsable de recolectar la mejor solución local de los trabajadores y trasmitir el mejor de todos ellos entre dichos trabajadores para iniciar una nueva iteración [8]. Algunos métodos híbridos basados en combinar búsqueda local heurística con programación entera han sido también paralelizados para resolver el CVRP [9]. Otros autores han resuelto el VRPTW utilizando el esquema maestro-trabajador en el que el componente maestro coordina los operadores genéticos y mantiene la selección de padres mientras que los trabajadores ejecutan de forma concurrente los operadores de reproducción y mutación [10]. La paralelización maestro-trabajador también se ha aplicado para resolver la versión de dos fases del VRPTW [11], concluyendo que se obtiene una ganancia en tiempo de cómputo requerido para generar soluciones de una determinada calidad. Algunos algoritmos paralelos aplicados en el VRPTW utilizan diferentes estrategias para inicializar soluciones, búsqueda tabú o algoritmos genéticos como métodos heurísticos de búsqueda, y procedimientos de post-optimización que aplican operadores de búsqueda local [12]. El VRPTW también ha sido resuelto con algoritmos paralelos basados en enfriamiento simulado haciendo uso de arquitecturas SMP, y utilizando OpenMP y MPI [13], [14].

Esta investigación es, para nuestro conocimiento, la primera que compara ambos paradigmas paralelos en meta-heurísticas poblacionales para resolver el VRPTW haciendo uso de procesadores multi-núcleo.

B. Procedimiento

A continuación describimos la heurística basada en enfriamiento simulado y su paralelización utilizando ambos paradigmas.

B.1 Algoritmo Secuencial

Los componentes principales del algoritmo aquí paralelizado son la representación de las soluciones, el método de construcción de soluciones y la condición de parada, los operadores de búsqueda según las estructuras de vecindario, la estrategia para mantener la diversidad de las soluciones, y la selección de soluciones vecinas a la actual.

- *La representación de las soluciones* se realiza mediante una codificación entera. Dado que el método utilizado es poblacional, la población P consta de un total de p individuos (soluciones), P=$\{I_1, I_2, \ldots, I_p\}$, cada uno de los cuales representa las rutas recorridas por los K vehículos para suministrar a los N clientes. Así, cada individuo, I_i, se representa mediante cromosomas, C_{ik}, que consiste en un conjunto variable de genes, $C_{ik} = \{0, G^1_{ik}, G^2_{ik}, ..., G^l_{ik}, 0\}$ que representan la ruta del k-ésimo vehículo en el i-ésimo individuo, con $1 \leq G^j_{ik} \leq N$. Así, el cromosoma $C_{5,3} = \{0, 7, 4, 15, 11, 0\}$ indica que el tercer vehículo del quinto individuo parte del centro logístico, y visita a los clientes 7, 4, 15, y 11, antes de volver a dicho centro logístico.
- *Las soluciones iniciales* se construyen utilizando tres estrategias, todas ellas teniendo en cuenta que es obligatorio satisfacer las ventanas temporales y las restricciones de capacidad. La primera consiste en asignar clientes aleatoriamente a los vehículos hasta que sean suministrados dichos clientes. La segunda estrategia es similar a la primera, pero los clientes son insertados de acuerdo en el orden correspondiente a su identificador de nodo. La tercera heurística (Time Window based Insertion Heuristic, TWIH) asigna clientes a los vehículos en un orden ascendente de ventanas temporales, visitando primeramente aquellos clientes cuyo tiempo de servicio comienza antes.
- *Los operadores de búsqueda* tienen como objetivo mejorar la calidad de las soluciones creadas mediante las heurísticas de inicialización. En la literatura del VRPTW encontramos un elevado elevado número de operadores de mutación [1], [15], de entre los cuales hemos implementado diez con el objetivo de diversificar la búsqueda. Algunos de ellos se basan en elegir un cliente y relocalizarlo en un orden diferente en el mismo vehículo, otras se basan en modificar el vehículo asignado a los clientes, mientras que otros operadores dividen, crean, o eliminan rutas.
- *El Enfriamiento Simulado (Simulated Annealing, SA)* [16] es una meta-heurística de

búsqueda local que simula el proceso seguido por los átomos de un metal cuando se funde y posteriormente se enfría lentamente. En términos computacionales, este proceso implica que las soluciones iniciales tienen asociadas una temperatura inicial elevada, Ti, la cual se va reduciendo progresivamente según un factor de enfriamiento, T_{factor}, hasta que cae debajo de un determinado umbral, T_{umbral}, de forma que en un instante dado, aquellas soluciones del vecindario, generadas a través de un operador de variación (mutación), que mejoren a la solución de partida serán directamente aceptadas, mientras que si la empeoran se aceptará con una probabilidad que decrece conforme se reduce la temperatura actual, t. La variable t se incluye dentro de la denominada función de Metrópolis [17] y actúa como factor de probabilidad para aceptar o rechazar una determinada solución. La estrategia aquí utilizada, denominada Multiple-Temperature Simulated Annealing (MT-SA), optimiza una población de p individuos utilizando los diez operadores de mutación comentados anteriormente, a la vez que se establecen esquemas de enfriamiento diferentes en cada individuo [18] con el objetivo de diversificar la búsqueda. En concreto, se establece un intervalo de temperaturas iniciales $[Ti_{min}, Ti_{max}]$, de forma que la temperatura inicial del individuo I_1 es Ti_{min}, la del individuo I_p es Ti_{max}, y el resto se distribuyen regularmente a lo largo de este intervalo.

B.2 Algoritmos Paralelos

El objetivo de la implementación paralela aquí presentada (pMT-SA) no es tanto reducir el tiempo de ejecución del algoritmo secuencial (MT-SA), sino mejorar la calidad de las soluciones obtenidas por MT-SA sin incrementar el tiempo de ejecución. Con el objetivo de implementar algoritmos paralelos que presenten las mismas características del código secuencial (una simulación paralela del código secuencial), tanto las paralelizaciones basadas en el modelo maestro-trabajador como en el modelo de islas utilizan comunicaciones síncronas, esto es, no se hace uso de las primitivas de paso de mensajes asíncronas (MPI), ni la cláusula *nowait* (OpenMP). Ambos paradigmas han sido adaptados en nuestro estudio como mostramos a continuación:

i) Paradigma maestro-trabajador con OpenMP: la hebra maestra inicializa la población de soluciones (cada una conteniendo un conjunto válido de rutas que visitan a todos los clientes a la vez que se cumplen las restricciones), de forma que, en cada iteración, dicha hebra maestra distribuye los p individuos de la población P entre el número de hebras ejecutadas (NP), incluyendose a sí misma, de forma que cada hebra es responsable de optimizar p/NP individuos de acuerdo a los operadores de búsqueda en conjunción con la función de Metropolis en la que se basa la aceptación de soluciones en el enfriamiento simulado. Una vez las hebras trabajadoras han computado sus soluciones asignadas, las retornan a la hebra maestra, la cual determina la mejor solución global, reemplazando un porcentaje de individuos con esa solución, y volviendo a distribuir trabajo entre las hebras. La hebra maestra también es responsable de controlar la condición de parada y de retornar la mejor solución obtenida;

ii) Paradigma de islas con MPI: cada proceso (isla) inicializa y optimiza p/NP individuos de forma autónoma. De forma periódica, la mejor solución de cada isla es enviada a un proceso central el cual, de forma temporal, es responsable de determinar la mejor solución global y distribuirla entre las restantes islas. Esas islas copian la solución recibida en un determinado porcentaje de soluciones de la población, tras lo cual continúan el proceso de búsqueda. Cuando la condición de parada se cumple, todas las islas envían las soluciones al proceso central, el cual devuelve la mejor solución global.

IV. Análisis Experimental

El computador utilizado consta de un Intel Core 2 Quad Processor Q6600 (4 núcleos, 2.40 GHz, 8MB de caché y 4 GB RAM). El algoritmo secuencial MT-PSA, programado en C++, se ha paralelizado utilizando MPI (MPICH2 versión 1.2.1p1), y OpenMP (versión 3.1). Para obtener resultados estadísticos significativos, se han lanzado 15 ejecuciones independientes de pMT-SA usando NP={1,2,3,4} procesos/hebras para cada benchmark.

A. Benchmarks

La evaluación de pMT-SA en la resolución del VRPTW se lleva a cabo utilizando el conjunto de benchmarks propuesto por Solomon [19], que se compone de 56 instancias divididas en seis categorías, todas ellas con 25 vehículos y 100 clientes. Cada benchmark incluye las capacidades máximas de los vehículos, así como las características de los clientes (coordenadas geográficas, demandas, ventanas temporales y tiempo de servicio). La distancia entre dos clientes es calculada a partir de sus coordenadas cartesianas utilizando distancias euclídeas. Las seis categorías son C1, C2, R1, R2, RC1, y RC2, donde los benchmarks de categoría R tienen los clientes distribuidos aleatoriamente, los de la categoría C están concentrados, siendo la categoría RC una mezcla de ambas. Por otro lado, los problemas de la categoría 1 (C1, R1, y RC1) tienen ventanas temporales más estrechas que los de la categoría 2 (C2, R2, y RC2). Las implementaciones paralelas se evaluan en dos instancias del problema de cada uno de esos seis grupos de benchmarks: C103, C108, C203, C208, R103, R108, R203, R208, RC103, RC108, RC203, y RC208, que son representativos de todos los tipos C/R/RC.

B. Configuración Paramétrica

El algoritmo secuencial utiliza una población de 200 individuos ($|P|$=p=200), que en el caso de aplicar MPI son distribuidos entre las islas, y en el caso de aplicar OpenMP islas son gestionadas mediante *pragma omp parallel for*. La probabilidad de aplicar el operador de mutación es del 25%. Si se aplica la mutación, cada una de las diez variantes del operador de mutación se aplica con una probabilidad aleatoria entre el 5% y el 15% (todas ellas han de sumar 100%). Como se comentó previamente, cada individuo tiene su propio esquema de enfriamiento, de forma que se establece un intervalo de temperaturas iniciales Ti=[10,1000] y un esquema de enfriamiento lento ($T_{cooling}$=0.995), mientras que la temperatura mínima es T_{stop}=0.001. Si no se cumple la condición de parada y la temperatura actual cae por debajo de T_{stop}, la temperatura se reinicializa (t=Ti) y el proceso de búsqueda continúa. Cuando los procesos o hebras se comunican para compartir sus mejores soluciones, la mejor se copia en el 25% de las soluciones de la población (paradigma maestro-trabajador) o en cada isla (paradigma basado en islas). De acuerdo a nuestros resultados preliminares, este porcentaje (25%) representa un valor compromiso adecuado que permite la independencia del proceso de búsqueda derivado del modelo de islas y la presión elitista derivada de la aplicación de la meta-heurística, mientras que un porcentaje mayor deteriora el rendimiento. Ambas implementaciones paralelas se ejecutan utilizando dos, cuatro, y ocho procesos/hebras, y dado que el procesador utilizado consta de cuatro núcleos con multi-hebra simultánea, utilizar dos, cuatro, y ocho procesos permite no sólo analizar la mejora ofrecida por el incremento del número de núcleos, sino también las ventajas proporcionadas por la tecnología multi-hebra. Cuando se comparan diferentes algoritmos o implementaciones, es posible determinar que técnica es mejor que otra si una de ellas obtiene un mejor rendimiento (mejores soluciones) en un tiempo computacional equivalente. Para los investigadores en optimización computacional, este coste suele medirse en términos de un mismo número de evaluaciones de la función objetivo. No obstante, este criterio asume que el coste de las restantes operaciones es similar en todas los algoritmos o implementaciones comparadas. Sin embargo, resolver el VRPTW requiere no sólo evaluar el fitness de las soluciones, sino también verificar las restricciones, actualizar las temperaturas, aceptar o rechazar soluciones de acuerdo al criterio de Metropolis, etc. Además, es desconocido si el paradigma basado en islas utilizando MPI necesitará mayor tiempo de cómputo en llevar a cabo el paso de mensajes, que la paralelización con OpenMP en gestionar el modelo de ejecución fork-and-join. Por tanto, el tiempo de ejecución parece la mejor forma de medir el coste computacional, siendo además un criterio recientemente utilizado a la hora de resolver problemas de enrutado de vehículos [20].

C. Resultados y Discusión

Dado que el objetivo de esta investigación es evaluar y comparar el rendimiento de diferentes implementaciones paralelas del algoritmo MT-SA, la primera tarea es ajustar los parámetros. Como se comentó anteriormente, el rendimiento de la paralelización basada en islas suele verse influida por el ratio de comunicación, que debería ser cuidadosamente ajustado en cada aplicación particular. Con este objetivo, la implementación paralela basada en el modelo de islas se evalúa en el conjunto de benchmarks analizado utilizando tres ratios de comunicación (CR={100,1000,10000}). Con los resultados obtenidos a partir de 15 ejecuciones diferentes, se observa que la mejor configuración corresponde a CR=100, parámetro que será el utilizado en los siguientes resultados mostrados.

TABLA I
COMPARATIVA ENTRE EL MODELO DE ISLAS CON MPI FRENTE AL MODELO MAESTRO-TRABAJADOR CON OPENMP (CRITERIO DE PARADA: TIEMPO DE EJECUCIÓN IGUAL A 60 SEGUNDOS).

	NP=1		*NP*=2			
	Serial		*MPI*		*OMP*	
	MED	*S.D.*	*MED*	*S.D.*	*MED*	*S.D.*
R103	1629.5	48.9	1406.9	37.4	1504.7	28.0
R108	1464.3	65.9	1194.0	51.4	1290.7	57.4
R203	1509.8	99.7	1204.6	71.7	1368.4	58.7
R208	1261.2	122.4	986.8	72.4	1103.5	98.2
C103	1456.7	62.5	1167.1	51.3	1432.8	90.8
C108	1663.0	75.0	1257.3	79.1	1451.7	55.9
C203	1381.8	102.2	982.7	57.8	1212.2	87.1
C208	846.6	42.9	749.5	39.7	818.3	39.8
RC103	1776.8	68.7	1515.5	49.1	1618.9	36.8
RC108	1656.7	60.6	1418.3	58.1	1514.0	64.1
RC203	1657.7	121.0	1359.7	66.3	1537.2	103.7
RC208	1463.1	101.4	1194.7	75.1	1424.5	86.1
Media	1480.6	80.9	1203.1	59.1	1356.4	67.22

	NP=4				*NP*=8			
	MPI		*OMP*		*MPI*		*OMP*	
	MED	*S.D.*	*MED*	*S.D.*	*MED*	*S.D.*	*MED*	*S.D.*
R103	**1349.5**	34.7	1478.6	32.9	1521.7	31.8	1405.3	43.0
R108	**1068.3**	27.2	1140.2	23.3	1341.4	38.1	1146.8	53.0
R203	**1105.8**	45.1	1213.3	58.5	1389.4	52.8	1172.2	59.0
R208	**877.3**	44.0	1011.8	80.0	1124.0	29.2	979.5	50.9
C103	**1045.9**	35.4	1250.6	88.4	1329.6	51.8	1208.1	79.7
C108	**1109.8**	88.2	1238.3	108.6	1172.8	55.1	1250.4	95.1
C203	**873.2**	57.1	1008.4	96.8	1164.4	70.9	971.5	125.6
C208	**715.9**	15.6	791.7	28.7	783.1	31.5	774.7	25.7
RC103	**1412.4**	34.5	1514.1	36.6	1641.2	31.9	1517.7	37.9
RC108	**1328.2**	40.6	1417.4	30.0	1523.2	43.9	1434.9	46.9
RC203	**1241.4**	63.1	1350.4	67.2	1562.4	68.2	1355.7	59.2
RC208	**1082.2**	55.6	1224.3	47.6	1311.0	52.2	1193.5	82.9
Media	**1100.8**	45.1	1219.9	58.2	1322.0	46.5	1200.9	63.2

Una vez ajustado el ratio de comunicaciones para el modelo basado en islas con MPI, el siguiente paso es comparar el rendimiento del modelo de islas con el basado en el modelo maestro-trabajador. La Tabla I muestra la mediana estadística (MED), en términos

de distancia viajada obtenida por el modelo de islas implementado con MPI y el esquema maestro-trabajador utilizando OpenMP en ejecuciones de 60 segundos. Además de la mediana, se ofrecen resultados de la desviación estandar (S.D.) como medida de dispersión. A partir de los resultados obtenidos en las 15 ejecuciones independientes, se observa que el modelo de islas programado con MPI y 4 procesos obtiene los mejores resultados en todas las instancias, aunque el rendimiento de esta estrategia decae cuando se utilizan 8 procesos. Este comportamiento se debe al efecto denominado sobresubscripción (oversubscription) [21], por el que un incremento en el número de procesos por encima del número de núcleos disponibles conlleva un mayor número de comunicaciones (paso de mensajes), lo cual provoca un overhead que no compensa el uso del esquema multi-hebra simultáneo (implementado por cada núcleo en el microprocesador). Por el contrario, el paradigma maestro-trabajador programado con OpenMP no obtiene resultados tan buenos como el modelo de islas programado con MPI al usar 4 hebras, pero sí mejora su propio rendimiento cuando se hace uso de 8 hebras, es decir, se aprovecha del esquema multi-hebra simutáneo [22]. Por último, señalar que la desviación estandard de las 15 ejecuciones independientes es mayor en la implementación secuencial, lo cual denota que el paralelismo también incrementa la robustez del algoritmo.

V. Conclusiones

Los problemas de enrutado de vehículos son NP-duros, por lo que resulta habitual hacer uso de métodos heurísticos para su resolución. Sin embargo, en determinadas aplicaciones es necesario abordar instancias de gran tamaño u obtener soluciones de alta calidad en un tiempo reducido, resultando conveniente incorporar técnicas que mejoren la eficiencia de dichas heurísticas. Este trabajo analiza los beneficios proporcionados por paralelizaciones de meta-heurísticas poblacionales basadas en los modelos maestro-trabajador y modelo basado en islas, haciendo uso de MPI y OpenMP, con el objetivo de mejorar la calidad de las soluciones obtenidas por el algoritmo secuencial sin incrementar el tiempo de cómputo. A partir de los resultados obtenidos, se han extraido dos conclusiones principales: el esquema maestro-trabajador programado con OpenMP toma ventaja tanto del esquema multi-hebra simultáneo como de la memoria compartida siendo capaz de mejorar su rendimiento utilizando un número de hebras superior al número de núcleos disponibles; el modelo de islas programado con MPI mejora al paradigma maestro-trabajador con OpenMP cuando el número de procesos no es mayor que el número de núcleos, mientras que su rendimiento decae en caso contrario.

Agradecimientos

Investigación financiada por el Ministerio de Ciencia e Innovación a través de los proyectos TIN2008-01117 y TIN2012-32039. R. Baños agradece el contrato postdoctoral *Juan de la Cierva* financiado por el Ministerio de Economía y Competitividad.

Referencias

[1] N.A. El-Sherbeny, *Vehicle routing with time windows: an overview of exact, heuristic and metaheuristic methods*, Journal of King Saud University (Science) 22(3), 123-131 (2010).

[2] Y. Nagata, O. Bräysy, W. Dullaert, *A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows*, Computers & Operations Research 37, 724-737 (2010).

[3] O. Bräysy, M. Gendreau, *Vehicle routing problem with time windows, part II: metaheuristics*, Transportation Science 39, 119-139 (2005).

[4] C.A., Coello, G.B. Lamont, D.A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*, Genetic and Evolutionary Computation Series, Springer (2007).

[5] É. Taillard, *Parallel iterative search methods for vehicle routing problems*, Networks 23(8), 661-673 (1993).

[6] F. Badeau, F. Guertin, M. Gendreau, J-Y. Potvin, E. Taillard, A parallel tabu search heuristic for the vehicle routing problem with time windows, Transp. Res.-Part C 5(2), 109-122 (1997) .

[7] A. Beham, Parallel tabu search and the multiobjective vehicle routing problem with time windows, Springer Lecture Notes in Computer Science 4739, 829-836 (2007).

[8] C. Rego, *Node-ejection chains for the vehicle routing problem: Sequential and parallel algorithms*, Parallel Computing 27(3), 201-222 (2001).

[9] C. Gröer, B. Golden, E. Wasil, A parallel algorithm for the vehicle routing problem, INFORMS Journal on Computing 23(2), 315-330 (2011).

[10] J. Berger, M. Barkaoui, *A parallel hybrid genetic algorithm for the vehicle routing problem with time windows*, Computers & Operations Research 31(12), 2037-2053 (2004).

[11] H. Gehring, J. Homberger, *Parallelization of a two-phase metaheuristic for routing problems with time windows*, Journal of Heuristics 8(3), 251-276 (2002).

[12] A. Le Bouthillier, T.G. Crainic, *A cooperative parallel meta-heuristic for the vehicle routing problem with time windows*, Computers & Operations Research 32(7), 1685-1708 (2005).

[13] Z.J. Czech, W. Mikanik, R. Skinerowicz, Implementing a parallel simulated annealing algorithm, Springer Lecture Notes in Computer Science 6067, 146-155 (2009).

[14] A. Debudaj-Grabysz, R. Rabenseifner, Load balanced parallel simulated annealing on a cluster of SMP nodes, Springer Lecture Notes in Computer Science 4128, 1075-1084 (2006).

[15] G.B. Alvarenga, G.R. Mateus, G. de Tomic, *A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows*, Computers & Operations Research 34(6), 1561-1584 (2007).

[16] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, *Optimization by simulated annealing*, Science 220(4598), 671-680 (1983).

[17] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A. Teller, E. Teller, *Equation of state calculations by fast computing machines*, The Journal of Chemical Physics 21(6), 1087-1092 (1953).

[18] R. Baños, C. Gil, J. Ortega, F.G. Montoya, *A parallel multilevel metaheuristic for graph partitioning*, Journal of Heuristics 10(3), 315-336 (2004).

[19] M.M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Operations Research 35(2),254-65 (1987).

[20] H. Derbel, B. Jarboui, S. Hanafi, H. Chabchoub, *An iterated local search for solving a location-routing problem*, Electronic Notes in Discrete Mathematics 36, 875-882 (2010).

[21] C. Iancu, S. Hofmeyr, Y. Zheng, F. Blagojevi, Oversubscription on Multicore Processors, IEEE International Parallel and Distributed Processing Symposium, 2010, pp. 1-11.

[22] S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, R.L. Stamm, D.M. Tullsen, Simultaneous multithreading: A platform for next-generation processors. IEEE Micro 17 (1997) 12-19.

Virtualización Remota de GPUs: Evaluación de Soluciones Disponibles para CUDA

Carlos Reaño[1], Adrián Castelló[1], Sergio Iserte[1], Antonio J. Peña[2], Federico Silla[3], Rafael Mayo[1], Enrique S. Quintana-Ortí[1] y José Duato[3]

Resumen— **En el presente artículo realizamos una comparativa de las distintas soluciones disponibles para la utilización de GPUs compatibles con CUDA de forma remota. Para ello, además de presentar las características de cada una de ellas, describiendo sus ventajas e inconvenientes, hemos realizado una serie de pruebas con la finalidad de evaluarlas y poder compararlas entre ellas en base al ancho de banda y la latencia que cada solución presenta.**

En cuanto a la latencia se refiere, DS-CUDA ha mostrado muy buenos resultados en copias que utilizaban memoria de CPU paginable, mientras que rCUDA ha obtenido la mejor latencia de las distintas soluciones estudiadas cuando las copias utilizaban memoria de CPU no paginable. En lo referente al ancho de banda, generalmente rCUDA ha presentado los mejores resultados. No obstante, cuando el tamaño de los datos transferidos era pequeño, entre 1MB y 4MB, su rendimiento bajaba considerablemente, sobre todo en el caso de las copias que utilizaban memoria de CPU no paginable.

Palabras clave— **CUDA, GPGPU, virtualización, computación de altas prestaciones.**

I. Introducción

En los últimos años, los clusters HPC (*High Performance Computing*, computación de altas prestaciones) han evolucionado hacia plataformas heterogéneas integrando tanto CPUs como hardware de propósito específico y aceleradores, como por ejemplo las GPUs (*Graphics Processing Units*, unidades de procesamiento gráfico) [1]. Normalmente, en este tipo de sistemas se instalan uno o más aceleradores en cada nodo del cluster. No obstante, este enfoque presenta varios inconvenientes:

- Coste de adquisición: además de incrementar el coste de adquisición, el uso de aceleradores también aumenta los costes de mantenimiento, administración y espacio [2].
- Consumo energético: por ejemplo, en el caso de las GPUs, éstas pueden incrementar la cantidad de energía total requerida por un nodo HPC en un porcentaje considerable [3].
- Utilización de los aceleradores: no es muy común que todos los aceleradores del cluster sean utilizados completamente todo el tiempo, ya que muy pocas aplicaciones presentan un nivel de paralelización tan elevado.

[1]DICC, Universitat Jaume I (UJI), 12.071 - Castellón (España), e-mail: carregon@gap.upv.es, {adcastel, siserte, mayo, quintana}@icc.uji.es.
[2]MCS, Argonne National Laboratory, 60.439 - Illinois (USA), e-mail: apenya@mcs.anl.gov.
[3]DISCA, Universitat Politècnica de València (UPV), 46.022 - Valencia (España), e-mail: {fsilla, jduato}@disca.upv.es.

Bajo estas premisas, la virtualización de aceleradores en el ámbito de la HPC parece ser una buena idea, puesto que conduce a la instalación de menos aceleradores. De esta manera, se reducen los costes de adquisición y el consumo, mientras que se incrementa la utilización de los mismos.

En particular, en este artículo nos centraremos en GPUs como aceleradores. A la hora de acelerar las aplicaciones mediante GPUs, las mejoras se obtienen descargando en éstas las partes de código computacionalmente intensivas. Para llevarlo a cabo, han aparecido diversas soluciones como, por ejemplo, CUDA [4] u OpenCL [5]. En este trabajo únicamente consideraremos las aplicaciones aceleradas con CUDA, por ser la opción más extendida.

Cuando se virtualizan GPUs, las GPUs físicas se instalan sólo en algunos nodos del cluster. Posteriormente, son compartidas con el resto de nodos de forma remota. Así, los nodos que tienen GPUs se convierten en servidores que proporcionan servicios GPGPU (*General Purpose Computing on GPUs*, computación de propósito general en GPUs) al resto de nodos dentro del cluster.

Las soluciones más conocidas para la virtualización de GPUs compatibles con CUDA son: rCUDA [6], GVirtuS [9], DS-CUDA [10], vCUDA [11], GViM [12], GridCuda [13] y V-GPU [14]. Todas estas soluciones presentan una arquitectura similar compuesta por dos partes: una parte cliente, instalada en el nodo que solicita servicios de aceleración a la GPU, y una parte servidora instalada en el sistema que posee físicamente la GPU.

Evidentemente, el hecho de utilizar una GPU de forma remota introduce algunos sobrecostes con respecto a utilizar una GPU local, principalmente:

- El sobrecoste de gestión introducido por la propia solución de virtualización de la GPU.
- El sobrecoste introducido por la red de comunicaciones utilizada para acceder a la GPU remota.

Para minimizar el impacto del sobrecoste de la red, es muy importante que las soluciones que virtualizan las GPUs aprovechen al máximo el ancho de banda de la misma, e intenten reducir la latencia al mínimo. En este artículo realizamos una comparativa de las distintas soluciones de virtualización de GPUs disponibles para CUDA. Para ello, además de presentar las características de cada una de ellas, describiendo sus ventajas e inconvenientes, realizaremos un conjunto de pruebas con la finalidad de evaluarlas y poder compararlas entre ellas en base al ancho de

banda y la latencia que presentan.

El resto del artículo está estructurado de la siguiente manera: en la Sección II describimos cada una de las tecnologías disponibles; a continuación, evaluamos algunas de ellas en la Sección III; para finalizar, en la Sección IV comentamos las conclusiones.

II. Soluciones disponibles para el uso de GPUs de forma remota

En esta sección describimos las características más relevantes de las soluciones que en la actualidad permiten utilizar GPUs compatibles con CUDA de forma remota.

A. rCUDA: remote CUDA

rCUDA [6], [7], [8] proporciona a las aplicaciones acceso transparente a GPUs instaladas en nodos remotos, de manera que las aplicaciones no son conscientes de que realmente están accediendo a dispositivos instalados en otros nodos del cluster. Para ello, la arquitectura de rCUDA está estructurada como un sistema cliente-servidor distribuido, tal y como muestra la Figura 1. Así, las peticiones de las aplicaciones que requieren GPUs son redirigidas por el cliente de rCUDA al servidor a través de la capa de comunicaciones.

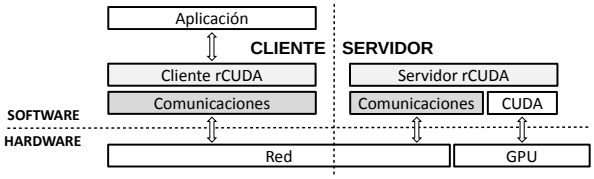


Fig. 1. Arquitectura de rCUDA.

Desde el punto de vista de las aplicaciones, el cliente rCUDA presenta la misma interfaz que la biblioteca de CUDA original en tiempo de ejecución, de tal forma que éstas se comportan de manera similar a como lo harían si tuvieran acceso local a las GPUs. Cuando el servidor recibe las peticiones, las procesa y ejecuta accediendo a las GPUs físicas. Posteriormente, envía los resultados de nuevo al cliente, el cual los hace llegar a la aplicación.

Por su parte, la capa de comunicaciones que conecta el cliente rCUDA con el servidor proporciona una API genérica de comunicaciones que posibilita la implementación de módulos de comunicaciones específicos para cada tecnología de red subyacente, permitiendo así el máximo aprovechamiento de la misma. Actualmente existen módulos de comunicaciones específicos para Ethernet e InfiniBand.

La última versión de rCUDA soporta la versión 5 del Runtime API de CUDA, a excepción de los módulos de interoperabilidad con gráficos. Además, también soporta el resto de bibliotecas CUDA, tales como CUBLAS [15], CUFFT [16] o Thrust [17]. No es necesario realizar ninguna modificación en las aplicaciones para utilizar rCUDA.

B. GVirtuS: Generic Virtualization Service

GVirtuS [9] intenta llenar el hueco existente entre los clusters propietarios equipados con dispositivos que permiten GPGPU, y los clusters virtuales de pago de altas prestaciones empleados a través de redes públicas o privadas. GVirtuS permite a una máquina virtual acceder a una GPU de una manera transparente. Es independiente del *hypervisor* de la máquina virtual, y no está limitado únicamente a virtualizar GPUs compatibles con CUDA. Su rendimiento ha sido evaluado en diversos escenarios, como por ejemplo proporcionando servicios GPGPU en un cluster HPC basado en *cloud computing* y compartiendo remotamente dispositivos que ofrecen servicios GPGPU entre nodos HPC.

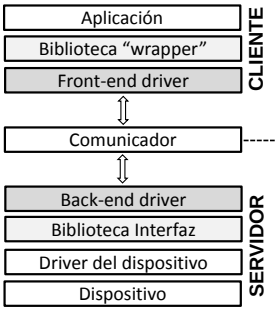


Fig. 2. Arquitectura de GVirtuS.

En la Figura 2 podemos ver la arquitectura de GVirtuS. La aplicación sigue utilizando el mismo interfaz que la biblioteca original virtualizada. El cliente utiliza el *front-end* de GVirtuS para realizar peticiones. El servidor, por su parte, emplea el *back-end* de GVirtuS para atender dichas peticiones. La comunicación entre ambos puede realizarse utilizando diversas tecnologías:

- Sockets Unix
- TCP/IP
- Máquinas Virtuales Xen [18]: XenLoop
- Máquinas Virtuales VMWare [19]: VMCI (*Virtual Machine Communication Interface*)
- Máquinas Virtuales KVM [20]: VMchannel

La última versión disponible de GVirtuS cubre un subconjunto de funciones de la versión 3.2 del Driver API y del Runtime API de CUDA. No permite el uso del resto de bibliotecas CUDA. Por otro lado, soporta parte de las funciones ofrecidas por OpenCL, y no requiere cambios en las aplicaciones. Aunque los mejores resultados los obtiene con máquinas virtuales KVM en conjunción con QEMU [21], en este trabajo no estamos analizando el uso de GPUs con máquinas virtuales, y por ello utilizaremos su versión TCP/IP para acceder a GPUs remotas.

C. DS-CUDA: Distributed Shared CUDA

DS-CUDA [10] es un *middleware* que proporciona acceso a GPUs CUDA distribuidas sobre los nodos una red. De este modo el usuario, al igual que ocurría con rCUDA, puede utilizar todas las GPUs de los distintos nodos de la red como si éstas estuvieran

instaladas en su propio nodo.

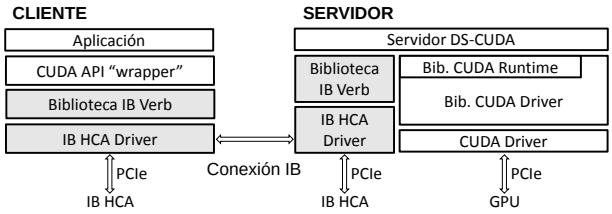


Fig. 3. Arquitectura de DS-CUDA.

La Figura 3 muestra la arquitectura de DS-CUDA. Al igual que en las soluciones anteriores, en el nodo cliente, la aplicación interactúa con una biblioteca que sustituye a la original de CUDA. En el nodo servidor, el servidor de DS-CUDA redirige dichas llamadas a la biblioteca nativa de CUDA. La comunicación entre el cliente y los nodos servidor se realiza utilizando *InfiniBand Verbs* por defecto, aunque también permite utilizar llamadas a procedimiento remoto (RPC, *Remote Procedure Calls*).

La última versión de DS-CUDA soporta un subconjunto de funciones de la versión 4.1 de CUDA. No permite copias entre zonas de memoria GPU, ni con memoria CPU no paginable, ni tampoco copias de tamaño superior a 32MB. Además, es necesario recompilar las aplicaciones enlazando con la biblioteca de DS-CUDA.

Sin embargo, un aspecto importante de DS-CUDA es que proporciona tolerancia a fallos, lo cual puede resultar muy útil si tenemos en cuenta que muchas GPUs, como las GeForce, en ocasiones podrían presentar errores de memoria debido a que carecen de mecanismos ECC (*Error Check and Correct*, detección y corrección de errores).

D. vCUDA

vCUDA [11] es una solución que proporciona servicios GPGPU a máquinas virtuales. De manera similar a los casos anteriores, vCUDA permite a aplicaciones ejecutándose en máquinas virtuales hacer uso de GPUs, lo cual puede resultar beneficioso para el rendimiento de aplicaciones dentro del ámbito de la HPC. Como en el resto de soluciones, la idea clave en su diseño es la interceptación de las llamadas al API de CUDA y su redirección.

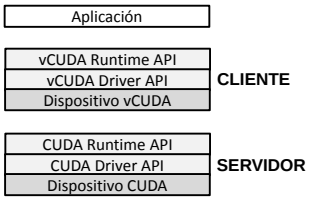


Fig. 4. Arquitectura de vCUDA.

En la Figura 4 exponemos cómo está estructurado vCUDA. La aplicación se ejecuta en una máquina virtual y accede a las bibliotecas de vCUDA, que sustituyen a las de CUDA. Las llamadas interceptadas por vCUDA, son redirigidas al sistema operativo anfitrión, donde sí se utilizan las bibliotecas originales de CUDA. Las comunicaciones entre la máquina virtual y la máquina real se realizan utilizando el protocolo XML-RPC [22], que utiliza HTTP para el transporte y XML para la codificación.

La última versión de vCUDA parece dar soporte a la versión 1.1 del Runtime API de CUDA, aunque en las publicaciones consultadas no hay referencias en este sentido.

E. GViM: GPU-accelerated Virtual Machines

GViM [12] es un sistema diseñado para virtualizar y gestionar los recursos de un sistema de propósito general acelerado por procesadores gráficos. Está construido a partir de soluciones de virtualización existentes, integrando nuevos mecanismos que permiten un soporte mejorado para máquinas virtuales aceleradas por GPUs.

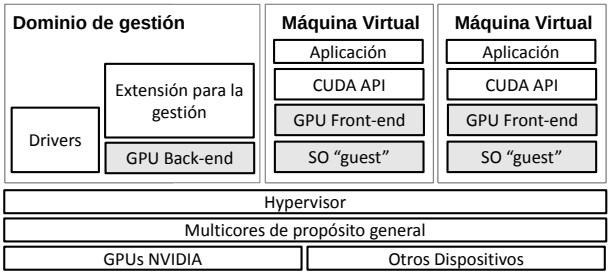


Fig. 5. Arquitectura de GViM.

La Figura 5 presenta la arquitectura de un sistema GPGPU virtualizado con GViM. La plataforma hardware está formada por procesadores de propósito general (por ejemplo, x86) y aceleradores gráficos especializados, en este caso GPUs NVIDIA. Las máquinas virtuales ejecutan aplicaciones que pueden requerir acceso a las GPUs de forma concurrente. El control de las GPUs físicas y del resto de dispositivos lo realiza íntegramente el dominio de gestión. Esto implica que todos los accesos a las GPUs desde las máquinas virtuales son enrutados a través del *GPU Front-end* a dicho dominio de gestión, y éste se encarga de redirigirlos a los dispositivos reales mediante el *GPU Back-end*.

La última versión de vCUDA parece dar soporte a la versión 1.1 de CUDA, aunque no se específica si cubre todas las funcionalidades de la misma.

F. GridCuda: A Grid-Enabled CUDA Programming Toolkit

GridCuda [13] proporciona una plataforma para que los usuarios desarrollen programas utilizando el API de CUDA, explotando los recursos GPGPU disponibles en redes computacionales para ejecutar dichos programas. Cuando una función CUDA es invocada desde la aplicación usuario, es redirigida de forma transparente a GPUs remotas por medio de llamadas a procedimiento remoto (RPC).

GridCuda está formado principalmente por dos componentes: el cliente y el servidor, tal y como podemos ver en la Figura 6. El cliente GridCuda

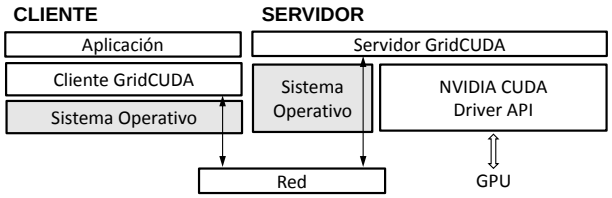


Fig. 6. Arquitectura de GridCuda.

es el encargado de redirigir las llamadas CUDA de las aplicaciones a las GPUs remotas. Por otro lado, el servidor GridCuda es utilizado por los nodos de la red que disponen de GPUs para recibir las peticiones del cliente, e invocar al driver de CUDA para llevar a cabo las operaciones solicitadas.

La última versión de GridCuda parece dar soporte a la versión 2.3 de CUDA, aunque no hemos encontrado información sobre si cubre todas las funcionalidades de la misma.

G. V-GPU: GPU Virtualization

Partiendo de una PaaS (*Platform as a Service*, plataforma como servicio) para juegos online basada en computación con GPUs, V-GPU [14] es una herramienta comercial dirigida a crear una infraestructura de GPUs a gran escala para soportar millones de jugadores online. Podemos ver cómo está estructurada en la Figura 7.

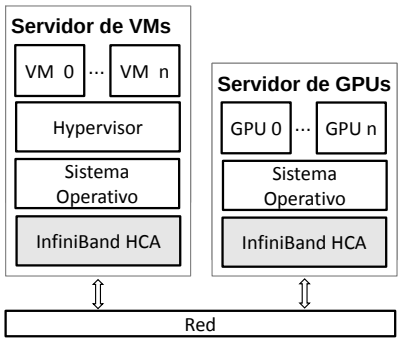


Fig. 7. Arquitectura de V-GPU.

De forma análoga a las soluciones anteriores, V-GPU intenta desacoplar las GPUs de las máquinas virtuales (del mismo modo en el que en su día se desacoplaron las bases de datos de los servidores web). Utiliza una red de altas prestaciones, como puede ser InfiniBand o 10GbE. Al igual que en el resto de soluciones presentadas, en la medida en la que el ancho de banda entre el servidor de GPUs y el servidor de máquinas virtuales sea suficiente, es posible conectar las GPUs que deseemos.

Aunque no hay información explícita al respecto, a partir de la información que encontramos en su web (http://www.zillians.com/products/vgpu-gpu-virtualization/see-it-in-action/), parece que la última versión de V-GPU soporta la versión 4.0 del Driver API y del Runtime API de CUDA. No es necesario modificar las aplicaciones para su uso y, además, también soporta las bibliotecas de CUDA CUBLAS y CUFFT.

III. Evaluación

En esta sección se describen los experimentos que hemos realizado para evaluar las distintas soluciones disponibles para la virtualización de GPUs remotas. De todas las soluciones descritas en la Sección II, únicamente están disponibles de forma libre rCUDA, GVirtuS y DS-CUDA. Así pues, éstos serán los marcos de trabajo que analizaremos.

Tal y como hemos comentado en la Sección I, vamos a comparar las soluciones disponibles de forma libre en base al ancho de banda y la latencia que presentan. Para ello, hemos utilizado la aplicación `bandwidthTest` disponible en los *NVIDIA CUDA Samples* [23]. Se trata de un *benchmark* que mide el ancho de banda de copias a memoria de la GPU. Permite realizar mediciones de copias entre zonas de memoria de la GPU, de memoria de la GPU a memoria de la CPU, y viceversa. La memoria de la CPU a utilizar puede ser paginable o no paginable, ésta última también conocida como *pinned*.

Para los experimentos hemos utilizado dos servidores equipados cada uno con:

- Dos procesadores Intel Xeon E5-2620 de 6 cores a 2.00GHz
- 32 GB de memoria SDRAM DDR3 a 1333 MHz
- 1 GPU NVIDIA Tesla K20
- 1 Mellanox ConnectX-3 single-port InfiniBand Adapter
- CentOS Linux Distribution 6.3, con Mellanox OFED 1.5.3 (drivers InfiniBand y herramientas administrativas), CUDA 3.2 (para GVirtuS), CUDA 4.1 (para DS-CUDA), CUDA 5.0 (para rCUDA), con NVIDIA driver 285.05.

Ambos equipos están interconectados mediante una red InfiniBand FDR a través de un switch Mellanox SX6025.

A. Latencia

En primer lugar hemos medido la latencia que presentan las distintas soluciones. Para ello hemos realizado copias de un tamaño reducido (64 bytes), utilizando una GPU remota, con los siguientes tipos de memoria:

- Copias de memoria CPU paginable a memoria GPU (referenciado como `Pag. H2D` en la Tabla I).
- Copias de memoria GPU a memoria CPU paginable (referenciado como `Pag. D2H` en la Tabla I).
- Copias de memoria CPU no paginable a memoria GPU (referenciado como `Pin. H2D` en la Tabla I).
- Copias de memoria GPU a memoria CPU no paginable (referenciado como `Pin. D2H` en la Tabla I).

Nótese que las copias de memoria GPU a memoria GPU no han sido evaluadas puesto que ninguna de las soluciones estudiadas permite copias directas entre dos GPUs ubicadas en distintos nodos re-

motos. Únicamente permiten copias entre distintas zonas de memoria dentro de una misma GPU o distintas GPUs ubicadas en el mismo nodo remoto, y en ninguno de estos casos los datos circulan por la red.

En la Tabla I podemos ver los resultados obtenidos. Se han realizado 100 repeticiones del test, y en esta tabla mostramos el mínimo tiempo obtenido de todas ellas, asumiendo que éste es el tiempo más próximo a la latencia real.

Como podemos observar, DS-CUDA presenta los mejores resultados cuando las copias involucran memoria CPU paginable. Sin embargo, DS-CUDA tiene el inconveniente de que no permite copias con memoria CPU no paginable, como se explicó en la Sección II-C.

Respecto a GVirtuS, muestra una latencia alta, aunque relativamente uniforme para los distintos tipos de copias, que suele situarse entre los 160 y los 200μs.

Por último, rCUDA presenta buenos resultados cuando se utiliza memoria CPU no paginable. El caso contrario se produce cuando las copias se realizan utilizando memoria paginable, donde llega a obtener peores resultados que el resto de soluciones en el caso de copias de memoria GPU a memoria CPU.

TABLA I
LATENCIA DE LAS DISTINTAS SOLUCIONES ESTUDIADAS COMPARADA CON CUDA.

Tecnología	Latencia (μs)			
	Pag. H2D	Pin. H2D	Pag. D2H	Pin. D2H
CUDA	34,3	4,3	16,2	5,2
rCUDA	94,5	23,1	292,2	6,0
GVirtuS	184,2	200,3	168,4	182,8
DS-CUDA	45,9	--	26,5	--

B. Ancho de banda

A continuación, para medir el ancho de banda, hemos realizado copias de diversos tamaños (de 1 a 30MB), utilizando los distintos tipos de memoria disponibles. Recordar que en el caso de DS-CUDA no es posible realizar copias utilizando memoria CPU no paginale, ni copias de más de 32MB de tamaño.

En la Figura 8 presentamos los resultados para copias de memoria de CPU paginable a memoria de GPU. En este caso, GVirtuS presenta los peores resultados, obteniendo un ancho de banda medio de unos 400MB/s. DS-CUDA, por su parte, obtiene un ancho de banda medio de 1,76GB/s. Mientras que rCUDA incluso supera a CUDA, con un ancho de banda medio de 3,4GB/s, frente a los 2,8GB/s de CUDA. No obstante, para copias de tamaño inferior a 3MB, en el caso de DS-CUDA, y de 4MB, en cuanto a CUDA se refiere, éstos muestran un mejor ancho de banda que rCUDA.

El hecho de que rCUDA supere a CUDA en este tipo de copias se debe a que rCUDA utiliza internamente memoria no paginable [24], la cual alcanza un mayor ancho de banda que la memoria paginable.

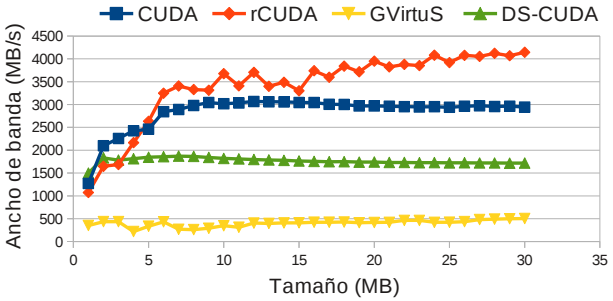


Fig. 8. Ancho de banda en copias de memoria CPU paginable a memoria GPU de las distintas soluciones estudiadas.

Tal y como podemos observar en la Figura 9, para copias de memoria de GPU a memoria de CPU paginable, CUDA nativo, con un ancho de banda medio de 2,7GB/s, supera ampliamente a todas las soluciones de virtualización de GPUs analizadas, las cuales presentan unos anchos de banda medios de 1,3GB/s, 193MB/s y 1GB/s, para rCUDA, GVirtuS y DS-CUDA, respectivamente.

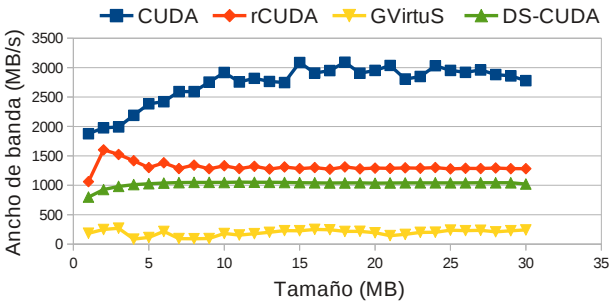


Fig. 9. Ancho de banda en copias de memoria GPU a memoria CPU paginable de las distintas soluciones estudiadas.

Finalmente, en el caso de copias utilizando memoria CPU no paginable, Figuras 10 y 11, los resultados siguen un mismo patrón, según el cual CUDA nativo obtiene los mejores resultados, seguido muy de cerca por rCUDA, y con GVirtuS obteniendo el peor rendimiento con diferencia. Así, para copias de memoria de CPU a memoria de GPU, CUDA obtiene un ancho de banda máximo de casi 6GB/s, frente a los algo más de 5,6GB/s de rCUDA, el cual comienza a obtener resultados comparables a los de CUDA nativo a partir de copias de más de 4MB de tamaño. GVirtuS, por su parte, alcanza un ancho de banda máximo de 485MB/s.

Cuando se trata de copias de memoria de GPU a memoria de CPU (Figura 11), los resultados siguen las mismas pautas, aunque en el caso de CUDA y rCUDA, el ancho de banda se sitúa en niveles más altos, llegando a superar ambos los 6GB/s. Asimismo, rCUDA obtiene unos resultados más próximos a CUDA que en la figura anterior. No sucede lo mismo en el caso de GVirtuS, cuyo ancho de banda disminuye en 100MB/s de media con relación a las copias de memoria CPU a memoria GPU.

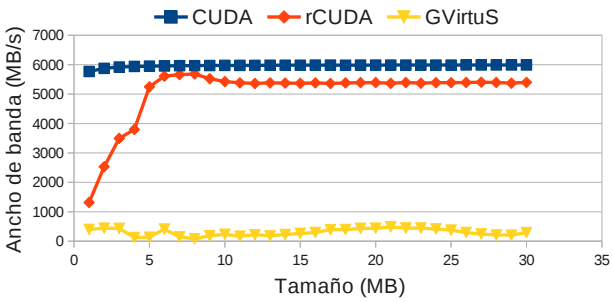


Fig. 10. Ancho de banda en copias de memoria CPU no paginable a memoria GPU de las distintas soluciones estudiadas.

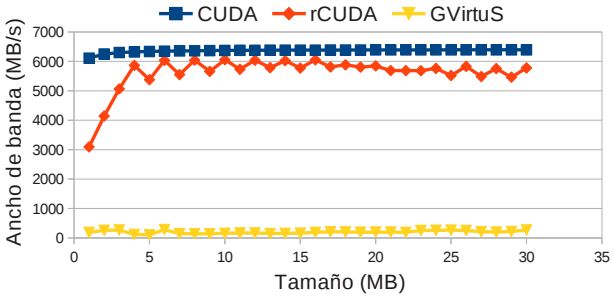


Fig. 11. Ancho de banda en copias de memoria GPU a memoria CPU no paginable de las distintas soluciones estudiadas.

IV. Conclusiones

En el presente artículo hemos realizado una comparativa de las distintas soluciones disponibles para la utilización de GPUs compatibles con CUDA de forma remota. Para ello, además de presentar las características de cada una de ellas, describiendo sus ventajas e inconvenientes, hemos realizado una serie de pruebas con la finalidad de evaluarlas y poder compararlas entre ellas, en base al ancho de banda y la latencia que presentan.

En cuanto a la latencia se refiere, DS-CUDA ha mostrado muy buenos resultados en copias que utilizaban memoria de CPU paginable, mientras que rCUDA ha obtenido la mejor latencia de las distintas soluciones estudiadas cuando las copias utilizaban memoria de CPU no paginable.

En lo referente al ancho de banda, generalmente rCUDA ha presentado los mejores resultados. No obstante, cuando el tamaño de los datos transferidos era pequeño, entre 1MB y 4MB, su rendimiento bajaba considerablemente, sobre todo en el caso de las copias que utilizaban memoria de CPU no paginable.

Agradecimientos

Por un lado, el personal de la Universitat Politècnica de València ha sido subvencionado por el Ministerio de Economía y Competitividad español (MINECO) y por fondos FEDER bajo el acuerdo TIN2012-38341-C04-01.

Por otro lado, los investigadores de la Universitat Jaume I de Castelló han sido financiados por el MINECO, fondos FEDER (TIN2011-23283), la Fundación Caixa-Castelló/Bancaixa (P11B2011-19) y Mellanox Technologies.

Para finalizar, añadir que parte del equipo utilizado en los experimentos fue donado por Mellanox Technologies y NVIDIA Corporation.

Referencias

[1] Sanders, Jason and Kandrot, Edward, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional, 2010.

[2] Renato J. O. Figueiredo, Peter A. Dinda, and José A. B. Fortes, *Guest Editors' Introduction: Resource Virtualization Renaissance*, in IEEE Computer, 38(5), 28 (2005).

[3] Jeremy Enos, Craig P. Steffen, Joshi Fullop, Michael T. Showerman, Guochun Shi, Kenneth Esler, Volodymyr V. Kindratenko, John E. Stone y James C. Phillips, *Quantifying the impact of GPUs on performance and energy efficiency in HPC clusters*, in Green Computing Conference, 2010.

[4] NVIDIA Corporation, *The NVIDIA CUDA API Reference Manual Version 5.0*, NVIDIA Corporation, October 2012.

[5] A. Munshi, Ed., *OpenCL 1.2 Specification*, Khronos OpenCL Working Group, 2011.

[6] J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, and F. Silla, *An efficient implementation of GPU virtualization in high performance clusters*, in Euro-Par 2009 Workshops, ser. LNCS, vol. 6043, 2010, pp. 385-394

[7] J. Duato, A. J. Peña, F. Silla, R. Mayo and E. S. Quintana-Ortí, *Performance of CUDA virtualized remote GPUs in high performance clusters*, in International Conference on Parallel Processing (ICPP), 2011.

[8] J. Duato, A. J. Peña, F. Silla, J. C. Fernández, R. Mayo and E. S. Quintana-Ortí, *Enabling CUDA acceleration within virtual machines using rCUDA*, in International Conference on High Performance Computing (HiPC), 2011.

[9] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, *A GPGPU transparent virtualization component for high performance computing clouds*, in Euro-Par -Parallel Processing, 2010.

[10] Oikawa, Minoru; Kawai, Atsushi; Nomura, Kentaro; Yasuoka, Kenji; Yoshikawa, Kazuyuki; Narumi, Tetsu, *DS-CUDA: a middleware to use many GPUs in the cloud environment*, High Performance Computing, Networking, Storage and Analysis (SCC), Nov. 2012

[11] L. Shi, H. Chen, and J. Sun, *vCUDA: GPU accelerated high performance computing in virtual machines*, in IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 2009.

[12] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, *GViM: GPU-accelerated virtual machines*, in 3rd Workshop on System-level Virtualization for High Performance Computing, 2009.

[13] Liang, Tyng-Yeu and Chang, Yu-Wei *GridCuda: A Grid-Enabled CUDA Programming Toolkit*, in Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA), 2011.

[14] Zillians, Inc., *V-GPU: GPU Virtualization*, http://www.zillians.com/vgpu.

[15] NVIDIA Corporation, *CUDA CUBLAS User Guide Version 5.0*, NVIDIA Corporation, October 2012.

[16] NVIDIA Corporation, *CUDA CUFFT User Guide Version 5.0*, NVIDIA Corporation, October 2012.

[17] NVIDIA Corporation, *Thrust Quick Start Guide Version 5.0*, NVIDIA Corporation, October 2012.

[18] The Linux Foundation, *Xen Project*, http://www.xen.org.

[19] VMware, Inc., *VMWare Virtualization for Desktop and Server, Public and Private Clouds*, http://www.vmware.com/.

[20] Red Hat Emerging Technologies, *KVM*, http://www.linux-kvm.org/.

[21] Fabrice Bellard, *QEMU: open source processor emulator*, http://www.qemu.org/.

[22] UserLand Software, Inc., *XML-RPC*, http://www.xmlrpc.com/.

[23] NVIDIA Corporation, *NVIDIA CUDA Samples Version 5.0*, NVIDIA Corporation, October 2012.

[24] Antonio J. Peña, *Virtualization of Accelerators in High Performance Clusters*, PhD thesis, University Jaume I, Castellón, Spain, 2013.

Mejoras en eficiencia y precisión de una aproximación Multi-GPU para el método SPH

J.M. Domínguez, A.J.C. Crespo, A. Barreiro y M. Gómez-Gesteira[1]

Resumen—**En este trabajo se presenta una avanzada implementación Multi-GPU del método SPH, que es un método numérico para la resolución de problemas de dinámica computacional de fluidos, especialmente útil con problemas de superficie libre e interacción entre fluido y estructuras. Esta implementación permite combinar la potencia de múltiples GPUs mediante el uso de MPI. Se resuelven de forma eficaz problemas típicos como el balanceo dinámico de carga y el solapamiento entre cálculo y comunicaciones. También se abordan con éxito otros problemas surgidos de la posibilidad actual de simular grandes extensiones de fluido con una alta resolución, así como la falta de precisión, minimizando su impacto sobre el elevado rendimiento del código descrito aquí. Se analiza la eficiencia de la nueva versión Multi-GPU de DualSPHysics para diferentes tamaños de simulación y hasta 128 GPUs. Finalmente, una simulación con más de 1,000 millones de partículas es utilizada para mostrar las enormes posibilidades de la nueva implementación.**

Palabras clave—**HPC, GPU, Multi-GPU, SPH, Smoothed Particle Hydrodynamics, Dinámica de fluidos.**

I. INTRODUCCIÓN

La dinámica de fluidos computacional (CFD) supone una herramienta fundamental para la investigación científica, además de ser muy útil para resolver problemas complejos. En el campo de la ingeniería nos permite simular el comportamiento del fluido en escenarios reales de gran complejidad, en lugar de construir modelos físicos que resultan ser mucho más costosos. Otra ventaja de esta herramienta es la posibilidad de obtener datos físicos difíciles o imposibles de obtener en modelos reales. Sin embargo, estos métodos numéricos pueden suponer un elevado coste computacional como es el caso del método SPH (Smoothed Particle Hydrodnamics), utilizado en este trabajo. SPH es un método lagrangiano desarrollado a finales de los años 70 para problemas de astrofísica [1], aunque más adelante se empezó a aplicar en otros campos. En el ámbito de la dinámica de fluidos destaca en el estudio de la hidrodinámica de problemas de superficie libre, flujos violentos e interacción entre olas y estructuras costeras. En el método SPH, el fluido se representa mediante un conjunto de puntos o nodos distribuidos en el espacio donde se conocen las propiedades físicas (densidad, masa, velocidad...). Estos nodos o partículas se mueven por el espacio describiendo el fluido de forma lagrangiana y sus propiedades van cambiando con el tiempo debido a la interacción con las partículas de alrededor.

Uno de los principales inconvenientes de este método radica en su elevado coste computacional cuando se desea estudiar problemas de ingeniería reales, debido al gran número de partículas necesario para representar geometrías complejas con el detalle requerido. Por tanto, es fundamental desarrollar implementaciones paralelas de SPH capaces de combinar los recursos de múltiples máquinas para simular millones de partículas en tiempos razonables. En los últimos años, las GPUs (Graphics Processing Units) han sufrido un rápido desarrollo convirtiendose en procesadores de cálculo paralelo de gran potencia, hasta el punto de convertirse en componentes claves en soluciones de High Performance Computing (HPC). Tanto es así que el primer puesto de la lista de los supercomputadores más potentes del mundo de noviembre de 2012 está ocupado por un supercomputador basado en tecnología GPU llamado Titan (http://www.top500.org/lists/2012/11). Así, con la aparición del lenguaje de programación CUDA [2] que facilita la programación de estos dispositivos, esta tecnología se ha vuelto en una opción muy atractiva para el desarrollo de aplicaciones con una gran carga de cálculo que exhiban un alto grado de paralelismo. De hecho, muchas implementaciones en GPU han mostrado speedups de uno o dos órdenes de magnitud con respecto a implementaciones en CPU. Sin embargo, el uso de una GPU no es suficiente para aplicaciones de ingeniería reales que involucran varios millones de partículas debido a las limitaciones de memoria y a los elevados tiempos de ejecución. Por tanto, es imprescindible desarrollar soluciones que puedan combinar recursos de múltiples GPUs eficientemente.

En este trabajo se muestra una implementación del método SPH usando Compute Unified Device Architecture (CUDA) y Message Passing Interface (MPI). La solución Multi-GPU para SPH presentada aquí ha sido implementada en el código DualSPHysics, que se trata de un código *open-source* disponible en www.dual.sphysics.org, y cuya formulación está basada en el código SPH llamado SPHysics [3, 4]. La versión Single-GPU ha logrado speedups de dos órdenes de magnitud con respecto a CPU [5]. Así, la versión Multi-GPU de DualSPHysics parte de una versión Single-GPU optimizada para CPU y GPU [6]. Pueden encontrarse otros artículos previos de los mismos autores que describen el uso de Multi-GPU con SPH [7] y [8] donde múltiples GPUs de distintas máquinas son utilizadas mediante MPI, aunque también hay trabajos publicados

[1] EPHYSLAB Environmental Physics Laborarory, Universidad de Vigo, Campus As Lagoas s/n 32004 Ourense, e-mail: jmdominguez@uvigo.es

donde se usan otras opciones como hilos para combinar la potencia de hasta 6 GPUs alojadas en la misma máquina [9].

En este trabajo se describen las implementaciones Single-GPU y Multi-GPU del método SPH implementadas en el código DualSPHysics. Se muestran los resultados de eficiencia con hasta 128 GPUs, explicando el origen de la pérdida de eficiencia y cómo mejorarla. Se trata el problema de la precisión en casos de grandes dimensiones y finalmente se demuestra la capacidad del código expuesto aquí presentando una aplicación realista con más de mil millones de partículas.

II. Implementación Single-GPU

La implementación CUDA del método SPH puede dividirse en tres pasos fundamentales: (i) creación de la lista de vecinos, (ii) cómputo de fuerzas mediante la interacción entre las partículas y (iii) actualización del estado de las partículas para el siguiente instante de la simulación. Estos tres pasos se repiten hasta completar la simulación y todos ellos están implementados en GPU. En el código DualSPHysics todos los datos de las partículas se mantienen en la memoria de la GPU reduciendo al mínimo las transferencias de memoria entre CPU y GPU. De hecho, estas trasferencias sólo se realizan al iniciar la simulación y cuando se desea grabar resultados, lo que ocurre cada muchos pasos de cálculo y por tanto su consumo de tiempo no es significativo. La lista de vecinos implementada en GPU es similar a la utilizada en versiones CPU y conocida como Cell-linked List [10], donde la ordenación de partículas se realiza mediante una implementación CUDA del algoritmo Radix sort [11]. La actualización del estado de las partículas se realiza fácilmente de forma paralela usando diferentes hilos de ejecución de la GPU. La interacción entre partículas es el paso más costoso ya que consume más del 95% del tiempo total de ejecución. En este paso cada hilo de ejecución GPU se encarga de calcular todas las interacciones de una partícula con todas las partículas que estén dentro de la distancia de interacción (que se denomina aquí *2h*), y acumulando todas sus aportaciones. Una descripción más completa puede encontrarse en [5], así como las optimizaciones aplicadas en [6].

III. Implementación Multi-GPU con MPI

En la implementación Multi-GPU se hace uso de MPI para comunicar diferentes máquinas CPU que pueden contener varias GPUs. De esta forma, podemos combinar la potencia de varias GPUs alojadas en una o varias máquinas conectadas en red. Sin embargo, el uso de MPI implica algunos costes computacionales extra debido a: (i) ejecución de procesos para mantener equilibrada la distribución de carga de trabajo entre los distintos procesos, (ii) intercambio de datos entre procesos vecinos y (iii) puntos de sincronización.

A. *Subdivisión del dominio*

El dominio físico de la simulación es dividido en subdominios o bloques de partículas asignados a diferentes procesos MPI. De esta forma, cada subdominio tiene dos subdominios vecinos (uno a cada lado), excepto los extremos que sólo tienen uno. Como las partículas interactúan con todas las partículas dentro de un radio de *2h*, es imprescindible que en cada paso de cálculo, todos los procesos MPI obtengan los datos de las partículas situadas a una distancia *2h* de los límites del subdominio. A esto le llamamos halo del proceso (o subdominio), o también borde del subdominio vecino. Para determinar los halos de cada proceso se utiliza la información obtenida durante la creación de la lista de vecinos. En esta etapa todas las partículas son ordenadas en celdas de tamaño *2h* y por tanto el halo está formado por una rebanada celdas del proceso vecino, donde todas las partículas están almacenadas de forma consecutiva. Esta forma de dividir en subdominios y reordenar las partículas en la dirección de división del dominio proporciona importantes ventajas: (i) nunca se mezclan las partículas propias del subdominio y de los halos por lo que no se pierde tiempo en reordenar las partículas de los halos para mezclarlas con el resto, ni después en separarlas; (ii) cada proceso puede ajustar los límites de su subdominio en función de las partículas fluidas reduciendo el número de celdas empleado y su consumo de memoria; (iii) como los datos de las partículas son almacenados en posiciones de memoria consecutivas su envío a los procesos vecinos es mucho más rápido; (iv) es posible solapar el cálculo de fuerzas entre las partículas propias de cada subdominio con la recepción y envío de partículas a los procesos vecinos.

B. *Comunicación entre procesos*

El incremento en el número de procesos supone un aumento en el tiempo dedicado a la comunicación entre procesos. Una opción para eliminar esta pérdida de eficiencia consiste en solapar comunicaciones con cálculo. Con esta finalidad se utilizan envíos asíncronos y recepciones síncronas. Así un proceso puede enviar datos a otro mientras continúa con la ejecución de otras tareas. Al inicio de cada paso de cálculo, todos los procesos seleccionan las partículas que cambian a otros subdominio y se las envían. Mientras, continúan con otras tareas de la lista de vecinos hasta recibir las nuevas partículas y finalizar la ordenación de todas las partículas. Para calcular las fuerzas, cada proceso envía sus bordes a los procesos vecinos y necesita recibir las partículas de sus halos. Pero mientras no recibe los halos puede calcular la interacción entre sus propias partículas, solapando de esta manera la transferencia de datos más costosa con el proceso de cálculo más pesado. Una vez recibido el primer halo se procede a calcular la interacción de sus partículas del borde de su subdominio con el halo recibido mientras se sigue esperando por el segundo para después realizar el mismo proceso. Finalmente, es necesario un punto de sincronización para calcular cual es la duración (tiempo físico) del siguiente paso de cálculo.

C. *Balanceo dinámico de carga*

Se han implementado dos aproximaciones distintas para balancear la carga de cálculo entre los procesos. En la primera se trata de repartir las partículas de fluido de forma uniforme entre todos los procesos involucrados. Esta aproximación es adecuada cuando todos los procesos tienen una potencia de cálculo similar ya que el tiempo de cálculo depende principalmente del número

de partículas de fluido. La segunda aproximación se basa en el tiempo de cálculo requerido por cada proceso en los últimos pasos de tiempo, así cada proceso puede asignar un peso diferente a cada rebanada de su subdominio en función de las partículas que contenga y después repartir estas rebanadas de forma que a todas las GPUs se les asigne una carga de trabajo similar. Esta aproximación permite adaptar el código a clústeres heterogéneos de GPUs sacando el máximo rendimiento, ya que a las GPUs se les asigna más o menos partículas en función de su potencia. Es importante comentar que aunque el balanceo se comprueba con relativa frecuencia, sólo es necesario aplicarlo cada muchos pasos de cálculo y por tanto su importancia en el tiempo total de ejecución es mínima. Los resultados del balanceo dinámico de carga en función del número de partículas o del tiempo de ejecución pueden consultarse en [8].

D. Resultados de eficiencia

En este apartado los resultados de eficiencia son analizados para demostrar la capacidad de la implementación Multi-GPU para simular un elevado número de partículas. El rendimiento ha sido medido como el número de pasos por segundo y haciendo uso de dos métricas distintas: (i) strong scaling S(N) que determina como varia el rendimiento para distinto número de procesos (N) manteniendo constante el tamaño del problema y (ii) weak scaling s(N) mide como varia el rendimiento en función también del número de procesos, pero manteniendo constante el tamaño del problema por proceso.

El caso de estudio empleado para obtener los resultados se muestra en la figura 1. Consiste en un *dam-break* donde el ancho del tanque se ajusta para obtener el número de partículas deseado. Nótese que modificando el ancho podemos cambiar el número de partículas sin afectar al número de pasos necesario para completar la simulación y manteniendo así el número medio de partículas vecinas que interactúan con cada partícula. Este caso se ha utilizado para medir el rendimiento con 1 a 1,024 millones de partículas simulando 0.6 segundos físicos.

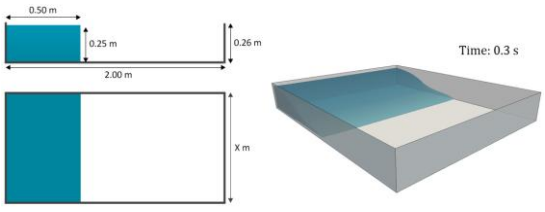


Fig. 1. Case de studio: *Dam-break*.

Las simulaciones fueron llevadas a cabo en el Barcelona Supercomputing Center (BSC-CNS). Este sistema está formado por 256 GPUs Tesla M2090 repartidas en 128 nodos. Todos los resultados presentados aquí fueron obtenidos usando CUDA 4.0, simple precisión y con Error-correcting Code Memory (ECC) desactivado. Los resultados de eficiencia (*strong scaling* y *weak scaling*) se muestran en la tabla 1 y 2.

TABLA I

EFICIENCIA USANDO *STRONG SCALING*.

GPUs	*Número total de partículas*		
	6M	**12M**	**24M**
2	97.5 %	99.5 %	98.5 %
4	93.9 %	97.8 %	97.2 %
8	87.3 %	94.1 %	94.9 %
16	76.8 %	87.0%	92.4 %
32	62.2 %	76.2 %	85.1 %

TABLA II

EFICIENCIA USANDO *WEAK SCALING*.

GPUs	*Número de partículas por GPU*		
	1M/GPU	**4M/GPU**	**8M/GPU**
2	92.4 %	96.8 %	>99.9 %
4	88.8 %	95.8 %	>99.9 %
8	85.3 %	96.4 %	>99.9 %
16	86.0 %	98.1 %	>99.9 %
32	86.2 %	98.0 %	>99.9 %
64	86.3 %	98.1 %	>99.9 %
128	85.1 %	97.4 %	>99.9 %

Usando un máximo de 128 GPUs Tesla M2090, se ha logrado una eficiencia del 97.4% simulando 4M por GPU y cerca del 100% con 8M por GPU. Téngase en cuenta que la mayor simulación realizada fue de 1024M con 128 GPUs.

E. Pérdida de eficiencia

Gracias a la posibilidad de combinar muchas GPUs de forma eficiente, podemos simular grandes dominios con una alta resolución. Sin embargo para usar cientos de GPUs son necesarias nuevas mejoras con respecto a la implementación descrita en [8].

Las dos fuentes principales de pérdida de eficiencia cuando el número de procesos MPI se incrementa son la sincronización y el intercambio de datos entre procesos.

Un punto de sincronización es imprescindible después del cálculo de fuerzas, ya que el tiempo de cada paso de cálculo es variable y es necesario calcularlo de forma conjunta para todos los procesos, de forma que todos ellos deben esperar por el más lento produciendo una pérdida de eficiencia. Éste es un problema de difícil solución ya que el balanceo de carga perfecto entre todas las GPUs es imposible, o al menos de forma eficiente, debido a la propia naturaleza lagrangiana del método SPH.

El tiempo consumido en las comunicaciones entre distintos dispositivos es significativo debido a que el solapamiento con el cómputo no es perfecto. Este intercambio de datos implica cuatro pasos: (i) trasferencia de GPU a CPU, (ii) envío de datos con MPI, (iii) recepción de datos y (iv) transferencia de CPU a GPU. Por tanto, las nuevas mejoras han sido aplicadas para reducir estos tiempos de comunicación. El tiempo de las transferencias CPU↔GPU se redujo a la mitad usando pinned memory. Además el uso de transferencias asíncronas permite solapar estos tiempos con cálculo. Por otro lado, se ha eliminado el uso de un buffer intermedio empleado en los envíos y recepciones con MPI reduciendo así el tiempo de comunicación y la memoria CPU empleada. Finalmente, gracias al uso de *streams* CUDA y a las transferencias asíncronas de memoria se ha mejorado el solapamiento entre cómputo y comunicaciones. Ahora el cálculo de fuerzas de cada

subdominio se solapa con todo el proceso de envío y recepción de los dos halos. Antes de esta mejora el solapamiento con el cálculo sólo se producía con la espera del primer halo y el cálculo de fuerzas con este halo es el que se solapaba con la recepción del segundo halo no siendo suficiente para cubrir su espera. La figura 2 muestra el porcentaje del tiempo total dedicado a las tareas propias de Multi-GPU, tales como sincronización, intercambio de datos y operaciones de balanceo de carga. Puede observarse cómo estas últimas mejoras consiguen reducir estos porcentajes a la mitad para diferente número de GPUs y distinto número de partículas.

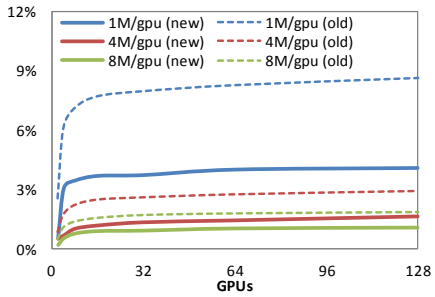


Fig. 2. Porcentaje de tiempo consumido por las tareas propias de Multi-GPU tras las últimas mejoras.

IV. PRECISIÓN Y RENDIMIENTO

Todos los resultados obtenidos en este trabajo fueron obtenidos usando simple precisión, ya que es suficiente para los casos estudiados aquí. En general, con el método SPH para fluidos no es necesario el uso de cálculos en doble precisión. La precisión de los resultados se mejora aumentando la resolución y por tanto el número de partículas. Sin embargo, gracias al uso de múltiples GPUs, el tamaño de las simulaciones ha aumentado de forma drástica pudiendo aparecer problemas derivados de la falta de precisión en los cálculos. Los problemas de precisión surgen con la simulación de dominios muy extensos con una resolución también muy elevada. De momento, alcanzar esta situación en casos 3D es complicado, porque antes se alcanza el límite de partículas que se pueden simular, pero en casos 2D puede ser más factible.

El origen de este problema está en el uso de simple precisión para las variables de posición de las partículas. El tipo de datos empleado tiene un tamaño de 32 bits de los cuales 1 bit es para el signo, 8 para el exponente y los 23 restantes para la mantisa, esto permite representar valores que van de 1.175494351e-38 a 3.402823466e38. Así, la mantisa tiene una precisión de 23 bits (24 de forma implícita) que en decimal viene a ser de 7 dígitos. Esto significa que cuando dos partículas están próximas al cero (0.xxxxxxx) y se calcula su distancia para la interacción se tiene una precisión de 7 cifras decimales, mientras que la distancia entre el mismo par de partículas situadas en la posición 1000.xxx sólo tendría una precisión de 3 cifras decimales. El mismo problema se produce al incrementar la posición en una cantidad muy pequeña, este incremento se hace con precisión en partículas situadas cerca del cero pero no en las alejadas.

En la figura 3 se muestra el efecto de la falta de precisión en la posición de las partículas. Este caso consiste en la simulación 2D de un tanque de agua con un pistón en el extremo izquierdo para generar olas. El caso tiene una longitud de 18m y la distancia de interacción entre partículas (*2h*) es de 0.012374m. Se simulan 25 segundos físicos aunque el pistón no comienza a moverse hasta el segundo 4.5. En la imagen superior correspondiente al segundo 2 de la simulación puede verse la diferencia de velocidad de las partículas según su posición. Esta diferencia se ve muy marcada a los 8 y 16 metros debido a que la representación interna de los valores es binaria y por tanto los saltos de precisión se producen en potencias de 2. En el segundo 5, a partir de los 16m ya se observan unos puntos negros que representan las partículas de fluido excluidas de la simulación por alcanzar valores de densidad asumidos como no válidos. En el segundo 10 las partículas excluidas aparecen desde los 8m y finalmente en el segundo 25, el 71.9% de partículas de fluido ha sido expulsado de la simulación.

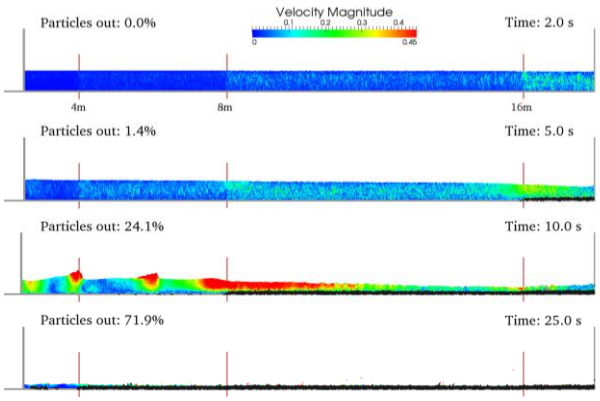


Fig. 3. Distintos instantes de una simulación 2D de un tanque de olas.

La solución consiste en aumentar la precisión con la que se almacena la posición de las partículas. La opción más evidente es usar variables de doble precisión (64 bits, 52 para la mantisa) que proporcionan una precisión de 15 cifras decimales. Sin embargo, su implementación en GPU implica una pérdida de rendimiento importante en el cálculo de fuerzas, aunque éste se siga realizando en simple precisión. Esta pérdida de rendimiento se debe al aumento de registros necesarios en el kernel de interacción, lo que provoca una reducción en la ocupación de la GPU y por tanto del rendimiento. Por otro lado, la capacidad de cómputo en simple y doble precisión de las GPUs no está compensada. Dependiendo del modelo de GPU, la capacidad de cálculo en doble precisión puede ser varias veces menor. Otro inconveniente de su uso es la pérdida de eficiencia en Multi-GPU ya que el solapamiento entre cálculo y comunicación nunca es perfecto, y los datos de partículas que se intercambian entre procesos en cada paso de cálculo aumenta en un 42.8%.

Una solución alternativa es seguir empleando simple precisión para la posición, pero en lugar de guardar la posición real de la partícula, se almacena la posición relativa a la celda en que se encuentra dicha partícula. Como ya se explicó en los detalles de la implementación, las partículas están repartidas en celdas de tamaño *2h* para reducir la búsqueda de vecinos en el cómputo de fuerzas. Así podemos usar esa división en

celdas para definir la posición en función de su celda. Esto supone que la posición en cada eje nunca será mayor de *2h*, con independencia de las dimensiones del caso a simular. Las ventajas de este método son que el tamaño de la posición pasa de 3x4 bytes a 4x4 bytes en lugar de los 3x8 bytes de la opción con doble precisión y por otro lado ya no es necesario el uso de doble precisión en el cómputo de fuerzas. Los cálculos en doble precisión sólo son necesarios al actualizar la posición y esta parte del código apenas consume tiempo de ejecución. El mayor inconveniente de esta implementación es el incremento en la complejidad del código, que no deja de ser un factor importante al tratarse de un código usado por múltiples investigadores de distintas universidades que está en continua evolución. Así, en la figura 4 puede verse como usando esta última implementación se solucionan los problemas de precisión que aparecían en la simulación del tanque de olas. Después de 25 segundos de simulación apenas se pierde el 0.3% del fluido, lo cual es debido únicamente a las inestabilidades que provoca el pistón.

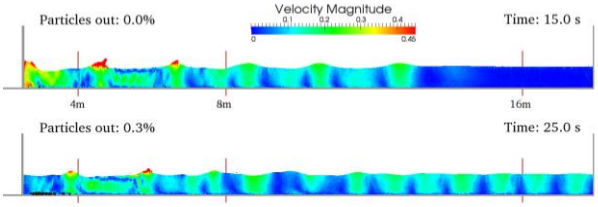


Fig. 4. Distintos instantes de la simulación del tanque de olas mejorando la precisión en la posición de las partículas.

La figura 5 muestra el error medio que se produce en la posición de las partículas en la simulación del tanque de olas según la distancia a la que se desplace a la derecha el tanque. En esta grafica se puede ver el error de las tres implementaciones descritas: posición en precisión simple (PosSimple), posición en precisión doble (PosDouble) y posición en precisión simple pero en función de la celda (PosCell). El error está medido como la diferencia de posición de las partículas de cada implementación con respecto a una implementación completa del código en doble precisión y este valor es expresado como porcentaje de *2h*. Así puede verse como el error de las opciones PosDouble y PosCell se mantiene constante aunque la simulación se desplace más de 8,000m a la derecha, mientras que el error de PosSimple es varios ordenes de magnitud mayor y se incrementa con la distancia al cero.

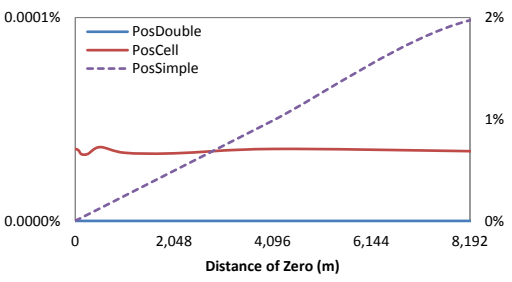


Fig. 5. Error en posición de las partículas para diferentes distintas implementaciones al alejar el caso de estudio del cero. El eje izquierdo mide el error de PosDouble y PosCell, mientras que el de PosSimple se mide en el eje derecho.

Ambas implementaciones (PosDouble y PosCell) solucionan el problema de precisión, pero también suponen un coste importante en la velocidad de ejecución. En la figura 6 se muestra la pérdida de rendimiento con respecto a la implementación en simple precisión usando distintos modelos de GPU. Para ello se empleó el mismo caso 3D usado en el estudio de eficiencia de Multi-GPU y representado en la figura 1. La pérdida de rendimiento varía según el tipo de GPU utilizada. El uso de doble precisión en la GPU más moderna (Tesla K20) supone una pérdida del 20% mientras que en la GTX 480 alcanza cerca del 30%. Sin embargo, con la implementación PosCell es menor del 8% en la Tesla K20 y menor del 15% en la GTX 480.

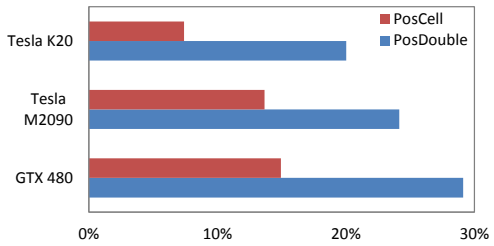


Fig. 6. Pérdida de rendimiento con respecto a la implementación de posición en simple precisión, usando el caso de estudio del *dam-break* 3D de 4M de partículas.

V. APLICACIÓN

Uno de los principales objetivos de la implementación multi-GPU propuesta aquí es poder simular una gran cantidad de partículas en tiempos de ejecución razonables, lo cual es fundamental para modelar aplicaciones del mundo real que requieren una elevada resolución. Por ello, el código DualSPHysics se aplicaen una simulación con más de 10^9 partículas. Esta simulación consiste en la interacción de una ola de gran tamaño con una plataforma petrolífera con dimensiones reales que pueden verse en la figura 7, y las dimensiones del fluido son 170m x 114m x 68m. La distancia inicial entre partículas es de 6 cm y el número total de partículas es de 1,015,896,172 (1,004,375,142 de fluido). Esta aplicación realista fue elegida debido al elevado número de partículas necesario para representan con suficiente resolución los detalles de la plataforma.

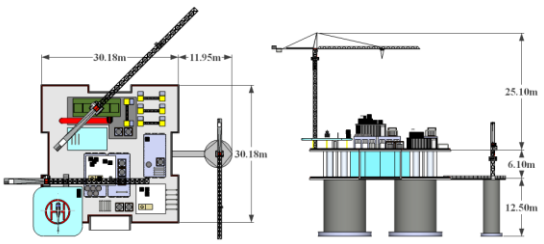


Fig. 7. Dimensiones reales de la plataforma petrolífera empleada en la simulación.

La simulación fue llevada a cabo utilizando 64 GPUs Tesla M2090 del Barcelona Supercomputing Center (BSC-CNS). Pueden verse distintos instantes de la simulación en la figura 8. Para simular 12 segundos de tiempo físico fueron necesarios 237,065 pasos de cálculo, cuya ejecución supuso 79.1 horas. Los datos

fueron grabados cada 0.04 segundos, lo que representa más de 8980 GB de resultados.

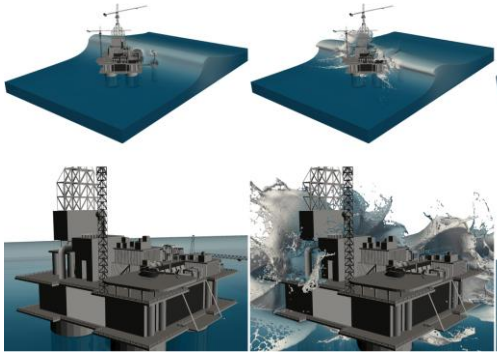


Fig. 8. Instantes 2.2s y 3.2s de la simulación (con más de 10^9 partículas) de una ola de gran tamaño impactando con una plataforma.

Tras aplicar las mejoras de eficiencia presentadas en el apartado E de la sección relativa a la implementación Multi-GPU, se ha logrado una importante reducción en el tiempo de ejecución. El tiempo de ejecución de esta misma simulación antes de aplicar dichas mejoras fue de 91.9 horas (un 16% más lento) y con la versión optimizada del código menos de 80 horas. La figura 9 muestra el tiempo de ejecución consumido por las tareas propias de Multi-GPU antes y después de las mejoras. Puede verse como el tiempo dedicado al intercambio del halo fue reducido de forma drástica aunque el tiempo de sincronización se mantiene constante.

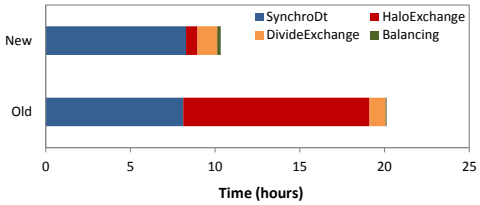


Fig. 9. Tiempo consumido por las tareas propias de Multi-GPU durante la simulación de la ola impactando con la plataforma.

VI. CONCLUSIONES Y TRABAJO FUTURO

Se ha presentado una potente implementación Multi-GPU del método SPH usando MPI, capaz de combinar eficientemente los recursos de múltiples GPUs alojadas en una o distintas maquinas.

El código presentado aquí muestra una elevada eficiencia usando un número importante de GPUs. Con 128 GPUs Tesla M2090 del Barcelona Supercomputing Center se ha logrado una eficiencia del 92.1% simulando 1M/GPU y cerca del 100% simulando 4M/GPU y 8M/GPU.

Aunque ciertas tareas propias de Multi-GPU (e inevitables) suponen una pérdida de rendimiento, principalmente la sincronización y el intercambio de datos en el cómputo de fuerzas, se ha logrado una significativa mejora que reduce estos tiempos a la mitad.

Se ha mostrado como el origen de los problemas de precisión que pueden surgir en los casos de grandes dimensiones con una elevada resolución vienen de la falta de precisión al almacenar la posición de las partículas. También se ha aportado una solución a este problema que minimiza su repercusión en el rendimiento del código.

Finalmente, la capacidad de esta implementación para simular aplicaciones reales con una elevada resolución ha sido demostrada con la simulación de una gran ola impactando contra una plataforma petrolífera de dimensiones realistas. Esta simulación de 12 segundos de tiempo físico y con más de 10^9 partículas fue llevada a cabo en menos de 80 horas con 64 GPUs Tesla M2090.

AGRADECIMIENTOS

El presente trabajo ha sido financiado por la Xunta de Galicia, Programa de Consolidación e Estructuración de Unidades de Investigación Competitivas (Grupos de Referencia Competitiva), cofinanciado por European Regional Development Fund (FEDER) y por el Ministerio de Economía y Competitividad bajo el proyecto BIA2012-38676-C03-03. Los autores también desean agradecer el soporte proporcionado por EPSRC EP/H003045/1 y Research Councils UK (RCUK) fellowship. Y finalmente al Barcelona Supercomputing Center (BSC-CNS) por el uso de sus instalaciones bajo las actividades FI-2012-2-0006, FI-2012-3-0004 y FI-2013-1-0018.

REFERENCIAS

[1] R.A. Gingold and J.J. Monagham, *"Smoothed particle hydrodynamics: theory and application to non- spherical stars"*, Mon Not R Astr Soc 181: 375-389, 1977.

[2] Nvidia, CUDA Programming Guide, 4.0. (2011), http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf

[3] M. Gómez-Gesteira, B.D. Rogers, A.J.C. Crespo, R.A. Dalrymple, M. Narayanaswamy and J.M. Dominguez, *"SPHysics - development of a free-surface fluid solver - Part 1: Theory and Formulations"*, Computers & Geosciences, 48: 289-299, 2012.

[4] M. Gómez-Gesteira, A.J.C. Crespo, B.D. Rogers, R.A. Dalrymple, J.M. Dominguez and A. Barreiro, *"SPHysics - development of a free-surface fluid solver - Part 2: Efficiency and test cases"*, Computers & Geosciences, 48: 300-307, 2012.

[5] A. J. C. Crespo, J. M. Dominguez, A. Barreiro, M. Gómez-Gesteira and B. D. Rogers, *"GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods"*, PLoS ONE, doi: 10.1371/journal.pone.0020685, 2011.

[6] J.M. Dominguez, A.J.C. Crespo and M. Gómez-Gesteira. *"Optimization strategies for CPU and GPU implementations of a Smoothed Particle Hydrodynamics method"*, Computer Physics Communications, 184(3): 617-627, 2012.

[7] D. Valdez-Balderas, J.M. Dominguez, A.J.C. Crespo and B.D. Rogers, *"Towards accelerating Smoothed Particle Hydrodynamics simulations for free-surface flows on multi-GPU clusters"*, Journal of Parallel and Distributed Computing, doi: 10.1016/j.jpdc.2012.07.010, 2012.

[8] J.M. Dominguez, A.J.C. Crespo, D. Valdez-Balderas, B.D. Rogers and M. Gómez-Gesteira, *"New multi-GPU implementation for Smoothed Particle Hydrodynamics on heterogeneous clusters"*, Computer Physics Communications, 184: 1848-1860, 2013.

[9] E. Rustico, G. Bilotta, A. Hérault, C. Del Negro and G. Gallo, *"Advances in multi-GPU Smoothed Particle Hydrodynamics simulations"*, IEEE Transactions on Parallel and Distributed Systems, vol 99, 2013.

[10] J.M. Dominguez, A.J.C. Crespo, M. Gómez-Gesteira and J.C. Marongiu, *"Neighbour lists in Smoothed Particle Hydrodynamics"*, International Journal for Numerical Methods in Fluids, 67: 2026-2042, 2010.

[11] N. Satish, M. Harris and M. Garland, *"Designing efficient sorting algorithms for manycore GPUs"*, Proceedings of IEEE International Parallel and Distributed Processing Symposium 2009, 2009.

MPCM: Codificador hardware para cámaras de alta velocidad

Estefanía Alcocer, Otoniel López-Granado, Manuel P. Malumbres[1] y Roberto Gutiérrez[2]

Resumen— **En la última década, las mejoras a nivel de integración VLSI y en las tecnologías de captura de imágenes han encabezado una carrera frenética para proporcionar sistemas de procesamiento de vídeo capaces de capturar y comprimir secuencias de vídeo con resoluciones y tasas de frames muy elevadas. Estas cámaras de vídeo de alta velocidad se utilizan en aplicaciones científicas e industriales, como en tests de accidentes automovilísticos, en investigación en combustión, ensayos de materiales, dinámica de fluidos, visualización de flujos, etc, que demandan la captura y almacenamiento de vídeo en tiempo real a muy altas velocidades (más de 1000 fps) y con formatos de alta definición. Por lo tanto, la capacidad de almacenamiento de datos, el ancho de banda de la comunicación, el tiempo de procesamiento y el consumo de energía son parámetros críticos que deben ser considerados cautelosamente en su diseño. En este trabajo, se propone una implementación en FPGA de un codificador rápido y sencillo llamado MPCM, el cual obtiene calidades similares a la codificación PCM tradicional, pero que es capaz de reducir los requisitos de ancho de banda hasta 1.4 veces, lo que permite a cámaras de muy alta velocidad capturar de manera continua en un dispositivo de almacenamiento masivo como una SSD.**

Palabras clave— **PCM, codificación de imagen, diseño FPGA, alta velocidad, circuitos integrados.**

I. Introducción

La compresión de vídeo ha sido una tecnología de gran éxito que ha encontrado su aplicación comercial en muchas áreas, en aplicaciones científicas e industriales como el almacenamiento de vídeo, imágenes médicas de alta calidad y aplicaciones de vigilancia y seguridad, en la industria audiovisual (cine y televisión) y en la amplia gama de aparatos de vídeo disponibles en el mercado como cámaras digitales, DVD, Blue-Ray, DVB, etc.

En la última década, las mejoras a nivel de integración VLSI y en las tecnologías de captura de imágenes han encabezado una carrera frenética para proporcionar sistemas de procesamiento de vídeo capaces de capturar y comprimir secuencias de vídeo con resoluciones y tasas de frames muy elevadas. Hoy en día, se pueden encontrar en el mercado cámaras de vídeo de ultra alta velocidad como Phantom v641 [1], que es capaz de capturar vídeo de alta resolución (2560 x 1600 píxeles) a 1.450 frames por segundo (fps). Estas cámaras de vídeo son especialmente adecuadas para aplicaciones científicas o industriales, como tests de accidentes automovilísticos, explosivos y pirotecnia, balística, seguimiento de proyectiles, investigación en combustión, ensayos de materiales, dinámica de fluidos, visualización de flujos, etc, que demandan la captura de vídeo en tiempo real a muy altas velocidades y con formatos de gran definición. Por lo tanto, la capacidad de almacenamiento de datos, el ancho de banda de la comunicación, el tiempo de procesamiento y el consumo de energía son parámetros críticos que deben ser considerados cautelosamente en el diseño de cámaras de vídeo de alta velocidad.

Actualmente, la mayoría de cámaras de alta velocidad almacenan las imágenes capturadas en un módulo SDRAM de hasta 64 GB [1], [2], sin realizar compresión, usando Pulse Code Modulation (PCM) [3], generando una enorme cantidad de datos sin comprimir que necesitan ser procesados para su transmisión o almacenamiento posterior. Por tanto, el bus de comunicación interna no es lo suficientemente rápido para transferir el vídeo desde la cámara, o la velocidad de escritura del dispositivo de almacenamiento no es lo suficientemente alta como para guardar los datos en tiempo real [4]. Consecuentemente, el enfoque de la utilización de la memoria SDRAM como almacenamiento de vídeo es factible ya que el ancho de banda de memoria es lo suficientemente alto, pero cuando la memoria se agota, la cámara deja de grabar y tiene que guardar el vídeo en un dispositivo de almacenamiento secundario en crudo o en formato comprimido. Esta es una limitación, ya que dependiendo de la resolución de captura de la cámara, sólo se registrarán unos pocos segundos en la memoria, y por lo tanto, no se realizará una captura continua.

Con el fin de superar estas restriciones, sería de interés reducir las necesidades de almacenamiento de vídeo a través de codificadores hardware que cumplan con los requisitos de la aplicación, es decir, alto frame rate y alta definición de imagen. Por lo tanto, si somos capaces de llevar a cabo un tipo de codificación muy rápida, reduciremos los recursos de almacenamiento, y será posible la grabación en tiempo real, al igual que en las cámaras convencionales.

Muchos codificadores hardware basados en diferentes algoritmos de codificación se utilizan en sistemas reales [5], [6], [7], [8], [9]. La mayoría de ellos son Circuitos Integrados de Aplicación Específica (ASICs) dedicados a algoritmos específicos de codificación que no están diseñados para trabajar en tiempo real con altos frame rates y con formatos de vídeo de alta definición.

Sin embargo, se han hecho varios estudios sobre la codificación en cámaras de alta velocidad. En [10] los autores presentan un codificador JPEG basado en FPGA, que es capaz de comprimir hasta 500

[1]Dpto. de Física y Arquitectura de Computadores, Univ. Miguel Hernández. Elche, e-mail: {ealcocer, otoniel, mels}@umh.es

[2]Dpto. de Ingeniería de Comunicaciones, Univ. Miguel Hernández. Elche, e-mail: roberto.gutierrez@umh.es

frames/s con una resolución de $1280x1024$. Además, en [11] una versión mejorada del algoritmo Fast Boundary Adaptation Rule (FBAR) [12] en conjunto con la modulación de código de pulso diferencial (DPCM) se aplica para aumentar la eficiencia R/D, aunque no se proporcionan los tiempos de codificación.

En general, las limitaciones impuestas por las aplicaciones de captura de vídeo con altos frame rates descartan la mayor parte de las técnicas de codificación existentes (por ejemplo, la codificación por predicción o por transformada), ya que son mucho más complejas que PCM y/o utilizan la codificación predictiva y entrópica (lo que elimina las propiedades de acceso aleatorio y escalabilidad). Por ello, un algoritmo de codificación que tenga propiedades similares a las de PCM (baja complejidad, acceso aleatorio, y escalabilidad), pero con mejor eficiencia de codificación, sería interesante . El codificador de imagen Modulo-PCM (MPCM) [13] cumple estos requisitos. Para codificar una imagen, MPCM elimina ciertos bits de cada píxel siendo éste un procesamiento muy simple. La complejidad se puede encontrar en el decodificador, donde los bits eliminados de cada píxel, se predicen usando los restantes bits del píxel y la información que el decodificador calcula por interpolación entre píxeles vecinos previamente decodificados.

En este trabajo se implementa un códec rápido basado en Modulo Pulse Code Modulation (MPCM) [13] en la FPGA de Xilinx XC7Z020-1CLG484CES. Los resultados muestran que el codificador MPCM basado en FPGA obtiene un rendimiento de hasta 409,84 MBytes/seg a altas tasas de compresión, lo que permite almacenar en una memoria no volátil 2501 frames por segundo a una resolución de $1280x1024$. Además, en este trabajo se presenta una implementación hardware del sistema de decodificación MPCM, que es capaz de reproducir un vídeo de alta definición Full HD a 204 frames por segundo.

El resto del trabajo se organiza de la siguiente manera: En la sección 2 se presenta un breve resumen del codificador Módulo-PCM. En la sección 3 se describe la arquitectura propuesta. Una evaluación detallada de la implementación de esta arquitectura se muestra en la sección 4 en términos de R/D, retardo de codificación, consumo de energía y área ocupada en el dispositivo hardware. Por último, en la sección 5 se presentan algunas conclusiones.

II. Sistema de codificación

En esta sección se describe el algoritmo MPCP para la codificación de una señal unidimensional. Consideramos $\widetilde{x}_n$ $(n \in N)$ una señal continua en amplitud y discreta en tiempo cuyos valores en amplitud se extienden entre $[A_{min}, A_{max}]$. Tomamos x_n como la señal digital resultante de la cuantización escalar uniforme de $\widetilde{x}_n$ con una tasa fija de B bits/muestra y un paso de:
$\Delta = (A_{max} - A_{min})/2^B$.

La manera más fácil para reducir el bit-rate de

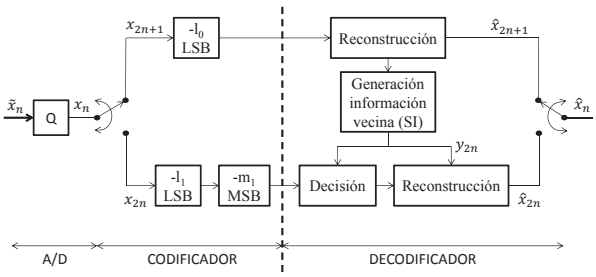


Fig. 1. Diagrama de bloques del algoritmo MPCM

$\widetilde{x}_n$ es eliminar los l-LSBs (bits menos significativos) de cada palabra-código de $\widetilde{x}_n$. Para lograr una reducción de tasa más eficiente, nos basamos en el algoritmo de codificación MPCM (Figura 1), donde las muestras de $\widetilde{x}_n$ se dividen en dos grupos: $S_0 = \{x_{2n+1} | n \in i = 0, 1, 2, \ldots\}$ y $S_1 = \{x_{2n} | n = 1, 2, \ldots\}$, los cuales se codifican con precisiones distintas. Como se muestra en la Figura 1, cada muestra de S_0 se codifica eliminando los l_0-LSBs de su palabra-código mientras que en cada muestra de S_1 se eliminan los m_1-MSBs (bits más significativos) y l_1-LSBs. Entonces, el codificador trabaja a una tasa media de $R = B - (l_0 + l_1 + m_1)/2$ bits/muestra.

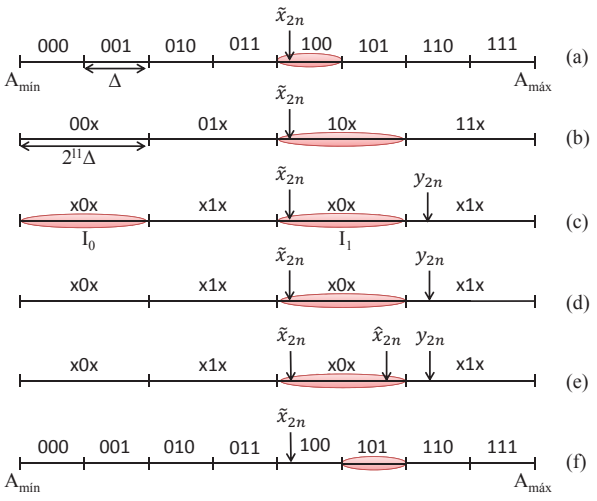


Fig. 2. Intervalos de cuantización y palabras-código para B=3, l_1=1, m_1=1: (a)Tras la conversión A/D, (b)Tras la eliminación de l_1 bits, (c)Tras la eliminación de m_1 bits, (d)Tras la decisión del intervalo I_0 o I_1, (e) tras la reconstrución y (f) tras la cuantización final. El símbolo X representa los bits eliminados. Los intervalos marcados son los representados por la palabra de código seleccionadas en cada paso para los valores mostrados de x_{2n} y y_{2n}.

Ya que la codificación de x_{2n+1} es equivalente a una cuantización uniforme con un incremento en el intervalo de cuantización igual a $2^{l_0}\Delta$, el decodificador puede directamente reconstruir las muestras de S_0 (Figura 1). En cuanto a la codificación de las muestras en S_1 (Figura 2(a)), la eliminación de los l_1-LSBs de x_{2n} es equivalente a cuantizar uniformemente su valor original con un incremento en el intervalo de cuantización igual a $2^{l_1}\Delta$ (Figura 2(b)). Tras eliminar los m_1-MSBs, la palabra código resultante identifica a un conjunto de 2^{m_1} intervalos no consecu-

tivos $\{I_i | i = 0, \ldots, 2^{m_1} - 1\}$, teniendo cada intervalo una longitud de $2^{l_1}\Delta$ (Figura 2(c)).

En el decodificador MPCM, para decidir que intervalo I_i pertenece a $\hat{x}_{2n}$, se explota la correlación entre las muestras de la señal mediante la ayuda de predicciones para cada muestra en S_1 basadas en muestras de S_0 previamente decodificadas. La predicción y_{2n} actúa como SI (Side Information) para decidir el intervalo (Figura 2(d)). La precisión del SI depende del grado de correlación entre las muestras x_n y la distorsión introducida por la codificación de S_0. Con el fin de limitar el impacto de la distorsión de codificación en la calidad del SI, a la hora de codificar l_0 debe ser menor que l_1 ($l_0 < l_1$). Si el proceso de decisión se realiza sin error para x_{2n}, sus m_1-MSBs se recuperan adecuadamente. Una vez que el decodificador ha estimado los m_1-MSBs de x_{2n}, intenta recuperar sus l_1-LSBs que finalmente proporciona la reconstrucción de $\hat{x}_{2n}$. Esta reconstrucción se realiza usando el intervalo de cuantización I_i donde supuestamente se encuentra x_{2n} y su SI (y_{2n}).

Para una descripción más detallada del algoritmo MPCM, se remite al lector a [13], [14].

III. Implementación hardware

Con el fin de hacer frente al gran ancho de banda que necesitan las cámaras de alta velocidad de hoy en día, tanto el codificador como el decodificador MPCM se han implementado en una arquitectura hardware. El lenguaje de descripción utilizado para construir el diseño es VHDL. La implementación propuesta en hardware se ha desarrollado en una FPGA Zynq-7000 de la familia Xilinx, específicamente sobre el modelo ZC702 que incluye el XC7Z020-1CLG484CES SoC (System-on-Chip) [15].

A. Arquitectura del codificador

La arquitectura implementada del codificador se ilustra en la Figura 3. La imagen original capturada por los sensores de la cámara se almacena en un bloque de memoria cuya lectura está determinada por un bloque de control. En esta estructura, 16 píxeles se leen en cada ciclo de reloj con el fin de acelerar el proceso de codificación tanto como sea posible dentro del alcance de la memoria interna del dispositivo. Esta memoria actúa como un buffer de frames interno para leerlos en aplicaciones de alta velocidad y es implementado con block RAMs. De esta manera, utilizamos 52 blocks RAMs Dual-port de 36 Kb, configurando sus puertos a $512x64$ bits, donde 64 bits de salida, es decir 8 píxeles, se leen en cada puerto de salida de la memoria. Estos píxeles son procesados en el siguiente bloque, sin retardo, en el mismo ciclo de reloj, donde son codificados eliminando los correspondientes bits l_0 o l_k y m_k. Finalmente, obtenemos las muestras codificadas de la imagen las cuales serán enviadas al dispositivo final de almacenamiento.

En el diseño propuesto, la imagen se divide en bloques de 4 muestras diferentes $x_{0,0}[n_1, n_2]$, $x_{0,1}[n_1, n_2]$, $x_{1,0}[n_1, n_2]$ y $x_{1,1}[n_1, n_2]$ de tal manera

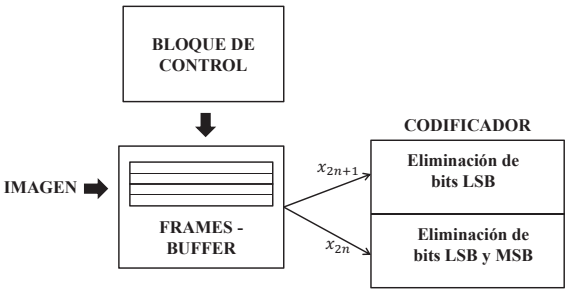


Fig. 3. Diseño de la arquitectura del codificador

que:

$$x_{p,q}[n_1, n_2] = x[2n_1 + p, 2n_2 + q]$$

con p y $q \in \{0, 1\}$. Así, la primera muestra se codifica usando PCM, eliminando unicamente los LSBs l_0, y el resto de partes utilizando MPCM, al extraer los LSBs l_k y los MSBs m_k. En la Figura 4 se muestra un diagrama con los pasos realizados en nuestro algoritmo hardware. Hay que tener en cuenta que tanto la lectura como la codificación de los 16 píxeles se lleva a cabo en el mismo ciclo de reloj.

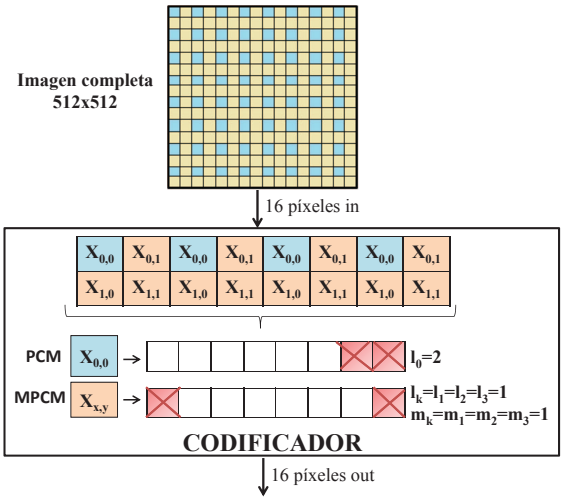


Fig. 4. Ejemplo de algoritmo de codificación con $l_0 = 2$, $l_k = 1$ y $m_k = 1$

B. Arquitectura del decodificador

En la Figura 5 se muestra la arquitectura propuesta para el decodificador. En este enfoque, el buffer intermedio se llena con muestras codificadas hasta que se completan las tres primeras líneas, entonces se inicia el proceso de decodificación. El decodificador está dividido en dos pasos: decisión y reconstrucción. La recuperación de la imagen original se realiza como sigue:

- Tres columnas de datos se procesan en el bloque de decisión, donde la señal PCM se reconstruye. Entonces se generan SIs precisos a partir de las muestras PCM y así se selecciona uno de los posibles intervalos 2^{m_k} donde cada señal decodificada MPCM se situará.

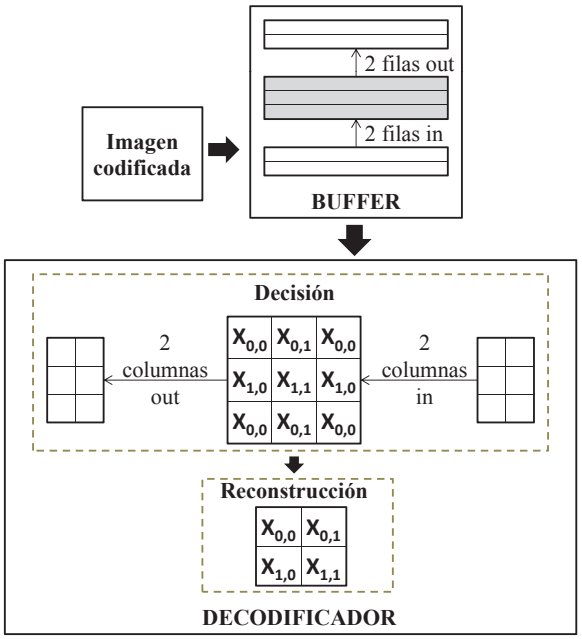


Fig. 5. Diseño de la arquitectura del decodificador

- En el bloque de reconstrucción del decodificador, se recuperan los bits menos significativos (LSB) que fueron eliminados en el proceso de codificación mediante la utilización de su SI y el intervalo correspondiente a cada muestra MPCM. Tras completar estos pasos, se obtienen cuatro píxeles decodificados.
- En cada ciclo de reloj, dos columnas de muestras codificadas salen del bloque del decodificador y dos nuevas entran en él. Este proceso continua iterativamente hasta que todas las muestras de las tres filas del buffer son decodificadas.
- Entonces, el buffer se desplaza. Dos filas salen del buffer y dos nuevas entran. Todas las operaciones anteriores continuan hasta que todas las muestras codificadas de entrada son procesadas.

El diseño propuesto del decodificador ha sido completamente segmentado, esto hace que cada una de las etapas mencionadas anteriormente para la decodificación se realicen concurrentemente. Además, como se explicó anteriormente, se obtienen 4 píxeles decodificados en cada ciclo de reloj. En esta propuesta, la frecuencia de operación se ha establecido para conseguir estos 4 píxeles, pero con el fin de organizar toda la imagen decodificada, se ha utilizado el módulo Phase-Locked-Loop (PLL), mediante el cual se generan múltiples relojes a partir de la señal de reloj de entrada determinada. Por lo tanto, en cada ciclo, se almacenan 4 píxeles en una memoria intermedia con una frecuencia fija, pero estos píxeles se leen con una frecuencia 4 veces mayor, logrando así una salida en serie sin retardo. Los buffers utilizados en esta arquitectura se han implementado usando únicamente block RAMs de 18 Kb de doble puerto.

IV. Resultados

En esta sección se presenta la evaluación del sistema completo en términos de PSNR, tiempos de codificación/decodificación, uso de área del dispositivo FPGA, máximo frame rate y mejoras en velocidad comparándolos con algoritmos secuenciales en CPUs. Las arquitecturas han sido sintetizadas, emplazadas y enrutadas utilizando la herramienta Xilinx ISE 14.3, y han sido simuladas y verificadas mediante Matlab/Simulink través de la herramienta System Generator. Estas propuestas han sido diseñadas para el dispositivo Zynq AP SoC previamente mencionado. El área ocupada del dispositivo, la frecuencia máxima y la estimación de consumo de energía se han medido desde la herramienta Xilinx ISE 14.3. En nuestros experimentos, hemos evaluado los resultados de cinco imágenes en escala de grises (Zelda, Lena, Peppers, Barbara y Baboon) con una resolución de $512x512$ píxeles y 8 bits por muestra. Por otra parte, hemos asignado los valores óptimos de los parámetros de codificación/decodificación con el fin de obtener el máximo PSNR para una tasa de bits dada. La asignación de valores de los parámetros para el codificador/decodificador MPCM se proponen en [14].

A. Evaluación del codificador

En la Tabla I se muestra el PSNR obtenido para todas las imágenes evaluadas en función del bit-rate (R). Como se esperaba, para tasas altas de (R), lo que significa eliminar pocos bits en el proceso de codificación, el algoritmo MPCM porporciona generalmente mejor PSNR debido a que no se produce una pérdida significativa en el proceso, y en consecuencia, no se introducen grandes errores en el proceso de decodificación. Por lo tanto, a menos bit-rate (R), menor valor de PSNR. Hay que tener en cuenta que, para cada tasa, los parámetros l_0, l_1, m_1 no son necesariamente los mismos para todas las imágenes. Cada imagen tiene unos parámetros apropiados para obtener la calidad óptima en la imagen recuperada.

En cuanto al retardo de codificación/decodificación, el codificador propuesto funciona a una frecuencia de reloj máxima de 204,96 MHz. Además, el algoritmo requiere 16.387 ciclos para llevar a cabo el proceso completo de codificación para una resolución de imagen de 512x512 píxeles. Por lo tanto, necesitamos 79,952 ms para codificar cualquier imagen de la mencionada resolución, siendo 12 veces más rápido que el algoritmo secuencial en una CPU Intel Core 2 a 1,8 Ghz con 5 GBytes de RAM. Ya que el proceso de codificación sólo depende de la resolución de la imagen, en la Figura 6 se muestra el máximo frame-rate obtenido en la arquitectura propuesta. Como se observa, la implementación hardware del codificador MPCM es capaz de comprimir hasta 3558 frames por segundo para la resolución HD-Ready y hasta 1668 frames por segundo para la resolución Full-HD.

El proceso de codificación de alta velocidad hace que las cámaras de alta velocidad sean capaces de capturar y grabar continuamente sin las restricciones

TABLA I
PARÁMETROS ÓPTIMOS Y VALORES DE PSNR PARA TODAS LAS IMÁGENES EVALUADAS

	R=1bpp		R=2bpp		R=4bpp		R=6bpp	
Imagen	(l_0,l_1,m_1)	PSNR	(l_0,l_1,m_1)	PSNR	(l_0,l_1,m_1)	PSNR	(l_0,l_1,m_1)	PSNR
Zelda	(4,8,0)	33.83	(0,8,0)	36.57	(1,3,2)	40.07	(2,1,1)	48.07
Lena	(4,8,0)	31.79	(3,6,1)	33.71	(1,4,1)	37.74	(2,1,1)	47.80
Peppers	(4,8,0)	30.57	(3,7,0)	32.14	(4,4,0)	34.88	(2,2,0)	44.62
Barbara	(4,8,0)	24.87	(3,7,0)	26.59	(4,4,0)	33.67	(2,2,0)	44.56
Baboon	(4,8,0)	22.53	(3,7,0)	24.22	(4,4,0)	32.20	(2,2,0)	43.85

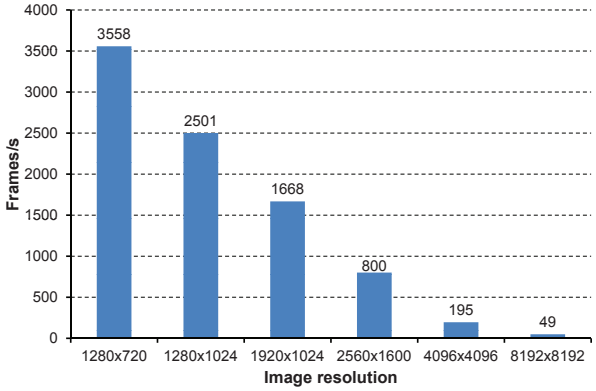


Fig. 6. Máximo frame por segundo en el codificador para diferentes resoluciones de imagen

de tamaño de la memoria RAM interna. Por ejemplo, a una tasa de compresión de 1 bpp, el sistema de codificación tiene un rendimiento de 409,84 MBytes/s, que es menor que el ancho de banda disponible en dispositivos de almacenamiento típicos como un SSD (Solid State Drive) (de hasta 600 MBytes/s). Dependiendo de la aplicación final, si es necesaria una calidad de imagen mayor, la implementación hardware MPCM propuesta podría comprimir a una tasa de 4 bpp con buena calidad y un rendimiento de ancho de banda de 1640 Mbytes/s, que extenderá el tiempo de captura en el módulo de memoria interna de la cámara hasta 1,4 veces o permitirá su transmisión a través de un enlace punto a punto Ethernet 40 Gbps.

Los elementos básicos de una FPGA son los CLBs (Bloques de Lógica Configurable). La arquitectura CLBs incluye: LUTs de 6 entradas, capacidad de memoria y de registro dentro de la LUT y la funcionalidad de registro de desplazamiento. Las LUTs en el dispositivo Zynq-7000 AP SoC pueden ser configuradas o bien como una LUT de 6 entradas (ROMs de 64-bit) con una salida, o como dos LUTs de 5 entradas (ROMs de 32-bit) con salidas separadas pero direcciones comunes o entradas lógicas. Cada salida de una LUT puede opcionalmente ser registrada en un flip-flop. Cuatro de estas LUTs y sus ocho flip-flops además de multiplicadores y lógica aritmética forman un slice, y dos slices forman un bloque lógico configurable (CLB). Cuatro de los ocho flip-flops por slice (un flip-flop por LUT) pueden configurarse como latches opcionalmente. Entre el 25-50% de todos los slices pueden también utilizar sus LUTs como RAM-distribuidas de 64-bit o como registros de desplazamiento de 32-bit. [15].

TABLA II
ÁREA USADA EN FPGA - IMPLEMENTACIÓN CODIFICADOR

	Used	**Available**	**% use**
No. of Slices	14	13300	1%
No. of Slice Registers (as Flip Flops)	17	106400	1%
No. of Slice LUTs	35	53200	1%
No. of RAMB36	52	140	37%
FMax(MHz)	204,96	-	-
Consumption (mW)	305	-	-

En la Tabla II se presentan los resultados de la implementación del codficador en términos de recursos hardware usados, indicando el número de Slices, Flip-Flops, LUTs y bloques RAMs de 36 KB utilizados. Adicionalmente, se muestra una estimación del consumo de potencia que proporciona la herramienta XPower de Xilinx ISE 14.3, obteniendo solamente un consumo de 305 mW debido a la gran segmentación realizada en el diseño del codificador. Como se puede observar, sólo un 1% de toda el área disponible en la FPGA es utilizada, por tanto dada la gran cantidad de área disponible en la FPGA, ésta podríamos usarla para implementar múltiples codificadores idénticos que podrían ejecutarse concurrentemente. De este modo, diferentes frames podrán ser codificados simultáneamente con el fin de incrementar el tiempo disponible de grabación de una cámara de alta velocidad. Para aprovechar las ventajas de ello, simplemente tendríamos que considerar el uso de una memoria externa para almacenar los diferentes frames, teniendo en cuenta el uso de los bloques de RAMs como buffers intermedios.

B. Evaluación del decodificador

En lo que se refiere al decodificador, la frecuencia máxima de reloj se ha establecido en 100MHz, siendo la menor latencia 713 ciclos. Esta frecuencia se toma como compromiso debido al uso de otra frecuencia 4 veces mayor, proporcionada por el módulo PLL, ya que analizando el retardo de la implementación, se podrían conseguir frecuencias mayores (aprox. 160 MHz), como se discutierón en la sección III-B. De esta manera, el decodificador MPCM es capaz de recuperar 400 Mpíxels por segundo a esa frecuencia. Por otro lado, el algoritmo requiere 66.240 ciclos para realizar el proceso de decodificación de imagen con

una resolución de 512x512 píxeles, por tanto se necesitan 662 μs para decodificar cualquier imagen con esta resolución, llegando a ser 70 veces más rápido que el algoritmo de decodificación secuencial en un Intel Core 2 CPU a 1.8 Ghz con 5 GBytes de RAM.

La Figura 7 muestra el máximo frame rate obtenido en la decodificación de la arquitectura propuesta. Como se muestra, la implementación hardware del decodificador MPCM es capaz de recuperar más de 434 frames por segundo para una resolución HD-Ready o más de 204 frames por segundo para una resolución Full-HD, que corresponde a un rendimiento de 50 MBytes/s, pudiendo así reproducir vídeo de alta definición a altas tasas de frames por segundo.

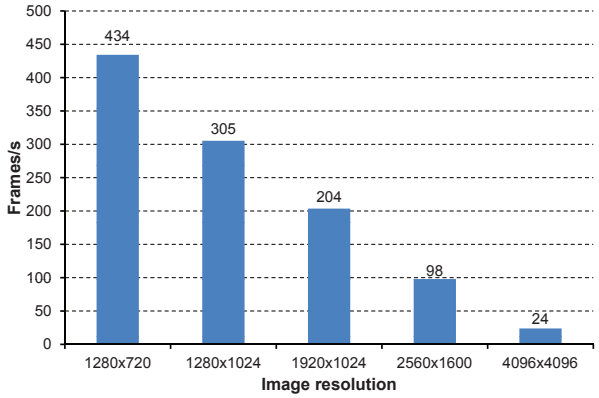


Fig. 7. Máximo frame por segundo en el decodificador para diferentes resoluciones de imagen

De forma análoga al estudio realizado en el codificador, el área utilizada en el dispositivo es menor del 1%. El área ocupada puede variar dependiendo de los parámetros l_0, l_1, m_1, pero en cualquier caso será menor que un 1%. Como se indicaba en la sección III-B, los buffers utilizados han sido modelados en bloques RAMs de 18 Kb de doble puerto con el fin de aprovechar la ventaja de un consumo menor comparado con las memorias distribuidas, además de ser más rápidas.

V. Conclusiones

En este artículo hemos presentado una implementación eficiente del codec MPCM en una FPGA. Se ha mostrado la calidad de las imágenes reconstruidas en términos de PSNR a diferentes tasas de compresión. En lo concerniente a la velocidad de codificación, los resultados muestran que nuestra propuesta de implementación es capaz de comprimir una imagen de resolución Full-HD a 1668 frames por segundo. El máximo ancho de banda alcanzado por nuestra implementación es de 409.84 MBytes/s lo que permite la grabación continua de una cámara de alta velocidad actual con una resolución de imagen HD-Ready ($1280x720p$) y una calidad razonable. Pero, si la aplicación final requiere una calidad de imagen superior, nuestro codificador es capaz de proporcionar más de 1640 MBytes/s a una tasas de compresión 2:1, duplicando el tiempo de captura sobre la memoria RAM interna de la cámara de alta velocidad. El área utilizada de la FPGA es menor al 1% del área total disponible, lo que nos da la posibilidad de replicar varios sistemas de codificación y por tanto, diferetes frames pueden ser comprimidos de manera paralela.

También hemos desarrollado en hardware el módulo del decodificador MPCM. El diseño propuesto es capaz de recuperar imágenes a 204 frames por segundo para una resolución Full-HD, con un área ocupada del dispositivo menor al 1%.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad con el proyecto de I+D con referencia TIN2011-27543-CO3-03 y por la Generalitat Valenciana con la ayuda de referencia ACOMP/2013/03.

Referencias

[1] Vision Research PHANTOM v641, "http://www.visionresearch.com/Products/High-Speed-Cameras/v641," .
[2] PHOTRON FASTCAM SA-X, "http://www.photron.com/index.php," .
[3] N. S. Jayant and P. Noll, Prentice-Hall, Inc, 1974.
[4] P. Gemeiner, W. Ponweiser, P. Einramhof, , and M. Vincze, "Real-time slam with high-speed cmos camera," *14th International Conference on Image Analysis and Processing ICIAP*, pp. 297–302, September 2007.
[5] L.H. Chen, W.L. Liu, O.T.C. Chen, and R.L. Ma, "A reconfigurable digital signal processor architecture for high-efficiency MPEG-4 video encoding," in *IEEE Conference on Multimedia and Expo*, 2002.
[6] J. Ritter, G. Fey, and P. Molitor, "Spiht implemented in a XC4000 device," in *IEEE 45th Midwest Symposium on Circuits and Systems*, 2002.
[7] I. Urriza, J.I. Artigas, J.I. Garcia, L.A. Barragan, and D. Navarro, "Vlsi architecture for lossless compression of medical images using discrete wavelet transform," in *Conference on Design Automation and Test in Europe*, 1998.
[8] J. Ahmad and M. Ebrahim, "Fpga based implementation of baseline jpeg decoder," *International Journal of Electrical & Computer Sciences*, vol. 9, no. 9, pp. 371–377.
[9] A. Descampe, Devaux F., Rouvroy G., Macq B., and Legat J.D., *An Efficient FPGA Implementation of a Flexible JPEG2000 Decoder for Digital Cinema*, Ph.D. thesis, Université catholique de Louvain, 2002.
[10] Xi CHEN, Lin ZENG, Qinglin ZHANG, and Wenxuan SHI, "A novel parallel JPEG compression system based on FPGA," *Journal of Computational Information Systems*, vol. 7, no. 3, pp. 697–706, 2011.
[11] Yan Wang, Shoushun Chen, and A. Bermak, "Fpga implementation of image compression using dpcm and fbar," in *Integrated Circuits, 2007. ISIC '07. International Symposium on*, sept. 2007, pp. 329 –332.
[12] Dominique Martinez and Marc M. Van Hulle, "Generalized boundary adaptation rule for minimizing rth power law distortion in high resolution quantization," *Neural Networks*, vol. 8, no. 6, pp. 891 – 900, 1995.
[13] J. Prades-Nebot, A. Roca, and E. Delp, "Modulo-pcm based encoding for high speed video cameras," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, oct. 2008, pp. 153 –156.
[14] Marleen Morbee Ph.D. Thesis, "Optimized information processing in resource-constrained vision systems," 2011.
[15] Xilinx Zynq-7000, "Zynq-7000 all programmable soc overview, advance product specification - ds190 (v1.2) available on: http://www.xilinx.com/support/documentation-/data_sheets/-ds190-Zynq-7000-Overview.pdf," August 2012.

Análisis de estrategias paralelas basadas en GOP sobre el nuevo estandar de vídeo HEVC

Otoniel López-Granado, Manuel P. Malumbres, Hector Migallón [1] y Pablo Piñol

Resumen— **HEVC es el nuevo estandar de codificación de video, siendo capaz por termino medio de reducir a la mitad el tamaño del bitstream con respecto al anterior estandar H.264/AVC obteniendo la misma calidad.**

Nuestro interés se centra en aplicar técnicas de procesamiento paralelo al codificador HEVC para reducir significativamente las demandas computacionales sin alterar el rendimiento del mismo. Para ello, proponemos varias alternativas de paralelización del codificador HEVC específicas para arquitecturas multicore. Nuestras propuestas están basadas en el uso de OpenMP a un nivel grueso de paralelización llamado 'GOP-based'. Las aproximaciones basadas en GOP codifican simultáneamente varios grupos de frames consecutivos. Dependiendo de cómo estos GOPs (Grupo de imágenes) se agrupan y distribuyen puede ser crítico obtener buenos rendimientos.

Palabras clave— **algoritmos paralelos, Codificación de vídeo, HEVC, multicore, rendimiento**

I. Introducción

EL nuevo standar de codificación de video 'High Efficiency Video Coding (HEVC)' ha sido desarrollado por el 'Joint Collaborative Team on Video Coding (JCT-VC)' que fue establecido por 'ISO/IEC Moving Picture Experts Group (MPEG)' y 'ITU-T Video Coding Experts Group (VCEG)'. Este nuevo estandar reemplazará al actual estandar H.264/AVC [1] para dar respuesta a las actuales y futuras demandas del mercado Multimedia. La resolución 4K de video ya etá en el mercado en la actualdad y no tardará mucho en ser una realidad cotidiana una resolución de 8K. Por otra parte, el nuevo estandar da soporte a una mayor profundidad de color (10 bits). HEVC pretende doblar la eficiencia en codificación con respecto a H.264/AVC mostrando una calidad visual similar con la mitad de bitrate.

En cuanto a complejidad se refiere, el decodificador HEVC no incrementará significativamente su complejidad con respecto a H.264/AVC [2]. Sin embargo, el codificador es varias veces más complejo que H.264/AVC lo que hará que este tema sea de interés científico en los proximos años. La versión actual del software de referencia de este nuevo estandar de vídeo, llamado 'HEVC test model (HM)', es HM 10.0 que se corresponde con el borrador de la especificación del estandar HEVC 10 [3]. En [4] se puede encontrar una buena descripción de las características de este nuevo estandar.

En la literatura podemos encontrar algunos trabajos relacionados con el análisis de complejidad y estrategias de paralelización de este nuevo estandar, como en [5] [6] y [7]. Muchas de las propuestas de paralelización existentes sobre el HEVC están centradas en el decodificador, y tratan de buscar las mejores optimizaciones para decodificar en tiempo real vídeo de alta definición (HD) y Ultra alta definición (UHD).

Actualmente, hay muy pocos trabajos relacionados con la paralelización del codificador HEVC. En [8] los autores proponen una paralelización de grano fino en el módulo de estimación de movimiento, permitiendo realizar la predición de movimiento en todos las unidades de predicción (PUs) disponibles en la unidad de codificación (CU) al mismo tiempo. En [9] los autores proponen una paralelización dentro del modulo de predicción Intra que consiste en eliminar la dependencia de los datos entre subbloques de una CU, obteniendo buenos resultados en términos de speed-up.

En este artículo analizaremos las estrategias de paralelización disponibles en el estandar y su viabilidad de desarrollo en el software de referencia HM. Además presentamos varias alternativas de paralelización a nivel de GOP para el codificador HEVC especialmente diseñadas para los perfiles Low-delay.

El resto del artículo se organiza de la siguiente manera: En la sección II, se muestra un resumen de los diferentes perfiles disponibles en el HEVC y los test de las condiciones comunes de evaluación. La sección III presenta las diferentes estrategias de paralelización de alto nivel propuestas en el estandar HEVC. La sección IV presenta nuestras propuestas de paralelización a nivel de GOP para los perfiles de aplicación Low-delay, mientras que en la sección V se presenta una comparativa de las diferentes estrategias propuestas. Finalmente, conclusiones y trabajo futuro se muestran en la sección VI.

II. Perfiles HEVC

En [10] el JCT-VC define las condiciones comunes de test y las configuraciones del software de referencia que se deben usar para realizar los experimentos con el HEVC, con el fin de poder comparar las diferentes aportaciones realizadas a éste.

Se definen un total de 24 secuencias de vídeo catalogadas en 6 clases. También se definen los parámetros de cuantización (QP) y los ficheros de configuración para el codificador. El hecho de usar estas condiciones comunes permite realizar fácilmente comparativas entre las diferentes propuestas que hacen los investigadores. También el JCT-VC indica cómo calcular las curvas 'Rate-Distortion (RD)' y el % de ganancia en bitrate, usando la medida Bjontegaard-Delta (BD) [11].

Las categorías desde A hasta E incluyen secuencias de vídeo natural a diferentes resoluciones y frame-

[1]Dpto. de Física y Arquitectura de Computadores, Univ. Miguel Hernández, e-mail: otoniel@umh.es.

rate, mientras que la categoría F contiene vídeo sintético.

Cuatro valores de QP se indican a la hora de realizar las comparativas: 22, 27, 32, 37. Para cada punto se obtiene el bitrate y el Peak Signal-to-Noise Ratio (PSNR), pudiendo realizarse las curvas de Rate-Distortion y calcular el BD.

Con el software de referencia [12] se distribuyen también los ficheros de configuración. Hay 8 test diferentes que son una combinación de 2 profundidades de color: Main (8 bits) y Main10 (10 bits) con 4 modos de codificación: All Intra (AI), Random Access (RA), Low-Delay B (LB), y Low-Delay P (LP).

En el modo All Intra, cada imágen se codifica como un I-frame, es decir, se codifica sin estimación/compensación de movimiento. De esta forma, cada frame es independiente de los demás en la secuencia de vídeo. Este modo proporciona menores niveles de compresión que los otros 3, porque el uso de P-frames y B-frames obtienes mayores niveles de compresión que los I-frames para el mismo nivel de calidad. Por otra parte, el proceso de codificación para el modo All Intra es más rápido que en los otros 3 porque no se pierde tiempo en realizar estimación de movimiento. Este modo es perfecto para aquellas aplicaciones que requieren de una codiifcación rápida y no están limitadas en el ancho de banda o el almacenaje.

El modo Random Access combina I-frames y B-frames. Un B-frame es un frame que usa estimación/compensación de movimeinto para obtener mejores niveles de compresión. Cada bloque de un B-frame puede usar hasta 2 frames de referencia, así en el proceso de codificación se mantienen 2 listas de frames de referencia. El tamaño del GOP GOP (Group Of Pictures / Grupo de imágenes) es de 8. Los frames de referencia se pueden localizar antes o después del frame que se está codificando actualmente. De esta manera, en este modo, el orden de codificación no es el mismo que el orden de visualización. Con el fin de permitir el acceso rápido a la secuencia (fast forward), se inserta un I-frame periódicamente. Dependiendo del frame-rate de la secuencia, este periodo varía, siendo siempre múltiplo de 8 (el tamaño del GOP).

Los modos Low-Delay (LP and LB) codifican cada frame en orden de visualización. Primero se inserta un I-frame y después sólo se usan P-frames (o B-frames) para el resto de la secuencia. El tamaño de GOP es 4, y todos los frames de referencia son previos al que se está codificando actualmente. Estos dos modos obtienen mayores tasas de compresión que el modo AI y evitan el retraso que introduce el modo RA. Este modo se usa para aplicaciones como vídeo conferencia que tienen restricciones de ancho de banda y tiempo.

III. Estrategias de paralelización de alto nivel en el estandar HEVC

Las estrategias de paralelismo de alto nivel se pueden clasificar siguiendo un esquema jerárquico. Esta clasificación se debe aplicar con cautela teniendo en cuenta los recursos hardware disponibles con el fin de realizar una implementación adecuada y eficiente. De esta forma, definimos los siguientes niveles desde el mas grueso hasta el más fino: GOP, tile, slice, y wavefront. Cuando se diseña una versión paralela del HEVC, primeramente analizamos el hardware disponible para determinar qué nivel es el más adecuado.

El nivel más grueso de paralelización, GOP-based, se basa en romper la secuencia de vídeo en GOPs de manera que cada GOP se procesa de forma independiente a los demás. En general, esta aproximación es la que mejor eficiencia de paralelización reporta. Sin embargo, la eficiencia en codificación puede verse afectada por cómo definimos estos GOPs y cómo eliminamos la interdependencia entre ellos.

Los Tiles se usan para dividir una imágen horizontalmente o verticalmente en varias sub-imagenes. Al usar Tiles las dependencias en la predicción se rompen en los bordes entre Tiles. Tiles consecutivos se representan en orden 'raster scan'. El orden de los Coding Tree Blocks (CTBs) se mantiene también en 'raster scan'.

Los Slices siguen el mismo concepto que en anterior estandar H.264/AVC permitiendo que una imágen se parta en grupos consecutivos de Coding Tree Units (CTUs). Hay una rotura en las dependencias en la predicción en los bordes de los slices que causa una pérdida de eficiencia en la codificación. El uso de slices está más justificado en temas de 'error resilience', más que en técnicas de paralelismo.

Por último, los Wavefronts dividen una imágen en filas de CTU, donde cada fila se procesa por un hilo diferente. Las dependencias entre filas se mantiene expcepto para el estado del contexto del aritmético CABAC [13], que se reinicializa al comienzo de cada fila de CTU. Para mejorar la eficiencia, en vez de reiniciar normalmente los contextos en CABAC, estos se heredan de la fila anterior.

Todas estas técnicas de paralelización son más útiles con resoluciones de imágenes superiores al HD. Para imágenes pequeñas, la decodificacion en tiempo real se puede realizar con un sólo proceso. Para resoluciones grandes de imágen puede ser util forzar un número mínimo de particiones de imágen para garantizar un nivel mínimo de paralelismo en el decodificador.

El software de referencia HM no soporta directamente muchas de estas técnicas de paralelización, sobre todo por el diseño de la implementación. En la siguiente sección presentaremos algunas aproximaciones de paralelización a nivel de GOP (GOP-based) que pueden implementarse en arquitecturas hardware multicore.

IV. algoritmos paralelos

En las secciones anteriores hemos revisdo las pricipales características del estandar de vídeo HEVC. Hemos paralelizado el software de referencia usando los modos LB y AI, ambos combinados con el per-

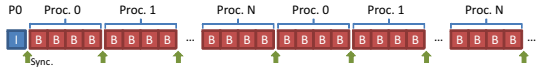


Fig. 1. Opción I: Distribución paralela.

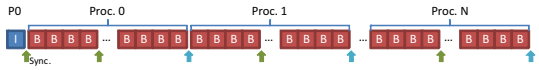


Fig. 2. Opción II: Distribución paralela.

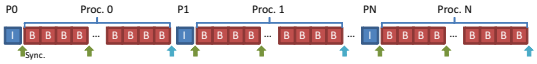


Fig. 3. Opción III: Distribución paralela.

Fig. 4. Opción IV: Distribución paralela.

fil Main (8 bits). Estos dos modos son utiles para aplicaciones con restricciones de tiempo, así que pensamos que pueden beneficiarse de las estrategias de paralelización. Obviamente, este trabajo puede exterderse fácilmente al modo Low-Delay P. Sin embargo, esto nos es así para el modo Random-Access debido a la forma en que usa los frames de referencia, usando referencias pasadas y futuras.

Los algoritmos paralelos desarrollados estan diseñados para un nivel de paralelización a nivel de GOP. Primeramente, decir que el tamaño de GOP para el modo AI es 1, porque todos los frames se computan como I-frames (sin frames de referencia). En el modo LB el tamaño de GOP es 4, sin embargo, este valor puede cambiarse. Por otra parte, hemos considerado algoritmos síncronos donde el proceso de sincronización se realiza después de computar los GOPs. Hemos desarrollado 4 aproximaciones:

- Opción I: (LB) en esta opción asignamos de forma secuencial cada GOP a cada proceso en la ejecución paralela, de manera que cada proceso codificará GOPs aislados.
- Opción II: (LB) en esta aproximación dividimos la secuencia en tantas partes como el número de procesos paralelos disponibles, de manera que cada proceso codificará grupos de GOPs adyacentes.
- Opción III: (LB) similar a Opción II, excepto que cada proceso comienza la codificación insertando un I-frame.
- Opción IV: (AI) similar a Opción I donde cada GOP se asigna secuencialmente a cada proceso, pero el GOP consiste en sólo un I-frame.

La Figura 1 muestra la distribución paralela para la Opción I. Los procesos de sincronización están localizados después de cada GOP. El proceso maestro (Proc. 0 o P0) computa el primer frame como I-frame. Tras esto, todos los procesos codifican un GOP de 4 B-frames. Todos los procesos, excepto el proceso maestro, codifica su primer B-frame sin frames de referencia, de esta manera, el número de bits necesarios para codificar este primer B-frame es similar al que usa el I-frame.

Para aumentar el rendimiento de los algoritmos paralelos propuestos, cada proceso contiene sus propios buffers de trabajo. Este hecho cambia el patrón real de frames de referencia utilizados. Por ejemplo, en la versión secuencial, el segundo B-frame de un GOP usa los frames *-1 -2 -6 -10* como frames de referencia, donde (*-1* significa el frame previo, y así sucesivamente). Como el tamaño de GOP es 4, el frame *-2* apunta al último frame del GOP previo. En el procesamiento paralelo, como asignamos GOPs aislados a cada proceso, el GOP previo no es el adyacente en la secuencia original, así, el frame *-2* no apunta al frame dos posiciones antes del frame actual. Si, por ejemplo, el número de procesos es 6, entonces el GOP previo para este proceso se localiza en la secuencia de vídeo 6 GOPs atrás con respecto al GOP actual. Así pues, el segundo B-frame de un GOP apuntará al frame *-22 (-2-(6-1)x4=-22)* en la secuencia de vídeo original. Se puede concluir que el algoritmo paralelo y el secuencial producirán bitstreams diferentes. Esto se analizará en la sección V, así como su impacto en términos de PSNR y bitrate.

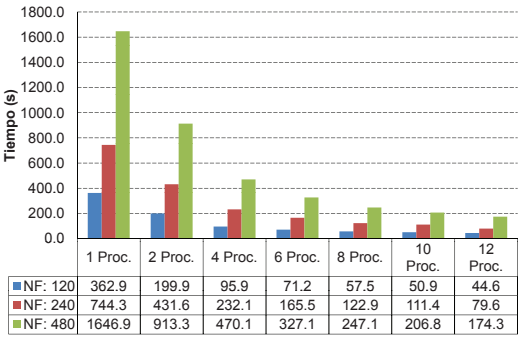
En la Figura 2 se puede ver una representación de distribución de la versión paralela Opción II. A igual que en la Opción I, los procesos de sincronización se localizan después de cada GOP. Así, todos los procesos, excepto el proceso principal, codifican su primer B-frame sin frames de referencia. En este caso, la lista de referencias no se modifica porque cada proceso trabaja con GOPs adyacentes. En el ejemplo anterior, para el segundo frame del GOP, el patrón de referencias solo se altera para los primeros tres GOPs. A partir de ahí, todos los frames de referencia están disponibles en el buffer privado de cada proceso.

La Figura 3 muestra la distribución paralela para la Opción III, donde se usa una estructura similar a la de la Opción II. Aquí cada proceso comienza codificando el primer frame como un I-frame. En este caso, las ejecuciones paralela y secuencial pueden ser identicas si en la versión secuencial realizamos un pequeño cambio en el fichero de configuración, cambiando el parámetro *IntraPeriod* de acuerdo con el número de procesos de la ejecución paralela. La Tabla I presenta los valores del parámetro *IntraPeriod* cuando computamos 240 y 480 frames. Más aún, el I-frame que se incluye debe ser un IDR (Instantaneous Decoding Refresh), así que debemos poner el parámetro *DecodingRefreshType* a 2.

Finalmente, en la Figura 4 se muestra la distribución paralela para la Opción IV. Remarcar, que la estructura paralela es simalar a la de la Opción I, pero el GOP siempre consiste en un I-frame. Los I-frames son de tipo IDR, con lo cual no hay diferencias entre las ejecuciones paralelas y secuencial.

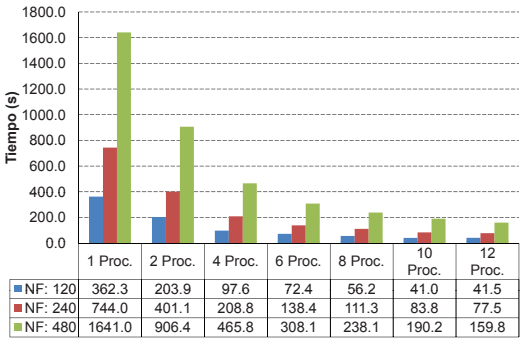
V. Experimentos

Analizaremos los algoritmos paralelos descritos en la sección IV, en términos de rendimiento paralelo, PSNR y bitrate. Se ha utiliado el paradigma de pro-



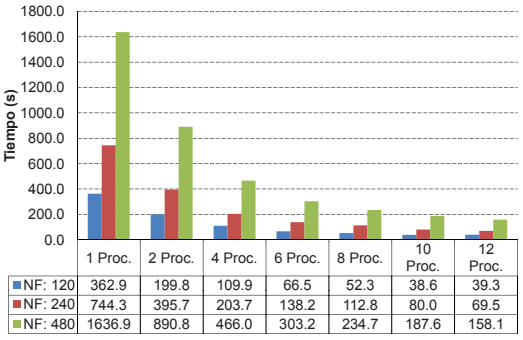
	1 Proc.	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	362.9	199.9	95.9	71.2	57.5	50.9	44.6
NF: 240	744.3	431.6	232.1	165.5	122.9	111.4	79.6
NF: 480	1646.9	913.3	470.1	327.1	247.1	206.8	174.3

(a) Opción I.



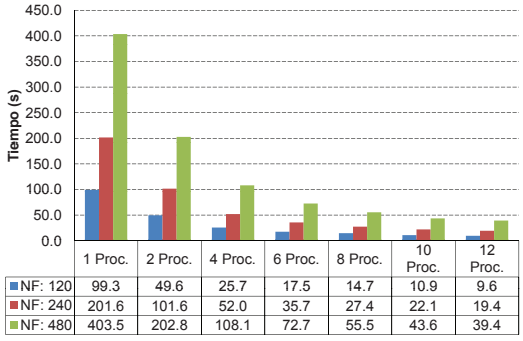
	1 Proc.	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	362.3	203.9	97.6	72.4	56.2	41.0	41.5
NF: 240	744.0	401.1	208.8	138.4	111.3	83.8	77.5
NF: 480	1641.0	906.4	465.8	308.1	238.1	190.2	159.8

(b) Opción II.



	1 Proc.	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	362.9	199.8	109.9	66.5	52.3	38.6	39.3
NF: 240	744.3	395.7	203.7	138.2	112.8	80.0	69.5
NF: 480	1636.9	890.8	466.0	303.2	234.7	187.6	158.1

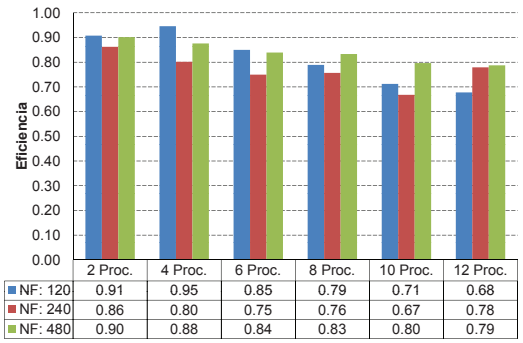
(c) Opción III.



	1 Proc.	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	99.3	49.6	25.7	17.5	14.7	10.9	9.6
NF: 240	201.6	101.6	52.0	35.7	27.4	22.1	19.4
NF: 480	403.5	202.8	108.1	72.7	55.5	43.6	39.4

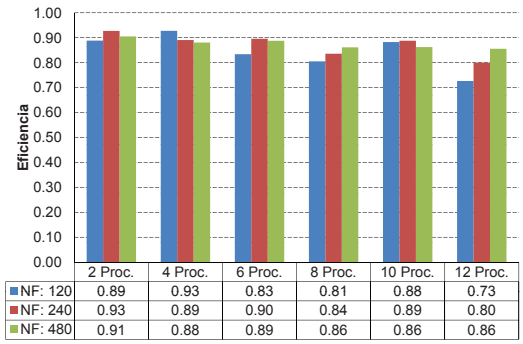
(d) Opción IV.

Fig. 5. Tiempos de cómputo de los algoritmos paralelos. Nº frames 120, 240 y 480.



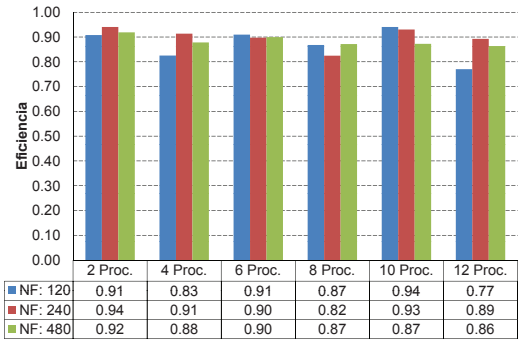
	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	0.91	0.95	0.85	0.79	0.71	0.68
NF: 240	0.86	0.80	0.75	0.76	0.67	0.78
NF: 480	0.90	0.88	0.84	0.83	0.80	0.79

(a) Opción I.



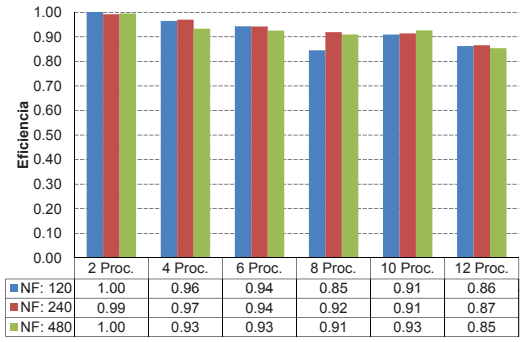
	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	0.89	0.93	0.83	0.81	0.88	0.73
NF: 240	0.93	0.89	0.90	0.84	0.89	0.80
NF: 480	0.91	0.88	0.89	0.86	0.86	0.86

(b) Opción II.



	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	0.91	0.83	0.91	0.87	0.94	0.77
NF: 240	0.94	0.91	0.90	0.82	0.93	0.89
NF: 480	0.92	0.88	0.90	0.87	0.87	0.86

(c) Opción III.



	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NF: 120	1.00	0.96	0.94	0.85	0.91	0.86
NF: 240	0.99	0.97	0.94	0.92	0.91	0.87
NF: 480	1.00	0.93	0.93	0.91	0.93	0.85

(d) Opción IV.

Fig. 6. Eficiencia de los algoritmos paralelos. Nº frames 120, 240 y 480.

gramación OpenMP [14].

La prataforma multicore utilizada es un HP Proliant SL390 G7 con dos Intel Xeon X5660, cada CPU con seis núcleos a 2.8 GHz, con lo que los experimentos usan hasta 12 procesos. La secuencia de vídeo utiliada es *BasketballPass_416x240.yuv*, que contiene

Nº procesos	240 frames	480 frames
2	120	240
4	60	120
6	40	80
8	*30*	60
10	24	48
12	20	40

TABLA I
OPCIÓN III: PARÁMETRO *IntraPeriod* PARA IGUALAR EJECUCIONES SECUENCIAL Y PARALELA.

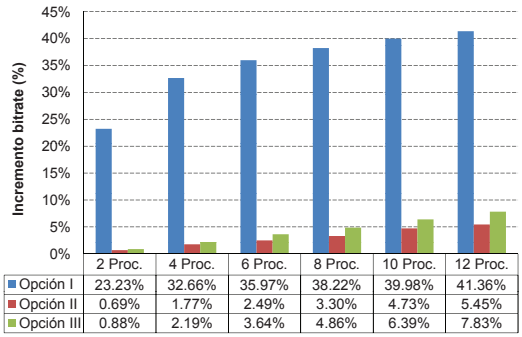


Fig. 7. Porcentaje de incremento de bitrate en los algoritmos paralelos. 480 frames.

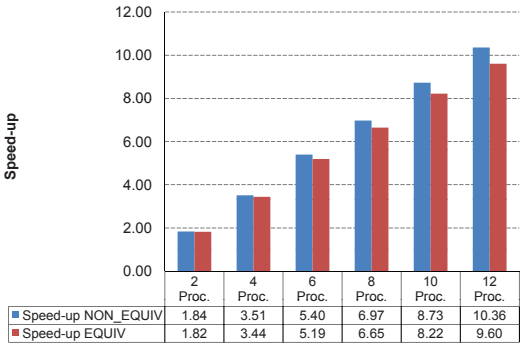


Fig. 8. Speed-up del algoritmo Opción III. 480 frames.

500 frames a $50Hz$ con una resolución de $416x240$ píxels. Hemos lanzado los algoritmos paralelos codificando 120, 240 y 480 frames en modo Low-delay. El valor de cuantización usado (QP) es 32;

En la Figura 5 mostramos los tiempos de cómputo para la Opción I, Opción II, Opción III, y Opción IV. Los resultados muestran un buen comportamiento paralelo en todos los casos. Cuando usamos sólo 1 proceso, todos los algoritmos propuestos muestran el mismo tiempo que la versión secuencial. Con respecto a la Opción IV, la ejecución secuencial de referencia no es la misma, ya que usamos el modo AI.

La Figura 6 muestra la eficiencia asociada a los resultados de la Figura 5. Esto confirma el buen comportamiento de todos los algoritmos paralelos propuestos, obteniendo cerca de eficiencias ideales en algunos casos y por encima de 0.85 en la mayoría de los experimentos realizados. Remarcar que la Opción IV obtiene una eficiencia media superior a 0.95.

Como se comentó en la sección IV, las versiones paralelas no dan el mismo resultado que la versión secuencial. En la Figura 7 mostramos cómo las versiones paralelas modifican el bitrate de la versión secuencial. Es importante decir que la Figura 7 muesta los resultados para la Opción I, II y III, pero no para la Opción IV, porque en este caso la versiones secuencial y paralela muestran el mismo bitrate. También se puede observar que el incremento del bitrate introducido por la Opción I no es aceptable. Este algoritmo cambia drásticamente la estructura de los frames de referencia y como consecuencia incrementa el tamaño del bitstream generado, como se muestra en la Figura 7. En todos los casos, el incremento del bitrate es mayor conforme aumentamos el número de procesos. Finalmente, el incremento de bitrate en la Opción III, se debe a que el frame inicial (I-frame) se codifica con mayor calidad que el B-frame inicial de la Opción II, como se especifica en la configuración del perfil Low-delay.

La Tabla II muestra el PSNR, i.e. una medida de calidad para los algoritmos paralelos II y III. Se puede observar que usando la Opción II la calidad del vídeo codificado decrece aunque en la Figura 7, se mostró que el bitrate aumentaba. Por el contrario, el incremento de bitrate de la Opción III mostrado en la Figura 7 se compensa con un aumento de la calidad como se puede observar en la Tabla II.

Finalmente, modificamos la configuración del perfil Low-delay para obtener el mismo PSNR y bitrate tanto en la versión paralela como en la secuencial del algoritmo Opción III. La Figura 8 muesta como *Speed-up NON_EQUIV* el speed-up cuando los algoritmos secuencial y paralelo obtienen resultados ligeramente diferentes, y como *Speed-up EQUIV* el speed-up cuando dan el mismo resultado. Se puede concluir que la Opción III obtiene buenas eficiencias que van desde 0.80 a 0.91.

VI. CONCLUSIONES

En este artículo hemos propuesto varios algoritmos paralelos para el codificador HEVC. Estos algoritmos se basan en un nivel de paralelización grueso a nivel de GOP. Estos algoritmos están especialmente diseñados para arquitecturas multicore. Tras implementar estos algoritmos en el software de referencia del HEVC en el perfil Low-delay se han realizado varios experimentos que muestran resultados interesantes como (a) La organización del GOP determina el rendimiento final de la codificación, siendo la mejor aproximación la Opción IV (modo AI) en términos de speed-up/eficiencia con respecto a su homólogo secuencial, y (b) Aunque la Opción III introduce un incremento de bitrate conforme aumentamos el número de procesos, el rendimiento paralelo global y las mejoras en PSNR lo convierten en un buen candidato para ser usado en modo LB (Low-delay B).

En general, todas las versiones desarrolladas obtienen buenas eficiencias, mostrando que las aproximaciones de paralelización basadas en GOP deben ser tenidas en cuenta para reducir la complejidad del codificador HEVC. Como trabajo futuro, exploraremos una aproximación jerarquica, combinando técnicas basadas en GOP junto con paralelización a

Algoritmos	1 Proc	2 Proc	4 Proc	6 Proc	8 Proc	10 Proc	12 Proc
Opción II	33.23	33.23	33.20	33.18	33.17	33.16	33.15
Opción III	33.23	**33.26**	**33.31**	**33.35**	**33.39**	**33.44**	**33.47**

TABLA II

PSNRs (dB) de los algoritmos paralelos (Luminancia). 480 frames.

nivel de Slice o de Wavefront.

Agradecimientos

Esta investigación ha sido financiada por el Ministerio de Educación y Ciencia TIN2011-27543-C03-03, por el Ministerio de Ciencia e Innovación TIN2011-26254 y por la Generalitat Valenciana ACOMP/2013/003.

Referencias

[1] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16, 2012*.

[2] J. Ohm, G.J. Sullivan, H. Schwarz, Thiow Keng Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards - including high efficiency video coding (hevc)," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1669–1684, 2012.

[3] B. Bross, W.J. Han, J.R. Ohm, G.J. Sullivan, Y-K Wang, and T. Wiegand, "High efficiency video coding (HEVC) text specification draft 10," *Document JCTVC-L1003 of JCT-VC, Geneva*, January 2013.

[4] G.J. Sullivan, J.R. Ohm, W.J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1648 –1667, December 2012.

[5] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1685–1696, 2012.

[6] M. Alvarez-Mesa, C.C. Chi, B. Juurlink, V. George, and T. Schierl, "Parallel video decoding in the emerging HEVC standard," in *International Conference on Acoustics, Speech, and Signal Processing, Kyoto*, March 2012, pp. 1–17.

[7] E.A. Ayele and S.B.Dhok, "Review of proposed high efficiency video coding (HEVC) standard," *International Journal of Computer Applications*, vol. 59, no. 15, pp. 1–9, 2012.

[8] Qin Yu, Liang Zhao, and Siwei Ma, "Parallel AMVP candidate list construction for HEVC," in *VCIP'12*, 2012, pp. 1–6.

[9] Jie Jiang, Baolong Guo, Wei Mo, and Kefeng Fan, "Block-based parallel intra prediction scheme for HEVC," *Journal of Multimedia*, vol. 7, no. 4, pp. 289 –294, August 2012.

[10] Frank Bossen, "Common test conditions and software reference configurations," Tech. Rep. JCTVC-L1100, Joint Collaborative Team on Video Coding, Geneva, January 2013.

[11] Gisle Bjontegaard, "Improvements of the BD-PSNR model," Tech. Rep. VCEG-M33, Video Coding Experts Group (VCEG), Berlin (Germany), July 2008.

[12] HEVC Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-10.0/, ," .

[13] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 620–636, 2003.

[14] "Openmp application program interface, version 3.1," *OpenMP Architecture Review Board. http://www.openmp.org*, 2011.

K-MEANS CON ORDENACIÓN, ACTUALIZACIÓN Y DESIGUALDAD TRIANGULAR EN GPU

Antonio J. Lázaro-Muñoz[1], Nicolás Guil-Mata[1], José Mª González-Linares[1] y Juan Gómez-Luna[2]

Resumen—**k-means es una técnica ampliamente usada en procesos de agrupamiento no supervisados, que busca dividir un conjunto de *n* puntos en *k* particiones. En este artículo se realiza un estudio de su implementación sobre tarjetas gráficas usando CUDA. Al algoritmo básico que permite realizar el proceso de k-means se le han añadido varias mejoras como la reordenación de los puntos del conjunto o el uso de la desigualdad triangular para acelerar el cálculo de la asignación de puntos a particiones. Para cada mejora se ha calculado, utilizando diferentes conjuntos de datos, su impacto sobre el tiempo de ejecución y se han analizado algunas propiedades como la divergencia, la ocupación o el número de instrucciones lanzadas para justificar los resultados y proponer modificaciones que mejoren el tiempo total de ejecución.**

Palabras clave—**CUDA, GPU; Algoritmos de clustering, k-means, Desigualdad Triangular.**

I. INTRODUCCIÓN

EL *clustering* es un método de aprendizaje no supervisado que es utilizado en diversos campos como análisis de reconocimiento de patrones y bioinformática [1, 2]. Esta técnica divide un conjunto de datos en clusters, de tal manera que minimiza la similaridad entre clusters mientras que maximiza la similaridad entre los datos pertenecientes al mismo cluster [3, 4]. El algoritmo *k-means* [5] es un método de clustering muy popular y ampliamente usado en varias aplicaciones.

El algoritmo *k-means* asigna cada punto o dato al cluster más similar. Para ello se obtiene la distancia mínima entre cada punto y los centroides de todos los clusters, donde dichos centroides se definen como la posición media de todos los puntos pertenecientes al mismo cluster. Este algoritmo es llevado a cabo mediante un proceso iterativo. Inicialmente las posiciones de los centroides de los clusters son inicializadas aleatoriamente o directamente con los *k* primeros puntos del conjunto de datos. Después de que todos los puntos han sido asignados a un cluster, se actualizan los centroides de éstos con respecto a los puntos que pertenecen al mismo. El proceso itera o bien hasta un número de iteraciones previamente fijado o hasta que no se realizan más cambios en los clusters.

En los últimos años, la mejora en el rendimiento del algoritmo *k-means* ha sido objetivo de diversas investigaciones. El tiempo de ejecución en el algoritmo *k-means* incrementa a medida que crece la cantidad de puntos y su dimensionalidad. De esta manera, el proceso de clustering de grandes conjuntos de datos consume un gran tiempo de ejecución. Para solventar los grandes requisitos computacionales que requiere este proceso, su paralelización se ha llevado a cabo en diversas plataformas hardware [6, 7]. Recientemente, las unidades de procesamiento gráficas (en inglés Graphics Processing Units o GPUs) han destacado como grandes candidatas para la computación paralela en diferentes aplicaciones científicas.

Existen diferentes implementaciones del algoritmo de clustering *k-means* para tales procesadores gráficos[8, 9, 10, 11]. En [8] se propone una mejora del algoritmo *k-means* en GPU, en la que se emplea un enfoque centrado en el uso de los registros para datos con baja dimensionalidad, mientras que para datos de mayor dimensionalidad se utiliza un enfoque similar a la multiplicación de matrices que aprovecha el gran ancho de banda de la memoria compartida. En [9], se desarrolla el algoritmo *k-means* en GPU utilizando la propiedad de desigualdad triangular, para reducir el número de cálculos realizados por cada punto, pero presenta una gran divergencia entre los hilos GPU. Para reducir tal divergencia aplica una etapa de monitorización del número de operaciones por punto. Una vez conocida la cantidad de operaciones por punto, ordena éstos con respecto a esta cantidad. En [10] se introducen dos etapas de preprocesamiento para reducir la carga de trabajo y la divergencia de los hilos GPU: una ordenación previa de los datos y una actualización parcial de la ordenación. En [11] se utilizan streams GPU, para acelerar el proceso para una gran cantidad de datos. Los conjuntos de datos se dividen en distintas partes las cuales requieren ser transferidas a la GPU en cada iteración. La aceleración del proceso es conseguida por medio del solapamiento de transferencias de memoria y computación mediante streams GPU.

El paso de asignación de los puntos a los clusters es el paso más crítico del algoritmo. La cantidad de cálculos a realizar por cada punto se incrementa con la cantidad de clusters y la divergencia en los hilos está siempre presente. Por un lado, la cantidad de cálculos realizada por cada hilo se puede reducir aplicando la propiedad de desigualdad triangular [9], como hemos comentado anteriormente. Por otro lado, la divergencia de los hilos presente en el paso de asignación puede ser reducida con una etapa de ordenación de los datos [10].

En este artículo se ha llevado a cabo un análisis del método propuesto en [10]. Posteriormente se han introducido las mejoras de la desigualdad triangular y la utilización de la matriz RID propuestas en [9] y hemos evaluado los resultados. A raíz de dicho análisis hemos modificado el algoritmo para obtener una versión más eficiente.

[1]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: alazaro|nguil|jgl@uma.es
[2]Dpto. de Arquitectura de Computadores, Eléctronica y Tecnología Electrónica, Univ. Córdoba, e-mail: el1goluj@uco.es

El resto del artículo se organiza de la siguiente manera. En la sección II se realiza una breve descripción del algoritmo *k-means* y las mejoras propuestas en este artículo. En la sección III se presentan los resultados experimentales obtenidos para demostrar cómo se complementan la desigualdad triangular y la ordenación de los datos para conseguir un mejor rendimiento. En la sección IV, se realiza una revisión de todo lo discutido en este artículo y se proponen algunos trabajos futuros.

II. ALGORITMO K-MEANS CON ORDENACIÓN, ACTUALIZACIÓN Y DESIGUALDAD TRIANGULAR

El algorimo *k-means* es uno de los algoritmos de *clustering* más populares. Dado un conjunto de puntos de tamaño *n*, $R = \{r_1, r_2, ..., r_n\}$ en un espacio dimensional *d*, el objetivo de *k-means* es particionar R en k clusters $(k < n)$ $S = \{S_1, S_2, S_3, ...S_k\}$ tal que $\sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - \mu_i \|^2$ es minimizado y donde μ_i es el centroide de S_i. El algoritmo *k-means* iterativamente divide un conjunto de datos en k clusters. Primero selecciona como centroides de los clusters los k primeros puntos o un conjunto aleatorio de puntos de tamaño k. A continuación el algoritmo itera de la forma mostrada en el cuadro siguiente.

Algoritmo 1. Pseudocódigo del algoritmo *k-means*.

R: conjunto de datos de tamaño n.
n: número de elementos.
x_i: punto i del conjunto de datos.
$d(i, j)$: distancia del punto x_i al cluster j.
$p(i)$: cluster al que pertenece el punto i.
μ_j: centroide del cluster j.

Para todo $i \in [1, n]$ hacer
$dis \leftarrow \infty$
  Para todo $j \in [1, k]$ hacer
$d(i, j) \leftarrow$ distancia euclídea de x_i a S_j
Si $dis > d(i, j)$ entonces
$dis \leftarrow d(i, j)$
$p(i) \leftarrow j$
    Fin si
  Fin para
Fin para

Para todo $j \in [1, k]$ hacer
$\mu_j \leftarrow$ media de x_i cuyo $p(i) = j$
Fin para

Para llevar a cabo nuestro análisis hemos partido del método desarrollado en [10], cuya implementación se encuentra disponible en su web [12], y le hemos añadido la desigualdad triangular propuesta en [9]. Para la paralelización en GPU de la técnica propuesta por este artículo hemos utilizado el entorno CUDA [13]. En la figura 1 se muestra el diagrama de flujo del método implementado en este artículo.

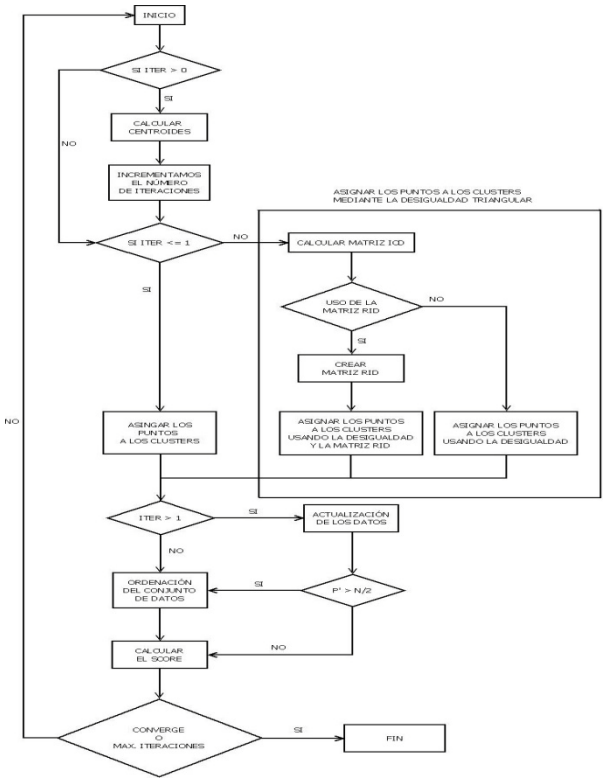


Fig. 1. Diagrama de flujo para el algoritmo *kpsmeans* usando desigualdad triangular y la matriz RID.

A. *Cálculo de los centroides.*

El cálculo de los nuevos centroides requiere obtener la media de las localizaciones de los puntos asignados a cada cluster. Para un vector desordenado es necesario recorrer, para cada uno de los clusters, los n puntos del conjunto de datos. Mediante la ordenación de los puntos en segmentos de datos, donde cada segmento contiene los puntos pertenecientes a un único cluster, sólo se tiene que utilizar el segmento de datos asociado al cluster para calcular su centroide.

La estrategia seguida para la paralelización de este paso en GPU es lanzar una cantidad de bloques de hilos igual al número de clusters k, donde cada bloque de hilos contiene *tpb* hilos (*tpb* hace referencia al número de hilos por bloque). Cada bloque de hilos trabajará en un determinado segmento de datos y sus hilos trabajarán con puntos pertenecientes al mismo cluster, con lo que se reducirá la divergencia y los accesos a memoria global serán de forma coalescente.

Para el cálculo de la media de las localizaciones de los puntos pertenecientes a un cluster, se aplica un proceso de reducción paralela en memoria compartida para sumar las coordenadas de los diferentes puntos.

B. *Asignación de los puntos a los clusters.*

Como hemos comentado anteriormente, el paso de la asignación de los puntos a los clusters es el más crítico.

Este paso en el algoritmo *k-means* standard tiene una complejidad de $O(ndk)$. En este proceso cada punto tiene que calcular su distancia a todos los clusters. La estrategia seguida para la paralelización de este paso en GPU es lanzar una cantidad hilos igual a n hilos organizados en n/tpb bloques.

Una de las modificaciones que incluye este trabajo es el uso de la desigualdad triangular propuesta en [9], con el fin de reducir el número de operaciones realizadas por cada punto. Acelerar el proceso de asignación de los puntos al cluster más cercano, por medio de la propiedad de la desigualdad triangular, se basa en la idea de que algunos cálculos de distancias son redundantes. Por ejemplo, dado un punto P_x y dos clusters con centroides C_i y C_j, cuando asignamos el punto P_x en el algoritmo *k-means* estándar necesitamos calcular dos distancias $d(P_x,C_i)$ y $d(P_x,C_j)$. Sin embargo, podemos calcular primero la distancia $d(P_x,C_i)$, y si se cumple que $d(C_i,C_j) > 2d(P_x,C_i)$, podemos inferir que $d(P_x,C_j) > d(P_x,C_i)$ sin calcular $d(P_x,C_j)$.

Consecuentemente, tenemos que calcular las distancias entre clusters antes de realizar este paso en cada iteración. Esto dará lugar a una matriz de distancias entre clusters, ICD , de tamaño kxk. Para acelerar los cálculos también se hará uso de una segunda matriz, RID, de tamaño kxk donde cada fila es una permutación $1,2,...,k$, representando una lista ordenada de los centroides más cercanos al centroide C_i. Por ejemplo, $RID_{2,3} = 5$ significa que el tercer cluster más cercano a C_2 es C_5.

Mediante el uso de la desigualdad triangular la complejidad computacional de este paso pasaría a ser $O(ndk')$ con $1 \le k' \le k$, donde k' indica la reducción en el número de cálculos realizados por cada punto.

Por lo tanto, el paso de asignación de los puntos a los clusters se desarrollaría de la siguiente manera. Supongamos que en una iteración anterior el punto P_x ha sido asignado al centroide i. Para encontrar el centroide más cercano en la iteración actual, el algoritmo iterará a través de los centroides, según la secuencia almacenada en la fila i-ésima de la matriz RID. Las iteraciones terminarán cuando algún centroide j cumpla la condición de la desigualdad triangular.

Inicialmente hemos seguido una estrategia de paralelización similar a la del código original consistente en lanzar una cantidad hilos igual a n hilos organizados en n/tpb bloques.

C. Ordenación del conjunto de datos.

En esta fase se lleva a cabo la ordenación de los puntos de cada cluster en su correspondiente segmento de datos. Para ello se tienen que realizar los siguientes pasos: 1) Contar los puntos pertenecientes a cada cluster, 2) Determinar los offsets del segmento de datos de cada cluster, mediante la aplicación de la suma de prefijos a los tamaños de los clusters, y 3) Cada bloque de hilos rellena su segmento de datos con los puntos asignados a su cluster correspondiente.

La ordenación completa de los puntos es requerida cuando existen muchos puntos mal asignados a los clusters. Sin embargo, cuando el número de puntos mal asignados a los clusters baja, la ordenación completa de los puntos resulta ineficiente y una actualización de los puntos resultará más eficiente. La actualización de los puntos mal asignados se puede realizar en los siguientes pasos:

1) Contar los puntos mal asignados a cada cluster.

En este paso se cuentan los puntos que están mal asignados a cada cluster. Esta cantidad se restará al tamaño del segmento de datos asociado al cluster. Esto es llevado a cabo mediante k bloques de tpb hilos cada uno. Cada bloque recorre su segmento de datos en S_k / tpb iteraciones, donde S_k es el tamaño del segmento de datos. En este paso se utiliza un proceso de reducción en memoria compartida, para sumar la cantidad de puntos mal asignados en cada iteración.

2) Determinar los k offsets del buffer.

Una vez que conocemos los nuevos tamaños de los segmentos de datos de los clusters, aplicamos un proceso de suma de prefijos para obtener los offsets de los segmentos de datos, que utilizaremos para llevar los datos mal asignados a un buffer temporal.

3) Mover las asignaciones erróneas al buffer.

En este paso se copian en el segmento del buffer en posiciones coalescentes, las asignaciones erróneas en el segmento de datos del cluster. Esto nos permite contar las nuevas asignaciones para cada cluster y modificar los nuevos offsets de los segmentos de datos para la siguiente iteración. Cada bloque de hilos recorre su segmento de datos y su segmento del buffer asociados con tpb hilos. Para copiar las asignaciones erróneas en posiciones coalescentes en el segmento del buffer, los índices de éstas se recogen en memoria compartida mediante un proceso de compactación. Las asignaciones erróneas de cada cluster se irán copiando en el buffer en las posiciones desde $offset_k$ hasta $offset_k + tpb - 1$.

4) Recoger las nuevas asignaciones del buffer.

Todos los bloques de hilos recorren el buffer desde el principio hasta el fin e incrementan los tamaños de los k segmentos de datos con el nuevo número de asignaciones encontradas para el cluster. De esta manera, obtenemos los nuevos tamaños de los segmentos de datos en el vector ordenado de la siguiente iteración.

5) Actualizar los offsets de los segmentos de datos.

Una vez conocidos los nuevos tamaños de los segmentos de datos para la siguiente iteración, aplicamos un proceso de suma de prefijos para obtener

los nuevos offsets de los segmentos de datos. Los nuevos offsets, pueden ocasionar que aparezcan nuevas asignaciones erróneas en algún segmento de datos de algún cluster. Esto es debido a que los nuevos offsets pueden solapar segmentos de datos pertenecientes a otros clusters. Para determinar la cantidad de datos que tienen que ser movidos a otro segmento, cada bloque de hilos tiene que recorrer de nuevo su segmento de datos asociado, contar los datos mal asignados y calcular los nuevos offsets del segmento del buffer.

La suma P' de los tamaños de los segmentos del buffer nos proporciona cuantos datos tienen que ser movidos en total. Si P' es mayor que $n/2$, quiere decir que en el siguiente paso se necesitarán más operaciones de lectura y escritura que en una ordenación completa del vector de datos. Por lo tanto si $P' > n/2$ la fase de actualización termina y se realiza una ordenación completa. De lo contrario, se pasa al siguiente paso.

6) Mover las asignaciones erróneas al buffer.

De nuevo, conociendo los nuevos offsets de los segmentos de datos, movemos las asignaciones erróneas al segmento del buffer.

7) Recoger datos del buffer.

En este paso, cada bloque de hilos realiza dos operaciones. Por un lado, recorre su segmento de datos para recoger los índices de los datos mal asignados al segmento y, por otro lado, recorre el buffer para recoger los índices de los datos asignados al cluster. Como último paso intercambia los datos mal asignados en su segmento de datos con los datos correctamente asignados al cluster, en el segmento del buffer.

D. Calcular el Score.

El score se calcula mediante la suma de todas las distancias de los puntos a sus centroides. La paralelización de este paso se lleva a cabo mediante k bloques con tpb hilos. Cada bloque realiza un proceso de reducción en memoria compartida para el cálculo de su propio score. Los k valores de score son devueltos a la CPU para realizar el proceso final de reducción y obtener el valor de score final.

III. RESULTADOS

El rendimiento obtenido con la implementación propuesta en este trabajo ha sido analizado con varios conjuntos de datos para observar el impacto de las distintas mejoras introducidas. Las pruebas se han realizado en un sistema mutiprocesador compuesto por 4 procesadores Intel® Core ™ 2 QuadQ6600 a 2.4 GHz, con una GPU Nvidia Tesla K20 y un sistema operativo Linux Ubuntu de 64 bits. El código GPU fue compilado y desarrollado con la versión de gcc 4.4.3 y la versión de CUDA 4.3.

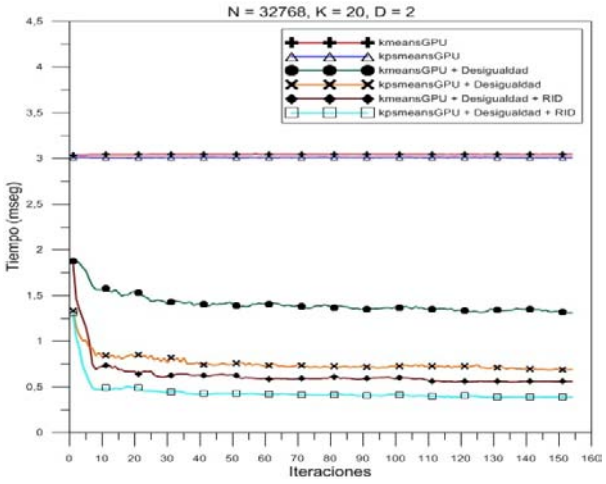


Fig. 2. Tiempo de ejecución por iteración de la asignación de puntos. Estos tiempos corresponde a los algoritmos *k-means* y *kpsmeans*, aplicando en ambos la propiedad de la desigualdad triangular y el uso de la matriz RID.

Para la evaluación de las mejoras se han usado implementaciones de referencia del algoritmo básico (*k-means*) y del algoritmo propuesto en [10] (*kpsmeans*), descargadas de [12], a las que se les ha añadido la desigualdad triangular y el uso de la matriz del clúster más cercano RID.

En la figura 2 podemos comprobar que los tiempos de ejecución para el paso de asignación de puntos, en los algoritmos *k-means* y *kpsmeans* son similares, ya que las mejoras por los pasos de ordenación y actualización de los puntos por clusters que son introducidos en el algoritmo *kpsmeans* no están orientados a mejorar este paso. Sin embargo, los tiempos obtenidos para ambos algoritmos cuando se emplea la desigualdad triangular si mejoran considerablemente. Por una parte, la desigualdad triangular disminuye el tiempo en alrededor de un 55% para la asignación de puntos en el algoritmo *k-means*, obteniendo tiempos de ejecución cercanos a 1.5 milisegundos por iteración. Esta diferencia es más acusada si hablamos del algoritmo *kpsmeans*. La desigualdad triangular ha reducido el tiempo en alrededor de un 80%, obteniendo tiempos de ejecución cercanos a 0.7 milisegundos por iteración.

La desigualdad triangular reduce el número de operaciones a realizar en el paso de asignación de puntos. A medida que las iteraciones aumentan, los puntos tienden a estar mejor asignados, por lo que la desigualdad triangular consume menos tiempo en asignarlos a su cluster más cercano. Las etapas de ordenación y actualización de los datos introducidas por el algoritmo *kpsmeans* aceleran la eficiencia de la desigualdad triangular. Esto es debido a que la ordenación y actualización de los datos hacen que la divergencia introducida por la desigualdad triangular se reduzca y que la carga de trabajo de los hilos en un mismo bloque sea balanceada.

En la tabla I se muestran los valores obtenidos para los valores de ocupación, divergencia e instrucciones lanzadas. Estos valores han sido obtenidos mediante la herramienta CUDA Visual Profiler proporcionada con CUDA 4.3. La ocupación mide la relación entre los warps activos por ciclo y el número de warps soportados por multiprocesador. Valores altos de ocupación indican un uso eficiente de los recursos hardware. La divergencia mide el número de patrones de ejecución distintos tomados por diferentes hilos dentro de un warp. En la tabla I se muestran los valores medios de ocupación, divergencia e instrucciones lanzadas en GPU.

TABLA I

VALORES DE OCUPACIÓN DIVERGENCIA Y NÚMERO DE INSTRUCCIONES LANZADAS EN GPU.

	Ocupación	Divergencia	Instrucciones Lanzadas
kmeans	0.727	17,545	5,017,634
kpsmeans	0.727	4,555	4,962,737
kmeans+Desigualdad Triangular	0.669	21,313	2,311,378
kpsmeans+Desigualdad Triangular	0.696	2,880	1,291,617
kmeans+Desigualdad Triangular+matriz RID	0.611	3,365	1,042,248
kpsmeans+DesigualdadTriangular+matriz RID	0.599	2,072	718,857

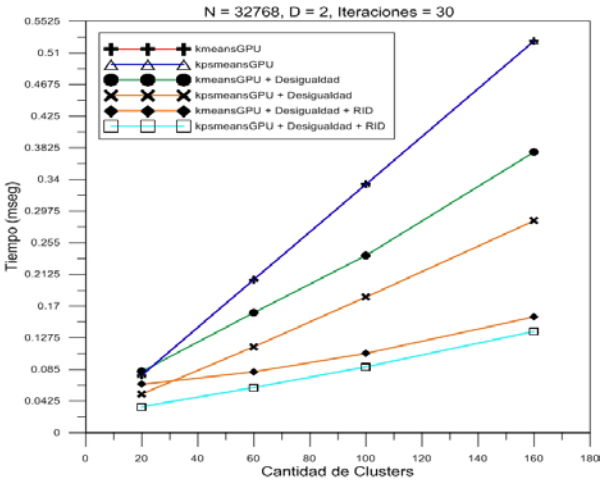


Fig. 3. Valores medios del tiempo de ejecución para el paso de asignación de puntos, cuando incrementa el número de clusters. Estos tiempos corresponde a los algoritmos *k-means* y *kpsmeans*, aplicando en ambos la propiedad de la desigualdad triangular y el uso de la matriz RID.

En la tabla I observamos que los valores de ocupación descienden para los dos algoritmos cuando se introducen las mejoras de la desigualdad triangular y el uso de la matriz RID. Esto es debido a que, tanto la desigualdad triangular como el uso de la matriz RID hacen que los hilos reduzcan su tiempo de computación haciendo que el número de warps activos por ciclo se reduzca. Además, podemos observar en la tabla I que el valor de divergencia para el algoritmo *k-means* cuando aplicamos desigualdad triangular ha aumentado debido a que los datos no están ordenados por clusters, por lo que los valores de divergencia aumentarán. Sin embargo, los valores de divergencia se reducen si se trata del algoritmo *kpsmeans* cuando aplicamos la desigualdad triangular. Este hecho es debido a que el algoritmo *kpsmeans* ordena los datos según su cluster.

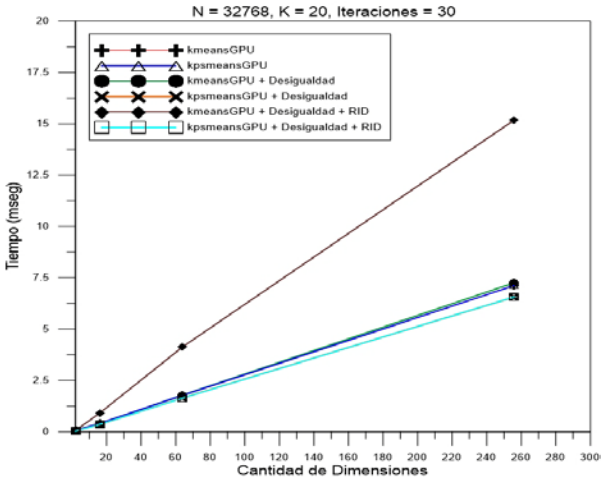


Fig. 4. Valores medios del tiempo de ejecución para el paso de asignación de puntos, cuando incrementa el número de dimensiones. Estos tiempos corresponde a los algoritmos *k-means* y *kpsmeans*, aplicando en ambos la propiedad de la desigualdad triangular y el uso de la matriz RID.

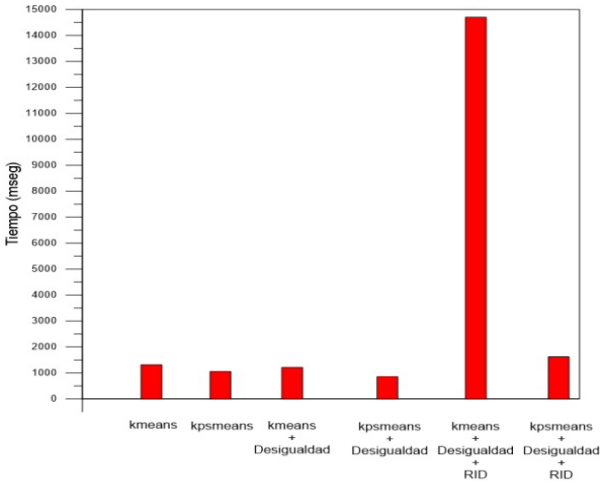


Fig. 5. Tiempos totales de ejecución en GPU para N=32768, K= 20 y D = 2. Estos tiempos corresponden a los algoritmo *k-means* y *kpsmeans* con las correspondientes mejoras de desigualdad triangular y el uso de la matriz RID.

De esta manera, los hilos dentro de un warp pueden trabajar con puntos pertenecientes al mismo cluster. Si todos los hilos dentro de un warp trabajan con puntos de un mismo cluster, entonces evitaremos patrones de ejecución distintos dentro de un warp. El uso de la matriz RID ha reducido de nuevo los valores de divergencia en ambos algoritmos. Esto se debe a que el uso de la matriz RID, como se ha comentado anteriormente, hace que los hilos terminen antes su computación y evita que tomen patrones de ejecución distintos. El número de instrucciones lanzadas también va disminuyendo como podemos observar en la tabla I. Este hecho pone de manifiesto la eficiencia introducida

por la desigualdad triangular, reduciendo el número de operaciones a realizar.

La figura 3 muestra el impacto en el tiempo de ejecución para el paso de la asignación de puntos a los clusters, cuando el número de éstos aumenta. En esta figura podemos observar que las diferencias entre todos los tiempos crecen a medida que el número de clusters aumenta y el número de dimensiones es pequeño. Esta diferencia es más notable cuando se hace uso de la matriz RID. Esto es debido a la reducción del número de operaciones realizada por la desigualdad triangular y el uso de la matriz RID comentada anteriormente y puesta de manifiesto en los valores de la tabla I.

En la figura 4 podemos observar el impacto en el tiempo de ejecución para el paso de la asignación de puntos a los clusters, cuando el número de dimensiones aumenta y el número de clusters es pequeño. En esta figura observamos que el uso de la matriz RID es indiferente e incluso ineficiente para ambos algoritmos. Esto es debido, a la gran cantidad de accesos a memoria global llevados a cabo tanto por la desigualdad triangular, como por el uso de la matriz RID. Para evitar esta ineficiencia se hace necesario buscar un posible reuso de los datos en estos pasos, para poder hacer uso de la memoria compartida y obtener mejor eficiencia a medida que la dimensionalidad del problema aumenta.

La figura 5 muestra los tiempos de ejecución totales en GPU, para los dos algoritmos con las correspondientes mejoras de la desigualdad triangular y el uso de la matriz RID. Estos tiempos son correspondientes a un conjunto de datos N = 32768, K = 20 y D = 2. En esta figura observamos que el tiempo total de ejecución ha incrementado en ambos algoritmos con el uso de la matriz RID. Aunque el uso de la matriz RID en todos los casos aumenta la eficiencia en el paso de la asignación de puntos a los clusters, la ordenación de esta matriz es un proceso que consume bastante tiempo e incrementa a medida que el número de clusters aumenta. Sin embargo, el tiempo total de ejecución desciende si solamente la desigualdad triangular es aplicada, ya que no hace uso de la matriz RID.

IV. CONCLUSIONES

Hemos modificado la versión GPU del algoritmo *k-means* propuesta en [10] introduciendo la propiedad de la desigualdad triangular propuesta en [9]. Hemos demostrado como esta propiedad acelera el paso de la asignación de los puntos a los clusters, reduciendo el número de operaciones a realizar por cada punto.

La etapa tanto de ordenación como de actualización de los puntos realizada en [10] ha propiciado tener mejores resultados en la desigualdad triangular, debido a que se reduce la divergencia de los hilos y elimina la etapa de monitorización del número de operaciones por puntos introducida en [9].

Para la implementación de la propiedad de la desigualdad triangular se han empleado dos matrices denominadas ICD y RID. Por una parte en la matriz RID se tiene que aplicar un proceso de ordenación por filas el cual incrementa su tiempo de ejecución a medida que el número de clusters aumenta. Esto puede consumir la mayor parte del tiempo de ejecución en el paso de la asignación de puntos, e incluso penalizar el tiempo total del algoritmo. Por otra parte, el uso de la matriz RID resulta indiferente cuando la dimensionalidad del problema aumenta.

En el tiempo total de ejecución habría que tener en cuenta un proceso de preprocesamiento de los conjuntos de datos. Los conjuntos de datos proporcionados para las diferentes pruebas ya han sido traspuestos y modificados para su lectura y almacenamiento eficiente en GPU. Esta situación no se da en la realidad por lo que, como trabajos futuros, se propone en primer lugar la inclusión de un paso de transposición de los datos en la GPU. En segundo lugar, se propone la realización de una implementación más eficiente de la ordenación de la matriz RID, cuyo tiempo de ejecución se vea débilmente afectado a medida que el número de clusters aumenta. Por último, se propone el estudio de una mejora en la implementación de la desigualdad triangular y el uso de la matriz RID, para hacer uso de la memoria compartida.

REFERENCIAS

[1] X. Wand, M. Leeser, *K-means Clustering for Multispectral Images Using Floating-Point Divide*, in Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2007.

[2] H. Zhou, Y. Liu, *Accurate Integration of Multi-view Range Images Using K-means Clustering*, Pattern Recogn., vol. 41, pp. 152-175, 2008.

[3] A. K. Jain, R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, 1988.

[4] P.-N. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining*, Addison-Wesley Companion Book Site, 2006.

[5] S. J. Phillips, *Acceleration of k-means and related clusteringalgorithms*, in ALENEX '02: Revised Papers from the 4th International Workshop on Algorithm Engineering and Experiments. London, Springer-Verlag, pp. 166-177, 2002.

[6] D. Judd, P. K. McKinley, A. K. Jain, *Large-Scale Parallel Data Clustering*, in Proccedings of the International Conference on Pattern Recognition (ICPR'96), vol. 4, 1996.

[7] I. S Dhillon, D. S. Modha, *A Data-Clustering Algorithm onDistributed Memory Multiprocessors*, in Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, Springer-Verlag, 2000.

[8] Y. Li, K. Zhao, X. Chu, J. Liu, *Speeding up K-means Algorithm by GPUs*, 10thIEEE International Conference on Computer and Information Technology (CIT 2010), 2010.

[9] J. Wu, B. Hong, *An Efficient k-means Algorithm on CUDA*, IEEE International Parallel & Distributed Proccesing Symposium, pp. 1740 – 1749, 2010.

[10] K. J. Kohlhoff, V. Pande, R. Atman, *K-means for parallelarchitectures using all-prefix-sumsorting and updating steps*, IEEE Parallel and Distributed Systems, vol. PP, pp. 1, 2012.

[11] R. Wu, B. Zhang, M. Hsu, *Clustering Billions of Data Points Usings GPUs*, Proceedings of the combined workshops on Un Conventional high performance computing workshop plusmemory access workshop, pp. 1-6, 2009.

[12] Clustering Algorithms for Massively Parallel Architectures Including GPU Nodes. https://simtk.org/home/campaign.

[13] NVIDIACUDA: Programming Guide, Version 5.0http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. October 2012.

Simulación paralela de dinámicas eco-evolutivas siguiendo un modelo basado en individuos

G. Barrionuevo-Rosales[1] , José Román Bilbao-Castro[2] , Jordi Moya-Laraño[3] y L.G. Casado[4]

Resumen— **En este artículo se define la paralelización de un Individual-Based Model (IBM) que sintetiza las dinámicas eco-evolutivas de un ecosistema en el que conviven varias especies que se alimentan unas de otras (red trófica). La incorporación del paralelismo es fundamental para obtener mejores resultados en estos modelos, siendo necesaria la búsqueda de métodos que proporcionen una mayor eficiencia sin modificar los conceptos básicos del ecosistema. La implementación de este tipo de simulaciones proporciona una herramienta útil que, combinada con los estudios de campo, permite el análisis minucioso de las relaciones entre la genética y las dinámicas de poblaciones de seres vivos. En la versión final de este artículo se muestra el modelo utilizado en la paralelización y los resultados sobre la escalabilidad del algoritmo.**

Palabras clave— **Modelo basado en Individuos, Paralelización, Eco-evolución, Genética, Individuo.**

I. Introducción

La evolución determina en gran medida los rasgos que caracterizan a los seres vivos. Esta información está incluida en su genética, que ha sido heredada generación tras generación sufriendo ciertas modificaciones debidas al conocido proceso evolutivo, sea este adaptativo o azaroso [1].

La ecología, por su parte, estudia otro tipo de propiedades que guardan más relación con el ecosistema que con el propio individuo. Esta ciencia se centra en la definición y descripción del ambiente, la observación y predicción de la cantidad y distribución de los animales en el espacio, así como en cómo afecta a estos elementos el comportamiento de los individuos y las interacciones que ocurren entre ellos [2].

La ecología evolutiva abarca la estrecha relación entre ambas ciencias [1], y en particular, el recientemente en auge estudio de las dinámicas eco-evolutivas es aquel que tiene como fin documentar cómo afectan las presiones de selección (factores ecológicos) a los cambios en las frecuencias génicas y cómo dichos cambios revierten recíprocamente en la dinámica ecológica de las poblaciones [3]. Sin embargo, esta tarea se vuelve muy compleja a la hora de llevar a cabo experimentos de campo, ya que para obtener algún resultado es necesario observar un número muy grande de individuos durante un intervalo de tiempo muy amplio, así como medir demasiados detalles que escapan a lo humanamente factible.

La experimentación que aquí se expone hace uso de un modelo basado en individuos (en adelante IBM: *Individual-Based Model*). Un IBM [4] es un modelo computacional que simula las acciones e interacciones de agentes autónomos, los cuales pueden estar englobados dentro de distintas especies, con el objetivo de evaluar los efectos generales que se producen en el sistema.

Haciendo uso de las investigaciones realizadas y los resultados obtenidos en biología [5], se va a definir e implementar un IBM que represente un ecosistema real (el suelo de un bosque caducifolio) lo más fielmente posible, donde convivan diferentes especies de animales haciendo uso de los recursos basales (hongos) disponibles. La implementación servirá para observar patrones en la eco-evolución animal de modo que se puedan contrastar con las investigaciones de campo.

En la sección II se explican los aspectos simulados en la experimentación. En la sección III se describe la versión secuencial del algoritmo, explicando los diferentes pasos que la componen. En la sección IV se muestran los resultados obtenidos en la ejecución secuencial. En la sección V se presentan las estrategias de paralelización de la versión secuencial. En la sección VI se mostrarán los resultados sobre la escalabilidad de la versión paralela. Finalmente en la sección VII se muestran las conclusiones y trabajos futuros.

II. Aspectos simulados en la experimentación

En la simulación existen una serie de conceptos que es necesario describir para entender la complejidad del problema. En esta sección se van a exponer y son los siguientes: creación de la genética y rasgos de los seres, desarrollo de los recursos basales, movimiento e interacciones entre individuos, gestión del metabolismo y reproducción.

A. Genética y rasgos

Dentro de este IBM los agentes se dividen en dos especies: los depredadores y las presas. Ambos comparten los mismos rasgos pero con diferentes valores y límites, y un comportamiento distinto. Cada individuo, independientemente de la especie a la que pertenezca, está compuesto por trece rasgos que son definidos a partir de trece pares de cromosomas.

[1]Grupo de investigación TIC146: Supercomputación-Algoritmos, Univ. Almería, e-mail: gbarrionuevo@ual.es.
[2]Grupo de investigación TIC146: Supercomputación-Algoritmos, Univ. Almería, e-mail: jrbcast@gmail.com.
[3]Estación Experimental de Zonas áridas, CSIC, e-mail: jordi@eeza.csic.es.
[4]Dpto. de Informática, Univ. Almería, e-mail: leo@ual.es.

Cada cromosoma está compuesto por veinte alelos que, sumados junto a los de su cromosoma par correspondiente, generan el valor inicial asociado al rasgo que se esté considerando. Sin embargo, los rasgos pueden estar correlacionados entre sí en mayor o menor medida, positiva o negativamente, con lo cual existe una dependencia directa entre ellos. Los trece rasgos característicos de cada individuo son: *tanque-de-energía*, *crecimiento*, *fenología*, *tamaño*, *asimilación*, *voracidad*, *velocidad*, *área-de-búsqueda*, *ratio-de-metabolismo*, *voracidad-en-Q10*, *velocidad-en-Q10* y *área-de-búsqueda-en-Q10*. Los tres rasgos en *Q10* determinan el cambio que sufren cada uno de ellos al variar la temperatura del entorno 10 ºC.

Además, cada ser vivo almacena una serie de variables de estado que deben ser actualizadas cada día a medida que se producen las interacciones con el entorno y con otros individuos.

B. Recursos basales

Por simplicidad, el espacio en el que se desarrolla este ecosistema está representado en una sola dimensión. Dicho espacio está dividido en celdas contiguas, y se distribuye de forma que la celda inicial se conecta con la final, formando un anillo.

En cada una de las celdas existe una cantidad determinada de recursos basales. Esta propiedad es inherente a las celdas y su comportamiento simula el crecimiento de una variedad de hongos del suelo. Al comienzo de la simulación su cantidad es inicializada en cada celda, siguiendo un patrón establecido. Los hongos crecen cada día dependiendo de la capacidad total de la celda y de la cantidad actual de hongo [6].

Estos recursos basales son consumidos por una de las dos especies: las presas. Esto provoca que la cantidad de hongo varíe cada día pudiendo ocurrir que llegue a extinguirse dentro de la celda e incluso en todo el espacio. Para evitar este último suceso, se ha establecido un umbral a partir del cual las esporas se propagan hacia la celda colindante que contenga menor cantidad de recursos basales.

C. Movimiento e interacciones

El movimiento y las interacciones de unos seres vivos con otros es la parte más compleja del modelo. Para simplificar, se ha establecido que en cada celda pueden estar situados en el mismo momento cualquier número de individuos de cualquier especie. Estos individuos se mueven por las celdas caminando un número de pasos que depende del rasgo *área-de-búsqueda*. Cada individuo solo tiene percepción de la celda actual y las dos colindantes. En caso de no encontrar alimento en estas celdas (ya sean hongos u otros individuos para cazar), el individuo puede realizar un salto aleatorio cuya longitud máxima está definida por el valor de *área-de-búsqueda*. Este procedimiento es común para todos los seres vivos. Sin embargo, cada especie se mueve atendiendo a una serie de variables diferentes:

Presas: Se mueven por el espacio buscando las celdas que contienen más hongos. En la presencia de depredadores la decisión no es tan básica, ya que también se considera disminuir el riesgo de depredación.

Depredadores: Caminan hacia las celdas que contienen un mayor número de presas. También consideran el número de depredadores existentes en una celda, ya que pueden cazar y ser cazados por otros individuos de su misma especie. Al igual que las presas, pueden quedar expuestos a los demás depredadores en cada movimiento.

Los encuentros entre depredador y presa (o bien depredador y depredador) tienen lugar cuando ambos se encuentran en la misma celda y cuando se active una variable aleatoria que depende del *tamaño* y la *voracidad* de ambos seres vivos: el cazador y el cazado. Si el encuentro se produce, entonces la víctima tendrá la oportunidad de escapar, dependiendo de su *voracidad*, *tamaño*, y *velocidad*. Durante el proceso se actualizan las variables de estado que almacenan el número de encuentros por día y los globales. De esta forma se lleva un registro minucioso de las interacciones entre todos los individuos.

Por último, según se ha documentado en animales caníbales [7], si la víctima es de la especie depredadora y escapa satisfactoriamente, puede llevar a cabo un contraataque si se siente segura, es decir, solamente contraataca cuando es atacada por un depredador externo que invade su territorio.

D. Metabolismo

Las dos especies de seres vivos presentes en la simulación deben metabolizar el alimento ingerido, convirtiendo éste en *tanque-de-energía* y, llegado a un determinado umbral, este *tanque-de-energía* se utiliza para mudar y crecer en *tamaño*. Una vez alcanzado un determinado número de mudas y recursos (comida), el individuo está listo para reproducirse, y buscará para ello algún individuo con el que aparearse ese mismo día.

Sin embargo el metabolismo va más allá, pues no sólo hay que considerar el crecimiento sino también el gasto del *tanque-de-energía*. Este gasto se debe a tres razones principalmente: el mantenimiento propio del individuo (metabolismo basal), la distancia que haya recorrido durante el día y el número de encuentros con depredadores que haya sufrido. Si el balance del gasto acumulado a lo largo de los días produce que el *tanque-de-energía* alcance un mínimo crítico, el animal morirá por falta de sustento.

E. Reproducción

En la simulación hay que tener en cuenta la reproducción como un suceso que hará escalar enormemente la cantidad de información y de procesamiento requeridos. Como ya hemos dicho antes, todos los individuos que alcanzan un determinado estado pasan a estar preparados para reproducirse. Además, si consiguen reproducirse varias veces perecerán por causa de senescencia reproductiva (envejecimiento por el coste de la reproducción).

El proceso de la reproducción incluye la mezcla de genes mediante el proceso de la meiosis y la recombinación. De esta forma, la evolución de la genética en los individuos es estudiada a lo largo de la simulación con el fin de entender aspectos relevantes y novedosos desde el punto de vista eco-evolutivo.

Cuando un individuo está listo para reproducirse, simplemente busca aleatoriamente otro compañero que también lo esté. Dado que los individuos son hermafroditas recíprocos (tienen los dos sexos), el proceso se realiza dos veces intercambiando el orden de los actores, para conservar así las distintas combinaciones del material genético. Este proceso genera nueva descendencia que en algunas ocasiones puede superar la decena de nuevos individuos en cada evento reproductivo.

III. Descripción del proceso secuencial

Hasta este punto nos hemos centrado en explicar los conceptos que se utilizan en cada etapa. Estos conceptos se han explicado en un orden lógico para facilitar la comprensión del proceso. El Algoritmo 1 muestra el pseudocódigo secuencial.

Algoritmo 1 IBM secuencial

1. Inicializar celdas *También los hongos*
2. Inicializar animales *Genes y primera generación*
   for $dia = 0$ **to** $DIAS$
3. Activar animales *Nacen si están preparados*
4. Crecer hongos *En todas las celdas*
5. Propagar hongos *Si hay exceso*
6. Tunear rasgos *Depende de los encuentros*
7. **if** $depredadores = true$
8. Mover presas con depredadores
9. **else**
10. Mover presas sin depredadores
11. Alimentar presas *Comen hongos*
12. Metabolizar y crecer presas *Depende del alimento y la distancia*
13. **if** $depredadores = true$
14. Morir depredadores por ambiente *Ayuda a la estabilidad*
15. Mover depredadores
16. Asimilar alimento depredadores
17. Metabolizar y crecer depredadores *Depende del alimento y la distancia*
18. Reproducir animales *Ambos presas y depredadores preparados*

Como podemos ver en los pasos 1 y 2 del algoritmo, en primer lugar se crea el mundo inicializando las celdas y los animales. Este proceso establece una cantidad de hongo para cada celda y genera los cromosomas para cada individuo, analizándolos y extrayendo de ellos los rasgos característicos, y colocando cada ser vivo en una celda aleatoriamente.

La simulación entonces comienza desde el día 0 hasta el número de días máximo indicado, repitiendo los pasos desde el 3 al 18. Cada día empieza activando los individuos que están preparados para nacer, en el paso 3. Entonces se ejecuta el algoritmo de crecimiento y propagación del hongo (pasos 4 y 5, respectivamente).

Antes de realizar el movimiento, en el paso 6 se modifican algunos rasgos de cada individuo basándose en los encuentros que ha podido sufrir en el día anterior. En los pasos 8 y 10 las presas caminarán hacia las celdas más convenientes, considerando depredadores, si éstos existen en la simulación, y quedando expuestos a ser devorados por ellos en su caso.

Las presas concluyen entonces su actividad alimentándose en el paso 11 y activando su metabolismo en el paso 12. Aún no se reproducirán pues pueden ser devorados por los depredadores en los siguientes pasos.

Los depredadores pueden morir aleatoriamente por causa del ambiente en el paso 14. Esta característica se incluye para ayudar a la estabilidad del sistema. Si sobreviven, se mueven y cazan en el paso 12 asimilando el alimento ingerido en el paso 15. Su metabolismo se activa en el paso 17.

Por último los individuos preparados de ambas especies se reproducen buscando aleatoriamente un compañero. Este proceso se realiza al final, en el paso 18, para asegurar que los individuos implicados son sólo aquellos que han sobrevivido al día actual.

IV. Resultados del proceso secuencial

Se ha llevado a cabo la ejecución de varias simulaciones con el objetivo de medir la eficiencia del algoritmo secuencial para distintos parámetros. El experimento se ha realizado sobre un procesador *Intel(R) Core(TM) i7 860* con 2.80GHz y 8GB de memoria RAM.

En la tabla 1 se puede ver el tiempo de ejecución obtenido para diferentes configuraciones, donde P indica el número de presas iniciales, D es el número de depredadores y C es el número de celdas del mundo. Las tiempos de ejecución se representan en segundos y aparecen en las columnas T_{100}, T_{200} y T_{300} que corresponden a la simulación de 100, 200 y 300 días, respectivamente.

TABLA I
Resultados del proceso secuencial. Tiempos de ejecución según los parámetros indicados.

P	D	C	T_{100}	T_{200}	T_{300}
500	50	100	9.013	32.966	61.904
1000	100	200	19.965	64.372	142.745
2000	200	400	37.134	121.607	228.322
4000	400	800	70.756	260.857	515.189

Los resultados obtenidos indican que la complejidad del sistema crece linealmente al incrementar el número de individuos y el espacio de la simulación. Este es un gran inconveniente para obtener resultados interesantes desde el punto de vista ecológico, ya que para ello se necesitará aumentar los valores de los parámetros iniciales. Además, la existencia de otros parámetros configurables hace que el número de simulaciones a realizar pueda ser muy elevado.

V. Propuesta de paralelización

Los resultados de la sección III muestran que este IBM presenta una complejidad computacional que se incrementa con el número de especies, el tamaño de la escena y el número de días de la simulación. Con la paralelización del código secuencial se pretende abordar problemas más reales donde los valores de los parámetros son más elevados que los usados en la versión secuencial. Por ejemplo, la selección natural puede actuar de manera débil y sobre una proporción minoritaria de los individuos, con lo que sólo el seguimiento de miles de individuos durante decenas o centenares de generaciones puede permitir vislumbrar los efectos evolutivos.

Primero identificaremos las limitaciones del paralelismo que aparecen en la versión secuencial y después propondremos una versión paralela del mismo.

A. Limitaciones del paralelismo en la versión secuencial

En todo IBM existen puntos de sincronización que limitan el paralelismo [8]. En este caso se debe actualizar toda la información referente al mismo día y todo el espacio antes de continuar con el día siguiente. Además, debe de finalizar cada etapa antes de continuar con la siguiente:

- Finalizar el crecimiento de los recursos basales, antes de comenzar con el movimiento y las interacciones.
- Finalizar todas las interacciones antes de calcular el metabolismo de los individuos, ya que éste depende del número de encuentros que ha experimentado el sujeto.
- Es necesario terminar la etapa anterior antes de realizar la reproducción, para saber cuántos seres vivos están preparados para llevarla a cabo.

B. Algoritmo paralelo propuesto

La propuesta para paralelizar este IBM se basa en una implementación híbrida que incluye el uso de hilos y paso de mensajes [9] [10]. El algoritmo se centra en una descomposición de datos basada en el área de simulación, que divide el espacio en varias zonas y las asigna a diferentes unidades de procesamiento [11]. Dentro de cada zona se opera mediante hilos, y las zonas se comunican entre sí mediante paso de mensajes. Sin embargo, debido a la naturaleza del problema, es necesario elegir cuidadosamente cómo separar estas zonas, de forma que se produzcan las menores inconsistencias en las fronteras de las sub-áreas seleccionadas. Estas fronteras se comunican con las de los demás procesos mediante el uso de paso de mensajes, ya que éste método permite una mayor escalabilidad en el número de procesadores a usar.

VI. Resultados del proceso paralelo

El proceso paralelo que hemos descrito anteriormente está enfocado a obtener mejores tiempos en las ejecuciones. En este punto aún no se dispone de una versión final paralela, pero se está en fase de desarrollo y se mostrarán los resultados en la versión definitiva del artículo.

VII. Conclusiones y trabajo futuro

El algoritmo secuencial que se ha planteado en este artículo muestra la complejidad de este tipo de simulaciones, donde se pretende encontrar los parámetros responsables de mantener el equilibrio del ecosistema simulado. Se puede concluir que la versión paralelizada reducirá el tiempo de cómputo de la solución, permitiendo realizar experimentos de IBMs más complejos y que tendrán un alto alcance en ecología tanto a medio como largo plazo, contribuyendo al avance del desarrollo teórico de esta ciencia.

Como trabajo futuro se propone estudiar nuevos métodos de balanceo de carga que puedan aportar un mayor beneficio [12] [13]. Se pretende incluir una mayor funcionalidad al modelo gracias al paralelismo: en un ecosistema real toda la dinámica e interacciones se realizan en paralelo. Además, se propone aprovechar las características de la computación sobre GPUs [14] y las arquitecturas heterogéneas para intentar obtener el mayor beneficio de las tecnologías disponibles en la actualidad.

Desde el punto de vista eco-evolutivo o funcional, se propone implementar espacios más realistas ya sea en dos o tres dimensiones, incorporar la humedad del ambiente e incluso analizar el comportamiento de las poblaciones en entornos fijados. Para ello puede ser útil la implementación de interfaces gráficas que ayuden al usuario y la incorporación de herramientas estadísticas para facilitar el estudio.

Agradecimientos

Este trabajo ha sido subvencionado por el Ministerio de Ciencia e Innovación (TIN2008-01117, TIN2012-37483-C03-03 y CGL2010-18602) y la Junta de Andalucía (P011-TIC7176) y financiado en parte por el Fondo Europeo de Desarrollo Regional (ERDF). J. R. Bilbao-Castro es un investigador Ramón y Cajal financiado por el Fondo Europeo Social y el Ministerio de Economía y Competitividad.

Referencias

[1] Douglas J. Futuyma, *Evolutionary Biology*, Sinauer, 3d ed. Sunderland, MA, 1998 1998.

[2] M. Begon, C.R. Townsend, and J.L. Harper, *Ecology: From Individuals to Ecosystems*, Wiley, 2009.

[3] F. Pelletier, D. Garant, and A. P. Hendry, "Eco-evolutionary dynamics," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1523, pp. 1483–1489, 2009.

[4] Donald L. DeAngelis and Wolf M. Mooij, "Individual-based modeling of ecological and evolutionary processes," *Annual Review of Ecology, Evolution, and Systematics*, vol. 36, no. 1, pp. 147–168, 2005, doi:10.1146/annurev.ecolsys.36.102003.152644.

[5] J. Moya-Laraño, O. Verdeny-Vilalta, J. Rowntree, N. Melguizo-Ruiz, M. Montserrat, and P. Laiolo, "Climate change and eco-evolutionary dynamics in food webs," *Advances in Ecological Research*, vol. 47, pp. 1–80, 2012.

[6] V. M. Savage, J. F. Gillooly, J. H. Brown, G. B. West, and E. L. Charnov, "Effects of body size and temperature on population growth," *American Naturalist*, vol. 163, no. 3, pp. 429–441, 2004.

[7] Quart Dong and Gary A Polis, "The dynamics of cannibalistic populations: a foraging perspective," in *Cannibalism: ecology and evolution among diverse taxa*, Cres-

pi BJ Elgar MA, Ed., pp. 13–37. Oxford University Press, New York, NY, USA, 1992.

[8] Dali Wang, Eric A. Carr, Louis J. Gross, and Michael W. Berry, "Design and implementation of a parallel fish model for south florida," in *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9 - Volume 9*, Washington, DC, USA, 2004, HICSS '04, pp. 90282.3–, IEEE Computer Society.

[9] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell, *PThreads Programming: A POSIX Standard for Better Multiprocessing*, O'Reilly, 1998.

[10] P.S. Pacheco, *Parallel Programming with MPI*, Morgan Kauffman Publishers, San Francisco, CA, 1997.

[11] Dali Wang, Michael W. Berry, and Louis J. Gross, "On parallelization of a spatially-explicit structured ecological model for integrated ecosystem simulation," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 571–581, 2006.

[12] Fengyun Lu, Simon Parkin, and Graham Morgan, "Load balancing for massively multiplayer online games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, New York, NY, USA, 2006, NetGames '06, ACM, doi:10.1145/1230040.1230064.

[13] Jungyoul Lim, Jaeyong Chung, Jinryong Kim, and Kwanghyun Shim, "A dynamic load balancing for massive multiplayer online game server," in *ICEC*, 2006, pp. 239–249.

[14] Wenwu Tang and David A. Bennett, "Reprint of: Parallel agent-based modeling of spatial opinion diffusion accelerated using graphics processing units," *Ecological Modelling*, vol. 229, no. 0, pp. 108 – 118, 2012, doi:10.1016/j.ecolmodel.2012.02.003.

Análisis del impacto de la jerarquía de memoria en clusters de multicores utilizando contadores de hardware.

Fabiana Leibovich[1], Laura De Giusti[1], Marcelo Naiouf[1], Franco Chichizola[1], Fernando G. Tinetti[1,2], Armando De Giusti[1,3]

Resumen—**El objetivo principal de la programación híbrida es aprovechar la jerarquía de memoria que las arquitecturas cluster de multicore proveen. En este sentido, partiendo de un mismo caso de estudio (multiplicación de matrices), este trabajo se enfoca en la comparación de dos soluciones híbridas (donde se combina pasaje de mensajes y memoria compartida) que utilizan diferentes estrategias de paralelización. Las mismas utilizan de distinta manera la jerarquía de memoria presente en la arquitectura de experimentación (cluster de multicore), en particular haciendo un uso diferente del nivel L1 de cache. En función de los tiempos de ejecución obtenidos y utilizando contadores de hardware se busca verificar que el factores de mayor incidencia sobre el tiempo de ejecución es la optimización del acceso a cache L1.**

Palabras clave— **arquitecturas paralelas, programación híbrida, cluster, multicore, pasaje de mensajes, memoria compartida, perf, contadores de hardware.**

I. INTRODUCCIÓN

DENTRO de los últimos grandes avances en las arquitecturas paralelas se encuentran los clusters de multicore [1]. Los mismos consisten en un conjunto de procesadores de múltiples núcleos interconectados mediante una red, en la que trabajan cooperativamente como un único recurso de cómputo. El esquema es el de un cluster tradicional pero donde cada nodo posee al menos un procesador multicore en lugar de un monoprocesador, tal como muestra la Figura 1. Esto permite combinar las características más distintivas de ambas arquitecturas (memoria distribuida y compartida), dando origen a los sistemas híbridos [2].

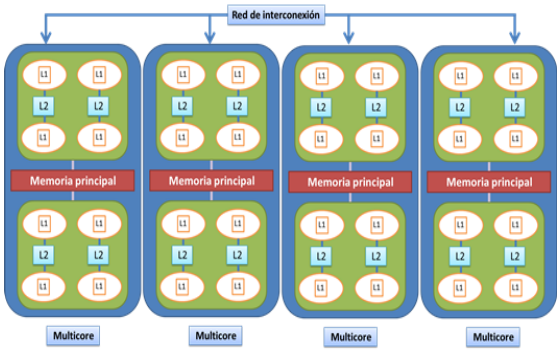


Fig. 1. Cluster de multicore.

Al diseñar un algoritmo paralelo es muy importante considerar la jerarquía de memoria con la que se cuenta, ya que es uno de los factores que incidirá directamente en la performance alcanzable del mismo. Los clusters de multicore introducen un nivel más en la jerarquía de memoria si se lo compara con los multicore: la memoria distribuida accesible vía red que permite la interconexión de los diferentes procesadores que conforman el cluster. Si se enumera la jerarquía de memoria, la misma queda conformada de la siguiente manera: niveles de registros y caché L1 propio de cada núcleo, cache compartida de a pares de núcleos (L2), memoria compartida entre los cores de un procesador multicore y finalmente memoria distribuida vía red [3]. Esta jerarquía puede ampliarse según los modelos de procesadores utilizados. Actualmente es frecuente encontrar multicore que utilizan un tercer nivel de cache (L3) [4].

Este artículo es una evolución de trabajos previos, en los que se utilizó como caso de estudio la multiplicación de matrices comparando en primera instancia la programación híbrida con la de pasaje de mensajes utilizando para ello diferentes estrategias de implementación [5][6][7][8][9]. Dados los resultados obtenidos, la investigación se enfocó en la comparación de soluciones híbridas que aprovechan de manera diferente la jerarquía de memoria disponible en la arquitectura utilizada en la experimentación (cluster de multicore).

El artículo está organizado de la siguiente manera: en la Sección II se detallan los objetivos y aportes del trabajo, mientras que la Sección III describe los fundamentos teóricos de la jerarquía de memoria y las características de la programación híbrida. En la sección IV se presenta la librería perf, utilizada para acceder a la información que almacenan los contadores de hardware mientras que en la Sección V se detalla el caso de estudio y las soluciones implementadas. La arquitectura utilizada y los resultados se muestran en la Sección VI. Por último, en la Sección VII se exponen las conclusiones y líneas de investigación futuras.

II. OBJETIVOS Y APORTES DEL TRABAJO

Como ya se mencionó, este trabajo es un avance de una investigación previa [9] en la que se compararon dos soluciones híbridas donde una de ellas aprovecha la localidad espacial y temporal de la cache L1 mientras que la otra no. En función de los tiempos de ejecución y la eficiencia obtenida, pudimos comprobar que la solución híbrida que aprovecha la jerarquía de memoria arroja mejores resultados que la solución que no lo hace.

En este trabajo se busca corroborar empíricamente esa conclusión desde otro punto de vista. Para ello se emplearon contadores de hardware que aportan información acerca de los eventos relacionados a la utilización de la memoria cache.

[1]Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática, Universidad Nacional de La Plata, 50 y 120 2do piso, La Plata, Argentina. {fleibovich, ldgiusti, mnaiouf, francoch, fernando, degiusti}@lidi.info.unlp.edu.ar
[2]Investigador CIC Prov. de Bs. As.
[3]Investigador CONICET

III. JERARQUÍA DE MEMORIA Y PROGRAMACIÓN HÍBRIDA

La performance de la jerarquía de memoria está determinada por dos parámetros de hardware: latencia (tiempo entre que un dato es requerido y está disponible) y ancho de banda: velocidad con la que los datos son enviados de la memoria al procesador. Tal como muestra la Figura 2, a medida que aumenta la capacidad se decrementa la velocidad pero asimismo disminuye el costo por bit.

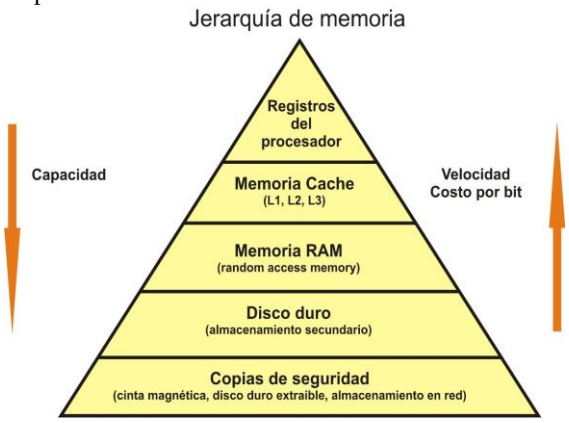


Fig. 2. Jerarquía de memoria tradicional

A continuación se describen los conceptos de localidad espacial y temporal de la cache. Luego se describe la programación híbrida como método de aprovechamiento de la arquitectura.

A. *Localidad temporal y espacial de la cache*

El concepto de localidad temporal hace referencia a que la cache mantiene los datos recientemente accedidos. Es decir que si se aprovecha el acceso a los datos teniendo en cuenta este factor, la latencia en el acceso a los mismos disminuirá.

Por otro lado, el concepto de localidad espacial hace referencia a que cada vez que un dato es llevado a cache, se obtiene una línea de la misma; de esta forma, si se accede a datos contiguos, también se disminuye la latencia en el acceso a los datos (en una sola lectura a memoria, los datos contiguos estarán ya en memoria cache) [10].

B. *Programación híbrida*

La programación híbrida combina la memoria compartida con el pasaje de mensajes [2][10], aprovechando las características propias de cada modelo de programación paralela. La comunicación entre procesos que pertenecen al mismo procesador físico puede realizarse utilizando memoria compartida (nivel micro), mientras que la comunicación entre procesadores físicos (nivel macro) se suele realizar por medio de pasaje de mensajes (memoria distribuida).

En el primer caso, los datos accedidos por la aplicación se encuentran en una memoria global accesible por los procesadores paralelos. Esto significa que cada procesador puede buscar y almacenar datos de cualquier posición de memoria independientemente uno de otro. Se caracteriza por la necesidad de la sincronización para preservar la integridad de las estructuras de datos compartidas.

En el pasaje de mensajes, los datos son vistos como asociados a un proceso particular. De esta manera, se necesita de la comunicación por mensajes entre ellos para acceder a un dato remoto. En este modelo, las primitivas de envío y recepción son las encargadas de manejar la sincronización.

El objetivo de utilizar el modelo híbrido es aprovechar y aplicar las potencialidades de cada una de las estrategias que el mismo brinda, de acuerdo a la necesidad de la aplicación. Esta es un área de investigación de interés actual, y entre los lenguajes que se utilizan para programación híbrida aparecen OpenMP [11] para memoria compartida y MPI [12] para pasaje de mensajes.

IV. PROFILING Y CONTADORES DE HARDWARE

El profiling es una técnica que se utiliza para obtener información acumulada de diferentes eventos que el hardware es capaz de contabilizar. Entre los eventos posibles podemos enumerar los relacionados a la CPU (número de ciclos, cantidad de instrucciones, etc.); eventos relacionados a la cache (cache misses, cache loads, cache stores, etc.); y eventos relacionados a la Translation Lookaside Buffer o TLB, entre otros.

Hay diferentes herramientas que permiten realizar profiling. La que se utiliza en este trabajo es perf [13], que se detalla a continuación.

A. *Perf*

Perf (performance counter subsystem para sistemas LINUX), es una librería dedicada a realizar profiling en función de los contadores de hardware disponibles en la arquitectura.

Los contadores de hardware son registros del procesador que cuentan eventos de hardware, como por ejemplo cantidad de instrucciones ejecutadas y cache-misses.

Una de las principales ventajas que ofrece perf frente a otras librerías dedicadas al profiling es que no se debe instrumentar, de manera que no es necesario modificar el código de nuestra aplicación para obtener la información que buscamos. Incluso en aplicaciones donde solo contamos con el binario de la misma, podremos acceder a la información buscada sin necesidad de tener el código fuente.

Sin embargo, perf posee una limitación importante, dado que no es una herramienta diseñada para aplicaciones distribuidas. De esta manera, pensando en un cluster de multicore, la librería solo toma en cuenta los contadores de hardware del procesador donde se lanza la ejecución. En este trabajo, en el que se utiliza la multiplicación de matrices cuadradas, esto no representa una limitación a la hora de contabilizar los eventos porque al ser una aplicación regular, ejecutándose en una arquitectura homogénea con balance de carga, todos los procesadores deben tener, aproximadamente, la misma información.

Para poder utilizar perf, será necesario indicar en la línea de comandos junto al ejecutable de la aplicación la lista de eventos que se quieren monitorizar, de la siguiente manera:

perf stat -e lista de eventos separados por comas nombreDelBinario [parámetros (dependiendo de la aplicación)]

Hay diferentes opciones en cuanto a los eventos, pueden utilizarse directamente los nombres de los que ya están definidos en perf o los que el fabricante del hardware utiliza para eventos específicos de cada arquitectura. Además, por ejemplo, los eventos pueden medirse a nivel de usuario, a nivel de kernel, etc… [13].

V. CASO DE ESTUDIO Y SOLUCIONES IMPLEMENTADAS

Dadas dos matrices A de m x p y B de p x n elementos, la multiplicación de ambas matrices consiste en obtener la matriz C de m x n elementos (C = A x B), donde cada elemento se calcula por medio de la ecuación que se muestra a continuación:

$$C_{i,j} = \sum_{k=1}^{p} A_{i,k} * B_{k,j}$$

Los estudios experimentales se realizaron, por un lado, implementando el algoritmo de multiplicación de matrices en su versión secuencial. Para las soluciones paralelas, se implementaron dos variantes. En una de ellas los resultados (matriz C) se calculan por bloques, mientras que en el segundo algoritmo paralelo, el cálculo de la matriz C se realiza subdividiendo las matrices A y B en bloques para ser procesados. En ambas soluciones se utilizó el modelo de programación híbrida.

Tanto la solución secuencial como las paralelas fueron desarrolladas utilizando el lenguaje C. Las soluciones paralelas utilizan para el pasaje de mensajes la librería OpenMPI [12], mientras que para memoria compartida emplean OpenMP [11].

A continuación se describen brevemente las soluciones implementadas y presentadas en [9]. En todos los casos la multiplicación se realiza almacenando las matrices A y C por filas y la B por columnas de manera de poder aprovechar la localidad espacial y temporal de la memoria cache en el acceso a los datos.

A. *Solución secuencial*

Se resuelve secuencialmente el valor de cada posición de la matriz C según la Ecuación 1. La diferencia con la solución secuencial tradicional se basa en que la matriz A es subdividida en filas y la B en columnas de bloques pequeños para aprovechar la localidad de la cache L1.

B. *Solución híbrida (I)*

El algoritmo utiliza una interacción de tipo master/worker, donde el master trabaja tanto de coordinador como de worker. El mismo divide la matriz C en bloques a procesar y posteriormente genera fases de procesamiento. Dado que todos los procesadores tienen la misma potencia de cómputo y que todos los bloques a procesar son del mismo tamaño, todos procesarán (aproximadamente) a la misma velocidad. De esta manera, en cada fase de procesamiento, el master reparte las filas de la matriz A y las columnas de la matriz B según el bloque correspondiente a cada worker, incluyendo un bloque para él. Procesa su bloque y luego recibe de todos los demás los resultados para poder así pasar a la siguiente fase de procesamiento. La cantidad de fases se calcula de la siguiente manera: si b es la cantidad de bloques que se deben procesar y w la cantidad de workers (incluyendo al master que funciona como worker también), la cantidad de fases es b/w. En esta solución existe un proceso por hoja (compuesta por dos procesadores quadcore) que internamente genera 8 hilos para hacer su procesamiento.

Es importante tomar en cuenta que cada proceso necesita almacenar las filas de la matriz A que va a procesar, las columnas de la matriz B y el bloque de la matriz C que genera como resultado.

C. *Solución híbrida (II)*

Al igual que en la solución anterior, el algoritmo utiliza una interacción de tipo master/worker, donde el master trabaja tanto de coordinador como de worker. El mismo, divide la matriz A en filas de bloques y la B en columnas de bloques. Dado que todos los procesadores tienen la misma potencia de cómputo y que todos los bloques a procesar son del mismo tamaño, todos procesarán (aproximadamente) a la misma velocidad. De esta manera, el master reparte las filas de bloques de la matriz A (según el subconjunto de filas de C que debe calcular cada uno) y todos los bloques de la B a cada worker, incluyéndose a sí mismo. Procesa sus filas de bloques de C y luego recibe de todos los demás los resultados. Es importante tener en cuenta que cada proceso necesita almacenar las filas de bloques de la matriz A que va a procesar, todas las columnas de bloques de la matriz B y las filas de la matriz C que genera como resultado.

Una vez que reparte los datos a los workers, se generan los hilos correspondientes para procesar por cada fila de bloques todas las columnas de bloques que la misma posee. Los demás procesos worker actúan de la misma manera recibiendo datos y enviando sus resultados al master.

VI. RESULTADOS OBTENIDOS

El hardware utilizado en la experimentación es un Blade de 16 servidores (hojas). Cada hoja posee 2 procesadores quad core Intel Xeón e5405 de 2.0 GHz y 10GB de memoria RAM. En todos los casos las características de la misma son las siguientes: memoria RAM compartida entre ambos procesadores; cache L2 de 2 X 6Mb compartida entre cada par de cores por procesador. El sistema operativo utilizado es Debian 6 de 64 bits [14][15].

Para poder comprobar que las diferencias de tiempos obtenidos por las soluciones implementadas están relacionadas particularmente al aprovechamiento de la cache L1 es que se han realizado pruebas experimentales en las cuales mediante la librería perf se accedió a la información almacenada en los contadores de hardware.

La librería provee acceso a diferentes contadores relacionados a la cache, tales como cache-misses, L1-dcache-loads, L1-dcache-load-misses, LLC-loads, entre otros. Debe tenerse en cuenta que los resultados obtenidos, como ya se mencionó anteriormente, corresponden al procesador donde se lanzó la ejecución de la aplicación.

Dadas las características de la aplicación, en la que la mayor parte del tiempo se acceden a datos de la cache que no son modificados (datos de las matrices A y B), los contadores de hardware más relevantes son cache-misses (cantidad de accesos a memoria que no pudieron ser servidos por ninguno de los niveles de cache) y L1-dcache-load-misses (cantidad de veces que se intentó leer un dato de cache L1 y este no se encontraba en la misma).

En la Tabla I y II se muestran los resultados obtenidos utilizando 16 núcleos de procesamiento, mientras que en la Tabla III y IV los obtenidos con 32 núcleos. En ambos casos los datos que se detallan son: tamaño de la matriz (n x n), tamaño del bloque de datos utilizado, tiempo de ejecución y valores arrojados por los dos contadores de hardware. El porcentaje de diferencia entre ambas soluciones ((Max-Min)/Min)*100 se muestra en la Figura 3 mientras que el porcentaje de diferencia obtenidas por ambos contadores, calculados de la misma manera que en el caso anterior para 16 y 32 núcleos se muestra en la Tabla V.

TABLA I

SOLUCIÓN HÍBRIDA (I) 16 NÚCLEOS.

Tam.	Tam Bloque H(I)	Tiempo ejecución H(I)	Cache-misses H(I)	L1-dcache-load-misses H(I)
1024	512	0,16	90833	39215937
2048	1024	1,59	7007919	1144669820
4096	2048	14,54	52331548	11710962997
8192	1024	107,4	353382411	150944601494
16384	1024	999,8	3319932856	1235070343976

TABLA II

SOLUCIÓN HÍBRIDA (II) 16 NÚCLEOS.

Tam.	Tam Bloque H(II)	Tiempo ejecución H(II)	Cache-misses H(II)	L1-dcache-load-misses H(II)
1024	64	0,20	111712	2780171
2048	64	1,24	622774	12055241
4096	64	8,58	2107495	84902512
8192	64	63,35	9620608	1422336083
16384	64	486,53	75615320	5153111864

TABLA III

SOLUCIÓN HÍBRIDA (I) 32 NÚCLEOS.

Tam.	Tam Bloque H(I)	Tiempo ejecución H(I)	Cache-misses H(I)	L1-dcache-load-misses H(I)
1024	512	0,17	204275	9676612
2048	1024	1,13	5363830	571470855
4096	2048	8,93	35368586	6018801855
8192	1024	61,20	247342621	76685071654
16384	1024	544,72	1984860405	625413462057

TABLA IV

SOLUCIÓN HÍBRIDA (II) 32 NÚCLEOS.

Tam.	Tam Bloque H(II)	Tiempo ejecución H(II)	Cache-misses H(II)	L1-dcache-load-misses H(II)
1024	64	0,17	115983	2410420
2048	64	0,87	331756	6447472
4096	64	5,25	949626	90814423
8192	64	35,41	5289732	351520078
16384	64	257,83	43687045	5850873036

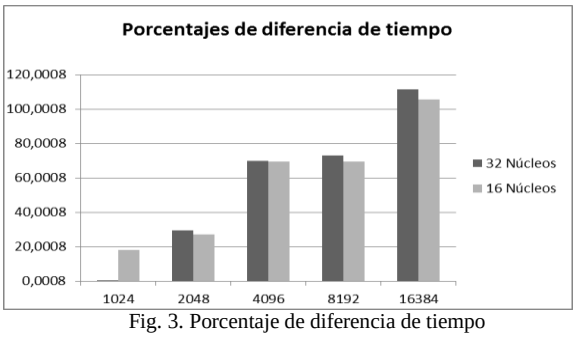


Fig. 3. Porcentaje de diferencia de tiempo

TABLA V

PORCENTAJE DE DIFERENCIA CONTADORES DE HARDWARE

Tam.	Cache-misses 16 núcleos	L1-dcache-load-misses 16 núcleos	Cache-misses 32 núcleos	L1-dcache-load-misses 32 núcleos
1024	22,98	1310,55	76,12	301,44
2048	1025,27	9395,20	1516,79	8763,48
4096	2383,11	13693,42	3624,47	6527,58
8192	3573,18	10512,44	4575,90	21715,27
16384	4290,55	23867,46	4443,36	10589,23

En función de los resultados obtenidos, se observa que en la solución híbrida (I) los tiempos de ejecución obtenidos para tamaños de matriz 1024, utilizando 16 núcleos, son mejores que los obtenidos por la solución híbrida (II). Mientras que para todas las demás pruebas, los resultados son exactamente inversos. Esto se debe a que para tamaños menores de 1024, en ambas soluciones los datos necesarios entran aproximadamente en Cache L1 y la forma de resolución del algoritmo I, al requerir menos operaciones que la solución II, arroja mejores resultados. Mientras que para tamaños mayores, los datos entrarán en Cache L1 para la solución híbrida (II) mientras que no para el algoritmo (I), y el impacto de ello, provoca que los fallos de Cache L1 del algoritmo (I) retrasen notablemente los tiempos de ejecución. Esto es verificado en la comparación de la información arrojada por los contadores de hardware.

VII. CONCLUSIONES

Los resultados obtenidos permiten analizar por un lado, que la solución híbrida (II) alcanza tiempos de ejecución menores que el algoritmo (I), tal como se muestra en el gráfico de porcentaje de diferencia de tiempos, ya que se aprovecha la jerarquía de memoria existente en el sistema. Esto se aplica también cuando se escala en la cantidad de núcleos de procesamiento utilizados. Esta diferencia entre un algoritmo y el otro se incrementa al aumentar el tamaño del problema dado que cuando los datos necesarios para ser procesados dejan de entrar en cache L1 en el algoritmo (I), sí lo hacen en el (II), verificando la idea original de la que parte este trabajo que es justamente el impacto del aprovechamiento de la arquitectura en la performance alcanzable por un algoritmo. Esto es verificado si analizamos los porcentajes de diferencia de cache-misses y L1-dcache-load-misses. En todos los casos, estos valores permiten ver que el algoritmo híbrido (I) genera siempre más de un 20% de fallos de cache que el algoritmo (II), llegando a porcentajes de diferencia de más del 100% a medida que aumenta el tamaño de las matrices.

Finalmente podemos concluir que el algoritmo híbrido (II) efectivamente aprovecha la localidad espacial y temporal de los datos en la cache L1, mejorando notablemente los tiempos de ejecución obtenidos a medida que crece el tamaño de los datos (independientemente de la cantidad de núcleos utilizados) a procesar si se lo compara con una solución que no lo aprovecha, tal como lo reflejan los contadores de hardware a los que podemos acceder mediante la librería perf, permitiendo verificar la hipótesis de la que parte este trabajo . Sin embargo, hay que destacar que la mejora reflejada en los contadores no tiene una relación lineal con la mejora de tiempos obtenidos. Esto permite concluir que la optimización por el camino de la utilización de jerarquía de memoria en principio estaría alcanzada (considerando desde la cache L1 hacia la memoria compartida). Para profundizar la optimización deberían analizarse otro tipo de optimizaciones tales como unrolling loops, entre otros. Es importante considerar que los algoritmos fueron compilados con el nivel máximo de optimización dado por el compilador (-O3).

Las líneas de investigación futuras incluyen el análisis de la eficiencia energética de los algoritmos que aprovechan la localidad de los datos y de los que no lo hacen, a fin de estudiar la influencia sobre dicho parámetro, y la adaptación de los algoritmos y su posterior análisis sobre clusters heterogéneos [16][17], incluyendo la combinación de multicore con gpu.

VIII. REFERENCIAS

[1] L. Chai, Q. Gao, D. K. Panda, “Understanding the impact of multi-core architecture in cluster computing: A case study with Intel Dual-Core System”, IEEE International Symposium on Cluster Computing and the Grid 2007 (CCGRID 2007), pp. 471-478. 2007.

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torzcon, White A, “Sourcebook of Parallel computing”, Morgan Kaufmann Publishers 2002, ISBN 1558608710, Capítulo 3.

[3] T. Burger, “Intel Multi-Core Processors: Quick Reference Guide, ”http:// cachewww.intel.com/cd/00/00/23/19/231912_231912.pdf, 2010.

[4] http://www.intel.com/support/sp/processors/xeon/sb/cs-007758.htm, 2012.

[5] F. Leibovich, S. Gallo, L. De Giusti, F. Chichizola, M. Naiouf, A. De Giusti, “Comparación de paradigmas de programación paralela en cluster de multicores: Pasaje de mensajes e híbrido. Un caso de estudio”, Proceedings del XVII Congreso Argentino de Ciencias de la Computación, CACIC 2011, Págs. 241-250, ISBN 978-950-34-0756-1.

[6] F. Leibovich, M. Naiouf, L. De Giusti, F. G. Tinetti, E. De Giusti, “Hybrid Algorithms for Matrix Multiplication on Multicore Clusters”, Julio 2012, WorldComp’12.

[7] G. Andrews, “Foundations of Multithreaded, Parallel and Distributed Programming”, Addison Wesley Higher Education 2000, ISBN-13: 9780201357523.

[8] F. Leibovich, L. De Giusti, M. Naiouf, “Parallel Algorithms on Clusters of Multicores: Comparing Message Passing vs Hybrid Programming”, Julio 2011, WorldComp’11.

[9] F. Leibovich, F. Chichizola, L. De Giusti, M. Naiouf, F. Tirado Fernández, A. De Giusti, “Programación híbrida en clusters de multicore. Análisis del impacto de la jerarquía de memoria”, Proceedings del XVIII Congreso Argentino de Ciencias de la Computación, CACIC 2012, Págs. 306 – 315, ISBN: 978-987-1648-34-4.

[10] A. Grama, A. Gupta, G. Karpyis, V. Kumar, “Introduction to Parallel Computing”, Pearson – Addison Wesley 2003, ISBN: 0201648652, Segunda Edición, Capítulo 3.

[11] https://computing.llnl.gov/tutorials/openMP, 2012.

[12] http://www.open-mpi.org, 2012.

[13] https://perf.wiki.kernel.org/index.php/Main_Page, 2013.

[14] HP, "HP BladeSystem". http://h18004.www1.hp.com/products/blades/components/c-class.html, 2011.

[15] HP, "HP BladeSystem c-Class architecture", http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00810839/c00810839.pdf , 2011.

[16] W.C. Feng, "The importance of being low power in high-performance computing", Cyberinfrastructure Technology Watch Quarterly (CTWatch Quarterly), 2005.

[17] J. Balladini, E. Grosclaude, M. Hanzich, R. Suppi, D. Rexachs, E. Luque, "Incidencia de los modelos de programación paralela y escalado de frecuencia de CPUs en el consumo energético de los sistemas de HPC", XVI Congreso Argentino de Ciencias de la Computación, pp. 172-181, 2010.

Paralelismo en la factorización NMF

EdgardoMejía-Roa[1] [2], Carlos García[1], Alberto Pascual-Montano[3] y Francisco Tirado[1]

Resumen— **En los últimos años se ha incrementado el interés de la comunidad científica en la Factorización NMF (*Non-negative Matrix Factorization*) como método de proyección empleado en el análisis exploratorio de datos para reducir sus dimensiones y facilitar así la búsqueda e interpretación de patrones ocultos. No obstante, este algoritmo requiere de una cantidad elevada de recursos de cómputo, lo que dificulta su aplicación en conjuntos de datos de gran tamaño.**

En este artículo presentamos tres implementaciones de NMF con el objetivo de estudiar el impacto en el rendimiento que aportan diferentes tecnologías que aprovechan el paralelismo a nivel de datos que expone este algoritmo.

La primera implementación explota el paralelismo de grano grueso mediante el paradigma clásico del paso de mensajes, MPI (*Message-Passing Interface*), en el que los datos se distribuyen entre distintos procesadores. A pesar de la reducción en el tiempo de cómputo, la latencia producida por los diversos puntos de sincronización y de comunicación entre procesos puede suponer un importante cuello de botella en sistemas con un elevado número de procesadores, llegando en el caso de datos de menor tamaño a anular cualquier ganancia de rendimiento.

La segunda versión, implementada con CUDA (*Compute Unified Device Architecture*), aprovecha el paralelismo de grano fino en un procesador de gráficos GPU (*Graphics-Processing Unit*), alcanzando un rendimiento comparable al de un *cluster* de varias decenas de procesadores superescalares. Sin embargo, muchos dispositivos GPU disponen de poca memoria, por lo que datos de grandes dimensiones deben transferirse y procesarse por bloques de forma secuencial, incrementando drásticamente los tiempos de cómputo.

Finalmente, la tercera implementación combina ambas tecnologías. Comparado con la versión *mono-GPU* anterior, se obtiene un *Speedup Superlineal* cuando se emplea el *mínimo* número de GPUs que evita el procesamiento por bloques de las porciones de datos asignadas a cada dispositivo.

Palabras clave— **Factorización de Matrices, Non-negative Matrix Factorization, NMF, MPI, GPU, multi-GPU, CUDA, CUBLAS, Paralelismo de grano fino, Paralelismo a nivel de datos, Análisis exploratorio, Producto de Matrices, Álgebra Lineal**

I. Introducción

La *Factorización NMF* (*Non-Negative Matrix Factorization*) [1] permite representar una matriz de datos de grandes dimensiones como la combinación lineal de un pequeño conjunto de elementos denominados *factores*, lo que reduce significativamente la cantidad de datos a analizar. A diferencia de otras técnicas de factorización parecidas (*PCA*, *SVD*, *ICA*, etc.), en NMF se impone la restricción tanto a *factores* como a coeficientes de no utilizar valores negativos, permitiendo únicamente combinaciones aditivas. De esta manera, se obtiene una representación de los datos basada en la acumulación total o parcial de partes o secciones con significado propio, es decir, donde cada uno de estos denominados *factores* posee una interpretación en el contexto del análisis. Gracias a esta propiedad, se ha incrementado el interés de la comunidad científica en este método. Por ejemplo, se ha aplicado en el campo de la Bioinformática [2], [3], en el reconocimiento facial [4], en estudios sobre el color [5], en la transcripción de señales musicales [6], etc. No obstante, los requisitos de cómputo de NMF son elevados, lo que dificulta su aplicación en el análisis de matrices de gran tamaño. Por tanto, una implementación eficiente que reduzca significativamente el tiempo de cómputo es clave para la viabilidad de este algoritmo.

En este artículo presentamos tres implementaciones de NMF con el objetivo de estudiar el impacto en el rendimiento que aportan diferentes tecnologías que aprovechan el paralelismo a nivel de datos que expone este algoritmo.

La primera implementación, basada en el estándar de paso de mensajes *MPI* (*Message-Passing Interface*) [7], explota el paralelismo de grano grueso mediante una descomposición estática de dominio en la que los datos se distribuyen entre distintos procesos. Dado que esta técnica suele emplearse en sistemas de memoria distribuida, las latencias producidas por los diversos puntos de sincronización y de comunicación entre procesos pueden tener un impacto importante en el rendimiento, especialmente en grandes sistemas multiprocesadores.

A continuación, la segunda versión aprovecha el paralelismo de grano fino que proporciona un procesador de gráficos *GPU* (*Graphics-Processing Unit*). Gracias a modelos de programación como *CUDA* (*Compute Unified Device Architecture*) [8], que permiten utilizar unidades GPU para cómputo de propósito general, la comunidad científica ha comenzado a emplear este tipo de dispositivos como sistemas de alto rendimiento *"de bajo coste"* (en comparación con un *cluster* clásico) [9], [10]. No obstante, en muchas arquitecturas el acceso se realiza a través de un bus de datos (por ejemplo, *PCI Express*) y se dispone de una memoria "privada" de capacidad limitada, por lo que datos de grandes dimensiones deben transferirse y procesarse por bloques de forma secuencial.

Finalmente, la tercera implementación combina ambas tecnologías. Intuitivamente, esta versión evita las situaciones de desventaja de las anteriores. Por un lado, el utilizar múltiples GPUs permite superar las limitaciones de memoria y la necesidad del procesamiento por bloques. Por otro, basta con pocas unidades para procesar datos de gran tamaño.

Hasta donde sabemos, sólo unos pocos trabajos han presentado implementaciones paralelas de NMF. Algunos de ellos (por ejemplo, [11], [12], [13], [14])

[1]Grupo ArTeCS, Universidad Complutense de Madrid.
[2]e-mail: edgardomejia@fis.ucm.es
[3]Centro Nacional de Biotecnología-CSIC, Madrid, e-mail: pascual@cnb.csic.es

están diseñados para ejecutarse únicamente en sistemas multiprocesadores clásicos. Otros, como [15], [16] y [6], sí utilizan dispositivos GPU, pero no toman en cuenta matrices de grandes dimensiones que superen la capacidad de memoria.

El resto del artículo está organizado del siguiente modo. En la sección II se describe la Factorización NMF y una de sus aplicaciones en el área de Bioinformática. Posteriormente, las secciones III, IV y V detallan las implementaciones del algoritmo utilizando, respectivamente, MPI, CUDA y la combinación de ambas. En la sección VI se muestran algunas pruebas de rendimiento. Finalmente, la sección VII presenta las conclusiones.

II. La Factorización de Matrices no Negativas (NMF)

Matemáticamente, la Factorización NMF [1] se describe como la descomposición de una matriz de entrada $\mathbf{V} \in \mathbb{R}^{n \times m}$ en otras dos matrices de menor tamaño, $\mathbf{W} \in \mathbb{R}^{n \times k}$ y $\mathbf{H} \in \mathbb{R}^{k \times m}$ (con $k \ll n, m$), cuyo producto se aproxima a la primera, es decir, $\mathbf{V} \approx \mathbf{WH}$. La matriz $\mathbf{W}$ contiene el conjunto de k factores, mientras que en $\mathbf{H}$ aparecen los coeficientes de la combinación lineal que permite reconstruir $\mathbf{V}$. El algoritmo consiste en modificar iterativamente las matrices $\mathbf{W}$ y $\mathbf{H}$ hasta que su producto se aproxime a $\mathbf{V}$. Tales modificaciones (compuestas de productos de matrices, reducciones y otras operaciones algebraicas) se derivan de la minimización de alguna función objetivo que describa la diferencia entre $\mathbf{WH}$ y $\mathbf{V}$. Existen diversas funciones objetivo que producen distintas ecuaciones [17], [1]. En este artículo nos centraremos en las ecuaciones (1) y (2) descritas en [17].

$$H_{pj} \leftarrow H_{pj} \frac{\sum_{i=1}^{n}(W_{ip}V_{ij}) / \sum_{q=1}^{k}(W_{iq}H_{qj})}{\sum_{i=1}^{n} W_{ip}} \quad (1)$$

$$W_{ip} \leftarrow W_{ip} \frac{\sum_{j=1}^{m}(V_{ij}H_{pj}) / \sum_{q=1}^{k}(W_{iq}H_{qj})}{\sum_{j=1}^{m} H_{pj}} \quad (2)$$

Como ya se mencionó en la introducción, tanto NMF como muchas de sus variantes [18] tienen diversas aplicaciones en el campo de la Bioinformática. Una de ellas es el *Análisis de expresión de genes* [2], [19], donde $\mathbf{V}$ representa una matriz con la expresión, o nivel de actividad, de n genes en m experimentos (ver figura 1). En este caso, NMF divide genes y experimentos en k grupos correspondientes a alguna característica local o patrón en los datos, indicando en $\mathbf{W}$ el grado de pertenencia de cada gen a cada uno de estos k grupos, y en $\mathbf{H}$ el grado de influencia de estos grupos en cada experimento.

III. Implementación mediante Paso de Mensajes (MPI)

En esta implementación se utiliza un esquema clásico de partición de datos en el que cada proceso MPI tiene en memoria dos porciones de la matriz de entrada $\mathbf{V}$ (un conjunto de filas y otro de columnas) y una copia completa de las matrices $\mathbf{W}$ y $\mathbf{H}$.

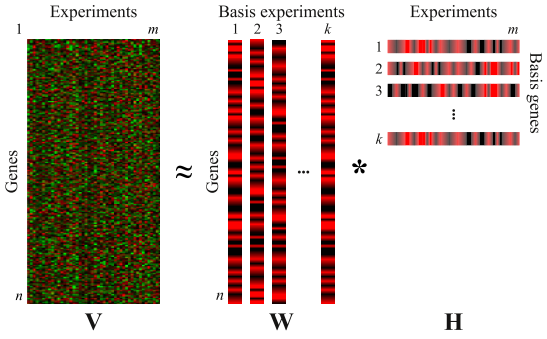


Fig. 1
NMF aplicado al análisis de expresión de genes.

Como se aprecia en la figura 2, cada proceso se encarga de actualizar un conjunto de b_m *columnas* de $\mathbf{H}$, es decir, un bloque de dimensiones $k \times b_m$. Para ello, se necesitan las correspondientes b_m *columnas* de $\mathbf{V}$ y *toda* la matriz $\mathbf{W}$.

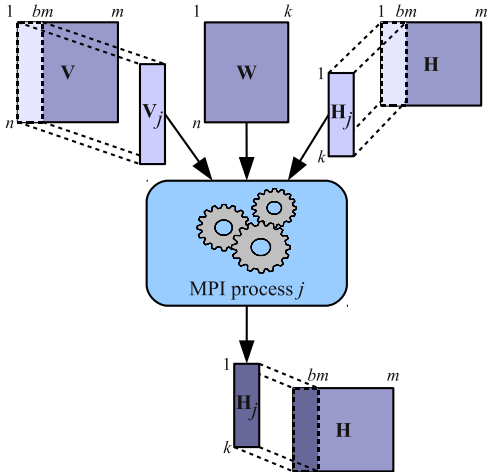


Fig. 2
Implementación en MPI. Actualización de la matriz $\mathbf{H}$.

Un esquema muy similar se utiliza para procesar la matriz $\mathbf{W}$, aunque en este caso se trata de bloques de b_n *filas* ($b_n \times k$). Para actualizarlos, son necesarias las correspondientes b_n *filas* de $\mathbf{V}$ y *toda* la matriz $\mathbf{H}$. Es importante resaltar que esta última debe tener todas sus columnas actualizadas, es decir, previamente cada proceso debe haber sincronizado su contenido con sus pares a través de comunicaciones de tipo `all_gather`. Análogamente, después de actualizar $\mathbf{W}$, todos los procesos deben sincronizar sus respectivas copias privadas. Si bien estas operaciones suponen ciertas latencias, son necesarias para mantener la coherencia entre los datos de todos los procesos. No obstante, para evitar un continuo proceso de empaquetado y desempaquetado de las columnas de $\mathbf{H}$ durante las comunicaciones, esta matriz se guarda en memoria *"por columnas"*, es decir, de manera transpuesta. Esto también beneficia la localidad espacial en el resto de operaciones aritméticas.

IV. Implementación en Procesadores de gráficos (GPU)

Esta versión se basa en el modelo de programación *CUDA* [8], de NVIDIA, que representa una GPU como un coprocesador de propósito general capaz de ejecutar pequeñas porciones de código secuencial, denominadas *kernels*, en cientos o miles de hilos de manera simultánea. Para ello, CUDA proporciona una extensión del lenguaje `C` y una biblioteca de funciones que permite 1) reservar memoria en la GPU, 2) transferir datos entre la GPU y la CPU, y 3) ejecutar los mencionados *kernels*.

Este modelo permite explotar un paralelismo de grano mucho más fino que otras tecnologías de procesamiento paralelo. Además, simplifica enormemente la implementación en GPU de algoritmos no relacionados con el dibujado de gráficos en pantalla [10]. No obstante, como se mencionó en la introducción, muchos de estos dispositivos disponen de una memoria de capacidad limitada, lo que impone un procesamiento por bloques mediante un esquema similar al de la versión MPI.

La figura 3 muestra el proceso de actualización de la matriz **H** en el que de manera iterativa se van procesando bloques de b_m columnas. Para ello, son necesarias las correspondientes columnas de **V** y una copia completa de **W**. Los productos de matrices se realizan utilizando la biblioteca de funciones *CUBLAS* [8], mientras que el resto de operaciones son llevadas a cabo por los *kernels* que hemos implementado. Tras actualizar las mencionadas columnas, estas se acumulan en el vector $\mathbf{ACC_H}$ (ver zona recuadrada de la figura 3) con el objeto de aprovechar la localidad temporal, ya que este vector se corresponde con el denominador de la ecuación (2), necesario para el posterior procesamiento de **W**. Todo este procedimiento se realiza de manera análoga para **W**, aunque en este caso se trata de bloques de b_n filas.

Puesto que $k \ll n, m$, las matrices **W** y **H** no adquieren dimensiones significativas, por lo que pueden permanecer residentes en la memoria de la GPU. Por tanto, únicamente **V** es transferida por bloques, lo que simplifica la implementación del algoritmo y reduce drásticamente el uso del bus de datos.

Esta implementación parece obtener mejores resultados al procesar las matrices como memoria lineal y disponer los hilos CUDA de manera contigua en memoria. La idea es garantizar que todos los hilos de cada *warp* (unidad de planificación de hilos CUDA) accedan a direcciones de memoria consecutivas. En el caso de operaciones que dependen del número de columnas de las matrices, se utilizan posiciones de relleno para imponer dimensiones múltiplo de 16 (valor equivalente a medio *warp*). Mantener de esta manera los datos alineados compensa con creces el gasto adicional de memoria y los cómputos irrelevantes, ya que permite unificar diversas peticiones de acceso a memoria en una única operación.

También es importante resaltar que de forma similar a la versión MPI, la matriz **H** se guarda en memoria *por columnas* (transpuesta). De esta manera, **W** y **H** tienen el mismo *"ancho"* (k) permitiendo utilizar los mismos *kernels* en ambas reglas de actualización.

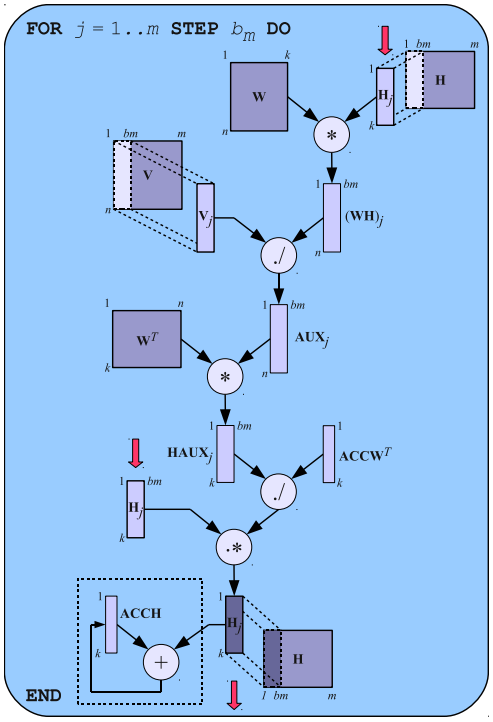


Fig. 3
Implementación en GPU. Actualización de la matriz **H**. Operaciones en círculos representan "kernels" de CUDA. Los símbolos ".*" y "./" denotan operaciones elemento a elemento entre matrices.

Por último, las transferencias se realizan de forma asíncrona, lo que permite solaparlas con la ejecución de los *kernels*. La concurrencia, en estos casos, se gestiona mediante *Streams* y *Eventos* de CUDA.

V. NMF en sistemas multi-GPU

Esta implementación combina los niveles de paralelismo de las dos versiones anteriores. En ella, las matrices se distribuyen entre varios procesos MPI como se describe en la sección III. A continuación, cada proceso gestiona un dispositivo GPU y aplica el esquema descrito en la sección IV a la porción de datos que le fue asignada.

Este procedimiento, que equivale a paralelizar el bucle mostrado en la figura 3, supone una reducción de las transferencias de la matriz **V** entre GPU y CPU, ya que a cada dispositivo le corresponde un conjunto de datos de menor tamaño. De hecho, si dichas porciones son suficientemente pequeñas, no será necesario transferirlas a sus respectivas GPUs por bloques sino en una única operación al inicio del algoritmo.

No obstante, dado que todos los procesos MPI deben sincronizar sus respectivas copias de **W** y **H** cada vez que estas se modifican, dichas operaciones también implican transferencias entre GPU y CPU. Afortunadamente, el tamaño de estas matrices es bastan-

te más pequeño que el de **V**, con lo que sigue habiendo una reducción global de las transferencias.

VI. Pruebas y Resultados

En esta sección presentamos los resultados de las tres implementaciones del algoritmo NMF. Los experimentos se llevaron a cabo en las siguientes plataformas:

- Sistema Multiprocesador con 1036 nodos *IBM BladeCenter JS20* (cada uno provisto de dos procesadores *IBM PowerPC 970FX* de 2,2 GHz y 4 GB de RAM DDR-PC2700), interconectados por una red *Myrinet* de alto rendimiento. La sincronización entre los procesos se realizó utilizando `mpich v1.2.6`, compilado con `xlc`. Por último, los productos de matrices se ejecutan a través de llamadas a la biblioteca de funciones `ATLAS v3.6` [20].
- Un servidor de dos procesadores *Xeon Dual-Core*, equipado con cuatro GPUs *Tesla C1060* de NVIDIA. Cada dispositivo tiene 240 núcleos distribuidos en 30 multiprocesadores, 4 GB de memoria y una *Capacidad de cómputo de NVIDIA* de 1,3. Las tres implementaciones fueron compiladas con `gcc 4.4.5` y `CUDA/CUBLAS 3.2` en un Linux de 64 bits. Para la sincronización se utilizó `mpich v1.2.7p1` con la configuración más parecida a la del sistema IBM.

Dado que el rendimiento de cada implementación depende en gran medida de las dimensiones de la matriz de entrada, hemos utilizado tres conjuntos de datos, procedentes de diversos estudios biológicos, que representan distintas clases de tamaño: pequeña, mediana y grande.

- *"ALL_AML"* (5000×38): Conjunto de 5000 genes analizados en 38 muestras de médula ósea correspondientes a dos tipos de tejido tumoral: *"Acute Lymphoblastic Leukemia"* (*ALL*) y *"Acute Myelogenous Leukemia"* (*AML*) [19].
- *"Cáncer"* (54675×1973): Nivel de expresión génica de 54 mil genes en 1973 muestras tumorales. Conjunto de datos procedente de la base de datos *GEO* [21] bajo el identificador '`GSE2109`'.
- *"HG"* (22283×5896): Análisis de más de 22 mil genes en 5800 muestras de tejido humano sano y afectado. Disponible en la base de datos *ArrayExpress* [22] bajo el identificador '`E-TABM-185`'.

A fin de homogeneizar todas las pruebas en la medida de lo posible, las matrices **W** y **H** se inicializan utilizando valores aleatorios procedentes de la misma *"semilla"*. Además, la ejecución del algoritmo queda fijada a 440 iteraciones, independientemente de la *"distancia"* entre **WH** y **V**.

En la tabla I se muestran los tiempos de cómputo de referencia, obtenidos con uno de los procesadores del *cluster* IBM. En las tres pruebas, se observa que los tiempos de ejecución crecen de forma lineal en función del *rango de factorización* (k).

TABLA I
Tiempos de ejecución, en segundos, de 440 iteraciones en la versión secuencial de base para diferentes rangos de factorización (k).

k	*ALL_AML*	*Cáncer*	*HG*
2	3.42	1842.29	2788.65
3	3.86	2069.99	2291.20
4	4.14	2404.44	2577.53
5	4.63	2397.94	2580.13
6	5.07	2620.29	2881.39
7	5.59	3131.00	3182.16
8	5.89	3147.86	3442.17
9	6.35	3124.68	3410.78
10	6.66	3348.15	3726.54

A. Rendimiento de la versión en MPI

Las figures 4 y 5 ilustran, respectivamente, los tiempos de ejecución de la mayor de las matrices (*HG*) y la Eficiencia media obtenida para todos los rangos de factorización (k).

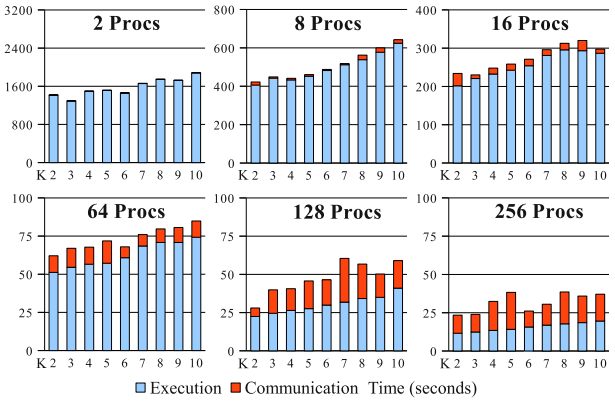


Fig. 4
Versión MPI: tiempos de cómputo y de comunicación entre procesos de la matriz *HG*, en segundos.

En matrices de medio y gran tamaño, las latencias producto de las comunicaciones entre procesos representan entre el 8 % y el 16 % del tiempo de ejecución con 32 CPUs, y aumentan gradualmente hasta alcanzar el 50 % con 256 procesadores. Esto representa una Eficiencia que decrece de 1,0 en el caso de dos procesadores, hasta 0,47 con 256 CPUs. Por su parte, en *ALL_AML*, la matriz de menor tamaño, estas latencias ocupan el 5 % del tiempo con dos procesadores, pero se incrementan rápidamente hasta el 70 % con sólo 16 CPUs.

A pesar de la reducción en el tiempo de cómputo, la escalabilidad está supeditada a que haya suficiente carga computacional y un bajo coste en las comunicaciones entre procesos.

B. Rendimiento de la versión en GPU

La capacidad de memoria de la GPU utilizada es más que suficiente para albergar cualquiera de las

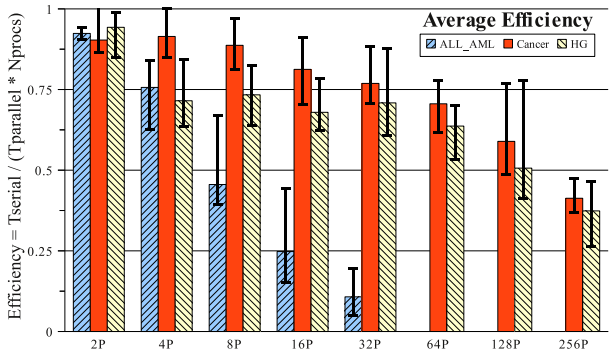


Fig. 5
Versión MPI: Eficiencia media obtenida para todos los rangos de factorización, en función del número de procesadores.

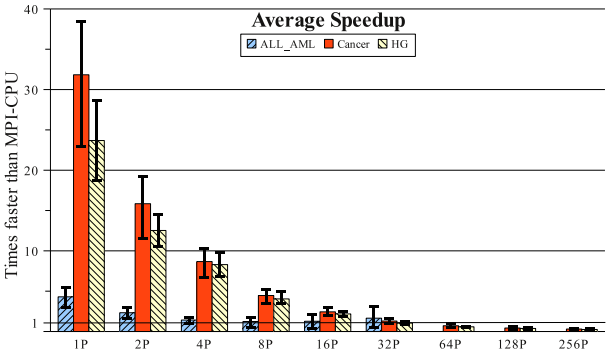


Fig. 7
Versión GPU: *Speedup* medio respecto de la versión base (*1P*) y MPI.

matrices de prueba. No obstante, este podría no ser el caso en muchas arquitecturas. Para estudiar el impacto en el rendimiento que tendría tal situación, hemos simulado que la GPU tiene menos de 800 MB de memoria, lo que impone un procesamiento por bloques de la matriz *HG*.

La figura 6 muestra los tiempos de ejecución y de transferencia de datos de la versión GPU. Por su parte, en la figura 7 se indica el *speedup* respecto a la versión MPI. Lo primero que se observa es la regularidad en los tiempos de cómputo respecto del parámetro k. Esto se debe a las posiciones de *relleno* para garantizar el alineamiento de los datos en memoria. En cuanto al rendimiento, en el caso de *ALL_AML* se alcanza un valor bastante modesto (menos de 6x respecto de la versión base), ya que no existe suficiente carga computacional para mantener a miles de hilos CUDA ejecutándose concurrentemente. Caso contrario al de la matriz *Cáncer*, donde se obtiene un pico de 38x. Sin embargo, con *HG*, el rendimiento cae a 29x debido al procesamiento por bloques, cuyas transferencias ocupan el 30 % del tiempo de ejecución. Por último, se aprecia que con matrices medianas o de grandes dimensiones, la versión MPI necesita al menos 32 procesadores para superar el rendimiento de un único dispositivo GPU.

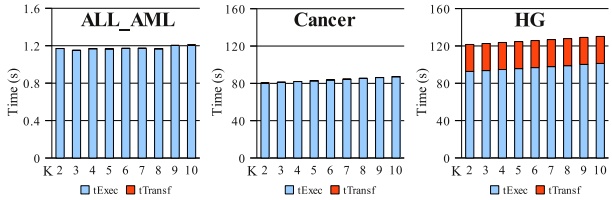
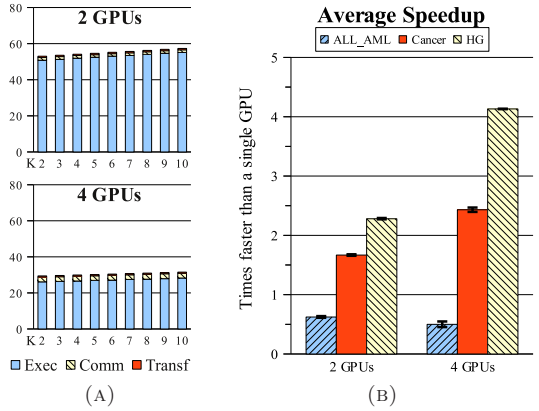


Fig. 6
Versión GPU: tiempos de cómputo y de transferencia de datos, en segundos.

C. Rendimiento de la versión MPI-GPU

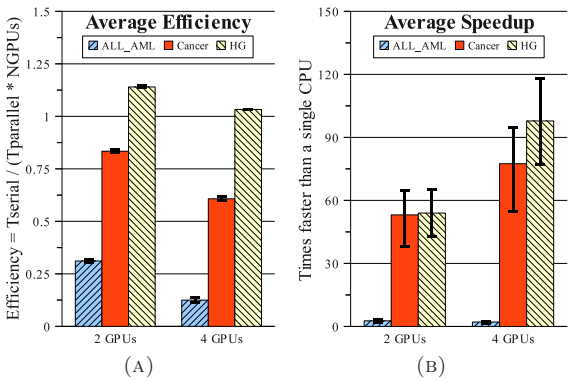
Al igual que en la versión anterior, hemos limitado la capacidad de memoria de los dispositivos. Las figuras 8a y 8b muestran, respectivamente, los tiempos de ejecución en *HG* y la ganancia de rendimiento respecto de la versión *mono-GPU*. Por su parte, en las figuras 9a y 9b se aprecia la Eficiencia obtenida y el *Speedup* medio respecto de la versión base.



(a) (b)

Fig. 8
Versión MPI-GPU: (a) tiempos de cómputo, transferencias de datos y de comunicación entre procesos, de *HG*. (b) *Speedup* medio respecto de la versión *mono-GPU*.

Como era de esperar, el rendimiento de *ALL_AML* es bastante bajo. Las latencias por transferencias de datos y comunicaciones entre procesos suponen el 50 % del tiempo de ejecución. En cuanto a la matriz *Cáncer*, las transferencias son despreciables (3 % del tiempo), pero las comunicaciones representan el 7 % y el 17 % en dos y cuatro GPUs, respectivamente, lo que provoca una caída en la Eficiencia de 0,83 a 0,61. Finalmente, con *HG* se obtiene un *Speedup superlineal* de 2,3x en dos GPUs, ya que las porciones asignadas a cada dispositivo no superan sus respectivas capacidades de memoria y pueden, por tanto, transferirse en una única operación al inicio del algo-



(A) (B)

Fig. 9
VERSIÓN MPI-GPU: (A) EFICIENCIA MEDIA RESPECTO DE LA VERSIÓN *mono-GPU*. (B) *Speedup* MEDIO RESPECTO DE LA VERSIÓN BASE.

ritmo. En el caso de cuatro GPUs sigue obteniéndose una ganancia *superlineal* (4,13x), pero la Eficiencia decrece ligeramente debido a las comunicaciones, que suponen el 9 % del tiempo.

Si bien el *Speedup* máximo respecto de la versión de base es de 120x, en la práctica, el rendimiento de esta implementación es comparable al de la versión MPI con 256 procesadores. En cuanto a la Eficiencia respecto de la versión *mono-GPU*, esta alcanza su valor más alto cuando se utiliza el *mínimo* número de GPUs que evita el procesamiento por bloques de las porciones asignadas a cada dispositivo. A partir de allí, las latencias por comunicaciones y transferencias de **W** y **H** comienzan a cobrar importancia.

VII. CONCLUSIONES

Aunque la tecnología MPI sigue siendo de uso generalizado, el coste de un sistema multiprocesador de alto rendimiento es, en muchos casos, de varios órdenes de magnitud por encima del de un dispositivo GPU. Por ello, se ha incrementado la utilización de estos últimos como alternativa de *"bajo coste"* en cuanto a relación MFLOP/$ se refiere, y la tendencia tecnológica es que esto se acreciente en el futuro. Si bien su (relativa) poca capacidad de memoria y las latencias asociadas a las transferencias de datos siguen siendo puntos críticos para el rendimiento, el creciente empleo de *clusters* de GPUs ha originado el desarrollo comercial de sistemas multi-GPU que facilitan el aprovechamiento de diversos niveles de paralelismo como los discutidos en este trabajo.

AGRADECIMIENTOS

El presente trabajo ha sido financiado mediante los proyectos BIO2010-17527 y TIN2008-00508. Además, E. Mejía-Roa es financiado por la beca *FPU* del Ministerio de Educación y Ciencia.

REFERENCIAS

[1] Daniel D. Lee and H. Sebastian Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, Oct. 21 1999.

[2] Pedro Carmona-Saez, Roberto Pascual-Marqui, F. Tirado, Jose M. Carazo, and Alberto Pascual-Montano, "Biclustering of gene expression data by non-smooth non-negative matrix factorization," *BMC Bioinformatics*, vol. 7, pp. 78, Feb. 2006.

[3] Karthik Devarajan, "Nonnegative matrix factorization: An analytical and interpretive tool in computational biology," *PLoS Comp Bio*, vol. 4, no. 7, pp. e1000029, July 25 2008.

[4] R. Ramanath, W.E. Snyder, and H. Qi, "Eigenviews for object recognition in multispectral imaging systems," in *Proc. of the AIPR Workshop*, 2003.

[5] R. Ramanath, R.G. Kuehni, W.E. Snyder, and D. Hinks, "Spectral spaces and color spaces," *Color Res.and Appl.*, vol. 29, pp. 29–37, 2004.

[6] Eric Battenberg and David Wessel, "Accelerating non-negative matrix factorization for audio source separation on multi-core and many-core architectures," in *Proc. of the ISMIR Conf.*, 2009.

[7] "Message Passing Interface (MPI) Forum," `http://www.mpi-forum.org`.

[8] "CUDA: Compute Unified Device Architecture," `http://www.nvidia.com/object/cuda_home.html`.

[9] Jhon Nickolls and William J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar.–Apr. 2010.

[10] "GPGPU: General-Purpose Computation on Graphics Hardware," `http://gpgpu.org`.

[11] Chao Dong, Huijie Zhao, and Wei Wang, "Parallel nonnegative matrix factorization algorithm on the distributed memory platform," *Int.J.of Parallel Program.*, vol. 38, pp. 117–137, 2010.

[12] Khushboo Kanjani, "Parallel Non Negative Matrix Factorization for document clustering," Tech. Rep., CPSC-659 (Parallel and Distributed Numerical Algorithms) course. Texas A&M University, 2007.

[13] Stefan A. Robila and Lukasz Grzegorz Maciak, "A parallel unmixing algorithm for hyperspectral images," in *Intell. Robots and Comp. Vision XXIV: Alg., Techn., and Active Vision*, Boston, MA, USA, Oct. 1 2006, vol. 6384 of *Proc. of the SPIE*, p. 63840F.

[14] Karthik Devarajan and G. Wang, "Parallel implementation of non-negative matrix algorithms using high-performance computing cluster," Scientific report, Fox Chase Cancer Center, 2006.

[15] Jan Platoš, Petr Gajdoš, Pavel Krömer, and Václav Snášel, "Non-negative Matrix Factorization on GPU," in *Netw. Digital Tech. I.* July 2010, vol. 87 of *CCIS*, pp. 21–30, Springer-Verlag.

[16] Noel Lopes and Bernardete Ribeiro, "Non-negative Matrix Factorization. implementation using Graphics Processing Units," in *IDEAL*. Sept. 2010, vol. 6283 of *LNCS*, pp. 275–283, Springer-Verlag.

[17] Daniel D. Lee and H. Sebastian Seung, "Algorithms for Non-negative Matrix Factorization," in *NIPS*. 2001, vol. 13, pp. 556–562, MIT Press.

[18] Alberto Pascual-Montano, José María Carazo, Kieko Kochi, Dietrich Lehmann, and Roberto D. Pascual-Marqui, "Nonsmooth nonnegative matrix factorization (nsNMF)," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 3, pp. 403–415, 2006.

[19] Jean Pierre Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov, "Metagenes and molecular pattern discovery using matrix factorization," *PNAS*, vol. 101, no. 12, pp. 4164–4169, Mar. 23 2004.

[20] "ATLAS: Automatically Tuned Linear Algebra Software," `http://math-atlas.sourceforge.net`.

[21] "GEO: Gene Expression Omnibus," `http://www.ncbi.nlm.nih.gov/projects/geo`.

[22] "ArrayExpress," `http://www.ebi.ac.uk/arrayexpress`.

Trasgo: Sistema en tiempo de ejecución para código paralelo abstracto y portable

Ana Moretón Fernández, Arturo Gonzalez-Escribano, Diego Llanos Ferraris

Resumen— **En este trabajo presentamos la evolución de Trasgo, un sistema de generación de programas paralelos a partir de código abstracto y portable. Trasgo se apoya en una biblioteca de funciones que posterga importantes decisiones de mapeo a tiempo de ejecución. El lenguaje de entrada es un lenguaje de coordinación explícitamente paralelo dónde el programador no necesita tomar decisiones relacionadas con la granularidad, los recursos del sistema, o la estructura de comunicación. A partir de estos códigos abstractos Trasgo permite generar programas eficiente que se adaptan al sistema real, explotando diferentes tecnologías de paralelización en sistemas híbridos y/o heterogéneos.**

Palabras clave— **Lenguajes de programación paralela, generación de código, sistemas run-time**

I. Introducción

DEBIDO a la evolución de los sistemas de cómputo paralelo, el desarrollo de aplicaciones que explotan paralelismo ha ido cambiando paulatinamente su foco. Desde las clásicas aplicaciones numéricas en supercomputadores de alto-rendimiento, hacia la computación de propósito general en cualquier ámbito y situación.

Hoy por hoy interesa explotar el paralelismo tanto de algoritmos básicos como de complejos programas modulares. Las nuevas aplicaciones paralelas muestran diferentes niveles de paralelismo, con diferentes estrategias de paralelización. Muchas aplicaciones muestran comportamientos dinámicos, dependientes de datos, dónde las políticas de balanceo automático de carga son claves. A su vez, las aplicaciones deberían ser flexibles, para poder ejecutarse en entornos cada vez más diversos, mezclando clusters de memoria distribuida con multicores y dispositivos aceleradores cómo GPUs. La enorme diversidad y heterogeneidad de las plataformas, y las diferentes formas de componer dichos elementos presentan problemas para el desarrollo de aplicaciones portables que sean capaces de adaptarse de forma eficiente al entorno de ejecución.

Hasta ahora han aparecido diversos modelos de programación que han sido adoptados con éxito tanto por la comunidad científica cómo los desarrolladores. En sistemas distribuidos las implementaciones eficientes del paradigma de paso de mensajes (p.e. las implementaciones de MPI). Para sistemas de memoria compartida las herramientas de control de threads de alto nivel (p.e. OpenMP). Para los dispositivos aceleradores, o sistemas híbridos que contengan aceleradores, modelos específicos (p.e. CUDA u OpenCL). En un intento de aunar diferentes tendencias y paradigmas los modelos PGAS (Partitioned Global Adress Space) (p.e. UPC, Chapel, X10).

Sin embargo, el programador que usa estos modelos o herramientas se enfrenta a diversas tareas que le obligan a razonar en términos de recursos físicos de una hipotética plataforma de ejecución, introduciendo código completamente ajeno a la lógica de la aplicación o algoritmo paralelo original, para preveer diversas situaciones. Estas tareas pueden incluir la maximización de la localidad de datos en los dispositivos, analizar costes de comunicación/sincronización, tomar decisiones sobre particiones de datos y granularidad de tareas, decidir las estrategias de paralelización para diferentes niveles de granularidad, escoger técnicas de mapeo, distribución de datos y planificación de tareas, o crear estructuras de comunicación y sincronización apropiadas.

En estos momentos es clave proveer a los desarrolladores con entornos y herramientas de programación productivas. Plataformas donde estos detalles queden ocultos y sea posible programar, probar y depurar de forma fácil, e independientemente de los detalles de la máquina dónde se va a ejecutar la aplicación. En este trabajo presentamos la evolución de Trasgo [1], un sistema integral de programación paralela de éstas características. Trasgo es un sistema de transformación de código fuente a código fuente (ver figura 1). Su entrada es un lenguaje de programación paralela explícito, pero de alto nivel, abstracto e independiente de los detalles y decisiones relacionados con la plataforma de ejecución. El sistema de transformación automática genera código para múltiples plataformas y/o clusters heterogéneos, explotando eficientemente diversas herramientas, lenguajes y modelos de programación. El código generado se asienta sobre un sistema run-time que adapta las decisiones relacionadas con la partición de datos y tareas, y las estructuras de comunicación y sincronización a la plataforma de ejecución. Presentamos las nuevas características de Trasgo, los principios utilizados para incluir en el sistema de generación herramientas para sistemas heterogéneos, y casos de

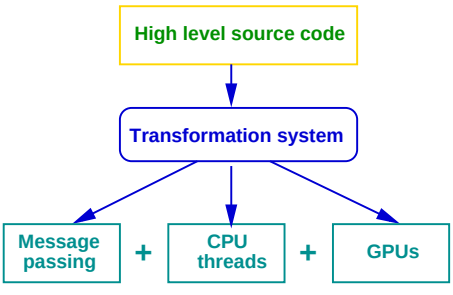


Fig. 1. Sistema de mapeo automático Trasgo

estudio para ejemplificar las propiedades del sistema.

El resto del artículo se organiza de la siguiente forma. En la sección II se repasa el modelo de programación asociado al sistema Trasgo. En la sección III se discuten las generalidades del sistema de transformación de Trasgo y los cambios conceptuales en las nuevas versiones. En la sección IV se presentan las nuevas funcionalidades incluidas en el sistema de run-time y como utilizarlas en la transformación de código. En la sección V se comentan dos casos de estudio y resultados experimentales preliminares. La sección VI presenta las conclusiones y retos planteados en el estado de desarrollo actual de Trasgo.

II. El modelo Trasgo

A. Modelo de programación

El modelo de programación de Trasgo se basa en representaciones abstractas, estructuradas y de alto nivel de los algoritmos o programas paralelos. Una de las claves es que dicha representación está restringida al paradigma de *paralelismo anidado*. Una representación jerárquica y estructurada, de las construcciones de paralelismo, pero con un modelo de sincronización restrictivo. Es misión del sistema de transformación encontrar las formas no estructuradas de comunicación o sincronización más apropiadas para cada caso y reescribirlas en el código generado. Al mantener una representación jerárquica y estructurada el modelo se asienta en la composición de procesos SP (Serie-Paralelo) [2], con propiedades semánticas claras y bien definidas [3]. El resultado es un modelo sencillo, fácil de entender y programar, libre de condiciones de carrera, o comportamientos estocásticos y situaciones de *abrazo-mortal* difíciles de predecir. Conceptualmente los puntos de sincronización generan estados globales coherentes que favorecen las pruebas y la depuración del código. El modelo Trasgo extiende características del modelo poliédrico [4] y de álgebras de datos.

B. Lenguaje de programación

Se pueden desarrollar diferentes front-ends para traducir un lenguaje de entrada concreto a la representación interna que utiliza el sistema Trasgo. En su versión actual, el lenguaje de entrada utilizado por defecto es cSPC [5], para el que existe un front-end completo. cSPC es una extensión del lenguaje C estándar, con primitivas de coordinación. En la figura 2 se muestra el código cSPC de un PDE solver basado en el método de Jacobi para resolver la ecuación del calor en un espacio bidimensional.

El código secuencial se encapsula en funciones con parámetros formales en los que el programador expresa explícitamente el comportamiento de entrada o salida de cada uno. El sistema usará esta información para determinar las dependencias de datos. Estas funciones secuenciales pueden ser del nivel de granularidad apropiado para el algoritmo, ignorando por completo decisiones relacionadas con la eficiencia. El sistema determinará si las tareas deben ser subdivididas o agrupadas. De hecho, las funciones secuenciales pueden implementar desde operaciones de grano muy fino, hasta wrappers que reciben grandes estructuras de datos y llaman a funciones de biblioteca especializadas para el cálculo secuencial, o para dispositivos aceleradores. En el ejemplo, la función *update* es una función secuencial que actualiza una celda (parámetro out) de la matriz que representa el espacio discretizado, con la media de los valores de otras cuatro celdas (parámetros in).

```
void update( in double left,
             in double right,
             in double up,
             in double down,
             out double myself ) {
   myself = ( left + right + up + down ) / 4;
}

coordination void jacobiSolver(
             in int iterations
             inout double matrix[][] ) {

  double inside[][] = matrix[1:$-1][1:$-1];

  loop ( i, [1:iterations] ) {

    parallel( Map( inside.shape(),
                   blocks,
                   rectangular2 ) ) {
      do: update(
           inside[ paridx(0)-1 ][ paridx(1) ],
           inside[ paridx(0)+1 ][ paridx(1) ],
           inside[ paridx(0) ][ paridx(1)-1 ],
           inside[ paridx(0) ][ paridx(1)+1 ],
           inside[ paridx(0) ][ paridx(1) ]
         );
   }
}
```

Fig. 2. cSPC: Extracto de un ejemplo. Jacobi solver.

Las funciones que contienen información de coordinación van precedidas de un modificador que lo indica. En estas funciones sólo se pueden declarar y seleccionar partes de las estructuras de datos. Pero no manipular sus valores. Esto debe hacerse siempre llamando a funciones secuenciales. En el ejemplo, la función *jacobiSolver* es una función de coordinación. Recibe una matriz de dos dimensiones como parámetro de entrada/salida, y un número de iteraciones.

La selección de partes de un array se realiza con una notación similar a la de Fortran95, o a las extensiones para arrays de los compiladores de Intel. El símbolo $ indica el último elemento del espacio de índices en esa dimensión. En el ejemplo, *inside* representa la parte de la matriz que excluye los bordes de la misma. Es la región a actualizar por el solver.

El lenguaje de programación ofrece una visión global de las estructuras de datos. No hay detalles sobre la gestión de threads, o la comunicación entre procesos. El programador trabaja en términos de procesos lógicos, no físicos. El paralelismo se expresa con combinaciones jerárquicas de una primitiva paralela unificada. La primitiva que permite expresar paralelismo recibe como parámetro el resultado de una función de mapeo que no se resolverá hasta el momento de la ejecución, cuando tanto los tamaños y propiedades de las estructuras de datos, como los características

de la plataforma de ejecución son conocidos.

La primitiva de paralelismo junto con la función Map trabajan con tres niveles de abstracción, que se corresponden con los tres parámetros de la función Map. Se expanden tantos procesos lógicos en paralelo como se indique en su primer parámetro, que es un dominio de índices lógicos. Este dominio puede extraerse de una estructura de datos (para generar paralelismo de datos) o explicitarse de forma directa (más apropiado para paralelismo de tareas).

El segundo parámetro es el nombre de un módulo de *Layout*. Dichos módulos implementan funciones de partición y asignación del dominio de índices sobre la topología virtual de procesadores. Actualmente el sistema Trasgo incluye las técnicas de partición más comunes. Nuevas técnicas se pueden añadir como plug-ins en el sistema de ejecución.

El tercer parámetro es el nombre de una función de *Topología virtual*. Estas funciones reorganizan los elementos y dispositivo de proceso reales en una topología virtual donde se crean relaciones de vecindad de acuerdo con las reglas específicas de cada función. De nuevo se incluyen las más comunes con un interfaz que permite desarrollar nuevos plug-ins.

En el momento de la ejecución estas funciones determinarán la estructura de tareas y el nivel más apropiado de granularidad siguiendo las políticas incluidas en cada función.

En el ejemplo de la figura 2, dentro de la función de coordinación se ejecuta una primitiva *loop* que implementa un bucle secuencial en el que la variable i recorre el espacio de índices dado como segundo parámetro. En cada iteración de este bucle se utiliza una primitiva paralela cuya función Map lanza tantos procesos lógicos en paralelo como celdas hay en la parte interna de la matriz, utilizando una función de layout que agrupará los procesos lógicos en bloques rectangulares, sobre una topología virtual en forma de rectángulo. Dentro de la primitiva de paralelismo aparecen una o más clausulas *do* que indican el código que se asociará a los procesos lógicos. En este caso sólo aparece una clausula *do* cuyo contenido se replica en todos los procesos lógicos. La clausula contiene una llamada a la función secuencial para actualizar el valor de la celda asociada al proceso lógico. La función *parindex(dim)* devuelve el índice asociado al proceso lógico para la dimensión deseada. Cada proceso lógico tiene una copia virtual de las estructuras de datos, de forma que todas las actualizaciones se ejecutan en paralelo independientemente de las agrupaciones, o secuencializaciones generadas por la implementación internamente. La semántica de alto nivel de la primitiva paralela implica que los procesos se sincronizan creando un estado global al final de la misma, en cada iteración. El código generado no tiene porque cumplir con dicha restricción mientras la semántica se mantenga.

En el caso de repartir pocos elementos sobre un mayor número de procesadores virtuales, éstos se dividen en grupos, lo que permite construir particiones recursivas de forma natural. Algunas de las políticas de layout permiten hacer un balanceo de carga dinámico en función de pesos arbitrarios, lo que permite representar algoritmos con particiones de datos recursivas dependientes de datos. En la figura 3 se muestra el código de una función de coordinación que utiliza la primitiva paralela para mapear un espacio de sólo dos índices sobre grupos de procesadores. Esto genera una bipartición recursiva de los procesadores en grupos de diferente tamaño, ajustando las capacidades de cómputo a los tamaños de los subconjuntos de datos. La clausula *reduce* se utiliza al final de la primitiva de paralelismo para construir un estado global a partir de los datos modificados por cada proceso lógico. En el ejemplo se utiliza para llamar a una función que realiza un fusionado de las dos partes del conjunto de datos. Cuando sólo queda un procesador activo en un grupo, la recursión continúa secuencialmente de forma natural, al tener que ejecutarse los dos procesos lógicos generados en cada etapa en el mismo dispositivo.

```
coordination recursiveDecomp( double data[] ) {

  if ( count(data) >= 2 ) {

    double parts[2][];
    splitElementsInTwoSets( data, parts );

    double weights[2] = { count(parts[0]),
                          count(parts[1]) };

    parallel( Map( [2],
                   groupBalance(weights),
                   linear ) ) {
       do:
          recursiveDecomp( parts[0] );
       do:
          recursiveDecomp( parts[1] );
       reduce:
          mergeElements( parts );
  }
}
```

Fig. 3. cSPC: Extracto de un ejemplo. Partición recursiva.

III. El sistema de transformación Trasgo

La figura 4 muestra un esquema de las capas del sistema de trasformación Trasgo. A la izquierda aparecen las diferentes representaciones que se van utilizando para el código y a la derecha las diferentes capas de transformación que se aplican.

El front-end traduce el lenguaje de entrada (p.e. cSPC) a una representación interna similar al código fuente pero en un formato XML. Esta representación en forma de etiquetas es apropiada para un lenguaje jerárquico y estructurado, y además disponemos de potentes herramientas estándar para identificar y localizar características del código (XPath) y ejecutar transformaciones (Xslt) en documentos XML.

Estas tecnologías se utilizan para escribir de forma compacta y reutilizable módulos de transformación de código. La capa central de transformaciones consta de un analizador y reconstructor de expresiones que identifican las dependencias a partir de las primitivas de coordinación y de los parámetros de las

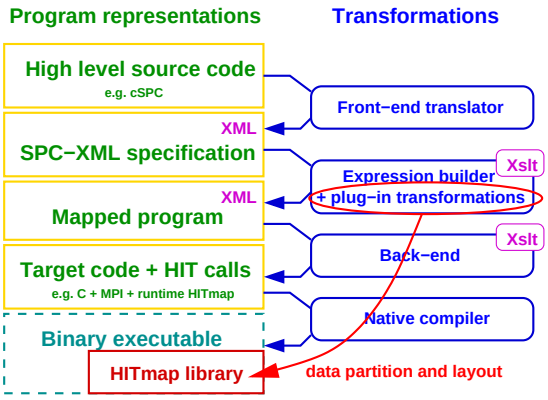


Fig. 4. Estructura del sistema de transformación.

llamadas a funciones. En versiones previas de Trasgo en esta capa se procedía también a generar expresiones y código para aplicar políticas de layout sobre topologías virtuales. En la versión actual de Trasgo esos módulos han sido movidos a la capa de run-time. Las estructuras de comunicación y/o sincronización también se construyen a partir de las dependencias de datos expresadas por la coordinación de tareas. Las expresiones relacionadas con el cálculo de comunicaciones llaman a las funciones de run-time para obtener la información necesaria, adaptándose en el momento de la ejecución a las partes de las estructuras de datos asignadas a cada procesador virtual por las políticas de layout. Este proceso es una generalización de las técnicas empleadas en el modelo poliédrico para generar código para sistemas de memoria distribuida [6].

El código transformado es reescrito por una capa de back-end en lenguaje C con llamadas a una biblioteca de funciones run-time denominada Hitmap, que incluye funcionalidades para la manipulación de estructuras de datos, módulos de mapeo (layout y topologías virtuales) y constructores de patrones de comunicación y sincronización. El código resultante se compila con un compilador nativo para generar el ejecutable.

IV. Biblioteca de funciones Hitmap

Hitmap [7] es una biblioteca de funciones para el mapeo automático y manipulación de arrays jerárquicos. Las versiones previas de Hitmap estaban orientadas al mapeo sobre un paradigma de paso de mensajes, más apropiado para sistemas de memoria distribuida. En esta sección se describen también las nuevas funcionalidades orientadas a mapear y manejar estructuras de datos entre los dispositivos dentro de un único nodo y cómo se integran con las anteriores.

A. Principales funcionalidades

En la figura 5 se muestra un esquema de los principales módulos de funcionalidad. Hitmap define objetos para declarar y operar con dominios de índices multidimensionales abstractos. Estos pueden usarse para declarar y seleccionar partes de arrays densos, u otras estructuras de datos dispersas [8]. Permite el manejo eficiente de tiles o particiones jerárquicas de dichas estructuras de datos. Incluye funciones para la declaración, reserva de memoria y liberación, copias entre zonas comunes seleccionadas, detección y construcción de *ghost zones* entre dominios de índices, etc.

Contiene también un sistema de plug-ins para la inclusión y utilización de módulos de construcción de topologías (para mapear los procesadores reales a una topología virtual) y de *layout* (para mapear dominios de índices sobre los procesadores de una topología virtual).

Finalmente, contiene también un conjunto de funciones para comunicar eficientemente tiles entre procesadores virtuales, y para construir patrones de comunicación reutilizables. Estas funciones utilizan internamente el estándar MPI para las comunicaciones. Su implementación explota técnicas eficientes de creación de tipos derivados y comunicaciones asíncronas. En Hitmap las comunicaciones se declaran en términos de procesadores virtuales y del resultado de un layout. Al cambiar la topología real de la plataforma de ejecución, o la política de layout aplicada, las estructuras de comunicación se calculan de forma diferente en tiempo de ejecución. De esta forma las estructuras de comunicación o sincronización se adaptan a la plataforma.

B. Módulos de mapeo

Los módulos relacionados con el mapeo se incorporan a la biblioteca a través de dos tipos de interfase. Uno para funciones de topología virtual y otro para funciones de layout. En el código se invocan por el nombre del plug-in.

En el caso de las topologías virtuales cada módulo puede definir los parámetros que el programador debe incluir en la llamada. En la mayor parte de los casos no es necesario ningún parámetro. El sistema incluye automáticamente un parámetro extra con una estructura que representa la información de la topología física. Esta información se obtiene en tiempo de ejecución y/o se completa con información de un fichero de definición de plataforma. El módulo devuelve una estructura que representa la asociación entre procesadores físicos y virtuales y las relaciones de vecindad entre procesadores. Actualmente Hitmap incluye módulos para construir topologías en forma de nube, cuadrados perfectos de procesadores, y varios tipos de paralelotopos multidimiensionales con diferentes restricciones.

En el caso de los módulos de layout el programa-

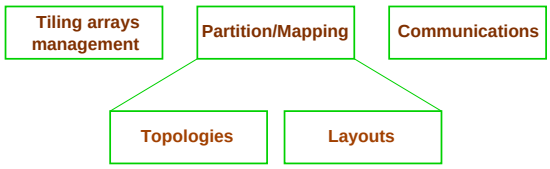


Fig. 5. Principales módulos de Hitmap.

dor debe suplir al menos dos parámetros obligatorios: Un dominio de índices y una topología virtual construida con alguna de las funciones anteriores. Cada módulo de layout puede definir otros parámetros necesarios. La función interna utilizará la información del tamaño de la topología virtual, sus relaciones de vecindad, y el dominio de índices para repartirlos. El resultado es un objeto con información sobre la zona de índices asignada al procesador local y métodos para obtener información de las zonas asignadas a otros procesadores o localizar índices. Actualmente Hitmap incluye módulos de layout para construir bloques multidimensionales, distribuciones cíclicas, balanceo de carga por pesos, particionado de grafos, etc.

C. Hitmap dentro de un nodo

Hitmap provee nuevas funcionalidades para manejar eficientemente los diferentes dispositivos dentro de un nodo en un clúster híbrido. Los diferentes procesos se asocian a dispositivos: Se asignan grupos de núcleos a un proceso de paso de menajes, y se asocia cada dispositivo acelerador a otro proceso de paso de mensajes. La comunicación entre dispositivos o grupos de dispositivos se ejecuta al mismo nivel que entre procesos de diferentes nodos. Internamente la comunicación entre los procesos de un mismo nodo se optimiza a través de la memoria compartida.

Se incluye en la nueva versión de Hitmap un nuevo tipo de módulos de mapeo: *Blocking/Tiling*. Estos módulos reciben un subdominio local y construyen la información necesaria para generar una rejilla de bloques o tiles de un tamaño/grano apropiado para computar de forma paralela en un conjunto de núcleos, o en el dispositivo acelerador asociado al proceso. El módulo utiliza la topología virtual para detectar el dispositivo asociado al proceso. En el caso de aceleradores como GPUs, el resultado indica los tamaños y formas de los bloques de threads. Cada módulo puede incluir diferentes políticas que el programador puede seleccionar en función del tipo de kernel, función secuencial o código a ejecutar, de los patrones de acceso, del uso de las memorias caché, etc.

Finalmente, en el caso de grupos de núcleos, se utiliza la información suministrada por el módulo para lanzar en paralelo tareas que subseleccionan los tiles correspondintes a cada bloque y ejecutan la función deseada utilizando alguna herramienta como OpenMP, TBBs, Cilk, o similar. En el caso de aceleradores, la comunicación de estructuras de datos (tiles) entre la memoria global y el dispositivo queda oculta en la llamada a las funciones. Se define un método común de lanzamiento de kernel o código, que internamente detecta el tipo de dispositivo asociado y ejecuta las acciones especiales necesarias para lanzar un kernel en el caso de una GPU.

D. Combinación de niveles

Las técnicas de partición se pueden combinar en diferentes niveles de forma natural. Dependiendo del tipo de aplicación puede resultar más natural una aproximación top-down o una bottom-up.

Por ejemplo, se puede usar primero una técnica de layout para distribuir un dominio entre procesos, y luego aplicar una técnica de blocking/tiling sobre el subdominio local del dispositivo o grupo de dispositivos. Por el contrario, se pueden calcular previamente los tamaños de los tiles y utilizar una técnica de layout para distribuir los bloques o tiles a lo largo de un conjunto de procesos.

También es posible anidar o construir estructuras recursivas que utilicen layouts en múltiples niveles para generar particiones recursivas a lo largo del conjunto de procesos.

V. Casos de estudio

Hemos seleccionado dos aplicaciones sencillas para probar la aplicación de estas combinaciones en clusters híbridos. La primera es el solver iterativo para PDEs basado en el método de Jacobi presentado en la figura 2. En este caso, en el código generado se utiliza primero un layout de bloques rectangulares para distribuir la estructura de datos matricial entre los procesos de paso de mensajes. Se utiliza posteriormente un módulo de Blocking básico para crear un tiling adecuado para los grupos de cores o GPUs. Los procesos asignados al mismo nodo se sincronizan en cada iteración para hacer las copias de las *ghost zones* o zonas fantasma entre diferentes procesos. Los procesos en diferentes nodos vecinos se comunican las zonas fantasma asíncronamente, de la forma habitual.

El segundo ejemplo es una factorización LU. En este caso, en el código generado se utiliza una técnica de Blocking/Tiling que calcula un tamaño de bloque genérico apropiado para comunicar entre los los diferentes dispositivos o grupos de dispositivos. Estos bloques se distribuyen a lo largo del conjunto de procesos utilizando una técnica de layout cíclica. El resultado es una distribución bloque-cíclica adaptada a las características de la plataforma de ejecución. Dentro de un nodo los procesos se sincronizan para copiar datos de las zonas compartidas. Entre nodos los procesos se comunican con operaciones colectivas mandando y recibiendo bloques o grupos de bloques según los patrones del algoritmo.

La biblioteca Hitmap ha demostrado previamente una buena eficiencia y escalabilidad en sistemas de memoria distribuida y compartida utilizando el paradigma de paso de mensajes [7]. Los experimentos preliminares con el nuevo sistema de capas incluyendo las abstracciones que se utilizan para los dispositivos dentro de un mismo nodo demuestran que la sobrecarga introducida por estas nuevas abstracciones es pequeña y fija, no afectando negativamente a la escalabilidad. En el caso de grandes cargas computacionales la sobrecarga es despreciable.

VI. Conclusión

En este trabajo se presentan innovaciones en el desarrollo de Trasgo, un sistema de programación capaz

de generar códigos eficientes y adaptables a partir de código abstracto y portable. Se muestra como ésto se consigue gracias a mover decisiones de mapeo a tiempo de ejecución dejando que las estructuras de comunicación y sincronización dependan de los resultados de las mismas. El sistema de generación de código se apoya en Hitmap, una biblioteca run-time que provee de las herramientas y utilidades necesarias para el mapeo en tiempo de ejecución. Se muestran nuevas técnicas orientadas a explotar múltiples niveles de paralelismo en clusters híbridos y/o heterogéneos.

El marco de trabajo planteado por el sistema Trasgo presenta aún varios retos importantes. Entre ellos se pueden destacar los siguientes. Incluir políticas automáticas de asignación de grupos de dispositivos a procesos de paso de mensajes para equilibrar la capacidad de computo. La integración de un sistema de generación automática del código de kernels para GPU u otros aceleradores a partir del lenguaje de entrada. Políticas de blocking/tiling que utilicen una mejor caracterización de los códigos para derivar tamaños de bloque apropiados. O la inclusión de técnicas de partición y layout irregulares que adapten la carga a las capacidades de diferentes dispositivos.

Agradecimientos

Este trabajo ha sido financiado parcialmente por el Ministerio de Industria (CENIT OCEANLIDER), Ministerio de Economía y programa FEDER de la Unión Europea (red CAPAP-H3 TIN2010-12011-E, red CAPAP-H4 TIN2011-15734-E, Proyecto MOGECOPP TIN2011-25639), y el proyecto HPC-EUROPA2 (project number: 228398), con el apoyo de la Comisión Europea (Capacities Area - Research Infrastructures Initiative).

Referencias

[1] A. Gonzalez-Escribano and D.R. Llanos, "Trasgo: A nested-parallel programming system," *The Journal of Supercomputing*, vol. 58, no. 2, pp. 226–234, 2011.

[2] A.J.C. van Gemund, "The importance of synchronization structure in parallel program optimization," in *Proc.* 11th *ACM ICS*, Vienna, Jul 1997, pp. 164–171.

[3] K. Lodaya and P. Weil, "Series-parallel posets: Algebra, automata, and languages," in *Proc. STACS'98*, Paris, France, 1998, vol. 1373 of *LNCS*, pp. 555–565, Springer-Verlag.

[4] C. Bastoul, "Code generation in the polyhedral model is easier than you think," in *Proc. PACT'04*. 2004, pp. 7–16, ACM Press.

[5] A. Moreton Fernandez, A. Gonzalez-Escribano, and D.R. Llanos, "Trasgo frontend: Hacia una generación automática de código paralelo portable," in *Proc. Jornadas Paralelismo'12*, 2012.

[6] U. Bondhugula, "Automatic distributed memory code generation using the polyhedral framework," Tech. Rep. IISc-CSA-TR-2011-3, IISc, 2011.

[7] A. Gonzalez-Escribano, Y. Torres, J. Fresno, and D.R. Llanos, "An extensible system for multilevel automatic data partition and mapping," *IEEE TPDS*, 2013 (to appear).

[8] J. Fresno, A. Gonzalez-Escribano, and D.R. Llanos, "Extending a hierarchical tiling arrays library to support sparse data partitioning," *The Journal of Supercomputing*, vol. 64, no. 1, pp. 59–68, 2012.

Codificación de Vídeo Escalable usando Motion Compensated JPEG2000 con Control de Bit-Rate

Carmelo Maturana-Espinosa, Joaquín García-Sobrino, J.J. Sánchez-Hernández y Vicente González-Ruiz

Dpto. de Informática. Grupo SAL. Universidad de Almería, Campus de Excelencia Internacional Agroalimentario, ceiA3. [1]

Palabras clave: Control del Bit-Rate, JPEG2000, Filtrado Temporal con Compensación Movimiento.

Hoy en día los dispositivos de codificación de vídeo necesitan realizar transmisión de datos hacia una gran variedad de terminales, posiblemente con diferentes resoluciones de pantalla, distinta capacidad de computación y a través de redes con ancho de banda variable. La codificación de vídeo escalable es una alternativa para soportar de forma óptima estas aplicaciones. Por otra parte, el control de bit-rate y la calidad de la imagen se han convertido en uno de los retos más importantes en la transmisión de vídeo ya que su adecuado tratamiento permite maximizar la calidad de los vídeos reconstruidos. En este trabajo estudiamos el problema de control de bit-rate para secuencias de imágenes comprimidas usando Motion JPEG2000 y estimación de movimiento. Proponemos una técnica iterativa y evaluamos su eficacia en comparación con el control de bit-rate que realiza el codec H.264/SVC, obteniendo resultados que avalan nuestra propuesta.

I. Introducción

Los avances tecnológicos y los estándares de codificación de vídeo, unidos al rápido desarrollo y mejoras de las infraestructuras de red, capacidad de cómputo y almacenamiento, han permitido un rápido incremento del número de aplicaciones que manejan vídeo. Esto, unido a proliferación de servicios multimedia en Internet y al auge que en los últimos años han experimentado las redes inalámbricas de banda ancha, han hecho que la comunicación y transmisión de vídeo a través de la red se haya convertido en un foco de interés tanto para la industria como para el mundo académico.

La transmisión de vídeo en tiempo real requiere una calidad de servicio que generalmente esté condicionada por: el ancho de banda, el retraso en la entrega y los errores de transmisión. La transmisión de vídeo presenta, por norma general, una serie de requisitos mínimos en cuanto a ancho de banda para conseguir una calidad aceptable de presentación. Por otro lado, la transmisión de vídeo requiere estrictos requisitos en cuanto al retraso en la entrega. Si los paquetes que componen el vídeo no llegan de la forma adecuada, la reproducción sufrirá pausas no deseadas. Por último, las aplicaciones de vídeo normalmente imponen límites en cuanto al número de errores permitidos ya que demasiados errores degradarían seriamente la calidad de reproducción. Además, en los casos de multidifusión de vídeo, la heterogeneidad de los receptores hacen difícil conseguir la eficiencia y flexibilidad requerida. Por lo tanto, es deseable contar con esquemas de codificación de vídeo escalable que permitan adaptarse a la situación concreta de las condiciones en las que se realiza la transmisión / reproducción.

Un esquema de codificación de vídeo escalable consiste básicamente en comprimir una secuencia vídeo en varios sub-streams. Uno de estos sub-streams se considera el sub-stream base que puede ser independientemente descodificado y proporciona generalmente una calidad visual de reproducción básica. Los demás sub-streams son mejoras del sub-stream base y pueden ser solamente descodificados junto con el sub-stream base para proporcionar una mejor calidad visual del vídeo. La descodificación del sub-stream base junto con determinadas combinaciones de los demás sub-streams produce secuencias de vídeo con mayor o menor pérdida de calidad, distinto tamaño de las imágenes del vídeo y diferente frame-rate. Esto es lo que se conoce como escalabilidad en calidad, espacial y temporal, respectivamente.

La codificación de vídeo escalable puede ayudar al BRC (Bit-Rate Control) debido a la capacidad de reconstrucción a distintas resoluciones y/o calidades a partir de bit-streams incompletos. Esta capacidad permite una fácil y rápida adaptación a las capacidades concretas de cada red y/o terminal. Se pueden conseguir diferentes formas de escalabilidad según el estándar de codificación usado. En nuestro caso, nos centramos en el estándar JPEG2000 porque nos permite conseguirlas todas.

El resto del Paper está organizado como sigue. En el Apartado II se presenta una visión general de JPEG2000, Motion JPEG2000 y H.264/SVC. Éste último se ha usado para compararlo a nivel de eficiencia con nuestra propuesta. Una breve descripción del codec MCTF+JPEG2000 básico (MCJ2K) además del algoritmo propuesto de un eficiente BRC para MCJ2K, se muestra en la Sección III. En la Sección

[1]Este trabajo ha sido financiado por subvenciones del Ministerio de Ciencia e Innovación de España (TIN2008-01117), la Junta de Andalucía (P08-TIC-3518 y P10-TIC-6548), y en parte financiado por el Fondo Europeo de Desarrollo Regional (ERDF).

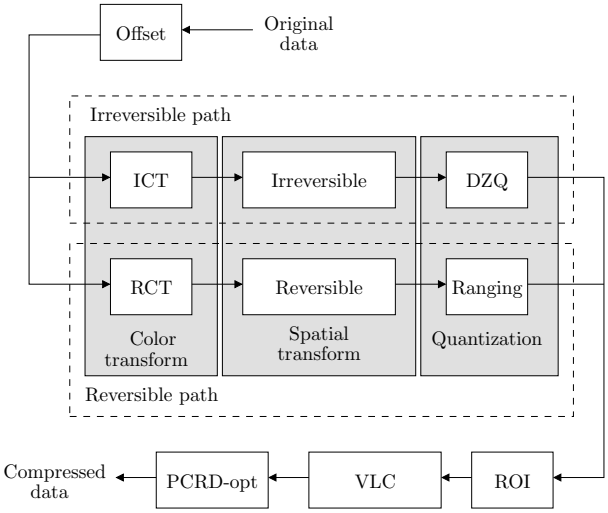


Fig. 1. Diagrama de bloques del compresor JPEG2000.

IV presentan los resultados experimentales. Por último, en la Sección V se muestran las conclusiones a las que llegamos en el estudio.

II. Trabajo relacionado

JPEG2000 [1], [2] es el estándar ISO/ITU más reciente de compresión de imágenes. El objetivo de este apartado es sólo proporcionar una breve descripción a muy alto nivel del codec JPEG2000 para el posterior entendimiento los siguientes apartados de este trabajo.

En la Figura 1 se muestra una versión simplificada del diagrama de bloques del codificador JPEG2000. En primer lugar, según se desee aplicar el codec con o sin pérdida se toma uno de los caminos representados. El primer paso, llamado Offset normaliza las muestras de cada bloque entre [-0.5, 0.5[provocando una pérdida irreversible en los datos, o teniendo encuenta el nº de bits/componente para permitir una decodificación reversible. A continuación, cada una de estas particiones se somete de forma opcional a una transformación de componentes con el objetivo de conseguir descorrelacionar las componentes de color. Dicha transformación de color puede ser irreversible (ICT), donde las componentes se representan en formato YCbCr. En caso de hacerse de forma reversible (RCT) la transformación se hace en un dominio de enteros, en el que se puede invertir las operación sin perdida de precisión. En cualquier caso a las componentes resultantes se les aplica la Transformada Wavelet (llamada transformada espacial en la Figura 1) y después son cuantificadas, eliminando generalmete ruido de alta frecuencia.

La siguiente etapa, consiste en la definición de la(s) region(es) de interés (ROI). Si las hubiera, los bloques comprendidos en dicha región serian extraídos en mayor medida que el resto, aportando una mayor calidad de ellos, haciendo uso de la codificación de longitud variable (VLC) que hace efectiva la compresión de los datos. Esto es, cada sub-banda se divide en codeblocks rectangulares, que se comprimen de forma independiente usando un codificador bitplane que hace tres pasadas sobre cada bitplane de cada codeblock consiguiendo como resultado un bitsream truncable de cada codeblock. Por último la etapa PCRD-opt (Post Compression Rate-Distortion Optimization), organiza los datos en el code-stream con el objeto de conseguir las diferentes formas de escalabilidad. Así durante la descompresión, para la escalabilidad en calidad, por ejemplo, las sub-bandas deben ser decodificadas usando un orden LRCP (capas de calidad, resolución, componentes y precintos) ya que es el orden de paquetes que guarda mejor relación entre el tamaño del code-stream y calidad de la imagen descomoprimida usando dichos paquetes.

El comité JPEG amplió el estándar JPEG2000 para la codificación de vídeo. El resultado se conoce como Motion JPEG2000 (MJ2K). MJ2K no incluye compensación de movimiento (cada frame se comprime de forma individual) y está orientado a generar vídeo comprimido altamente escalable que puede ser fácilmente editado.

H.264/AVC [3], [4] es un codificador de vídeo estándar que proporciona gran eficiencia de compresión. La versión escalable de H.264/AVC es H.264/SVC. En SVC se permite escalabilidad temporal, espacial y en calidad.

III. Propuesta

Son muchos los esfuerzos destinados al desarrollo y mejora de sistemas de codificación de vídeo eficientes. Estos estudios consiguen buenos resultados centrándose en características concretas de la "codificación ideal" de un vídeo como pueden ser un buen acceso aleatorio al contenido [5] o portabilidad en la transmisión por un canal estrecho [6]. Sin embargo, parece evidente que el mayor número de ventajas se consigue por medio de compensación temporal de movimiento, donde por ejemplo, puede llegar a conseguirse una notable reducción del bit-rate necesario para llevar a cabo la codificación de un vídeo. Aunque se trata de un aspecto importante de mejora del rendimiento de compresión, sigue siendo un proceso que no cuenta una metodología estándar. Existen numerosos estudios que aplican alguna estrategia de predicción de movimiento usando técnicas como MCTF [7] (Motion Compensated Temporal Filtering) o LIMAT [8] (Lifting-based Invertible Motion Adaptive Transform) y que muestran resultados interesantes.

Como se observa en las dependencias de la Figura 2, MCTF provee de una escalabilidad temporal y controles para minimizar el impacto de errores y pérdidas en los datos. El resultado de aplicar MCTF a una secuencia de imágenes es otra secuencia de imágenes filtradas en el dominio del tiempo, que pueden ser comprimidas eficientemente por un compresor de imágenes MJ2K.

En este trabajo se propone usar una técnica novedosa a la que hemos denominado MCJ2K (Motion Compensated JPEG2000), pretende aunar las ventajas de MCTF y Motion JPEG2000 en la com-

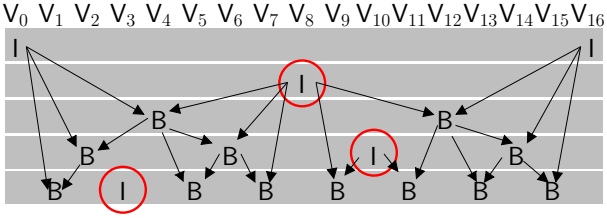


Fig. 2. MCTF sin suficiente correlación temporal.

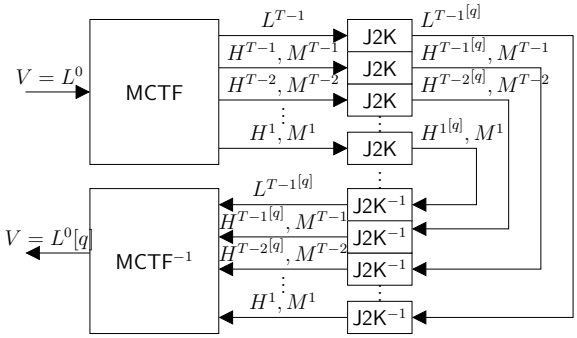


Fig. 3. Estructura básica de MCJ2K.

presión de secuencias de imágenes con compensación de movimiento.

MCJ2K proporciona compresión escalable en espacio, tiempo y calidad mediante la distribución de los datos en capas de calidad y la posterior ordenación de los mismos. Durante la descompresión para la escalabilidad en calidad, las imágenes de las sub-bandas temporales deben ser decodificadas usando un orden LRCP. Cada sub-banda temporal será un code-stream diferente, comprimido de forma independiente con MJ2K. El esqueleto de MCJ2K se puede ver en la Figura 3, donde el vídeo original es divido en sub-bandas temporales mediante el algoritmo de MCTF y cada una de las imágenes residuo (B) en cada sub-banda es comprimida usando J2K. Para la descompresión se invierte el proceso dando lugar al vídeo reconstruido, a partir de vectores de movimiento (M) comprimidos sin pérdidas (habitualmente), imágenes comprimidas con pérdidas (generalmente) intra (I) y residuos.

Es importante precisar que la implementación de MCTF debe sufrir ciertas modificaciones para adaptarse al esquema inicial (Figura 3) y siga una típica implementación recursiva "filter bank", con transformaciones hacia adelante y hacia atrás. Por ejemplo, en caso de no encontrar suficiente correlación temporal, las imágenes B son reemplazadas por I y las referencias en los vectores de movimiento se eliminan (Figura 2).

A. *El code-stream MCJ2K*

Al aplicar el algoritmo de compresión de vídeo de MCTF, el code-stream generado, es compuesto de múltiples conjuntos de información. Como se observa en la Figura 2 se obtienen tantas sub-bandas de información sobre texturas como número de niveles de resolución temporales (TRL) indicados para la compresión. Por otra parte, la información del movimiento se distribuye en tantos campos de movimiento como sub-bandas de altas frecuencias. El code-stream se compone de la siguiente información:

1. **Texturas:** Comprenden la secuencia de imágenes del vídeo divididas en sub-bandas. Cada sub-banda está compuesta por imágenes I y/o residuos B, que son comprimidos con un nº concreto de capas de calidad, de manera que se puede enviar sólo el nº de capas deseado de una sub-banda. Cada capa adicional proporciona información más precisa de las imágenes de la sub-banda. No hay límite en el nº de capas con la que se desee crear una sub-banda, pero un nº demasiado alto provoca ineficiencias, ya que cada capa necesita de una cabecera. Además puede que la información de la sub-banda no se pueda dividir en más partes (capas) y entonces se creen capas vacías sin información útil, sólo con cabeceras, siendo una capa que únicamente engorda el code-stream. En la Figura 3 podemos observar que existen dos tipos de sub-bandas, la L y las H, según el rango de frecuencia de las texturas que almacenan.
   (a) **L (Low):** Esta sub-banda no se relaciona con información de movimiento, siendo totalmente independiente del resto. Contiene sólo imágenes I.
   (b) **H (High):** Son generadas TRL - 1 sub-bandas, que contienen las imágenes residuos producidas por el algoritmo MCTF. Dichas imágenes sólo contienen texturas de altas frecuencias.
2. **Movimiento:** Para cada sub-banda con altas frecuencias, existe un conjunto de vectores de movimiento, comprimidos sin pérdida. Éstos indican la posición de cada uno de los bloques de cada imagen B, en cada sub-banda H. Para nuestra propuesta será considerado cada campo de movimiento como una capa de calidad.

B. *Reconstrucción de un code-stream incompleto*

Este repertorio de sub-bandas, campos de movimiento y capas hace posible un envío selectivo y óptimo del code-stream cuando es necesario truncarlo. Así en situaciones de transmisión reales, donde se produce un truncamiento del code-stream o incluso la pérdida completa de una sub-banda, el descompresor necesita poder reconstruir el vídeo usando un code-stream incompleto. En nuestra propuesta, en caso de faltar uno(s) campo(s) de movimiento, se usa la compensación del movimiento para interpolar los datos que faltan. De manera que, el descompresor interpola linealmente los frames que no son recibidos, manteniendo constante el framerate. Si los datos que faltan son una(s) sub-banda(s), las texturas no recibidas se interpretan como negro y se mezclan con las texturas de las que sí se dispone.

C. Un algoritmo de Control del Bit-Rate (BRC)

En este trabajo se evalúa el uso de una técnica de control del bit-rate que permite maximizar la calidad del vídeo descomprimido en un entorno de transmisión de ancho de banda inferior al necesario para enviar el vídeo sin truncar. La disminución de la distorsión que produce cada capa (entiéndase como un campo de movimiento o capa de calidad de una sub-banda), es aditiva [9], es decir, podemos calcular la distorsión (o calidad) de un vídeo reconstruido, como la suma de las calidades que proporciona cada capa de cada sub-banda o campo de movimiento de forma independiente, lo mismo ocurre entre las capas de calidad de cada imagen. Nuestro algoritmo BRC, se sirve de esta característica para conocer cual es el beneficio real (disminución de distorsión frente a aumento de bit-rate) de cada capa a la hora de la reconstrucción. Así, según nuestro algoritmo BRC, la selección del orden óptimo de entre todas las capas se lleva a cabo tratando de minimizar la curva Rate/Distorsión (R/D) del vídeo reconstruido, siendo R su bit-rate y D su error cuadrático medio (RMSE). Si la selección de capas es adecuada, cada capa adicional enviada al descompresor proporciona una disminución en la distorsión y un aumento en el bit-rate consumido. Es decir, añadir una capa a la reconstrucción queda representado en una gráfica R/D como un punto que se sitúa más a la derecha (más bit-rate) y más abajo (menos distorsión) que el punto anterior que carecía de dicha capa. Así la pendiente, (*slope* en el Algoritmo 1) en una curva R/D debería ser monótonamente no creciente, con lo que dibuja un triángulo rectángulo entre cada par de puntos, permitiendo calcular su inclinación. A mayor inclinación, mejor opción constituye la capa analizada. La labor de nuestro algoritmo BRC consiste en calcular las inclinaciones derivadas de añadir una capa al code-stream en cada sub-banda y campo de movimiento, para seleccionar la mejor, obteniendo así un nuevo code-stream con dicha capa adicional. Este nuevo code-stream se usa entonces para la siguiente iteración de este proceso de cálculo y selección.

En el pseudocódigo del Algoritmo 1 se describe este proceso, así como los parámetros de entrada y salida en la Tabla 1. El parámetro L^0 representa el vídeo original y t su duración, ésta es necesaria para calcular el valor real del ancho de banda consumido por un code-stream y así poder comparar con el ancho de banda disponible (B). Para la aplicación de nuestro algoritmo de BRC se debe disponer del code-stream completo (C), es decir, no truncado, generado previamente mediante la compresión del vídeo original. Esta compresión necesita de parámetros que también son necesarios para poder truncar y reconstruir dicho code-stream, éstos son, entre otros menos relevantes, el nº de TRL (T) y el nº de capas con las que se comprimió cada sub-banda y campo de movimiento, representado como ($Q[]$), este parámetro es un vector de enteros donde las primeras (T) posiciones hacen referencia a las sub-bandas de texturas, siendo éstas por ejemplo: L, H2 y H1 para un T=3. Las siguientes *T-1* posiciones hacen referencia a los campos de movimiento correspondientes a H2 y H1, en el ejemplo dado. La finalidad del algoritmo consiste en devolver un fichero (*E_list*) con el orden óptimo de envío de las capas.

En la *línea 1* del Algoritmo 1 se inicializa el parámetro *V[]*, el cual tiene la misma estructura que el parámetro *Q[]*, con la salvedad de que en este último se indican el nº de capas del code-stream sin truncar (C) y en *V[]* se indican el sub-conjunto de capas de C que constituye un code-stream truncado (*C_part*). En un principio se considera que no se está enviando ninguna capa, quedando representado por un valor igual a [0, ···, 0] en *V[]*. La estructura de datos *E[]* contiene un determinado *V[]*, que también se inicializa, y más datos calculados sobre la reconstrucción del vídeo (L^q) según las capas que dicho parámetro *V[]* indica, como son: la pendiente (S), el consumo de bit-rate (R) y la distorsión (D). En esta misma línea de pseudocódigo se representa la estructura de datos *E[V[], S, R, D]* como *E[V[]]* para indicar que el valor que se está inicializando es *V[]*. Lo mismo ocurre en la *línea 2* donde la misma estructura se representa como *E[S, R, D]* indicando que los valores asignados son sólo estos tres. Dicha inicialización consta de tres asignaciones: la pendiente, que se iguala a un valor mínimo, como podría ser 0 o un nº negativo; el bit-rate que se igualaría a 0, al no haberse transmitido aún nada, no hay ancho de banda consumido; y por último la distorsión que debe ser ahora máxima, dado que al no enviar nada la reconstrucción será peor en calidad que la que pueda generar cualquier truncado del code-stream.

En la *línea 3* se encuentra el bucle que condiciona el nº de iteraciones del algoritmo de BRC, el cual es útil iterar mientras queden capas para enviar y se disponga de ancho de banda B. Para la primera condición, se conocen el nº de capas totales en el parámetro *Q[]*, donde siempre debe existir alguna sub-banda (posición del vector) que sea mayor estricto que su correspondiente sub-banda o campo de movimiento en *V[]*.

En el bucle *for* se evalúan tantos *V[]*, entonces llamados *V_test*, como sub-bandas y campos de movimiento contenga el code-stream. Cada *V_test* consiste en la adición de una capa al code-stream representado por *V[]*. Así iterativamente de las líneas *7* a la *10* la función *extracting_layers* extrae code-streams truncados (*C_part*), que son reconstruidos (L^q) para calcular el ancho de banda usado, la distorsión y la pendiente del *V_test* evaluado.

En la *línea 12* la estructura de datos *E[]* se mantiene actualizada conteniendo el *V[]* y demás datos asociados a la mayor pendiente (S) encontrada hasta el momento. De forma que tras finalizar dicho *for* la mejor opción, entiéndase como el *V_test* que proporciona el code-stream que genera el vídeo reconstruido con una mayor pendiente en una curva R/D, es almacenada, en el fichero *E_list* (*línea 16*) para su posterior consulta durante el envío.

Input	
L^0	Original video
t	Duration of L^0
B	Available bandwidth
C	Complete code-stream
T	Number of TRL of the C
$Q[]$	Total number of layers of each sub-band
$Q[_i]$	Total number of layer of a sub-band
Output	
E_list	File with the list of optimal order
Temporal Structures	
C_part	Truncated one part C
L^q	Reconstructed video from C_part
$V[]$	Number of layers in each sub-band of a C_part
$V\_test[]$	Another instance of V[]
$V[_i]$	Number of layers in a sub-band
S, R, D	Slope, Bit-Rate and Distortion of a L^q
$E[V[], S, R, D]$	Data of a particular C_part and L^q
min, max	Values for S and D

Table 1: Parámetros.

```
1  E[V[]] = V[] = create_V (T);
2  E[S, R, D] = min, 0, max;
3  while exits_layers_unsent (Q[], V[]) and B do
4  |   V[] = E[V[]];
5  |   for i=1 until i=T do
6  |   |   if Q[i] > V[i] then
7  |   |   |   V_test[] = add (V[], V[i] + 1);
8  |   |   |   C_part = extracting_layers (V_test[], C);
9  |   |   |   L^q = decompressor (C_part, T);
10 |   |   |   S, R, D = info (L^0, t, L^q, E[R, D]);
11 |   |   |   if S > E[S] then
12 |   |   |   |   E[] = V_test[], S, R, D;
13 |   |   |   end
14 |   |   end
15 |   end
16 |   E_list = save (E[]);
17 end
```

Algorithm 1: Pseudocódigo BRC.

Notar que el primer code-stream que es seleccionado para ser enviado, consta de una sola capa de la sub-banda L. Como puede verse en la Figura 2 la sub-banda L compuesta sólo por imágenes I, no depende de ninguna otra sub-banda, y por ello, la primera opción resulta ser siempre ésta.

IV. Evaluación

La evaluación del algoritmo BRC se ha llevado a cabo con code-streams generados por MCJ2K. Además para ubicar la relevancia de los resultados se han comparado las curvas R/D dadas por el codec H.264/SVC. Para proporcionar una comparación realista se han considerado todas las posibles variables que afectan a ambos codec, con el fin de igualar condiciones. Así se han usado igual nº de TRL e imágenes, tamaño de GOP, ancho de banda y escalabilidad (que ambos codec permitan truncar el code-stream el mismo nº de veces). Aunque en este último aspecto, SVC no nos ha proporcionado tanta escalabilidad como MCJ2K en resoluciones altas, como es FULL-HD, ya que produce un error de computo al intentar codificar un vídeo a más de 4 puntos de truncadoEn resoluciones pequeñas (CIF), representada en la Figura 4 Fila 1, si se ha obtenido una escalabilidad comparable entre ambos codec.

En la Figura 4 se muestran los resultados de la evaluación de nuestro algoritmo BRC. En cada gráfica se muestran 3 curvas R/D. Dos curvas del codec MCJ2K *con* y *sin* BRC, en comparación con la curva dada por H.264/SVC. A excepción de las gráficas de la Figura 4 Fila 2, donde en lugar de una curva para SVC, se muestran dos, con 3 y 4 puntos de truncado, con intención de dar una visión del codec SVC más detallada, dado que no se ha podido equiparar la escalabilidad con MCJ2K en resolución alta. Las curvas para MCJ2K *sin* BRC se han obtenido extrayendo el mismo nº de capas de todas las sub-bandas, siendo el primer punto de truncado la reconstrucción de sólo las primeras capas. Esto es equivalente a codificar y extraer el vídeo completamente con un determinado nivel de cuantificación.

Como puede verse en la Figura 4 la aplicación de nuestro algoritmo de BRC siempre mejora la eficiencia en el envío de datos del codec MCJ2K. Esta mejora es máxima, a mínimo ancho de banda y nula en el máximo ancho de banda, coincidiendo así los últimos puntos de las curvas R/D de *con* y *sin* BRC. Esto se debe a que en máximo bit-rate se envían todas las partes del code-stream, resultando la reconstrucción completa del mismo. El decremento en esta eficiencia, respecto de no usar BRC, no es lineal, ya que el aporte de cada capa depende de parámetros de entrada y de las propias características del vídeo.

La cuantía de los beneficios de la optimización presentada aumentan de forma lineal al nº de TRLs y capas usado para la compresión (en nuestros ejemplos usamos 8 capas de calidad para las texturas y 1 para los vectores de movimiento), puesto que al estar compuesto de un mayor nº de partes, su ordenación y posibilidad de no envío de algunas de ellas tiene mayor relevancia. Esto queda de manifiesto en la Figura 4 Columna 1, donde a 3 TRL el ancho de banda mínimo necesario para transmitir cada vídeo (CIF y FULL-HD, respectivamente) se reduce al usar BRC de 123.264 a 68.112 Kbps y de 301.824 a 228.816 Kbps. Estas mejoras aumentan en los ejemplos a 5 TRL (véase la Figura 4 Columna 2), donde el mínimo ancho de banda necesario al usar BRC pasa de 103.538 a 20.724 kbps (vídeo CIF), y de 184.602 a 66.112 kbps (vídeo FULL-HD). Es decir, se consigue reducir a menos de la mitad el ancho de banda necesario para reconstruir los vídeos a su mínima calidad disponible, haciendo posible su transmisión bajo restricciones de bit-rate donde antes no se podría reproducir nada.

Desde el punto de vista de la calidad del vídeo, se puede valorar que todo este bit-rate ahorrado puede ser invertido en disminuir la distorsión usando un bit-rate similar. Así especialmente para 5 TRL (Figura 4 Columna 2), usando un ancho de banda similar obtenemos una disminución de la distorsión, gracias al uso del BRC, desde 28.034 a 22.684 dB y de 10.372 a 8.958 dB, para los vídeos a baja y alta resolución respectivamente.

Además el BRC presentado consigue explotar toda la escalabilidad del codec MCJ2K. Esto puede observarse en la Figura 4, comparando el mayor nº puntos en las curvas *con* que *sin* BRC. El nº de puntos de truncado para todos los code-stream de MCJ2K sin

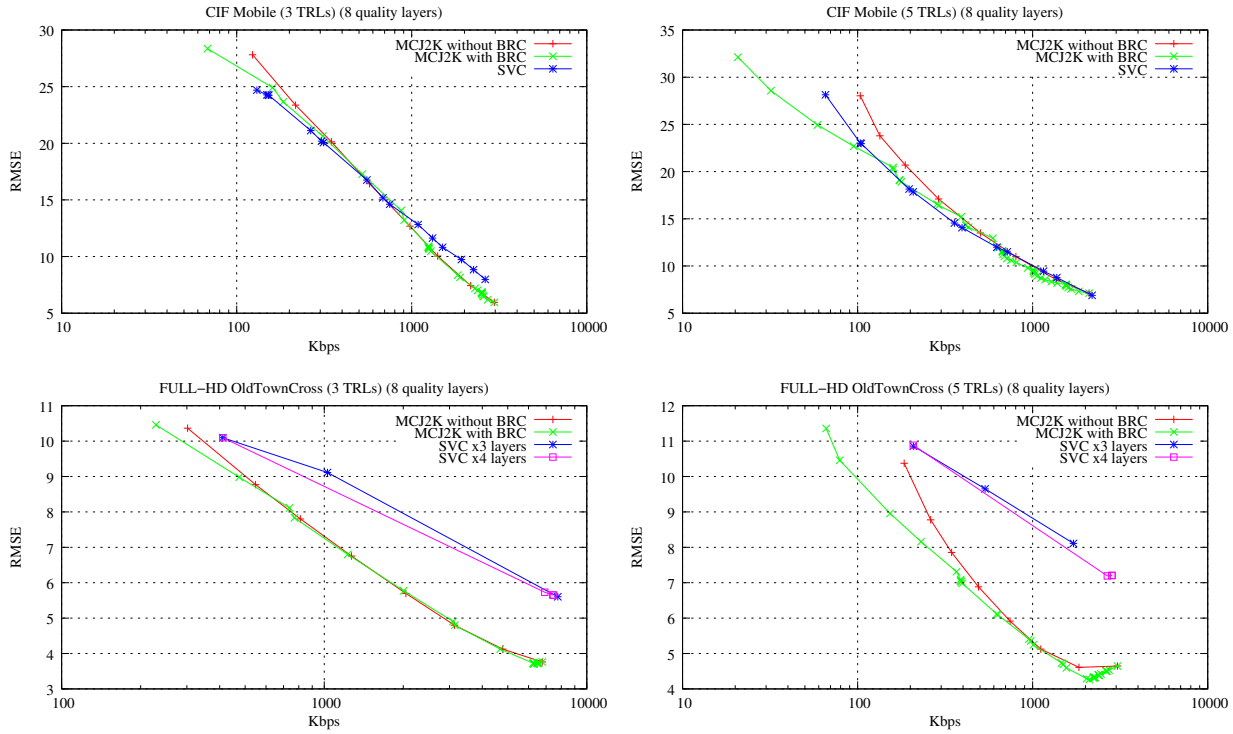


Fig. 4. Comparación entre nuestro BRC y el de H.264/SVC. Vídeos de baja y alta resolución (filas), de 3 y 5 TRL (columnas).

BRC es de 8. Gracias al uso del BRC se consiguen 26 y 44 puntos de truncado para 3 y 5 TRL, respectivamente. En el caso de SVC se obtienen 16 y 12 puntos de truncado a baja resolución con 3 y 5 TRL, respectivamente. Y 4 puntos a alta resolución, sea cual sea el valor de TRL. Así, especialmente en altas resoluciones la escalabilidad de los code-stream generados por MCJ2K con BRC, mejora un 69% y 81% para resoluciones bajas y altas, respectivamente. Esto representa una mayor escalabilidad de MCJ2K con BRC respecto a SVC de: 38% y 73% en los ejemplos de la Figura 4 Fila 1. Y aumento de un 85% y 91% para la Fila 2.

V. Conclusiones

MCJ2K carecía de un control de bit-rate que le permitiera optimizar la calidad de sus reconstrucciones en code-stream escalables, como sí tienen codec como H.264/SVC, con el que comparamos nuestra propuesta de BRC, que en este trabajo, aplicamos a code-stream generados por MCJ2K. Aunque es aplicable a otros tipos code-stream escalables.

La aplicación de nuestro BRC nunca provoca un empeoramiento respecto de no usarlo. Por tanto, siempre es provechoso su uso en términos de maximizar la calidad del vídeo. Se ha conseguido poder transmitir vídeo necesitando cinco veces menos ancho de banda. Además, el uso del algoritmo de BRC posibilita explotar completamente toda la escalabilidad proporcionada por el codec MCJ2K, superando hasta en un 91% a H.264/SVC.

Sin embargo, todas estas ventajas tienen un coste en tiempo de codificación del vídeo, donde para obtener el orden óptimo se requiere de iterativas extracciones y reconstrucciones.

Como línea de trabajo futuro es interesante aplicar planteamientos que mejoren la compresión máxima, en esta línea se puede investigar la compresión escalable de los vectores de movimiento. Junto a sus resultados se obtendrá además una mayor escalabilidad del code-stream a la actual.

Referencias

[1] The Joint Photographic Experts Group, *ISO/IEC 15444-1 (JPEG2000, Part 1)*.

[2] A. Skodras C. Christopoulos and T. Ebrahimi, "The jpeg2000 still image coding system: An overview," *IEEE Transactions on Consumer Electronics*, vol. 46, pp. 1103–1127, 2000.

[3] D. Marpe H. Schwarz and T. Wiegand, "Overview of the scalable video coding extension of the h.264/avc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, pp. 1103–1120, 2007.

[4] T. Wiegand and G.J. Sullivan, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, 2003.

[5] R. Leung and D. Taubman, "Transform and embedded coding techniques for maximum efficiency and random accessibility in 3d scalable compression," *IEEE Transactions on Image Processing*, vol. 14, pp. 1632–1646, 2005.

[6] D. Taubman and J. Thie, "Optimal erasure protection for scalably compressed video streams with limited retransmission," *IEEE Transactions on Image Processing*, vol. 14, pp. 1006–1019, 2005.

[7] D. Gibson and M. Spann, "Multi-frame motion estimation: Application to motion compensated prediction," in *Proceedings of the 1999 IEEE International Symposium onCircuits and Systems, ISCAS 99*, July 1999, vol. 4, p. 54–57.

[8] A. Secker and D. Taubman, "Lifting-based invertible motion adaptive transform (limat) framework for highly scalable video compression," *IEEE Transactions on Image Processing*, vol. 12, pp. 1530–1542, 2003.

[9] M. Antonini T. Andre, M.o Cagnazzo and M. Barlaud, "Jpeg2000-compatible scalable scheme for wavelet-based video coding," *EURASIP Journal on Image and Video Processing*, vol. 2007, 2007.

Implementación del Gradiente Conjugado en un sistema multi-GPU

Carmen Pena, Emilio J. Padrón y Margarita Amor[1]

Resumen— **El método del Gradiente Conjugado es uno de los métodos numéricos más eficientes para la resolución de grandes sistemas de ecuaciones lineales, sobre todo cuando estos presentan una matriz de coeficientes dispersa. Este trabajo aborda la implementación GPGPU del método del Gradiente Conjugado en sistemas CUDA multi-GPU. Aunque abundan las implementaciones de este tipo de algoritmos enfocadas a sistemas con una única GPU, no es habitual encontrar implementaciones que permitan aprovechar la presencia de múltiples GPUs en un nodo computacional.**

El desarrollo de la implementación CUDA se ha basado en la mejora progresiva de una versión mono-GPU, partiendo de una implementación directa que se va mejorando, y su posterior extensión para el trabajo con múltiples GPUs. Nuestra propuesta explora las principales características de CUDA 4 sobre una arquitectura Nvidia Fermi, consiguiendo una implementación eficiente del método iterativo. El trabajo analiza las principales vías de rendimiento y presenta alguno de los cuellos de botella encontrados.

Palabras clave— **GPGPU, CUDA, Gradiente Conjugado, Producto Matriz Dispersa-Vector (SpMV)**

I. Introducción

En este trabajo presentamos una adaptación GPGPU del método del Gradiente Conjugado [1], [2] (en adelante CG, acrónimo del inglés *Conjugate Gradient*) a una GPU (*Graphics Processing Unit*) de arquitectura Fermi de Nvidia usando CUDA. CG is una aproximación iterativa ampliamente usada para resolver sistemas de ecuaciones lineales.

Las GPUs son utilizadas últimamente como coprocesadores de próposito general, de forma que un amplio número de aplicaciones pueden beneficiarse de la gran potencia de cálculo de estos dispotivivos, en lo que se ha venido denominando GPGPU (General-Purpose Computing on Graphics Processing Units). No obstante, adaptar un algoritmo de forma óptima a esta arquitectura no es, en general, trivial.

Por un lado, el paralelismo de datos que estos dispositivos permiten principalmente explotar no siempre es el más adecuado en las distintas partes del código de un programa, y, por otro lado, los accesos a memoria (sobre todo determinados patrones de acceso) suelen verse muy penalizados respecto a operaciones de cálculo intensivo. A todo ello hay que unir el cuello de botella que presenta la transferencia de los datos a través de un bus para que la GPU pueda trabajar con ellos. Un problema adicional lo constituyen unos modelos, herramientas y entornos de programación todavía en fase de maduración en comparación con las aproximaciones tradicionales en CPU.

[1]Grupo de Arquitectura de Computadores, Universidade da Coruña, e-mail: carmen.pena.loures@udc.es, emilioj@udc.es y margarita.amor@udc.es

En concreto, CG presenta grandes retos en una implementación GPU debido a la existencia de estructuras y núcleos computacionales no especialmente adecuados para su ejecución eficiente en la GPU, tales como reducciones escalares u operaciones sobre estructuras de datos para matrices dispersas, lo que supone un patrón irregular de accesos. En este trabajo se estudian diferentes implementaciones de CG aprovechando algunas técnicas de optimización propias de las arquitecturas de Nvidia. Además, se pretende también aprovechar la presencia de múltiples GPUs en el sistema.

Para llevar a cabo la implementación CUDA de CG se ha empleado una metodología ágil de desarrollo. Así, comenzando con una versión GPGPU muy simple de CG, traducción directa del algoritmo secuencial en CPU, se han ido desarrollando, en iteraciones cortas de una o dos semanas, nuevas versiones con las que se solucionan defectos de las anteriores o se aprovechan nuevas características de CUDA. Cada versión es evaluada mediante *profiling*, analizando su rendimiento, problemas y posibilidades de mejora. De esta forma, la implementación para una única GPU se ha ido optimizando en lo posible para ser finalmente extendida a una implementación eficiente para un sistema multi-GPU.

La organización de este documento es la siguiente: La Sección II describe el método del Gradiente Conjugado y los principales trabajos relacionados con su implementación, o con la de alguna de sus operaciones clave, en sistemas GPGPU. La Sección III aborda nuesta implementación en CUDA, describiendo las principales iteraciones en su desarrollo hasta llegar a la versión multi-GPU. Por último, la Sección IV muestra el rendimiento de las soluciones implementadas y la Sección V las principales conclusiones extraídas del trabajo.

II. Gradiente Conjugado

CG es uno de los métodos numéricos más utilizados para la resolución de sistemas lineales de ecuaciones del tipo

$$Ax = b \tag{1}$$

donde A es una matriz cuadrada $(n \times n)$, simétrica $(A^T = A)$ y definida positiva $(y^T Ay > 0, y \neq 0)$, b es un vector de valores conocidos y x es un vector de incógnitas.

CG es un método iterativo con complejidad $\mathcal{O}(n)$ que minimiza la función cuadrática

$$f(x) = \frac{1}{2}x^T Ax - x^T b \tag{2}$$

obteniendo así el vector x solución de la Ec. 1. Para ello se sigue el siguiente algoritmo iterativo:

$$d_{(0)} = r_{(0)} = b - Ax_{(0)} \quad (3)$$

$$\alpha_{(i)} = \frac{r^T_{(i)} r_{(i)}}{d^T_{(i)} A d_{(i)}} \quad (4)$$

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)} \quad (5)$$

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A d_{(i)} \quad (6)$$

$$\beta_{(i+1)} = \frac{r^T_{(i+1)} r_{(i+1)}}{r^T_{(i)} r_{(i)}} \quad (7)$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)} \quad (8)$$

en el cual las letras griegas correponden a valores escalares y el resto son matrices (mayúsculas) y vectores (minúsculas).

Este tipo de métodos son especialmente útiles cuando se trabaja con grandes matrices dispersas, por lo que en la práctica se utiliza un número de iteraciones muy inferior a n, considerando que se alcanza la convergencia cuando el residuo (r en el algoritmo) es suficientemente pequeño.

La operación dominante durante una iteración de CG es el producto matriz dispersa-vector (SpMV en la literatura), con un coste de $\mathcal{O}(nz)$ operaciones (multiplicaciones y sumas), siento nz el número de elementos distintos de cero en la matriz dispersa. El resto son productos escalares (que se resuelven con una operación de reducción), operaciones escalar-vector (variantes de lo que se conoce como SAXPY) y operaciones entre escalares.

La Fig. 1 muestra un pseudocódigo para el método CG, con los distintos bloques de operaciones que involucran a matrices y vectores (*kernels*) identificados y relacionados con los pasos del algoritmo de las Ec. 3–8. Para el almacenamiento de la matriz dispersa se utiliza el formato CSR (*Compressed Sparse Row*) [3], que utiliza tres vectores para representar la matriz: A_{nz} (simplemente A en el pseudocódigo), que almacena todos los valores de la matriz que no son cero, recorridos por filas; *colidx*, que para cada elemento de A_{nz} indica la columna de la matriz dispersa en la que se encuentra ese elemento; y *rowptr*, que indica para cada fila de la matriz dispersa cuál es su primer elemento no-zero mediante su índice en A_{nz}.

En los últimos años proliferan trabajos sobre implementaciones de algunas de las operaciones algebraicas más comunes, como la SpMV, en entornos modernos de computación de altas prestaciones. Así, por ejemplo, en [4] se presentan diversas optimizaciones para entornos multi núcleos, mientras que en [5] se propone un nuevo formato, CSX (*Compressed Sparse eXtended*), para el almacenamiento de matrices dispersas que explota el acceso a las subestructuras dentro de la matriz en sistemas de memoria compartidas con múltiples unidades de procesamiento. Otro formato diferente para matrices dispersas, en este caso orientado a GPGPU, es propuesto en

```
(3)  1  x = 0;   r = b;   d = r;
     2  for (i = 0; i ≤ n; i++)                          INNER
     3    ρ += r_i · r_i;                                PROD
     5  for (cgit = 0; cgit < itmax; cgit++) {
(4)  7    for (i = 0; i ≤ n; i++) {
     8      sum = 0;
     9      for (j = rowptr_i; j < rowptr_{i+1}; j++)    SpMV
    10        sum += A_j · d_{colidx_j};
    11      q_i = sum;
    12    }
    13    for (i = 0; i ≤ n; i++)                        INNER
    14      γ += d_i · q_i;                              PROD
    15
    16    α = ρ/γ;
    17
(5) 18    for (i = 0; i ≤ n; i++) {
(6) 19      x_i += x_i + α · d_i;                        SAXPY2
    20      r_i += r_i − α · q_i;
    21    }
    22
(7) 23    ρ_o = ρ;
    24    for (i = 0; i ≤ n; i++)                        INNER
    25      ρ += r_i · r_i;                              PROD
    26
    27    β = ρ/ρ_o;
(8) 28    for (i = 0; i ≤ n; i++)                        SAXPY
    29      d_i = r_i + β · d_i;
    30  }
    31
    32  for (i = 0; i ≤ n; i++) {
    33    sum = 0;
    34    for (j = rowptr_i; j < rowptr_{i+1}; j++)      SpMV
    35      sum += A_j · x_{colidx_j};
    36    q_i = sum;
    37  }
    38
    39  for (i = 0; i ≤ n; i++) {                        INNER
    40    γ = b_i − q_i;                                 PROD2
    41    sum += γ · γ;
    42  }
    43
    44  rnorm = √sum;
```

Fig. 1: Pseudocódigo con los principales *kernels* computacionales identificados.

[6]: *Cocktail Format*. Esta aproximación está basada en el uso de la herramienta clSpMV, que en tiempo de ejecución analiza la matriz dispersa y recomienda cuál es el mejor formato para la plataforma sobre la que se está ejecutando.

En lo que respecta a implementaciones GPGPU de CG, destaca un trabajo similar al planteado en nuestra aproximación: [7]. En este trabajo la implementación paralela de CG en multi-GPU se basa en dividir en una serie de segmentos las filas de bloques consecutivos que se obtienen de la matriz dispersa en formato BCSR. Estos segmentos son posteriormente asignados a las distintas GPUs.

III. Implementación de CG MultiGPU

Esta sección describe la evolución de la implementación CUDA planteada, describiendo los aspectos más relevantes de cada versión.

A. Versión inicial (CG.naive)

Como versión de partida, se ha realizado una traslación directa del algoritmo secuencial, creando un *kernel* CUDA por cada uno de los bloques de código resaltados en la Fig. 1. Después de cada invocación de un *kernel* se realiza una sincronización. Por razones básicas de eficiencia, los valores escalares α y β se calculan en la CPU.

En concreto, se ha creado un *kernel* SpMV que utiliza directamente el formato CSR de la versión secuencial original, y que obtiene el resultado de cada

```
int i = blockIdx.x*blockDim.x + threadIdx.x;
int tid = threadIdx.x;

if (i < n) {
  sum[tid] = 0.0;
  for (j=rowptr[i]; j<rowptr[i+1]; j++)
    sum[tid] += a[j] * b[colidx[j]];

  q[i] = sum[tid];
}
```

Fig. 2: SpMV versión naive.

uno de los elementos del vector final utilizando un hilo CUDA para calcularlo. La Fig. 2 muestra el código fuente de este kernel, en el que vemos que cada hilo CUDA realiza el producto de una fila de la matriz A por el vector b, obteniendo un escalar.

También se ha creado un *kernel* de reducción, basado en la reducción en árbol eficiente en CUDA propuesta en [8], y que se utiliza para implementar los distintos bloques de código del algoritmo que calculan un producto escalar. Concretamente los bloques `innerPROD` e `innerPROD2` mostrados en la Fig. 1 son implementados con una primera etapa distinta en ambos, que corresponde a una primera etapa de reducción que se acompaña de una operación previa (producto de un elemento consigo mismo en el caso de innerPROD y resta de dos elementos y producto consigo mismo de esa resta en el caso de innerPROD2), seguida de una reducción normal que ya es común a todos los casos (kernel que hemos llamado `reduction`).

En la Tabla I, se muestran los datos obtenidos del *profiling* realizado sobre esta primera versión de CG en CUDA. Los parámetros analizados son el tanto por ciento de tiempo de GPU de cada *kernel*, la eficiencia de carga y almacenamiento de la memoria global, la eficiencia de ejecución de los *warps*, la sobrecarga causada por las ramas divergentes, la sobrecarga debida a las reemisiones de instrucciones por fallos en la memoria global y en su caché, y por último la ocupación conseguida frente a la teórica. Como puede observarse en esta tabla, el *kernel* que más tiempo de ejecución consume es el SpMV con mucha diferencia. Este *kernel* también tiene unos porcentajes bajos de eficiencia de carga y almacenamiento en memoria y una sobrecarga causada por la reemisión de instrucciones muy alta, de un 87.8 %. Estos problemas se atacan en las siguientes versiones.

B. *Versión CG con coalescencia y utilización de la memoria de texturas (CG.CT)*

El grado de coalescencia en los accesos a los vectores en el *kernel* SpMV de la versión *naive* es muy bajo, ya que el patrón de acceso a los datos almacenados no es eficiente. Los hilos de un mismo *warp* no acceden a datos consecutivos en memoria para los vectores a ni $colidx$ ya que los datos que va utilizando cada hilo están almacenados consecutivos por fila en memoria global (ver Fig. 2). Esto causa que no exista coalescencia en la lectura de estos dos vectores, provocando una degradación del rendimiento. Por otro

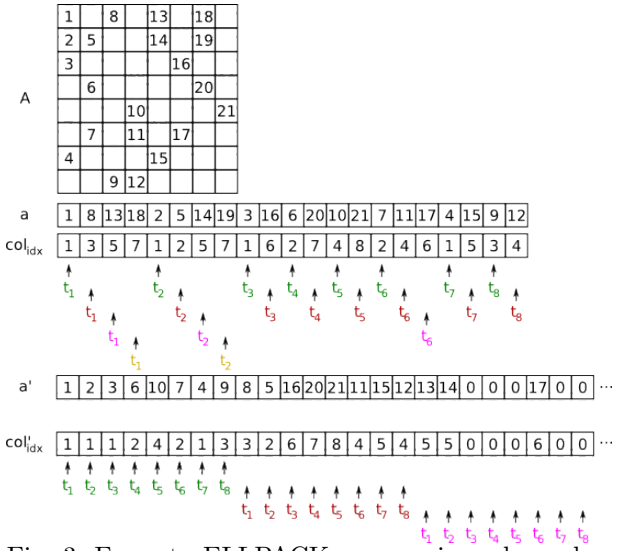


Fig. 3: Formato ELLPACK para mejorar la coalescencia en los accesos a a y $colidx$.

lado, el acceso al vector b es irregular lo que impide un acceso coalescente. Lamentablemente, en este caso el tamaño del vector no hace viable su almacenamiento en la memoria compartida, que podría solucionar este acceso ineficiente. En resumen, el grado de coalescencia como podemos ver en Tabla I, *Global Memory Load Efficiency*, es sólo de un 5.2 %.

Por último, cabe destacar que en este kernel cada hilo realiza un número arbitrario (y potencialmente diferente) de iteraciones del bucle `for`, que corresponde al número de elementos no nulos de la matriz A en la fila i que ejecuta la tarea i ($rowptr[i+1] - rowptr[i]$). Esto provoca un alto grado de divergencia en las ejecuciones de los hilos de un mismo *warp*, como se deduce del valor obtenido de 37.2 % en el parámetro *Branch Divergence Overhead* en la Tabla I.

Para corregir estos problemas se optó por analizar diversos formatos de almacenamiento de matrices dispersas para mejorar la coalescencia en los acceso a a y $colidx$. Existe una amplia variedad de formatos de almacenamiento de matrices dispersas, pero nos centraremos en este trabajo en aquellos formatos adecuados para matrices con estructura arbitraria y que al mismo tiempo son eficientes para la realización de operaciones matriciales. En concreto, hemos considerado el formato *ELLPACK* [9], que almacena la matriz dispersa de dimensión $n \times m_{nz}$, donde n es el número de filas y m_{nz} es el máximo número de elementos distintos de cero presentes en una fila. Aquellas filas con un número menor de no ceros se rellenan con ceros. Este formato es muy adecuado para una implementación GPU, pues permite conseguir coalescencia en los acceso, como se puede ver en el ejemplo de la Fig. 3.

En la matriz 8×8 del ejemplo, 8 hilos ejecutan el kernel en paralelo. Como el número máximo de elementos distintos de cero en una fila de la matriz es de cuatro, cada hilo hará como mucho cuatro iteraciones del bucle `for`. Con la nueva disposición planteada para los vectores, que favorece el acceso coalescente,

TABLA I: Datos de *profiling* (en %) de la versión CG.Naive.

	SpMV	saxpy	innerPROD2	innerPROD	reduction	saxpy2
TiempoGPU	97.7	0.2	0.0	1.6	0.3	0.2
Global Memory Load Efficiency	5.2	99.9	99.9	99.3	38.4	99.9
Global Memory Store Efficiency	39.5	100.0	24.1	24.6	38.6	100.0
Warp Execution Efficiency	62.8	100.0	100.0	100.0	100.0	100.0
Branch Divergence Overhead	37.2	0	0	0	0	0
Total Replay Overhead	87.8	6.1	1	0.9	5.2	8.4
Achieved Occupancy	64	74.1	64.8	68.3	27.4	75.2

```
if (i < n) {
  temp = 0.0;
  sum[tid] = 0.0;
  for (j=i; j<i+(m_nz-1) * n; j+=n)
    temp += a[j] * fetch_double(b,colidx[j]);

  q[i] = temp;
}
```

Fig. 4: SpMV versión CG.CT.

tendremos un tamaño de $n \times m_{nz}$, con m_{nz} como número máximo de iteraciones por hilo. En el ejemplo serían $8 \times 4 = 32$ elementos, rellenando con ceros cuando sea necesario.

La utilización de memoria de texturas presenta algunos beneficios en términos de rendimiento si los accesos a memoria no siguen los patrones que deben cumplir los accesos a memoria global, permitiendo alcanzar un mayor ancho de banda si se logra localidad espacial en los accesos. Uno de los problemas que nos encontramos en esta implementación es que las texturas no trabajan con valores de tipo `double`, por lo que hay que hacer una conversión del tipo de dato a `int2`, que sería el equivalente que podríamos usar en las texturas. Hemos optado, por tanto, en mejorar el rendimiento en el acceso al vector b mediante el uso de la memoria de texturas. El código de la Fig. 4 muestra este kernel *CG.CT*, en el que además, como la utilización de registros es más óptima que la utilización de memoria compartida, se ha usado la variable `temp` en lugar de `sum`.

La Tabla II refleja cómo ha aumentado la coalescencia y ha disminuido la divergencia en la ejecución de los hilos de un *warp*.

C. CG con Multistream (CG.MS)

El siguiente paso seguido para mejorar el rendimiento fue utilizar una ejecución concurrente asíncrona, que nos permita eliminar puntos de sincronización y solapar en lo posible cálculo y computación. En CUDA existen algunas llamadas a funciones asíncronas que permiten la ejecución concurrente entre la CPU y la GPU, devolviendo el control a la CPU antes de que el dispositivo haya completado la tarea requerida. Algunos ejemplos de estas funciones son las llamadas a los kernels, las copias de memoria entre GPUs, las copias de memoria entre CPU y GPU de bloques de 64 KB o menos, las copias de memoria realizadas por funciones asíncronas y las llamadas a funciones de reserva de memoria.

Dependiendo de la capacidad computacional CUDA del dispositivo en cuestión, también es posible lograr el solapamientos entre la ejecución de los kernels en la GPU y la transferencia de datos entre CPU y GPU; además de solapamientos entre la ejecución en GPU de distintos kernels de un mismo contexto CUDA; o incluso solapar la copia de datos de CPU a GPU con una transferencia desde GPU a CPU. Para lograr esto hay que utilizar, además de las llamadas apropiadas del API de CUDA, el concepto de *streams*, dividiendo todo el proceso de cálculo y transferencias en varios flujos de ejecución, En nuestra propuesta los *streams* procesan los vectores en fragmentos de igual tamaño, donde se solapa las transferencias de los fragmentos de un *stream* con el procesamiento de otros.

La cuestión más importante en los *streams* es la asociación entre estos y las llamadas a las funciones que tienen que ejecutar y el orden en que estas asociaciones se realizan, ya que la cantidad de solapamiento entre los distintos *streams* depende en gran parte del orden en que los comandos son distribuidos a cada uno de ellos. En nuestro código tenemos que realizar la transferencia de dos vectores y la ejecución del kernel que trabaja con ellos. Para que se pueda ejecutar el kernel, tienen que haber sido transferidos con anterioridad los dos vectores con los que trabaja, por lo que estas llamadas tienen que realizarse en un orden determinado:

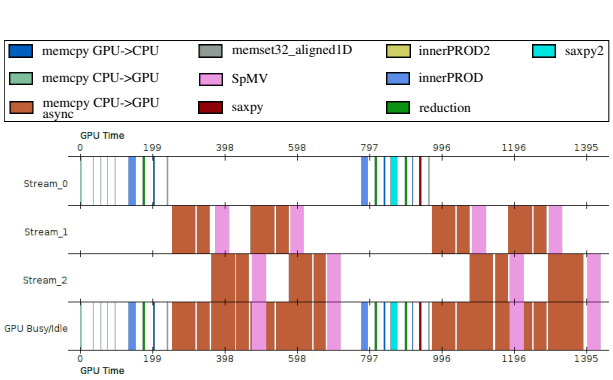
```
cudaMemcpyAsync(vector_x0, stream0);
cudaMemcpyAsync(vector_y0, stream0);
out_kernel<<<stream0>>>(vector_x0, vector_y0);

cudaMemcpyAsync(vector_x1, stream1);
cudaMemcpyAsync(vector_y1, stream1);
out_kernel<<<stream1>>>(vector_x1, vector_y1);
```

Se tienen que enviar los datos del *Stream 0* antes de ejecutar el *kernel* de este *stream*, y lo mismo sucede con el *Stream 1*. De esta forma se pueden solapar la ejecución del kernel en el *Stream 0* con la transferencia de datos CPU-GPU en el *Stream 1*, y la ejecución del kernel en el *Stream 1* con la siguiente transferencia de datos en el *Stream 0*, siempre respetando las dependencias dentro del mismo *stream*. Estas ejecuciones concurrentes se repiten hasta que se procesen todos los fragmentos de los vectores.

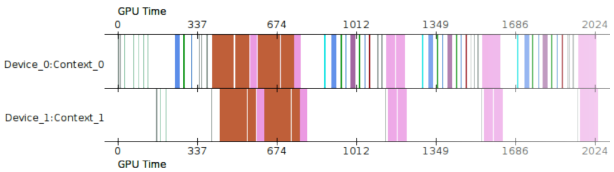
En la Fig. 5a se puede ver el resultado del *profiler* de CUDA con la ejecución de esta versión. En la figura podemos observar cómo se solapan las ejecución del *kernel* `SpMV` con las copias de memoria entre CPU y GPU.

TABLA II: Datos de *profiling* (en %) de la versión CG.CT.

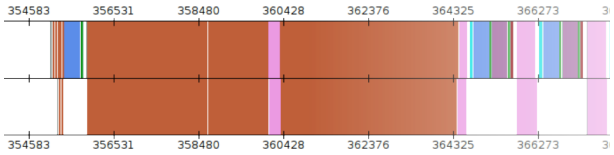
	SpMV	saxpy	innerPROD2	innerPROD	reduction	saxpy2
TiempoGPU	41.4	4.4	1.0	25.0	5.4	22.8
Global Memory Load Efficiency	62.6	99.9	99.9	99.8	77.8	99.9
Global Memory Store Efficiency	99.9	99.9	25	25	26.8	81.8
Warp Execution Efficiency	99.9	100.0	100.0	100.0	99.2	100.0
Branch Divergence Overhead	0.1	0.0	0.0	0.0	0.0	0.0
Total Replay Overhead	9.1	0.8	0.1	0.1	3.6	0.2
Achieved Occupancy	35.2	72	76.2	75.3	27	64.6



(a) Versión CG.MS: multi-stream



(b) Versión CG.MG: multi-GPU



(c) Versión CG.MSMG: multi-GPU con multi-hilo en CPU

Fig. 5: *Profiling* de las tres versiones más eficientes implementadas.

D. CG con Multi-GPU (CG.MG)

Esta nueva versión trata de aprovechar la presencia de múltiples GPUs en el sistema, con lo que la ejecución se realiza de forma concurrente en más de una GPU. Existen varias opciones para implementar esta mejora, consistiendo la más sencilla en que una única tarea o hilo de la CPU controle todas las GPUs, cambiando de un contexto a otro para ejecutar los diferentes *streams* en cada una. Una opción más elaborada consiste en que la CPU tenga una tarea por cada GPU para controlarlas, de forma que cada tarea maneja un contexto diferente. Abordamos primero la versión más simple, dejando la versión multi-hilo en CPU como una mejora sobre esta.

En esta primera versión simple multi-GPU todas las comunicaciones entre GPUs tienen lugar a través de la CPU, siendo un único hilo de CPU el que controla todas las GPUs cambiando de contexto cuando es necesario. Se usa memoria *page–locked host* que permite una comunicación más eficiente entre CPU-GPU y la técnica *peer–to–peer* que se puede usar en dispositivos de capacidad computacional CUDA 2.0 o superiores. *Peer–to–peer* consiste en utilizar en un espacio de direcciones unificado para la CPU y GPUs, de forma que se puede usar el mismo puntero para direccionar memoria de cualquiera de los dispositivos. También se pueden realizar copias de memoria entre dos GPUs usando la función `cudaMemcpyPeer()`, que permite un acceso más rápido que el acceso habitual a través de la CPU.

En la Fig. 5b se muestra la gráfica de *profiling* para una ejecución de esta versión donde se puede observar cómo se solapa perfectamente la ejecución de varios *kernels* en las dos GPUs utilizadas.

E. CG con Multistream y Multi-GPU (CG.MSMG)

Nuestra segunda aproximación multi-GPU crea un hilo en la CPU para cada GPU que se vaya a utilizar, aprovechando así además la presencia de múltiples núcleos computacionales en la CPU. Estos hilos controlan las declaraciones y ejecuciones del contexto de la GPU que tiene asignada.

Los hilos de la CPU se han implementado mediante la biblioteca estándar *POSIX threads*, y lógicamente la implementación sigue aprovechando la ejecución multi-stream ya implementada. En la Fig. 5c se presenta la gráfica de *profiling* de la ejecución de esta versión, mostrando cómo se solapa perfectamente la ejecución de las dos GPUs utilizadas.

IV. Resultados

En esta sección se presentan los datos de rendimiento obtenidos con las diferentes versiones de CG implementadas sobre dos plataformas diferentes:

Plataforma 1: Dos Nvidia TESLA S2050 con 3GB VRAM sobre Intel Xeon X5650 2.67GHz con 12 MB caché, 6 núcleos y 12GB de RAM.

Plataforma 2: Nvidia GeForce GTX 580 con 1536 MB sobre Intel Corei7-2600 3.40GHz con 8 MB caché, 4 núcleos y 8 GB de RAM.

En la Tabla III se muestran los tiempos de ejecución de las diferentes versiones en ambas plataformas, utilizando distintos tamaños n de la matrix A. La Plataforma 2 dispone de una única GPU, por lo que no se han ejecutado en ella CG.MG y CG.MSMG. La tabla presenta el tiempo de ejecución total, el tiempo de ejecución de una iteración de CG y la aceleración entre paréntesis. La aceleración máxima alcanzada es de 8,23x (CG.MG Plataf. 1).

TABLA III: Tiempo de ejecución en segundos sobre las dos plataformas.

Plataf. 1	**n = 1400**	**n = 7000**	**n = 14000**	**n = 75000**	**n = 150000**
CG.CPU	0.2006 0.0133	1.3387 0.0891	5.9090 0.3934	238.7122 3.1820	637.6806 8.4779
CG.naive	2.8530 (0.76) 0.0175	3.1061 (2.41) 0.0304	4.0625 (3.69) 0.1002	66.9248 (3.55) 0.8583	153.9046 (4.22) 2.0101
CG.CT	2.3676 (1.73) 0.0077	2.5468 (4.72) 0.0189	3.0756 (7.56) 0.0521	32.2509 (8.01) 0.3973	90.6883 (7.09) 1.1966
CG.MS	0.1378 (1.47) 0.0091	0.3492 (3.84) 0.0232	0.9460 (6.36) 0.0619	36.7937 (6.57) 0.4843	107.1265 (5.94) 1.4274
CG.MG	0.2475 (1.27) 0.0105	0.4210 (3.97) 0.0225	0.9057 (6.31) 0.0624	29.9743 (8.23) 0.3864	82.5860 (7.69) 1.1025
CG.MSMG	0.2796 (1.02) 0.0131	0.4702 (3.50) 0.0255	0.9082 (7.28) 0.0541	29.4410 (8.20) 0.3882	82.8173 (7.79) 1.0882
Plataf. 2	**n = 1400**	**n = 7000**	**n = 14000**	**n = 75000**	**n = 150000**
CG.CPU	0.1052 0.0069	0.6964 0.0460	2.9236 0.1957	126.6280 1.6961	335.6122 4.4502
CG.Naive	0.2256 (0.56) 0.0122	0.3818 (2.00) 0.0230	1.2475 (2.42) 0.0807	55.0469 (2.30) 0.7385	133.0815 (2.50) 1.7787
CG.CT	0.1028 (1.50) 0.0046	0.2297 (3.44) 0.0134	0.6127 (5.09) 0.0385	24.7105 (5.20) 0.3264	79.3388 (4.20) 1.0590
CG.MS	0.0814 (1.21) 0.0057	0.2278 (3.05) 0.0151	0.6595 (4.43) 0.0442	27.9402 (4.55) 0.3724	89.4715 (3.73) 1.1920

V. Conclusiones

En este trabajo se han presentado varias implementaciones GPGPU del método del Graciente Conjugado, todas ellas sobre la plataforma CUDA de Nvidia. Partiendo de una versión *naive* mono-GPU, traducción directa de la versión secuencial para CPU, se ha alcanzado una versión eficiente mono-GPU y posteriormente una versión multi-GPU. Durante el proceso se han analizado los principales cuellos de botella y problemas de rendimiento, rediseñando el algoritmo para ajustarse mejor a las características de la arquitectura CUDA.

Siendo la optimización del uso de la memoria uno de los objetivos a alcanzar para obtener un buen rendimiento, se ha mejorado el patrón de accesos para maximizar el ancho de banda. Así, mediante la mejora en el grado de coalescencia se ha obtenido una aceleración de $6x$. También se ha experimentado con el uso de la memoria de texturas, para intentar mejorar el problema de accesos irregulares, obteniéndose así una aceleración en torno a $8x$.

Otro de los problemas que se ha encontrado ha sido el cuello de botella que supone la transferencia de datos a la GPU. Para minimizarlo se ha intentado solapar dicha transferencia de datos con las ejecuciones de código en la GPU mediante el uso de funciones asíncronas y de *streams*. Para poder observar una mejora significativa en este aspecto se ha desarrollado una versión multi-GPU con la que se puede solapar la ejecución de una GPU con la ejecución de otra GPU distinta del sistema. Con esta solución se ha conseguido paliar en parte el problema de la transferencia de datos, además de aprovechar la presencia de múltiples GPUs en el sistema, obteniendo una aceleración de $8{,}23x$ con dos GPUs.

Referencias

[1] Magnus R. Hestenes and Eduard Stiefel, "Methods of conjugate gradient for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, pp. 409–436, 1952.

[2] Jonathan Richard Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Tech. Rep., School of Computer Science, Carnegie Mellon University, 1994.

[3] Nathan Bell and Michael Garland, "Efficient sparse matrix-vector multiplication on cuda," Tech. Rep. NVR-2008-004, NVidia, 2008.

[4] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel, "Optimization of sparse matrix–vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, no. 3, pp. 178–194, 2009.

[5] Kornilios Kourtis, Vasileios Karakasis, Georgios Goumas, and Nectarios Koziris, "Csx: an extended compression format for spmv on shared memory systems," in *Proc. of the 16th ACM symposium on Principles and practice of parallel programming (PPoPP'11)*, 2011, pp. 247–256.

[6] Bor-Yiling Su and Kurt Keutzer, "clspmv: A cross-platform opencl spmv framework on gpus," in *Proc. of the 26th ACM international conference on Supercomputing (ICS'12)*, 2012.

[7] Mickeal Verschoor and Andrei C. Jalba, "Analysis and performance estimation of the conjugate gradient method on multiple gpus," *Parallel Computing*, vol. 38, pp. 552–575, 2012.

[8] Mark Harris, "Optimizing parallel reduction in cuda," http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf, 2007.

[9] F. Vazquez, G. Ortega, J. J. Fernandez, and E. M. Garzon, "Improving the performance of the sparse matrix vector product with gpus," in *Proc. of the 10th IEEE International Conference on Computer and Information Technology (CIT'10)*, 2010, pp. 1146–1151.

Sincronización Paralela de Trayectorias para Detección de Conflictos entre Aeronaves

Eduardo de la Iglesia, Guillermo Botella, Carlos García, Manuel Prieto y Francisco Tirado

Resumen--El objetivo de este trabajo es realizar un estudio de viabilidad -en cuando a aceleración se refiere- de sincronización paralela de trayectorias para detección de conflictos en aeronaves, usando hardware gráfico. Se pretende evaluar la conveniencia de usar dispositivos GPU con este propósito.

Palabras clave--Trayectoria de Aeronave, EPP, Control de Tráfico Aéreo, AGDL, ADS-C, GPU, Sincronización de Trayectorias, Resolución de conflictos, CUDA.

I. INTRODUCCIÓN

La tarea de comparar múltiples trayectorias es un problema bastante costoso. Para conseguir la trayectoria de una aeronave suelen usarse predictores de trayectoria - Trajectory Predictor (TP) en inglés - que tratan de modelar el recorrido de un avión y estimar su trayectoria en un intervalo de tiempo futuro. Un TP debe de tomar una gran cantidad de datos de entrada para afinar la predicción lo más posible. De forma tradicional han sido implementados modelos de predicción [5] en los centros de control aéreo empleando información proporcionada por los radares, con los datos previos de planes de vuelo, evaluando la congestión del tráfico aéreo, la información meteorológica y las características físicas de cada modelo determinado de avión con el ánimo de prevenir riesgos.

En los últimos años se han hecho bastantes esfuerzos por facilitar la comunicación entre los centros de control aéreo y las aeronaves. Un ejemplo de ello es el uso de AGDL (Air Ground Data Link), que establece un canal de comunicación entre las propias aeronaves y los controladores aéreos [6]. Entre sus Aplicaciones se encuentra ADS (Automatic Dependent Surveillance) que permite a los aviones proporcionar automáticamente datos derivados de la navegación a bordo y de su estado (identificación, posición en cuatro dimensiones y datos adicionales) a los centros de control aéreo.

En este trabajo se tomarán como entrada las trayectorias predichas por cada aeronave y obtenidas en tierra por medio de la Aplicación ADS. Cabe destacar que la propia aviónica debería poder hacer predicciones de su trayectoria más exactas de lo que un sistema en tierra sería capaz de calcular, pues conoce de primera mano el estado actual (posición, velocidad, aceleración, cantidad de combustible, parámetros de vuelo, etc.) y modelo de comportamiento y actuación del avión concreto.

Una vez obtenida la predicción de trayectoria de varias aeronaves se puede utilizar para distintos fines. Un uso muy común es la comparación de la trayectoria obtenida por medio de ADS y la progresión de la aeronave que se tiene prevista en tierra para que el controlador pueda dar instrucciones válidas en su gestión del espacio aéreo como se refleja en [1] y [3]. Otra posible aplicación es la de comparar múltiples trayectorias de aeronaves para detectar posibles conflictos y permitir a los controladores su resolución, problema estudiado en [2], [4] y [7]. Debido a la complejidad de las trayectorias que se reciben por ADS, un algoritmo de comparación secuencial resulta costoso y poco eficiente. Además, en este contexto de tiempo real, el tiempo de respuesta de este algoritmo es crucial. Si vamos a detectar un conflicto entre dos aeronaves dentro de unos pocos minutos, el algoritmo de comparación no puede tardar demasiado en avisar de este conflicto al controlador porque se pondría en riesgo la seguridad de las aeronaves y se violaría la distancia de separación entre las mismas.

II. COMPARACIÓN DE TRAYECTORIAS

Se introducen algunos conceptos clave que permitan el entendimiento de las decisiones tomadas en capítulos posteriores:

Extended Projected Profile (EPP)

La Aplicación ADS (Automatic Dependent Surveillance) es el mecanismo por el cual la aeronave envía su trayectoria al sistema de tierra periódicamente, por demanda o ante eventos destacables mediante un ADS-C Report que contiene un EPP (Extended Projected Profile). Este EPP es calculado automáticamente por el FMS (Flight Management System) y contiene una lista completa de puntos desde la posición actual hasta el umbral de la pista de aterrizaje en el aeropuerto de destino. Estos son los puntos destacables para el vuelo, aquellos interesantes por diversos motivos (cambios de rumbo, cambios de altitud, entradas y salidas de aerovías, etc.).

Para cada uno de estos puntos, se provee la siguiente información: Posición (latitud y longitud), Nivel (altitud), Tiempo estimado de sobrevuelo y Tipo de punto (por ejemplo Top of Climb, Top of Descent). Además, el EPP puede contener otro tipo de información, como la velocidad estimada en cada punto, información meteorológica, masa actual de la aeronave, etc. Por medio del EPP se puede trazar la trayectoria futura completa de la aeronave.

Sampling

Para calcular el conflicto entre dos aeronaves se va a necesitar una forma de comparar dos EPP, que a priori son bastante diferentes, porque cada uno contiene diferentes puntos en diferentes tiempos. Se podría plantear una solución geométrica secuencial para resolver esta cuestión. Ésta es, por norma general, la manera más usada [8]: Uniendo los puntos adyacentes de un EPP mediante segmentos se podrían calcular todas las combinaciones de segmentos entre las dos trayectorias y comprobar si existen puntos de corte. Si existen puntos de corte en un intervalo de tiempo común para ambas trayectorias es cuando se puede hablar de conflicto. Se considera un conflicto entre aeronaves cuando sus trayectorias se encuentran lo suficientemente próximas no respetando el mínimo de separación que se considera segura.

En este caso, la intención de este trabajo para paralelizar la comparación de trayectorias está basada en utilizar primero un algoritmo paralelo de muestreo, por el que transformar el EPP en una serie de puntos que puedan ser temporalmente coincidentes con los de las trayectorias, como podemos observar a continuación:

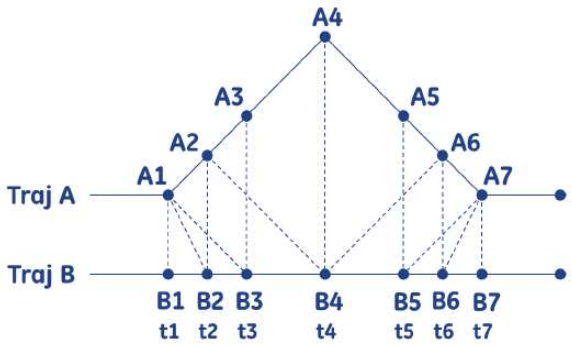


Figura 1: Muestreo de EPP aplicado a dos trajectorias

La "sincronización" de trayectorias en esta figura consiste en calcular la posición de ambas aeronaves en tiempos coincidentes t_1, t_2,..t_n para poder calcular, por ejemplo, la distancia que separa a dichas aeronaves en un determinado instante. Esta “sincronización” entre trayectorias puede hacerse por interpolación. Se usarán diversos métodos de interpolación con este objetivo. Como los puntos introducidos por el FMS en el EPP son los más significativos de la trayectoria habrá que estudiar posibles formas de obtener mejores resultados para después hacer una comparación geométrica o analítica entre varias aeronaves.

Las interpolaciones candidatas a utilizar son las siguientes:

- **Interpolación lineal** con gran tasa de muestreo

- **Interpolación cuadrática**

- **Interpolación polinómica o Spline** con menor tasa de muestreo

Se relacionará cada una de las interpolaciones con la distancia de seguridad admisible para cada aeronave teniendo en cuenta la tasa de muestreo que el propio método de interpolación recomiende en cada caso.

Geometría simplificada de conflicto

En un instante t, se puede calcular si una aeronave se encontrará posicionada dentro del espacio de seguridad de la otra aeronave de forma geométrica. Esta es la operación básica a ejecutar en cada procesador de GPU de forma paralela. Además habrá que tener en cuenta otras restricciones dependiendo del algoritmo de muestreo utilizado.

Resulta relevante intentar hacer esta comprobación geométrica en 3D, debido a que las trayectorias pueden sufrir tanto discrepancias verticales como horizontales. El objetivo pasa por conseguir la mayor eficiencia en el algoritmo encontrando un compromiso entre la tasa de muestreo y la distancia de seguridad. Una alta tasa de muestreo aumenta el número de operaciones a realizar, disminuye la distancia de seguridad entre aeronaves, y simplifica el cálculo principal hasta llegar a una operación atómica idealmente paralelizable como pueda ser calcular la distancia entre dos puntos. Por el contrario, una tasa de muestreo menor se queda a medio camino entre la paralelización absoluta y la necesidad de compartir más información entre los cálculos, o bien obliga a aumentar la distancia de seguridad entre aeronaves.

Detección múltiple de conflictos

De un modo general se debería poder usar otro algoritmo paralelo para la detección de conflictos de múltiples aeronaves teniendo en cuenta conceptos del espacio aéreo. El espacio aéreo está dividido de forma dinámica, donde cada porción de espacio es controlada por un controlador aéreo diferente bajo el concepto de AoI (Área de interés) y AoR (Área de responsabilidad). Cabe destacar que detectar conflictos debe ser una capacidad distribuida entre cada uno de los sectores de dicho espacio aéreo, puesto que cada controlador está interesado solamente en los conflictos que puedan acontecer en su área de responsabilidad. Existen algunas aproximaciones como [4] que han estudiado algoritmos distribuidos para la planificación de trayectorias en UAVs (Unmanned Aerial Vehicles, Vehículos Aéreos no Tripulados).

III. PARALELISMO EN LOS ALGORITMOS DE COMPARACIÓN DE TRAYECTORIAS

El objetivo es comprobar si una división del problema en etapas paralelizables constituye una sólida alternativa a su implementación secuencial. Se propone el siguiente esquema:

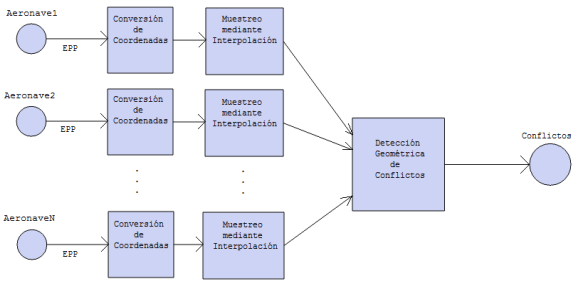


Figura 2: Etapas paralelizables de la Comparación de Trayectorias

Algoritmo de conversión de coordenadas

La entrada de esta etapa es un EPP, conjunto de puntos cuyas coordenadas geodésicas recibidas vienen definidas en latitud, longitud y altitud, comúnmente usadas en navegación. En el caso de las dos primeras, las unidades son grados, minutos y segundos. Es conveniente realizar un cambio de coordenadas a cartesianas (x, y, z) puesto que los cálculos geométricos que describe el problema de la sincronización de trayectorias serán simplificados.

Se usa el estándar WGS 84 (World Geodetic System) como sistema de referencia. El algoritmo se compone de operaciones aritméticas y trigonométricas simples para cada punto:

Dada la latitud ϕ y la longitud λ y altitud h de un punto:

$$\phi = (\phi_d + \phi_m / 60 + \phi_s / 3600) * \pi / 180$$
$$\lambda = (\lambda_d + \lambda_m / 60 + \lambda_s / 3600) * \pi / 180$$

las coordenadas cartesianas equivalentes son:

$$x = (N(\phi) + h)\cos\phi\cos\lambda$$
$$y = (N(\phi) + h)\cos\phi\sin\lambda$$
$$z = (N(\phi)(1-e^2) + h)\sin\phi$$

donde

$$N(\phi) = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}$$

Algoritmos de interpolación

Como segunda etapa, la intención es obtener un gran número de puntos que describan la trayectoria de la aeronave con respecto al tiempo a partir del EPP. Esto se puede conseguir por medio de diferentes métodos de interpolación:

1) Interpolación lineal

La interpolación es un método sencillo y viable para obtener puntos intermedios de la trayectoria de una aeronave. En este caso, la implementación de este algoritmo en GPU es eficiente. Para cada elemento, dados dos puntos de trayectoria (x_{i-1}, y_{i-1}, z_{i-1}) y (x_i, y_i, z_i) por los que la aeronave pasa en t_{i-1} y t_i respectivamente, podemos obtener un punto interpolado (x, y, z) en t, siendo $t_{i-1}<t<t_i$ mediante las ecuaciones:

$$x = x_{i-1} + (t - t_{i-1})\frac{x_i - x_{i-1}}{t_i - t_{i-1}}$$
$$y = y_{i-1} + (t - t_{i-1})\frac{y_i - y_{i-1}}{t_i - t_{i-1}}$$
$$z = z_{i-1} + (t - t_{i-1})\frac{z_i - z_{i-1}}{t_i - t_{i-1}}$$

2) Interpolación cuadrática

A pesar de la buena aproximación que supone hacer una simple interpolación lineal, se puede mejorar el muestreo con respecto al verdadero movimiento de la aeronave incluyendo otras variables como la velocidad estimada. Para cada elemento, dados dos puntos de trayectoria (x_{i-1}, y_{i-1}, z_{i-1}) y (x_i, y_i, z_i) por los que la aeronave pasa en t_{i-1} y t_i a velocidades v_{i-1} y v_i respectivamente, podemos obtener un punto interpolado (x, y, z) en t, siendo $t_{i-1}<t<t_i$ mediante las ecuaciones:

$$x = \frac{Ax_{i-1} + Bx_i}{C} \qquad y = \frac{Dy_{i-1} + Ey_i}{F}$$

$$z = z_{i-1} + (t - t_{i-1})\frac{z_i - z_{i-1}}{t_i - t_{i-1}}$$

siendo:

$$A = (v_{i\text{-}1(x)} - v_{i(x)})(t_i - t)^2 + 2v_{i(x)}(t_i - t_{i-1})(t_i - t)$$
$$B = (v_{i(x)} - v_{i\text{-}1(x)})(t - t_{i\text{-}1})^2 + 2v_{i\text{-}1(x)}(t_i - t_{i\text{-}1})(t - t_{i\text{-}1})$$
$$C = (v_{i\text{-}1(x)} + v_{i(x)})(t_i - t_{i\text{-}1})^2$$
$$D = (v_{i\text{-}1(y)} - v_{i(y)})(t_i - t)^2 + 2v_{i(y)}(t_i - t_{i\text{-}1})(t_i - t)$$
$$E = (v_{i(y)} - v_{i\text{-}1(y)})(t - t_{i\text{-}1})^2 + 2v_{i\text{-}1(y)}(t_i - t_{i\text{-}1})(t - t_{i\text{-}1})$$
$$F = (v_{i\text{-}1(y)} + v_{i(y)})(t_i - t_{i\text{-}1})^2$$

3) Interpolación polinómica

Para representar curvas es común el uso de polinomios por intervalos, cuya función es llamada spline. Se usará la versión de splines cúbicas debido a que se obtienen resultados más adecuados a las curvas mediante polinomios de grado bajo y se evitan indeseables oscilaciones.

La idea central es que en vez de usar un solo polinomio para interpolar los datos, se pueden usar segmentos de polinomios y unirlos adecuadamente (bajo ciertas condiciones de continuidad) para formar la interpolación. La funcion spline cúbica es una función s(t) definida así:

$$s(t) = \begin{cases} s_0(t) & si \quad t \in [t_0, t_1] \\ s_1(t) & si \quad t \in [t_1, t_2] \\ \vdots & \\ s_{n-1}(t) & si \quad t \in [t_{n-1}, t_n] \end{cases}$$

donde cada $s_i(t)$ es un polinomio cúbico que cumple $s_i(t_i) = x_i$, para todo i = 0,1,..,n y tal que s(t) tiene primera y segunda derivadas continuas en $[t_0,t_n]$. En nuestro problema de 3D, necesitaremos tres interpolaciones, una para cada coordenada, de forma que usaremos la notación $s_x(t)$, $s_y(t)$ y $s_z(t)$ que para referirnos a ellas.

Para la implementación de esta interpolación se ha decidido hacer uso de un método aproximado de splines cúbicas conocido como Curvas Cúbicas de Bezier [9]. Para calcular la curva entre dos puntos, P_0 y P_3, se cuenta con dos puntos de control intermedios (puntos de anclaje) P_1 y P_2.

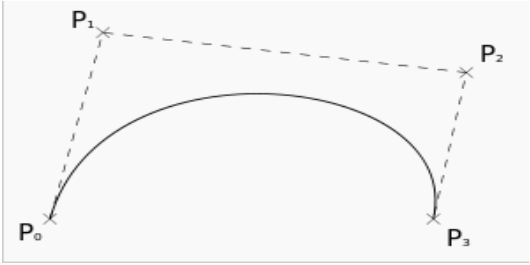


Figura 3: Curva de Bezier entre dos puntos

Algoritmo de detección de conflictos

1) Detección geométrica

Gracias al muestreo anterior se puede detectar un posible conflicto comparando punto a punto ambas trayectorias.

Se debe encontrar una relación entre la tasa de muestreo (número de puntos), la calidad de los puntos, y la distancia de seguridad entre aeronaves. Las dos primeras vienen determinadas por el algoritmo de interpolación y la última se puede considerar como un parámetro externo como dato incorporado en el EPP.

Si la tasa de muestreo es baja, esto es, los puntos de una misma trayectoria están bastante separados, hay dos opciones: aumentar la distancia de separación entre aeronaves o utilizar métodos suplementarios que permitan verificar que ningún conflicto es pasado por alto. Si la tasa de muestreo es alta, se puede comparar punto a punto con una distancia de separación mucho menor sin necesidad de añadir otros métodos.

La ecuación de distancia entre 2 puntos $P_1(x_1,y_1,z_1)$ y $P_2(x_2,y_2,z_2)$ es

$$|P_1P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Debido a que la distancia de seguridad entre aeronaves puede cambiar dinámicamente, se establece un umbral μ como entrada externa del sistema tal que:

Hay conflicto entre dos aeronaves A, B en t_i cuando $|P_A(t_i)P_B(t_i)| < \mu.$

Se podría optar por considerar μ como una entrada externa del sistema que tiene en cuenta la estela turbulenta de la aeronave, la metereología y otras condiciones para la detección de conflictos. En este caso lo vamos a usar como parámetro dependiente de un dato que se recibe en el EPP y que contiene la precisión de los puntos calculados por la aeronave, la *Figure of Merit*, medida que caracteriza de forma cuantitativa la utilidad dichos puntos estableciendo su exactitud.

2) Detección Grid-based

Lo más interesante de poder detectar los conflictos usando las técnicas propuestas anteriormente es que se puede tratar de generalizar el problema a N aeronaves y seguir aprovechando la capacidad de paralelización que nos permite la GPU. Por medio de la construcción de un grid espacio-tiempo 4D se pueden computar qué celdas están ocupadas y hacer el chequeo oportuno para detectar conflictos.

IV. RESULTADOS EXPERIMENTALES

Algoritmo de conversión de coordenadas

El EPP estándar contiene un máximo de 128 puntos, lo que implica un escaso paralelismo de datos.
El principal problema de paralelizar este algoritmo es que al tiempo de cómputo neto de GPU hay que añadir el tiempo de transferencia de memoria principal de los datos de entrada a la memoria de la tarjeta gráfica y el tiempo de transferencia del resultado de la memoria de la tarjeta gráfica a la memoria principal.
Se estudia el comportamiento del acelerador para la conversión de coordenadas con mayor número de puntos, pues parece indicado pensar que en el futuro la capacidad de la aeronave de calcular su trayectoria de forma realista será incrementar el número de puntos. En la siguiente gráfica se observa la porción de tiempo usada para cada operación de GPU. De acuerdo a la gráfica se sugiere una buena escalabilidad. El cómputo del kernel se acaba llevando más del 70% del tiempo total frente a la transferencia de memoria.

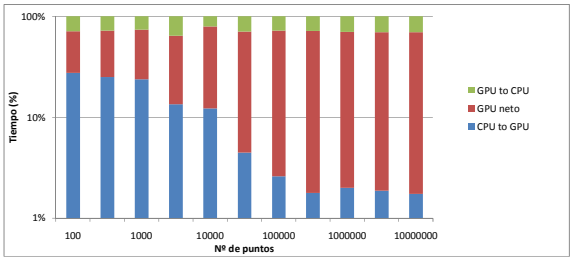


Figura 4: Uso total de tiempo GPU en la conversión de coordenadas

Algoritmos de interpolación

1) Interpolación lineal

La paralelización de la interpolación lineal se puede realizar de forma directa porque cada cálculo es independiente de los demás. En las gráficas de rendimiento se observa como el speedup frente al algoritmo secuencial es considerable.

Como se necesitan intervalos de tiempo bastante reducidos para después desarrollar la comparación, se obtiene la mayor cantidad de muestras de la trayectoria y los resultados son muy positivos (speedup pico 100x):

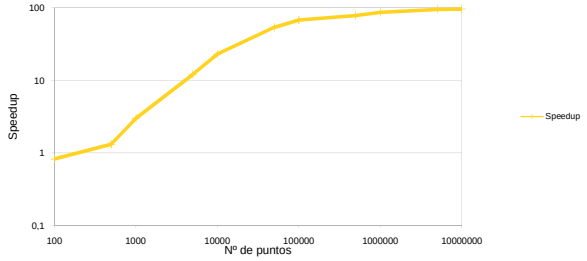


Figura 6: Speedup de la interpolación lineal

En la práctica, para intervalos en segundos de unas 4 horas de vuelo, el speedup frente a la implementación secuencial se encuentra entre 30x y 40x.

2) Interpolación cuadrática

Para la interpolación cuadrática se ha obtenido un rendimiento similar con la particularidad de que los puntos interpolados son de mayor calidad. Al introducir la velocidad en el cálculo, el número de operaciones en cada hilo es bastante mayor.

El speedup se mantiene con respecto a la interpolación lineal (CPU vs GPU). Los tiempos de ejecución son algo superiores, pero con este método las coordenadas obtenidas son bastante más reales y muy parecidas a la trayectoria recibida en el EPP desde la aeronave.

3) Interpolación polinómica

La principal diferencia en la implementación del kernel con respecto a las interpolaciones lineal y cuadrática vistas anteriormente es que se necesitan dos puntos de control para cada intervalo.

El algoritmo que calcula los puntos de control tiene una fuerte componente secuencial que provoca divergencia en los hilos. Cuando los hilos que se ejecutan a la vez toman caminos distintos (las sentencias condicionales hacen que en cada hilo se ejecuten instrucciones distintas), dichos caminos diferentes se serializan, penalizando en gran medida la componente paralela del algoritmo.

En este caso, se obtiene un speedup de 3x con respecto a la versión secuencial cuando la cantidad de puntos a interpolar es alta. Sin embargo, el tiempo de ejecución aumenta considerablemente con respecto a las interpolaciones anteriormente implementadas en paralelo, con el añadido de que el objetivo de éste método era conseguir una baja tasa de muestreo con mayor calidad de puntos interpolados. Además, la calidad de los puntos conseguidos no mejora en gran medida a la de los obtenidos por interpolación cuadrática, debido con seguridad a que los puntos recibidos por medio del EPP contienen los puntos más significativos de la trayectoria de la aeronave.

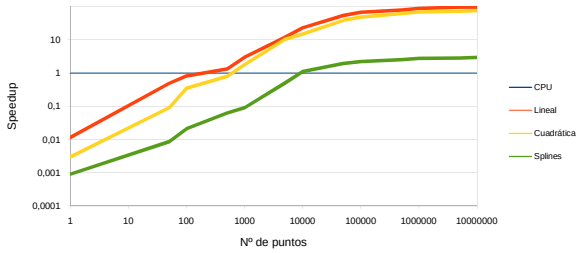


Figura 7: Comparativa de speedup para los métodos de interpolación

Algoritmo de detección de conflictos

4) Detección geométrica

La estrategia para la detección geométrica usada parte de dos trayectorias obtenidas mediante interpolación cuadrática con alta tasa de muestreo. Debido a eso, el algoritmo paralelo de comparación es directo.

La razón de elegir la interpolación cuadrática es debida a un pequeño estudio de precisión que tiene en cuenta los siguientes parámetros:

(a) Número de puntos necesarios para obtener suficiente ganancia con respecto a la implementación en CPU.

(b) Tiempo de ejecución parcial de la interpolación.

(c) Calidad de los puntos obtenidos.

(d) Distancia de separación necesaria.

Se puede decir que la distancia de separación μ es inversamente proporcional al número de puntos de trayectoria N e inversamente proporcional al *Figure of Merit Level (FoM).* Se mantendrá una distancia de separación mínima μ_{min} impuesta externamente y se añade otra constante (Q) que indica la calidad del método de interpolación:

$$\mu = \mu_{min} + \frac{Q}{N * FoM}$$

El cálculo geométrico de distancia 3D entre puntos que se puede hacer es sencillo debido al muestreo y a la conversión de coordenadas geodésicas a cartesianas.

Usamos la ecuación de distancia entre dos puntos en cada instante t bajo una serie de restricciones que conforman la distancia de seguridad vertical y horizontal de la aeronave.

Por otra parte, la ejecución de este kernel nunca provocará divergencia entre los hilos. Debido al algoritmo de interpolación cuadrática con alta tasa de muestreo implementado en la etapa anterior hemos conseguido que la comparación sea un problema SIMT (Single Instruction, Multiple Threads) óptimo para su implementación paralelizable, sin necesidad de usar instrucciones condicionales dentro del kernel. Se obtiene el siguiente speedup con respecto a una versión secuencial:

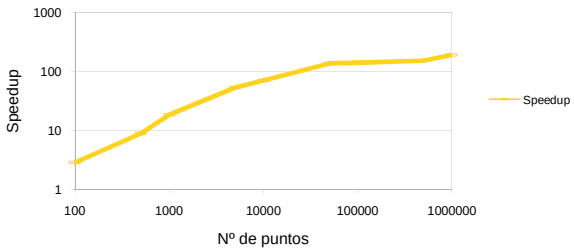


Figura 8: Speedup en el algoritmo de detección geográfica

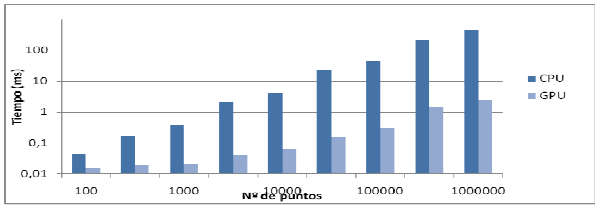


Figura 9: Tiempo CPU vs GPU para la de detección geográfica de conflictos

Análisis de rendimiento

En las tablas a continuación se pueden observar los tiempos de ejecución de cada etapa. Dependiendo del número de puntos de entrada, se pueden comparar también los tiempos de ejecución total en cada caso para las implementaciones en GPU y en CPU respectivamente:

Nº Puntos	Transferencia CPU to GPU	Conversión de Coordenadas	Interpolación Cuadrática	Detección Geométrica	Transferencia GPU to CPU	Total
100	0,009	0,014	0,051	0,015	0,009	0,098
500	0,009	0,017	0,126	0,019	0,010	0,181
1000	0,010	0,020	0,126	0,020	0,010	0,186
5000	0,012	0,044	0,169	0,038	0,031	0,294
10000	0,013	0,072	0,185	0,060	0,023	0,352
50000	0,015	0,225	0,399	0,155	0,100	0,895
100000	0,016	0,436	0,635	0,302	0,174	1,563
500000	0,050	1,958	2,780	1,417	0,787	6,992
1000000	0,108	3,688	5,003	2,267	1,611	12,677

Figura 10: Resumen de tiempos en la implementación GPU (ms)

Nº Puntos	Conversión de Coordenadas	Interpolación Cuadrática	Detección Geométrica	Total
100	0,002	0,042	0,010	0,054
500	0,004	0,165	0,083	0,251
1000	0,007	0,375	0,092	0,474
5000	0,168	2,037	0,343	2,548
10000	0,365	4,205	1,792	6,362
50000	1,337	21,359	12,078	34,774
100000	2,754	42,719	22,666	68,139
500000	13,805	216,621	132,267	362,693
1000000	27,920	432,776	261,121	721,817

Figura 11:Resumen de tiempos en la implementación CPU (ms)

CONCLUSIONES

Cuando un problema se puede descomponer hasta convertirlo en subtareas independientes que ejecutan una misma instrucción para cada uno de los datos de entrada, la capacidad de ejecutar rápidamente cálculos intensivos de la GPUs hace que el resultado de esta descomposición sea por norma general más eficiente que la versión secuencial del algoritmo. En el caso de la comparación de trayectorias, fruto de una necesidad real que nace en los actuales sistemas de Control de Tráfico Aéreo, el uso de GPUs parece altamente recomendable. La escalabilidad, el bajo coste y el mínimo periodo de aprendizaje inicial de CUDA hacen que sea una opción interesante a tener en cuenta para múltiples desarrollos. La tarjeta gráfica empleada, NVIDIA GeForce GTX660, es adecuada para este tipo de problema puesto que el número de procesadores y la elevada tasa de transferencia de memoria juegan un papel fundamental para obtener el rendimiento de cálculo esperado. Cabe destacar también la capacidad de aritmética de doble precisión de esta tarjeta, cumpliendo la necesidad que se tenía al inicio de este estudio, y que ha sido usada en la definición de coordenadas para todas las etapas descritas.

El objetivo de conseguir un algoritmo que responda en tiempo real al problema ha sido satisfecho, concatenando los diferentes algoritmos que hemos explicado y que han sido seleccionados tratando de primar el rendimiento y la simplicidad para conseguir una fuerte base a desarrollar en el futuro.

TRABAJO FUTURO

Como trabajo futuro quedan principalmente dos tareas. Por un lado, generalizar la comparación geométrica entre dos aeronaves a N de forma distribuida mediante un grid espacio-tiempo apropiado que pueda ser optimizado usando Quadtrees o distintos métodos ya evaluados por la comunidad científica. Por otra parte, complementar estos resultados con otros métodos y otras arquitecturas que cubran las situaciones para las que la paralelización pueda llegar a ser no adecuada, o necesite de otras cualidades que no perjudiquen el rendimiento en GPUs, como por ejemplo, debido a que no sea necesario muestrear las trayectorias para detectar conflictos tempranos. En este contexto, la posibilidad de utilizar un acelerador tipo Intel-Xeon-Phi reusando parte del código implementado en este trabajo podría ser muy recomendable.

Cabe también la posibilidad de estudiar en el futuro otras aproximaciones suponiendo que las trayectorias recibidas en el EPP puedan ser complementadas con métodos tradicionales de predicción de trayectorias, constituyendo sistemas más complejos e incluso colaborativos entre distintos Centros de Control Aéreo.

AGRADECIMIENTOS

Este trabajo se ha financiado con los proyectos TIN 2008/508 y TIN 2012/32180.

REFERENCIAS

[1] Torres, S.; Klooster, Joel K.; Liling Ren; Castillo-Effen, M., "Trajectory Synchronization between air and ground trajectory predictors," Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th , vol., no., pp.1E3-1,1E3-14, 16-20 Oct. 2011.

[2] Weber, R.; Cruck, E., "Subliminal ATC Utilizing 4D Trajectory Negotiation," Integrated Communications, Navigation and Surveillance Conference, 2007. ICNS '07 , vol., no., pp.1,9, April 30 2007-May 3 2007.

[3] Chan, D.S.K.; Brooksby, G.W.; Hochwarth, J.; Klooster, Joel K.; Torres, S., "Air-ground trajectory synchronization — Metrics and simulation results," Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th , vol., no., pp.1E1-1,1E1-14, 16-20 Oct. 2011.

[4] Mejia, J.S.; Stipanovic, D.M., "Safe trajectory tracking for the two-aircraft system," Electro/Information Technology, 2007 IEEE International Conference on , vol., no., pp.362,367, 17-20 May 2007.

[5] M.M. Paglione, H.F.Ryan.Trajectory Prediction Accuracy Report U.S. Deparment of Transportation ,1999.

[6] Jamrok, R., "Aircraft datalink communications for the future," Digital Avionics Systems, 2001. DASC. 20th Conference , vol.2, no., pp.6A6/1,6A6/9 vol.2, Oct 2001.

[7] F. J. Martín Crespo, The Collision Avoidance Problem: Methods and Algorithms, Universidad Rey Juan Carlos, 2010.

[8] Zhang Zhong; Zhang Xuejun, "An improved geometric approach to conflict resolution in curve trajectory," Electronic Measurement & Instruments, 2009. ICEMI '09. 9th International Conference on , vol., no., pp.1-220,1-224, 16-19 Aug. 2009.

[9] Ji-wung Choi; Curry, R.; Elkaim, G., "Path Planning Based on Bézier Curve for Autonomous Ground Vehicles," World Congress on Engineering and Computer Science 2008, WCECS '08. Advances in Electrical and Electronics Engineering - IAENG Special Edition of the , vol., no., pp.158,166, 22-24 Oct. 2008.

Aplicaciones de la Computación de Altas Prestaciones

Improving the Server Performance of CAR Systems Based on Mobile Phones

Víctor Fernández, Juan Manuel Orduña and Pedro Morillo [1]

Resumen— **Collaborative Augmented Reality (CAR) systems allow multiple users to share a real world environment including computer-generated images in real time. The hardware features of most current mobile phones include wireless network capabilities that offer a natural platform for CAR systems. However, the potential number of clients in CAR systems based on mobile phones is much larger than on CAR systems based on other kind of mobile devices, requiring a system design that takes into account scalability issues. This paper presents the experimental comparison of different CAR systems based on mobile phones with different server implementations. The performance evaluation results show that the best implementation is the one based on UDP messages instead of classical TCP connections, in order to improve the system throughput. The UDP-based implementation provides a significant improvement in system throughput, at the cost of loosing a very small percentage of updating messages. However, the effects of these small quantities of dropped messages cannot expand beyond some jitter (bounded within a short period of time) in a reduced number of clients of the CAR application. These results validate the proposed UDP-based implementation as the best option for large-scale CAR systems based on mobile phones.**

Palabras clave— **Collaborative Augmented Reality, Mobile Phones.**

I. Introduction

AUGMENTED Reality (AR) systems are nowadays widely used in applications such as medical procedures, scientific visualization, manufacturing automation, cultural heritage and military applications. The term Augmented Reality (AR) refers to computer graphic procedures or applications where the real-world view is superimposed by computer-generated objects in real-time [1], [2]. From the beginning of AR systems, the potential of collaborative AR (CAR) systems was exploited for different activities such as Collaborative Computing or Teleconferencing [3]. Wearable devices were used to provide CAR systems, where a wearable AR user could collaborate with a remote user at a desktop computer [4]. On other hand, mobile phones have become an ideal platform for CAR systems, due to the multimedia hardware that they include. As an example, Figure 1 shows a CAR system developed for collaborative training in industrial electricity. It shows on the left image the execution of the CAR tool on a Samsung Galaxy NOTE mobile phone, while the image on the center shows a real image of the the panelboard where technicians collaboratively operate, and the right image shows the execution of the CAR tool on a HTC Nexus One mobile phone.

Fig. 1. Example of a CAR application developed for training in industrial electricity.

The wide variety of current mobile phones, with different graphic and processing capabilites, and different operating systems, can have significant effects on the performance of a large-scale CAR system, in terms of system latency, frames per second or number of supported clients with certain latency levels. In previous works, we have characterized the behavior of different mobile phones and the server when used in Collaborative Augmented Reality applications, [5], [6]. The results showed that CAR systems throughput heavily depends on the kind of client devices, but for certain kind of devices, the system bottleneck is the server I/O.

In this paper, we propose a comparative study of different implementations of the CAR server, in order to improve the performance of CAR systems based on mobile phones. The performance evaluation results show the UDP-based implementation provides a significant improvement in system throughput with respect to other implementations, supporting more than one thousand clients at interactive rates (twice the number of supported clients of the TCP implementation). This improvement is achieved at the cost of loosing a very small percentage of updating messages but the effects of these dropped messages cannot expand beyond some jitter (bounded within a short period of time) in a reduced number of clients.

The rest of the paper is organized as follows: Section II shows some related work about CAR applications on mobile phones. Section III describes the different CAR implementations considered for comparison purposes, and Section IV shows the performance evaluation results. Finally, Section V presents some conclusion remarks.

The rest of the paper is organized as follows: Section II shows some related work about CAR applications on mobile phones. Section III describes the different CAR implementations considered for comparison purposes, and Section IV shows the perfor-

[1]Dpto. de Informática, Univ. of Valencia, Valencia, Spain. E-mails: {Victor.Fernandez-Bauset, Juan.Orduna,Pedro.Morillo}@uv.es

mance evaluation results. Finally, Section V presents some conclusion remarks.

II. Related Work

Augmented Reality superimposes multimedia content - 3D object, text, sound, etc - to real world through a display or screen. In order to locate digital contents on a specific image of the real world point, some references within the image are needed. These references are known as markers, and two methods are usually used: natural feature tracking and fiducial marker tracking. The former method uses interest point detectors and matching schemes to associate 2D locations on the video with 3D locations [7]. This process can be grouped in three big phases: interest point detection, creation of descriptor vectors for these interest points, and comparison of vectors with the database [8]. The latter method uses fiducial markers to find a specific position of real world. This process can be divided in three phases: edge detection, rejection of quadrangles that are too large or too small, and checking against the set of known patterns [7].

Any CAR application needs a device equipped with an on-board camera, CPU and display. The most common devices used for CAR applications are Tablet-PCs or mobile phones. We will focus on mobile phones, because they are more suitable for CAR applications [9].

There are few solutions based on fiducial marker tracking over mobile phones. In 2003, ArToolKit [10], one of the most well-known software libraries for developing Augmented Reality (AR) application, was released for Windows CE, and the first self-contained application was developed for mobile phones [11]. This software evolved later as the ArToolKitPlus tracking library [7]. A tracking solution for mobile phones that works with 3D color-coded marks was developed [12], and a version of ArToolKit for Symbian OS was developed, partially based on the ArToolKitPlus source code [13]. The research teams behind these works have worked on fiducial marker tracking, but not from the collaborative point of view. Also, there are many other works that focus on natural feature tracking [7], [14], [15], [16].

Although real-time natural feature tracking over mobile devices has been currently achieved [7], fiducial marker tracking is more widely used, because it allows simultaneous computational robustness and efficiency. A large number of locations and objects can be efficiently labeled by encoding unique identifiers on the markers. Additionally, the markers can be detected with angles near to 90 degrees [7].

The first CAR applications improved the conference system highlights, giving the feeling of real presence to remote collaborators [3]. The Rekimoto's Transvision system showed how to share virtual objects through handheld displays [17]. Also, Schmalstieg created a software architecture to develop CAR applications [18].

III. Server implementations

We have developed a multithreaded CAR server that supports simulated clients (simulated mobile devices) with the behavior measured in our previous work [5]. The system configuration consists of this server and a certain amount of mobile devices that are scanning the visual space of their video camera looking for a marker that will be converted into a 3D object in their display. After each updating of the object location, the mobile device sends a location update message (containing the new location) to each of its neighbor devices. The neighbor devices are those who participate in the same collaborative task, and we have denoted this set of neighbor devices as a *working group*. The messages are sent through the server (that is, it sends the location update message to the server, and then the server re-sends the message to the appropriate clients). For performance evaluation purposes, the destination clients return an acknowledgment message (ACK) to the server, which, in turn, forwards it to the source client. When the source client has received the ACK messages corresponding to the location update from all the clients in its working group, then it computes the average system response for that location update. Figure 2 illustrates the action cycle that takes place for each of the mobile clients in the system.

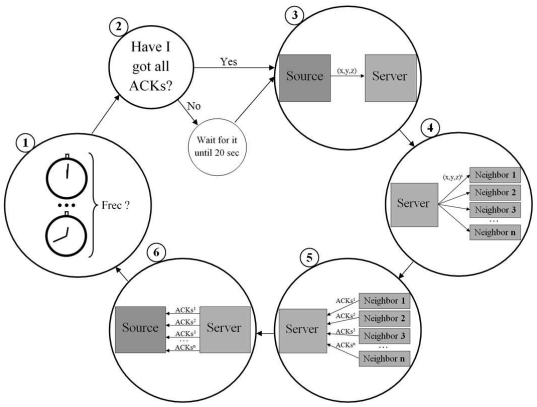


Fig. 2. Stages of the action cycle in each mobile device.

Once the message with the location update is sent, the action cycle performed by each client is composed of the following steps: first, it performs one new image acquisition followed by a marker detection stage. Then, the client waits until the cycle period (determined by the action frequency, a system parameter) finishes. Next, if the acknowledgments from all the neighbors have been received, a new message with the new marker location is sent. If not all the acknowledgments have been received, then it waits until a maximum threshold of 20 seconds, and then a new round of messages (with the latest marker location) are sent to the neighbors through the server. The neighbors simply return an ACK message to the sender device through the server. The server simply forwards the messages to the corresponding destination clients. It must be noticed that

the mobile devices will not send a new round of messages with a new location update until it has received the acknowledgment message from all its neighbors, even although new marker detection stages have been completed in the device.

This characterization setup considers that all the required static content in the scene has been loaded. According to recent works [19], in these cases the network bandwidth required is less than 50 kbps for performing this information exchange. Since we are using a Gigabit Ethernet, we ensure that the network bandwidth does not become a system bottleneck.

The system latency provided for each location update is computed by recording a timestamp when the first message is sent to the server. Next, a second timestamp is recorded with the last ACK message for that location update received from the server. The system response time is computed by subtracting these two timestamps. The server response time is computed by timestamping both each message forwarded from each client and the reception of the corresponding ACK message from the destination client. Also, the percentage of CPU utilization is measured both in the server and the mobile devices every half second.

A. TCP Implementation

The simulator starts generating a *Server Process*, and for every 50 clients it generates a *Client Process*. Figure 3 illustrates the general scheme of the Server Process. This process starts listening connections, and for each connection it generates a new array of X TCP sockets, where X is the number of clients that will be within a given working group. When all the clients have connected to the Server Process (the population size is a simulation parameter) then the Server Process generates as many *Server Threads* as needed. Each Server Thread is in charge of managing all the clients within a working group. Concretely, it starts the simulation by sending a welcome message to all the client sockets. When the simulation finishes, it collects statistics from all the clients in its working group. But the most important task performed by server threads is the generation of two threads for each of the clients in the working group: the *Server Receiver Thread (SRT)* and the *Server Processor Thread (SPT)*. The SRT associated to client i receives the location update messages from the client i. Next, it computes the correct destination clients (the neighbor clients, that is, the clients within the same working group) and it generates messages that will be stored in the queues of the Server threads managing these neighbor clients. The SPT associated to client i extracts the queued messages that the SRTs associated to other clients may have generated for client i, and it sends them to this client. Additionally, the server process collects and processes the statistics generated by the server threads, and it also measures the percentage of CPU utilization.

Figure 4 illustrates the general scheme of the

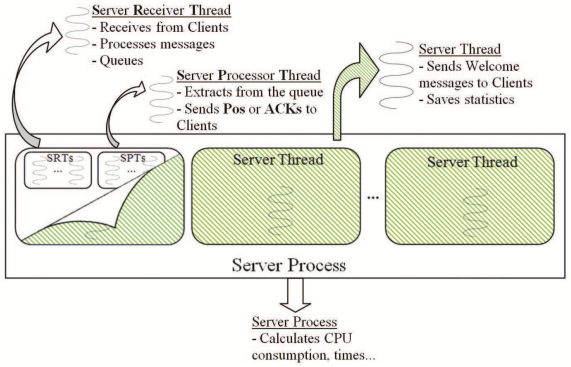


Fig. 3. General scheme of the server process in the TCP implementation.

Client Process. This process generates 50 client threads (we have assumed a maximum population size of 1000 client devices), and it also computes the percentage of CPU utilization, client latencies, etc.. Each Client Thread generates two threads for each client: the *Client Receiver Thread (CRT)* and the *Client Processor Thread (CPT)*, and when the welcome message from the Server Thread arrives to the associated socket, then the Client Thread starts the simulation, that consists of sending a given number of position update messages and receiving the corresponding acknowledgments from the neighbor clients.

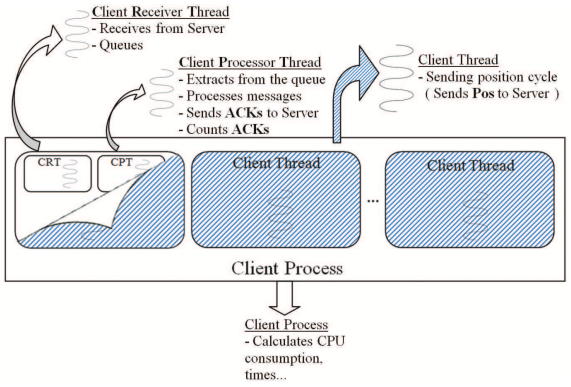


Fig. 4. General scheme of the client process in the TCP implementation.

Also, we developed another version where each Server Thread has a single SRT and a single SPT for managing all the clients in each working group, instead of one SRT and one SPT for each client. Using the `Select` function, the SRT receives messages from all the clients and it processes them. As it could be expected, we obtained better performance results with the Select version of the TCP implementation.

B. UDP Implementation

Finally, we have considered a connectionless oriented implementation for the CAR system, in order to study the effectiveness of TCP connections in a distributed environment like a CAR system. The motivation of this study are both the short message size (usually carry a position update consisting of a

bunch of bytes) and the huge amount of the messages generated by CAR systems. Although the UDP protocol can loose messages and the effects and size of these losses should be studied, we have also considered this implementation for comparison purposes. The UDP implementation is very similar to the TCP-Select implementation. The only difference is that in this implementation we have used UDP sockets. Since this implementation can drop messages, it also counts the number of dropped or lost messages (since both the number of iterations and the number of clients in each working group is known, each client can compute the number of message that should arrive).

IV. Performance Evaluation

We have performed different measurements on different simulated systems using these implementations. We have performed simulations with different number of clients and we have measured the response time provided to these clients (the round-trip delay for each updating message sent by a given client to the clients in its working group). In this way, we can study the maximum number of clients that the system can support while providing a response time below a given threshold value. In order to define an acceptable behavior for the system, we have considered 250 ms. as the threshold value, since it is considered as the limit for providing realistic effects to users in DVEs [20].

We have considered the system response time (in milliseconds) for each updating message sent by a given client to its neighbor clients as the time required for receiving the acknowledgments from all the clients in the working group of this given client. In order to measure the dispersion of this metric, we have measured the standard deviation for all the updating messages sent, as well. Also, we have computed the response time in the server (in milliseconds) as the time required by the destination clients to answer the server messages. We have measured both the average and the maximum values measured in the server for each simulation. Additionally, we have computed the percentage of the CPU utilization in the system server, since it can easily become the system bottleneck. The computer platform hosting the system server is a Intel Core 2 Duo E8400 CPU running at 3.00 GHz with 4 Gbytes of RAM, executing an Ubuntu Linux distribution with the 3.0.0-14-generic x86 64 operating system kernel. In order to study the system behavior for different levels of workload, we have repeated simulations with working group sizes of 10,15,20 and 25 clients. Due to space limitations, we only show here the results for the biggest size (25 clients in each working group).

Table I shows the results for a CAR system whose client devices are all of them HTC Nexus One, and where the working group size for each client is of five neighbor clients. This table shows the results for the three considered implementations, organized as three subtables with ten rows each, and labeled with the name of the implementation (TCP, TCP-Select and UDP). The most-left column in these subtables shows the number of clients in the system, that is, the population size. The values in this column range from 100 to 1000 clients in the system. The next two columns show the average value of the response times (in milliseconds) provided by the system to all the clients (labeled as "RT"), as well as the corresponding standard deviation values (column labeled as "Dev"). The fourth column (labeled as "CPU") shows the percentage of the CPU utilization in the server. Finally, the fifth and sixth columns (labeled as "RT_S" and "RT_SM", respectively) show the average and maximum values (in milliseconds) of the response time in the server for all the messages exchanged during the simulation.

TABLA I
Results for a working group size of 5 neighbors

	TCP implementation				
Size	RT	Dev	CPU	RT_S	RT_SM
100	62.37	22.68	9.9	19.36	20.9
200	63.77	22.21	15	20.12	22.43
300	66.71	22.66	22	25.15	28.28
400	68.68	22.5	32.7	27.14	29.56
500	71.04	23.56	45	27.14	30.9
600	71.5	24.18	48.6	26.58	31.95
700	72.37	25.01	59	27.17	35.42
800	72.85	26.01	68	27.87	34.54
900	75.01	28.98	79.2	28.61	35.89
1000	147.33	101.71	85	43.95	48.66
	TCP-Select implementation				
Size	RT	Dev	CPU	RT_S	RT_SM
100	65.77	21.56	8	20.44	24.96
200	67.12	22.71	11.2	21.5	23.54
300	67.52	22.6	19.8	23.67	26.62
400	67.64	22.88	28	23.21	27.28
500	69.12	23.23	31	25.31	29.34
600	69.14	23	39.6	25.28	28.74
700	69.37	23.45	47	24.53	30.72
800	75.75	26.63	54.5	26.96	35.16
900	70.24	24.81	59.6	24.02	31.87
1000	71.05	27.53	67	22.64	35.04
	UDP implementation				
Size	RT	Dev	CPU	RT_S	RT_SM
100	4.80	7.06	38.40	1.95	3.22
200	3.76	4.95	34.70	1.57	2.84
300	9.57	9.74	26.00	4.44	10.97
400	3.60	5.18	33.70	1.46	3.16
500	4.59	6.41	41.60	1.81	3.77
600	7.34	13.17	47.00	3.23	17.16
700	5.28	8.24	53.00	2.11	7.76
800	7.10	18.52	84.10	2.65	16.53
900	5.85	11.69	66.30	2.61	12.05
1000	7.15	15.47	69.50	2.87	15.18

Table I shows that none of the values in the RT column reaches the threshold value of 250 milliseconds in any of the considered implementations, showing that the system can efficiently support up to one thousand clients while interactively displaying the Augmented Reality content. Nevertheless, there are significant differences in this column among the considered implementations. Thus, the TCP implementation shows a huge rise in the response time when the system reaches one thousand clients, passing from around 75 milliseconds to more than 147 milliseconds as an average. The standard deviation

of these values are also more than three times the value shown for nine hundred clients. These values show that for that population size the system is approaching saturation. On the contrary, the TCP-Select implementation does not show an increase in neither the column RT nor the column Dev for a population of one thousand clients. Moreover, the UDP implementation shows RT values that are one order of magnitude lower than the ones shown by the other two implementations.

The third column in table I shows that the CPU utilization increases as the number of clients in the system increases. In the case of the TCP implementation, the system approaches saturation when the server reaches 85% of CPU utilization. For lower percentages of CPU utilization the response times do not significantly increase. It is worth mention that the UDP implementation provides RT values that are one order of magnitude lower than the ones provided by the TCP implementations, even for CPU utilization of around 70%. These values show that the latency provided by CAR systems greatly depends on the connection or connectionless scheme followed by the system to exchange information with the clients.

Finally, the columns RT_S and RT_SM show that most of the response time provided to clients is due to processing in the server. Thus, for example, the results for the TCP implementation and a system size of 900 clients show that, as an average, each client has to wait 75.01 milliseconds for receiving the acknowledgments from all the clients in its working group, but as an average the server must wait only 28.61 milliseconds to receive answers from clients. This difference highly increases for the case of one thousand clients, where the response time obtained by the server from clients is around 44 milliseconds but the average response time provided to clients is 147.33 milliseconds, around three times higher. It is also worth mention that the ratio between the RT_S and the RT columns do not significantly vary among the three implementations. Finally, the RT_SM column shows that the maximum values in the RT_S parameter do not exceed the value in the RT column for both TCP implementation, and they do not exceed twice the value in the RT column of the UDP implementation. Therefore, we can conclude that most of the time required to acknowledge each client update is due to the processing of the updates and acknowledgments in the server.

These results show that the best latencies when the system is far from saturation are provided with the UDP implementation. However, UDP is a connectionless-oriented protocol, and therefore it may drop messages when the system approach saturation.

Table II shows the results for a working group size of 25 clients. The most-left column in these subtables shows the number of clients in the system, that is, the population size. The values in this column range from 100 to 1000 clients in the system. The next two columns show the average value of the response times (in milliseconds) provided by the system to all the clients (labeled as "RT"), as well as the corresponding standard deviation values (column labeled as "Dev"). The fourth column (labeled as "CPU") shows the percentage of the CPU utilization in the server. Finally, the fifth and sixth columns (labeled as "RT_S" and "% lost", respectively) show the average values (in milliseconds) of the response time in the server for all the messages exchanged during the simulation and the percentage of messages dropped by the system. The latter column has been computed by subtracting the number of messages received by all the clients in a simulation (measured in the simulation itself) from the theoretical number of messages that clients should exchange for a given population size.

TABLA II
RESULTS FOR A WORKING GROUP SIZE OF 25 NEIGHBORS

	TCP-Select implementation				
Size	RT	Dev	CPU	RT_S	% lost
100	90.8	24.7	23.2	19.35	0.00
200	89.95	21.13	47	33.4	0.00
300	123.95	32.36	72	54.7	0.00
400	209.2	35.88	87.2	85.55	0.00
500	268.17	44.44	86	112.07	0.00
700	383.96	70.6	93.1	151.56	0.00
1000	566.44	133.33	93.1	166.79	0.00
	UDP implementation				
Size	RT	Dev	CPU	RT_S	% lost
100	9.86	6.78	72.50	4.06	0.83
200	21.70	14.73	82.00	9.84	1.18
300	26.01	21.91	79.60	11.61	0.69
400	39.41	30.66	81.90	18.26	0.83
500	48.68	39.68	83.80	22.84	0.74
700	79.70	97.87	85.10	37.26	0.76
1000	122.37	85.35	85.00	44.98	0.90

Table II shows that for this level of workload the system enters saturation in the TCP-based implementation. Effectively, the RT column shows that TCP-Select implementation reaches (and exceeds) this threshold value for a population of 500 clients. However, the UDP implementation does not reach even half of this value for the maximum population size considered, one thousand clients.

It is worth mention that for those cases when the system reaches saturation, the percentage of CPU utilization in the server is 85% or higher. The gap between 85% and 98% of CPU utilization for reaching the saturation point can be explained by the shared memory architecture of current multicore processors (the dual core processor in the computer platform used as simulation server), as shown in [6].

The "% loss" column shows that for the UDP implementation the percentage of lost messages is not higher than 1.2%. The effects of loosing some messages will consist of producing some jitter in the display of the clients. However, these percentage values ensure a reasonable quality in the visualization of the CAR system. In order to ensure that the effects of the UDP implementation in terms of dropped messages are consistent for all the workload levels considered, Figure 5 shows the average number of packets

dropped for each working group size considered.

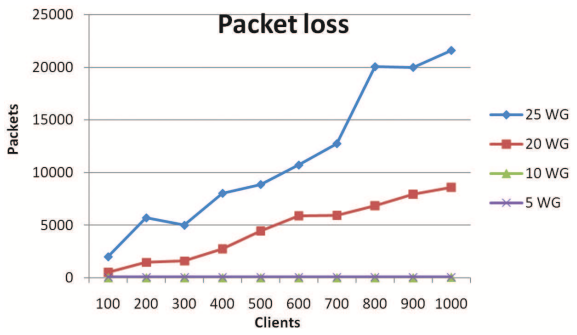


Fig. 5. Number of packets lost in the UDP implementation.

Figure 5 shows that for working group sizes of 5 and 10 neighbor clients there are no packet losses. For a working group size of 20 neighbors, the amount of lost packets reaches 8581 for a theoretical total of packets sent of 1.9 million packets. Analogously, for a working group size of 25 neighbors, the amount of lost packets reaches 21593 out of 2.4 million packets sent. Therefore, in the worst case the number of lost packets only represent a 1'18 % of the total amount of packets sent. This value represents only a small image flicker on some clients, and in very limited periods of time. As the information is sent more than once per second (since the action cycle of the HTC Nexus One is 167.11 ms.), this value can be considered an insignificant flickering.

Although they are not here for the shake of shortness, we repeated the same tests shown in this section here using a different client device, the Motorola Milestone, and we obtained analogous results. Those results were less interesting because of the bigger action cycle of the Milestone (698.34 ms.). With that action frequency the system saturation point was not reached even in the worst case of a working group size of 25 neighbors and a population of 1000 clients. We have shown here the results for the Nexus One as the worst case for the server implementation.

V. CONCLUSIONS

This paper has proposed the experimental comparison of different large-scale CAR systems based on mobile phones with different server implementations. The performance evaluation results show that the best implementation is the one based on UDP messages, instead of classical TCP connections, in order to support more than one thousand clients at interactive rates. These results validate the proposed UDP-based implementation as the best option for large-scale CAR systems based on mobile phones.

ACKNOWLEDGEMENTS

This work has been jointly supported by the Spanish MICINN and the European Commission FEDER funds, under grant TIN2009-14475-C04.

REFERENCIAS

[1] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *Computer Graphics and Applications, IEEE*, vol. 21, no. 6, pp. 34 –47, 2001.

[2] S. Cawood and M. Fiala, *Augmented Reality: A Practical Guide*, Pragmatic Bookshelf, 2008.

[3] M. Billinghurst and H. Kato, "Real world teleconferencing," in *Proc. of the conference on Human Factors in Computing Systems (CHI 99)*, 1999.

[4] Tobias Hallerer, Steven Feiner, Tachio Terauchi, Gus Rashid, and Drexel Hallaway, "Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system," *Computers and Graphics*, vol. 23, pp. 779–785, 1999.

[5] Victor Fernández Bauset, Juan M. Orduña, and Pedro Morillo, "Performance characterization on mobile phones for collaborative augmented reality (car) applications," in *Proceedings of the 2011 IEEE/ACM 15th DS-RT*, 2011, DS-RT '11, pp. 52–53.

[6] Victor Fernández Bauset, Juan M. Orduña, and Pedro Morillo, "On the characterization of car systems based on mobile computing," in *Proceedings of HPCC '12 (AHPCN workshop)*, 2012.

[7] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg, "Pose tracking from natural features on mobile phones," in *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Washington, DC, USA, 2008, ISMAR '08, pp. 125–134, IEEE Computer Society.

[8] Seung Eun Lee, Yong Zhang, Zhen Fang, S. Srinivasan, R. Iyer, and D. Newell, "Accelerating mobile augmented reality on a handheld platform," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, October 2009, pp. 419 –426.

[9] Michael Haller; Mark Billinghurst; Bruce Thomas, *Emerging Technologies of Augmented Reality: Interfaces and Design*, IGI Global, 2007.

[10] Dr. Hirokazu Kato, "Artoolkit," 2011, Available at http://www.hitl.washington.edu/artoolkit/.

[11] Daniel Wagner and Dieter Schmalstieg, "First steps towards handheld augmented reality," in *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, Washington, DC, USA, 2003, ISWC '03, pp. 127–135, IEEE Computer Society.

[12] Mathias Mahring, Christian Lessig, and Oliver Bimber, "Video see-through ar on consumer cell-phones.," in *ISMAR'04*, 2004, pp. 252–253.

[13] A. Henrysson, M. Billinghurst, and M. Ollila, "Face to face collaborative ar on mobile phones," in *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, October 2005, pp. 80 – 89.

[14] S. Srinivasan, Zhen Fang, R. Iyer, S. Zhang, M. Espig, D. Newell, D. Cermak, Yi Wu, I. Kozintsev, and H. Haussecker, "Performance characterization and optimization of mobile augmented reality on handheld platforms," in *Workload Characterization. IISWC 2009. IEEE International Symposium on*, 2009, pp. 128 –137.

[15] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-time detection and tracking for augmented reality on mobile phones," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 3, pp. 355 –368, May-June 2010.

[16] D. Wagner, D. Schmalstieg, and H. Bischof, "Multiple target detection and tracking with guaranteed framerates on mobile phones," in *Proceedings of ISMAR 2009*, 2009, pp. 57 –64.

[17] Jun Rekimoto, "Transvision: A hand-held augmented reality system for collaborative design," in *Virtual Systems and Multi-Media (VSMM)'96*, 1996.

[18] Z. Szalavari, D. Schmalstieg, A. Fuhrmann, and M. Gervautz, "studierstube: An environment for collaboration in augmented reality," *Virtual Reality*, vol. 3, pp. 37–48, 1998.

[19] Tuomas Kantonen, "Augmented collaboration in mixed environments," M.S. thesis, Helsinky University of Technology, 2009.

[20] T. Henderson and S. Bhatti, "Networked games: a qos-sensitive application for qos-insensitive users?," in *Proceedings of the ACM SIGCOMM 2003*. 2003, pp. 141–147, ACM Press / ACM SIGCOMM.

Resolución del problema del líder-seguidor con demanda endógena mediante algoritmos paralelos

A.G. Arrondo, J. Fernández [1], J.L. Redondo[2], P.M. Ortigosa[3]

Resumen— **En el problema de localización competitiva presentado en este artículo, el objetivo es maximizar el beneficio obtenido por una cadena (el líder) sabiendo que la competencia (el seguidor) reaccionará a su entrada en el mercado localizando un nuevo centro. En contraposición a lo que se hace comúnmente en la literatura, se supone que la demanda es endógena, es decir, que varía en función de las distancias y la calidad de las instalaciones. Este supuesto aumenta la complejidad del problema y, por lo tanto, el esfuerzo computacional necesario para resolverlo. Recientemente, se ha propuesto un algoritmo evolutivo de optimización global, llamado TLUEGO, para resolver este problema del líder-seguidor. Sin embargo, este problema es tan complejo y computacionalmente costoso que TLUEGO es incapaz de manejar problemas de gran tamaño. En este trabajo se proponen tres versiones paralelas de TLUEGO, capaces no solo de obtener la solución más rápidamente sino también de resolver problemas más grandes. En particular, se propone un algoritmo de programación de memoria distribuida (apto para multicomputadores), un algoritmo de programación de memoria compartida (apto para multiprocesadores, como la mayoría de los ordenadores de hoy en día) y un algoritmo híbrido (apto para clusters). Los estudios computacionales muestran que los tres paralelizaciones son eficientes.**

Palabras clave— **Algoritmos evolutivos, algoritmos evolutivos paralelos, MPI, OpenMP, programación híbrida**

I. Introducción

La ciencia de la localización trata de localizar uno o más comercios de forma que se optimize una o varias funciones objetivo (minimización de los costes de transporte, minimización del coste social, maximización de la cuota de mercado, etc). El lector interesado puede consultar [1] y [2] para una introducción al tema. Cuando una cadena minorista considera entrar o ampliar su presencia en un mercado donde ya existen otros comercios que ofrecen los mismos productos, la nueva instalación tendrá que *competir* por el mercado. La localización competitiva es un área de investigación muy activa (véase, por ejemplo, los siguientes artículos [3], [4] y [5]).

En la mayoría de los trabajos de localizacíon competitiva presentes en la literatura se asume que la demanda es fija, sin tener en cuenta las condiciones del mercado. Esto puede ser apropiado cuando se trata de productos esenciales. Sin embargo, la demanda puede ser endógena, es decir, puede variar dependiendo de varios factores. Como se muestra en [6], el gasto de los consumidores puede incrementarse por varias razones relacionadas con la localización de un nuevo centro: las ofertas pueden aumentar la utilidad de un producto; los gastos de marketing derivados del nuevo centro pueden aumentar la presencia del producto, dando lugar a la demanda creciente de los consumidores; o algunos clientes que no compraban en ningún centro por no tener ninguno cercano pueden verse ahora inducidos a ello. Por otro lado, la calidad de los centros puede afectar a la demanda, ya que un mejor servicio conduce a más ventas. El considerar la demanda variable aumenta en gran medida la complejidad y carga computacional del problema, pero hace que el modelo sea más realista.

En este trabajo, se considera demanda endógena (véase [7]). En particular, el escenario considerado, que fue introducido en [8], es un *duopolio*. Una cadena (el líder) quiere aumentar su presencia en el mercado localizando un nuevo centro en una región del plano, donde ya existen otros comercios de la competencia (el seguidor), y posiblemente del suyo propio, ofertando el mismo producto. El objetivo del líder es encontrar la localización y la calidad del nuevo centro que maximice su beneficio, sabiendo que, como reacción a su entrada en el mercado, la competencia localizará otro nuevo centro. Esta intrusión del seguidor en el mercado podría, no sólo reducir la ganancia obtenida por el líder, sino también modificar la posición en la que éste obtendría su máximo beneficio. Este tipo de problemas son conocidos como problemas de Stackelberg o de Simpson. En la literatura sobre Localización fueron introducidos por Hakimi [9], quien definió el término medianoide para el problema del seguidor, y centroide para el problema del líder.

Para resolver el problema del centroide, es necesario resolver muchos problemas medianoide, ya que la evaluación de la función objetivo del líder en un punto dado requiere la resolución del correspondiente problema del medianoide. El problema de medianoide fue analizado en [7], donde se propuso un algoritmo evolutivo llamado UEGO, que demostró ser fiable y robusto para la resolución de este problema. Por lo tanto, se considera UEGO como algoritmo para obtener la solución de los problemas medianoide. Debe tenerse en cuenta que la solución de un problema medianoide no es una tarea trivial, muy al contrario, es un problema de optimización global

[1]Dpto. Estadística e Investigación Operativa, Univ. Murcia, e-mails: {agarrondo,josefdez}@um.es

[2]Dpto. de Arquitectura y Tecnología, Univ. Granada, e-mail: jlredondo@ugr.es.

[3]Dpto. de Informática, Univ. Almería, e-mail: ortigosa@ual.es.

difícil de resolver (como la mayoría de los problemas de localización competitivos).

El problema centroide se estudió en [8]. Entre las tres heurísticas propuestas en ese trabajo, el algoritmo evolutivo de dos niveles llamado TLUEGO (que también sigue la estructura UEGO) fue el algoritmo más robusto y el que mejores resultados proporciona. Sin embargo, el tiempo de cálculo empleado por TLUEGO para resolver los problemas de tamaño pequeño era muy alto. Esto sugiere claramente la necesidad de una paralelización del algoritmo, sobre todo si se pretende resolver los problemas reales, con más puntos de demanda.

El objetivo de este trabajo es paralelizar el algoritmo TLUEGO, estudiando los modelos de programación más extendidos. El primero de ellos es un modelo basado en paso de mensajes, que puede ser usado en multicomputadores. El segundo es un modelo de memoria compartida, que puede ser usado en multiprocesadores (como la mayoría de PCs actuales). Y el último es una versión híbrida de los dos modelos anteriores, que pueda ser usada en clusters, donde varios nodos están conectados de acuerdo a un determinada topología (como los multicomputadores) y cada nodo está compuesto de varios procesadores que comparten la memoría (como los multiprocesadores).

El resto del trabajo se organiza como sigue: En la siguiente sección se define el problema. En la sección III se describe el algoritmo TLUEGO muy brevemente. En la sección IV se describen las versiones paralelas de TLUEGO desarrolladas. El artículo finaliza con los estudios computacionales (sección V) y con las principales conclusiones y líneas de trabajo futuro (sección VI).

II. Problema del líder-seguidor con demanda endógena

En esta sección se describe el problema del líder-seguidor con demanda endógena muy brevemente. El lector interesado puede dirigirse a [8] para una descripción más detallada.

El escenario considerado es el siguiente: una cadena, el líder, desea localizar un nuevo centro $nf_1 = (x_1, y_1, \alpha_1)$ en un área determinada del plano, donde existen m centros pre-existentes que ofrecen el mismo producto. Los primeros k centros de los m pre-existentes pertenecen a la cadena líder ($0 \leq k < m$) y el resto, es decir, $m - k$ centros pre-existentes, pertenecen a la cadena competidora, el seguidor. Se sabe que como reacción a la localización del nuevo centro por parte del líder, el seguidor localizará un nuevo centro $nf_2 = (x_2, y_2, \alpha_2)$. La demanda, que se supone endógena, se concentra en n puntos de demanda, cuya localización p_i y poder de compra son conocidos. La localización f_j y calidad de los centros pre-existentes también es conocida.

Siguiendo [10], suponemos que el comportamiento de los clientes es probabilístico, esto es, los puntos de demanda reparten su capacidad de compra entre los comercios proporcionalmente a la atración que sienten por ellos. La atracción que un punto de demanda siente por un comercio depende tanto de la localización del comercio como de su calidad (según la percibe el punto de demanda).

El objetivo es maximizar el beneficio obtenido por el líder teniendo en cuenta la entrada posterior del seguidor. Dicho beneficio se define como la diferencia entre las ganancia obtenida por la cuota de mercado capturada menos los costes.

Dado nf_1, el problema del seguidor es el problema del medianoide ($FP(nf_1)$), cuyo objetivo es encontrar la mejor localización (x_2, y_2) y calidad α_2 para su centro, de modo que se maximice su beneficio $\Pi_2(nf_1, nf_2)$ (una vez que el líder ha localizado su nuevo centro en nf_1). Denotemos por $nf_2^*(nf_1)$ una solución óptima de ($FP(nf_1)$). El problema del líder es el problema del centroide (LP), cuyo objetivo es encontrar la mejor localización (x_1, y_1) y calidad α_1 para su centro de modo que maximice su beneficio $\Pi_1(nf_1, nf_2^*(nf_1))$, y conociendo que el seguidor localizará otro nuevo centro $nf_2^*(nf_1)$ en el lugar y con la calidad que maximice su propio beneficio.

Como puede verse, el problema del líder (LP) es mucho más difícil de resolver que el problema del seguidor ($FP(nf_1)$). Note, por ejemplo, que para evaluar la función objetivo Π_1 en un punto dado nf_1, es necesario resolver primero el problema del medianoide ($FP(nf_1)$) para obtener $nf_2^*(nf_1)$. Además, con el fin de calcular el valor de la función objetivo Π_1 en nf_1 con precisión, el problema del seguidor ($FP(nf_1)$) tiene que ser resuelto con exactitud, ya que en otro caso, el error derivado de valores aproximados puede ser considerable.

Algorithm 1 TLUEGO

```
1: Init_population
2: for t = 1 to L do
3:    Create_species(new_t)
4:    Selection(max_spec_num)
5:    Optimize_species(n_t)
6:    Selection(max_spec_num)
7: end for
```

III. TLUEGO: algoritmo secuencial

TLUEGO es un algoritmo evolutivo multimodal basado en subpoblaciones (denominadas especies). Su estructura principal (ver Algoritmo 1) es muy parecida a la de otros algoritmos evolutivos, donde hay una fase de inicialización de la población y un bucle iterativo destinado a la evolución de la población hacia los óptimos, mediante la creación, selección y optimización de especies.

En este contexto, una especie es una esfera definida por su centro y su radio. El centro es una solución y el radio indica su área de atracción, que cubre una región del espacio de búsqueda, y por tanto, varias soluciones. La característica más importante de estas especies es que para poder evaluarlas y, por tanto, poder determinar su aptitud, es necesario ejecutar otro algoritmo de optimización global. Esto hace que

el proceso de evaluación sea especialmente costoso.

Durante el proceso de optimización, TLUEGO mantiene una lista de especies. Cada especie de la lista trata de ocupar un máximo local de la función objetivo, sin conocer el número total de máximos existentes. Esto significa que cuando el algoritmo comienza no sabe cuantas especies habrá al final. En consecuencia, la población consiste en una lista de especies cuyo tamaño varía continuamente debido a los operadores en el bucle, es decir, los procedimientos *Create_species, Selection* y *Optimize_species*. Sin embargo, es importante mencionar que existe un tamaño máximo de población (max_spec_num), definido por el usuario.

En el método *Create_species*, para cada especie en la lista, se generan un conjunto de nuevas soluciones con el fin de encontrar nuevas especies prometedoras. El parámetro new_t se refiere al número de evaluaciones de la función objetivo para la iteración actual, t. Observe que cada especie genera otras nuevas por sí misma, independientemente del resto. En contraste, el método *Selection* requiere el conocimiento de toda la lista de especies para poder seleccionar las mejores especies de la lista. TLUEGO introduce un optimizador local dentro del proceso de optimización, *Optimize_species*. Este mecanismo es aplicado a cada una de las especies de manera independiente, por tanto, el número de evaluaciones permitido para cada especies viene dado por n_t/max_spec_num. El lector interesado puede consultar [8] para una explicación más detallada del algoritmo TLUEGO.

IV. TLUEGO: ALGORITMOS PARALELOS

TLUEGO es un algoritmo basado en subpoblaciones, es decir, trabaja con una población de soluciones candidatas (especies). Esto viene a significar que las especies que conforman la lista son autosuficientes como para generar nuevas especies y para trasladarlas hacia la posición de los óptimos locales y globales de la función. De este modo hay un paralelismo intrínseco en este algoritmo que se basa en dividir la lista de especies entre los distintos procesadores.

Es importante destacar que el procedimiento de selección (*Selection*) se comporta como punto de sincronismo, el cual viene impuesto por la necesidad de mantener coherencia en los datos en la lista de especies. Esto implica que no podemos ejecutarlo hasta que los métodos que lo preceden hayan finalizado sus tareas. Por otro lado, la paralelización de los mecanismos de selección resulta, en este caso, improcedente, puesto que el coste computacional asociado a los mismos es insignificante. Por tanto, la paralelización de TLUEGO pasa por la paralelización de los métodos *Create_species* y *Optimize_species*.

A continuación se describen brevemente las tres versiones paralelas de TLUEGO desarolladas.

A. *Programación de paso de mensajes para TLUEGO*

La primera versión de TLUEGO ha sido diseñada para ser ejecutada en un multicomputador, y por tanto, el lenguaje de programación usado es MPI. Esta versión utiliza un modelo de comunicación maestro-esclavo, en el que un procesador (el maestro) se encarga de tomar las decisiones globales y de repartir la información, y el resto de procesadores (los esclavos) procesarán la información de forma concurrente. En cada nivel, el procesador maestro divide la lista de especies entre todos los procesadores (incluyéndose el mismo) y les asigna una tarea. Cada procesador realiza la tarea asignada sobre su sublista de especies y envía los resultados al procesador maestro, el cual se encarga de reunificar toda la información recibida para su posterior procesamiento en la siguiente etapa del algoritmo.

El Algoritmo 2 muestra la estructura principal del model maestro-esclavo implementado. En él, el procesador master ejecuta el algoritmo secuencial TLUEGO. El paralelismo se introduce en la evaluación de la función objetivo de las nuevas especies creadas en el procedimiento de creación, *Create_species_parallel*, y de la aplicación del optimizador local a cada una de las especies de la lista, *Optimize_species_parallel*.

En cada nivel t, el procesador maestro obtiene una nueva descendencia de soluciones candidatas para el líder. El cálculo del centro asociado al seguidor (aplicando UEGO) y la evaluación del valor objetivo del centro del líder, se llevan a cabo en paralelo. Con este fin, el procesador maestro divide la lista de soluciones candidatas entre el número de procesadores P disponibles (incluyéndose él mismo). Cada procesador recibe una sublista de especies, sobre la que aplicará UEGO para obtener la localización del seguidor asociada al líder. Es importante destacar que para ejecutar UEGO, cada procesador únicamente necesita conocer las características del centro del líder, es decir, la localización (x_1, y_1) y la calidad α_1. Por tanto, la cantidad de información involucrada en las comunicaciones es bastante pequeña.

El procesador maestro no recibirá información de los esclavos hasta que éste haya finalizado su trabajo. Cuando lo haga, pasará a un estado de recepción, donde cogerá las sublistas enviadas por los procesadores esclavos. Una vez que el maestro ha recibido toda la información, actualiza la lista de especies. Siguiendo con la estructura general de TLUEGO, la lista de especies resultante es ahora seleccionada (Línea 4 del algoritmo 2).

Para realizar el proceso de optimización, el procesador master divide la lista de especies entre todos los procesadores, de nuevo incluyéndose él mismo. En este caso, cada procesador aplica el optimizador local a cada una de las especies de la lista. Note que en este caso, las comunicaciones incluyen toda la especie, es decir, el centro de la especie y el radio asociado (nf_1, R). Cuando el procesador maes-

Algorithm 2 MPI TLUEGO algorithm

```
1: Init_population
2: for t = 1 to L do
3:    Create_species_parallel(new_t)
4:    Selection(max_spec_num)
5:    Optimize_species_parallel(n_t)
6:    Selection(max_spec_num)
7: end for
```

tro acaba su trabajo, comienza a recibir las sublistas de especies del resto de procesadores. La lista resultante será seleccionada en la siguiente etapa del algoritmo (Línea 6 del Algoritmo 2).

B. Programación de memoria compartida para TLUEGO

La segunda versión paralela de TLUEGO ha sido diseñada para ser ejecutada en un multiprocesador, y por tanto, se necesita un modelo de programación de memoria compartida. A diferencia de la versión anterior, los procesadores no necesitan mensajes para comunicarse entre sí, aunque sí un mecanismo para compartir los datos en memoria de forma coherente. Para esta implementación, se ha considerado OpenMP por ser un modelo portable y escalable, que da los programadores una interfaz simple y flexible para desarrollar aplicaciones paralelas.

Con respecto al modelo paralelo, esta versión se puede considerar un *pseudo* maestro-esclavo. OpenMP dispone de mecanismos para distribuir la lista de especies entre los diferentes procesadores sin la existencia de un procesador maestro. En este sentido, no existe un procesador maestro que controle globalmente la lista de especies, si no que se realiza de manera paralela entre todos los procesadores. Sin embargo, sigue siendo necesario que el procedimiento de selección *Selection* se realice por un único procesador. Por tanto, se han diseñado nuevos procedimientos de creación y optimización para esta versión paralela. La estructura del algoritmo es similar a la del algoritmo 2, aunque en este caso, a diferencia de la versión de paso de mensajes, no se puede considerar una técnica maestro-esclavo puramente, ya que no existe un procesador que tenga el control sobre los demás. A continuación, se describen las principales ideas de esta versión de memoria compartida.

El algoritmo paralelo desarrollado en esta subsección considera que la población se almacena en la memoria compartida. Cuando *Create_species_paral* se ejecuta, un único procesador genera nuevas soluciones candidatas. A continuación, mediante el uso de un bucle paralelo, cada procesador selecciona una nueva especie y la evalúa. Cuando un procesador ha terminado de realizar esta tarea, selecciona otra nueva especie. Este bucle se repite hasta que se evalúan todas las nuevas soluciones generadas. Tenga en cuenta que no es necesaria la exclusión mutua, ya que cada procesador accede a distintas áreas de la memoria. Además, tenga en cuenta que ningún procesador decide sobre de qué especies deben ser evaluadas por qué procesador, como ocurre en la versión anterior. Al contrario, todos los procesadores se mantienen en el mismo nivel jerárquico, gracias al uso del bucle paralelo.

El procedimiento *Optimize_species_paral* difiere del método *Create_species_paral* en la tarea que realiza cada procesador. En este caso, en lugar de evaluar las especies, se aplica el procedimiento de búsqueda local. Observe que el número de evaluaciones de la función objetivo requeridas para optimizar una especie puede ser muy diferente al requerido por otra especie, y por tanto, la carga computacional asumida por cada procesador es distinta. Mediante una planificación de tareas dínamicas de tamaño uno, se consigue equilibrar la carga computacional entre todos los procesadores, y así, reducir el tiempo de espera de los procesadores.

C. Programación híbrida para TLUEGO

La última versión paralela se ha desarrollado para ser ejecutada en un multicomputador donde cada nodo es un multiprocesador. La programación paralela combina el paso de mensajes entre todos los nodos con la paralelización de memoria compartida dentro de cada nodo. En este caso, se ha considerado MPI y OpenMP como herramientas para implementar esta versión paralela. El modelo paralelo combina un *modelo de grano grueso* con un *pseudo maestro-esclavo*.

En un modelo de grano grueso, cada elemento de proceso ejecuta un algoritmo de manera independiente al resto durante la mayor parte del tiempo [11]. La idea es que diferentes elementos de proceso trabajen con subpoblaciones más pequeñas y diferentes, de forma que, cuando se unan todas las subpoblaciones, se obtenga una población similar a la que la versión secuencial obtiene. Además, se pueden realizar migraciones de la información de un elemento de proceso a otro de acuerdo a una *política de migración*, que determina el *intervalo de migración* (la frecuencia con la que un número determinado de individuos se tiene que migrar), la *tasa de migración* (el número de individuos que se tienen que comunicar) y el *criterio de selección* (la política que se aplicará en la selección de los individuos que se comunicarán).

Para el problema considerado, cada nodo del multicomputador ejecuta TLUEGO. El tamaño de la población y el número total de evaluaciones de la función objetivo para el proceso de optimización está distribuido entre todos los nodos, N (ver algoritmo 3). En cuanto al procedimiento de migración, se supone que los nodos están conectados mediante una topología de anillo. Se consideran dos tipos de nodos (recolectores y trabajadores). La mitad de los nodos actúan como recolectores y la otra mitad como trabajadores. En cada comunicación, cada nodo se comporta o bien como un trabajador (remitente) o bien como un recolector (receptor), aunque los roles se intercambian en la siguiente comunicación. Un nodo trabajador envía su lista de población al si-

Algorithm 3 Hybrid TLUEGO algorithm at each node

```
Init_subpopulation
for t = 1 to L do
    Create_species_parallel(new_t/P)
    Selection(max_spec_num/P)
    Optimize_species_parallel(n_t)
    Selection(max_spec_num/P)
    Migration
end for
```

guiente nodo en el anillo. El nodo recolector une su propia lista con la recibida y aplica un proceso de selección, en el que selecciona las mejores especies. A continuación, la lista resultante se distribuye entre los dos nodos (recolector y trabajador). El proceso de migración ocurre en la primera mitad de los niveles del algoritmo.

Es importante destacar que con esta estrategia paralela de grano grueso, los recursos computacionales de cada nodo no están totalmente explotados, ya que se utiliza un solo procesador en el proceso de migración. Para hacer uso de todo el conjunto de procesadores y mejorar la eficiencia de esta versión paralela, se ha considerado que dentro de cada nodo se ejecuta una versión de memoria compartida en lugar de la versión secuencial de TLUEGO. De esta forma, se obtiene una versión híbrida que combina una implementación de paso de mensajes y de memoria compartida.

V. Estudios computationales

Los estudios computacionales de este trabajo se han realizado en el Supercomputador Ben Arabi del Centro de Supercomputación de Murcia, en concreto, en Arabi, que es un clúster con 816 cores, organizados en 32 nodos de 16GB de memoria cada uno, y 70 nodos con 8GB cada uno (en total 102 nodos). Cada nodo tiene 8 procesadores, divido en 2 Intel Xeon Quad Core (E5450) a 3.0 GHz. Para nuestros estudios, hemos considerado únicamente los nodos de 8GB. Los algoritmos se han implementado en C++.

Para evaluar el comportamiento de las versiones paralelas, se ha generado un conjunto de problemas variando el número n de puntos de demanda, el número m de centros pre-existentes y el número k de estos centros que pertenecen a la cadena. Las configuraciones (n, m, k) empleadas pueden observarse en la tabla I.

TABLA I
Configuraciones de los problemas test.

n	100			500		
m	2	5	10	2	15	25
k	0,1	0,1,2	0,2,4	0,1	0,5,10	0,7,15

Por cada configuración en la Tabla I, se generó un único problema escogiendo aleatoriamente los parámetros del problema uniformente en los intervalos definidos en [8]. Todos los problemas se han resuelto 10 veces y que por tanto, los resultados ofrecidos son valores medios. Es importante destacar que las tres versiones paralelas desarrolladas son fiables y robustas como la versión secuencial, ya que son capaces de encontrar prácticamente la misma solución que la versión secuencial en las 10 ejecuciones.

Para estudiar el rendimiento de los algoritmos paralelos se han considerado diferentes configuraciones del número de nodos considerados N y el número de procesadores dentro de cada nodo p.

Una de las principales metas en paralelismo consiste en incrementar el rendimiento de una aplicación con respecto a su ejecución en un uniprocesador. Una métrica muy usada para medir el rendimiento de una implementación paralela en procesadores homogéneos es la eficiencia, que estima lo bien utilizados que han estado los procesadores a la hora de resolver el problema. La eficiencia de una versión paralela (ejecutada sobre P procesadores) con respecto a la versión secuencial se computa como:

$$Eff(P) = \frac{T(1)}{P \cdot T(P)}$$

donde $T(1)$ es el tiempo empleado por TLUEGO secuencial, y el valor P viene dado por el número de nodos N multiplicado por el número p de procesadores utilizados en los nodos. $T(P)$ es el tiempo de CPU empleado por el algoritmo paralelo cuando se usan P procesadores.

La tabla II muestra los resultados medios (para todos los valores m y k) para $n = 100$. En la columna etiquetada por $Eff(E)$, se muestra el valor de la eficiencia y en la columna etiquetada por $Av(T)$, el tiempo medio empleado en las 10 ejecuciones por problema. Como puede observarse, la versión de paso de mensajes y la versión de memoria compartida obtienen una eficiencia ideal o casi ideal hasta $P = 8$ procesadores. Los valores de eficiencia decrecen conforme aumenta el número de procesador P. Esta tendencia, que también puede observarse en la versión híbrida, puede ser debida al aumento de las comunicaciones, a que no hay suficiente carga computacional como para dividirla entre muchos procesadores y a un posible desbalanceo de la carga.

TABLA II
Resultados de eficiencias para problemas con $n = 100$.

Versión paralela de TLUEGO	N	p	P	Av(T)	Av(Eff)
MPI	1	1	1	65470	-
	2	1	2	32774	1.00
	4	1	4	16507	0.99
	8	1	8	8934	0.92
	16	1	16	4861	0.84
	32	1	32	2920	0.70
	64	1	64	1800	0.57
OpenMP	1	1	1	65470	-
	1	2	2	32878	1.00
	1	4	4	16928	0.97
	1	8	8	8703	0.94
Híbrida	1	1	1	65470	-
	2	8	16	5013	0.82
	4	8	32	3047	0.67
	8	8	64	2006	0.51

TABLA III
RESULTADOS DE EFICIENCIAS PARA PROBLEMAS CON $n = 500$.

Versión paralela de TLUEGO	N	p	P	Av(T)	Av(Eff)
MPI	1	1	1	565358	-
	2	1	2	282562	1.00
	4	1	4	144663	0.98
	8	1	8	75520	0.94
	16	1	16	39194	0.90
	32	1	32	21699	0.81
	64	1	64	12290	0.72
OpenMP	1	1	1	565358	-
	1	2	2	283707	1.00
	1	4	4	143416	0.99
	1	8	8	73065	0.97
Híbrido	1	1	1	565358	-
	2	8	16	40751	0.87
	4	8	32	24953	0.71
	8	8	64	14773	0.60

Aparentemente, en términos de eficiencia, la versión híbrida de TLUEGO parece ser peor que la versión de paso de mensajes, ya que los valores de eficiencias obtenidos para $P = 16, 32, 64$ son más pequeños. Sin embargo, es importante destacar que la versión híbrida explota al máximo los recursos disponibles, mientras que la versión de MPI no (requiere el doble de nodos para obtener valores de eficiencias similares).

Finalmente, se analiza la *escalabilidad* de los algoritmos propuestos. En líneas generales, este concepto se puede definir como la capacidad de un algoritmo para gestionar una cantidad cada vez mayor de trabajo. La tabla III muestra el valor medio de eficiencia alcanzado por los tres métodos paralelos para instancias con $n = 500$ demandas puntos. Como puede verse, todos los algoritmos propuestos muestran un mayor rendimiento con el aumento del tamaño del problema, es decir, la eficiencia se mejora con respecto a los problemas anteriores (con $n = 100$), ver la tabla II. Al igual que antes se observa un decremento de la eficiencia conforme aumenta el número de procesadores, cuyo comportamiento se puede deber a lo ya comentado anteriormente.

VI. Conclusiones

Este estudio se ha basado en el problema del centroide (Stackelberg or Simpson) introducido en [8]. En dicho artículo, el algoritmo evolutivo TLUEGO probó ser la mejor técnica de entre todas las propuestas. Aun así, sólo fue capaz de resolver problemas de tamaño pequeño.

En este trabajo, se han propuesto tres implementaciones paralelas del algoritmo TLUEGO, ideadas para ser ejecutado en diferentes arquitecturas. Todas ellas presentan un buen comportamiento de rendimiento, obteniendo la eficiencia óptima o casi óptima hasta 8 procesadores. Además, su escalibilidad ha sido demostrada mediante la resolución de problemas con diferentes cargas computacionales, comprobándose que la eficiencia aumenta con el tamaño de problema.

Dependiendo de la arquitectura disponible, se puede elegir una u otra versión paralela de TLUEGO. Por ejemplo, si se dispone de un ordenador personal multicore, podremos utilizar la versión de memoria compartida de TLUEGO para resolver este problema del líder-seguidor con demanda endógena. O, si por el contrario, tenemos acceso a un cluster, podríamos utilizar la versión híbrida.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación (TIN2008-01117, ECO2011-24927), Junta de Andalucía (P10-TIC-6002), Programa CEI del MICINN (PYR-2012-15 CEI BioTIC GENIL, CEB09-0010) y Fundación Séneca (Agencia de Ciencia y Tecnología de la Región de Murcia, 00003/CS/10 y 15254/PI/10), en parte financiado por el Fondo Europeo de Desarrollo Regional (ERDF).

Referencias

[1] Z. Drezner, Ed., *Facility location: a survey of applications and methods*, Springer, Berlin, Berlin, 1995.

[2] Z. Drezner and H. W. Hamacher, Eds., *Facility location: applications and theory*, Springer, Berlin, 2002.

[3] H.A. Eiselt and G. Laporte, "Sequential location problems," *European Journal of Operational Research*, vol. 96, no. 2, pp. 217–231, 1996.

[4] M. Kilkenny and J.F. Thisse, "Economics of location: a selective survey," *Computers and Operations Research*, vol. 26, no. 14, pp. 1369–1394, 1999.

[5] F. Plastria, "Static competitive facility location: an overview of optimisation approaches," *European Journal of Operational Research*, vol. 129, no. 3, pp. 461–470, 2001.

[6] O. Berman and D. Krass, "Locating multiple competitive facilities: Spatial interaction models with variable expenditures," *Annals of Operations Research*, vol. 111, no. 1, pp. 197–225, 2002.

[7] J.L. Redondo, J. Fernández, A.G. Arrondo, I. García, and P.M. Ortigosa, "Fixed or variable demand? Does it matter when locating a facility?," *Omega*, vol. 40, no. 1, pp. 9–20, 2012.

[8] J.L. Redondo, J. Fernández, A.G. Arrondo, I. García, and P.M. Ortigosa, "A two-level evolutionary algorithm for solving the facility location and design (1|1)-centroid problem on the plane with variable demand," *Journal of Global Optimization*, To appear, (doi: 10.1007/s10898-012-9893-4).

[9] S.L. Hakimi, "On locating new facilities in a competitive environment," *European Journal of Operational Research*, vol. 12, no. 1, pp. 29–35, 1983.

[10] D.L. Huff, "Defining and estimating a trading area," *Journal of Marketing*, vol. 28, no. 3, pp. 34–38, 1964.

[11] E. Cantú-Paz, "A survey of applications and methods," Tech. Rep. IlliGAL 97003, University of Illinois at Urbana-Champaign, 1997.

Implementación de un modelo de flujo óptico robusto sobre un DSP multinúcleo

Francisco D. Igual, Guillermo Botella, Carlos García, Manuel Prieto, Francisco Tirado[1]

Resumen— **Este artículo propone una implementación eficiente de un modelo de flujo óptico robusto basado en gradiente sobre una plataforma hardware de bajo consumo basada en un procesador digital de señal (DSP) multinúcleo. El objetivo principal es evaluar la adaptación de este tipo de algoritmos a dichas plataformas, muy comunes en sistemas autónomos que aparecen en robótica, navegación, seguimiento o asistencia médica. Todos estos escenarios suelen presentar restricciones desde el punto de vista del procesamiento en tiempo real, y también en términos de energía. En nuestro trabajo, proponemos el uso de la plataforma C6678 de Texas Instruments, y realizamos una comparativa con otras arquitecturas de propósito general y de bajo consumo en términos de rendimiento y eficiencia energética.**

Palabras clave— **Estimación de movimiento, DSPs, arquitecturas de bajo consumo.**

I. Introducción

El problema de la estimación de movimiento se ha investigado en profundidad durante los últimos 50 años; sin embargo, todavía es considerado por la comunidad científica como un campo emergente y de especial interés, gracias al gran abanico de aplicaciones del mundo real que se basan en él: navegación, seguimiento, vigilancia, compresión de vídeo, robótica, automoción o neurociencia.

La estimación de movimiento determina *vectores de movimiento*, y describe la transformación de una imagen bidimensional en otra, típicamente tomando fotogramas contiguos en secuencias de vídeo. El movimiento, pues, se basa en tres dimensiones, pero las imágenes son la proyección de una escena tridimensional sobre un plano bidimensional. Por tanto, se trata de un problema mal condicionado [1], [2], [3], conocido comúnmente como "problema de la apertura". Para solucionar estos inconvenientes, resulta necesario incorporar cierto conocimiento externo acerca del comportamiento de los objetos, como restricciones de los cuerpos sólidos, u otros modelos que aproximen el movimiento de una cámara real (rotación, traslación y zoom).

El *flujo óptico* no se corresponde exactamente con el concepto de estimación de movimiento, aunque con frecuencia aparecen asociados. Se define el flujo óptico como el movimiento aparente de objetos de la imagen o píxeles entre fotogramas [1]. Realizando una clasificación del estado del arte en algoritmos y técnicas, podemos categorizar los métodos para estimar el flujo óptico en:

- *Métodos basados en correspondencias* [1], operan comparando posiciones de la estructura de la imagen entre fotogramas adyacentes e infieren la velocidad a partir del cambio en cada lugar.
- *Métodos de energía*, que usan filtros orientados espacio-temporales que responden óptimamente a determinadas velocidades.
- *Métodos diferenciales o basados en gradiente*, trabajan usando derivadas de la intensidad de la imagen en el espacio y el tiempo. La velocidad se obtiene a partir de cocientes de las medidas anteriores [3], [4]. La implementación usada en este artículo pertenece a esta familia, y se basa en el trabajo de Johnston [5], [6].

El *Modelo de Gradiente Multicanal* (McGM) se desarrolló como parte de un proyecto destinado a entender el sistema visual humano. Uno de los principales problemas del modelo McGM son los altos requisitos hardware necesarios para conseguir procesamiento en tiempo real. Trabajos previos [7], [8] han solventado estos requerimientos explotando el paralelismo de datos inherente a McGM sobre procesadores multinúcleo y aceleradores hardware como FPGAs y GPUs. Sin embargo, las limitaciones en términos de consumo de energía que aparecen en muchos escenarios conlleva el estudio de alternativas eficientes tanto desde el punto de vista de rendimiento como de consumo para la implementación de este tipo de algoritmos. En este trabajo proponemos una implementación eficiente del modelo McGM sobre el DSP C6678 de Texas Instruments (TI), que combina un pico teórico de rendimiento de 128 GFLOPS (miles de millones de operaciones en coma flotante por segundo) con un consumo estimado de 10W por chip, Además, una de sus principales ventajas es la facilidad de programación, ya que adopta modelos de programación muy extendidos tanto en modo secuencial como paralelo. Nuestra evaluación incluye además una comparación de la implementación sobre el DSP con otras arquitecturas modernas, incluyendo procesadores de propósito general y otras arquitecturas diseñadas específicamente para ser eficientes desde el punto de vista energético.

II. Modelo multicanal de gradiente (McGM)

El modelo McGM fue propuesto por Johnston *et al* [5], [6], e implementa un esquema de procesamiento visual propuesto por Hess y Snowden [9], que combina la interacción entre la visión ocular y la percepción cerebral, simplificando el modelo de visión humana [10].

La Figura 1 muestra un esquema simplificado de las etapas necesarias para el desarrollo e implementación del modelo. Desde el punto de vista de

[1]Departamento de Arquitectura de Computadores y Automática. Universidad Complutense de Madrid. {figual,gbotella,garsanca,mpmatias,ptirado}@ucm.es

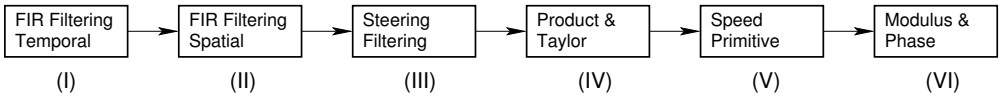


Fig. 1. Esquema del modelo McGM dividido en etapas.

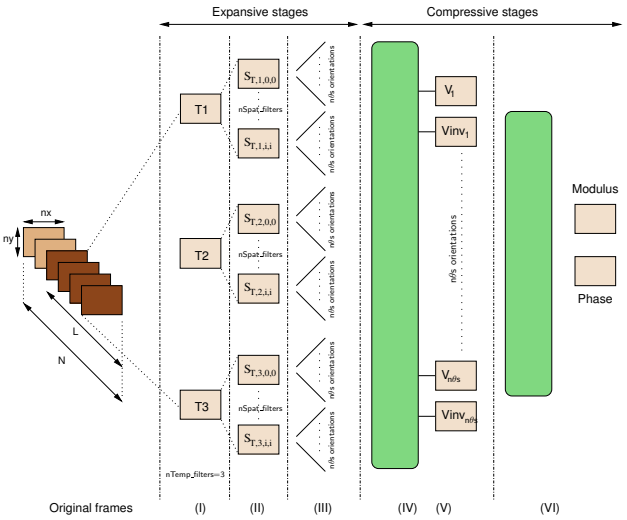


Fig. 2. Procesamiento de datos en el modelo McGM a través de las distintas etapas.

la cantidad de datos procesados, McGM puede considerarse un algoritmo expansivo, como puede apreciarse en la Figura 2. Además, la naturaleza del algoritmo hace que sea necesario concluir completamente el procesamiento de una etapa antes de comenzar la siguiente, por lo que la única forma de reducir el tiempo total de procesamiento se basa en optimizar cada una de las etapas de forma aislada. A grandes rasgos, las etapas que componen una implementación básica del modelo son [11]:

Filtrado temporal. Tomando como base el trabajo de Hess y Snowden [9], se modelan tres canales temporales diferentes (uno paso-baja con una frecuencia de corte en 10 Hz, y dos paso-banda a $5-10$ Hz y $12-18$ Hz). Para obtener la derivada temporal de la imagen, es necesario convolucionar cada secuencia con cada uno de estos filtros temporales.

Filtrado espacial. Se calculan las derivadas espaciales, de orden 0 hasta 6 en x y 0 hasta 2 en y. Para ello se usarán operadores diferenciales espaciales, donde será otra función núcleo (en este caso una Gaussiana), la que se derive para obtener estos operadores diferenciales de órdenes superiores.

Aplicación de filtros orientados (steering). En esta etapa, se proyectan los filtros espacio-temporales previamente calculados en distintas orientaciones, generando un banco de filtros a partir de combinaciones lineales de un conjunto de filtros básicos del mismo orden.

Desarrollo de Taylor. En esta etapa se lleva a cabo un desarrollo de Taylor, utilizando cada uno de los filtros orientados calculados previamente. Esta función representa una estructura robusta que recoge toda la información espacio-temporal previamente generada, y aproxima el valor de un píxel genérico a través de las derivadas de su vecindad. De acuerdo con el modelo original [5], el desarrollo se restringe a los tres primeros órdenes en la primera dimensión y al segundo orden en las dimensiones ortogonal y temporal.

Primitivas de velocidad. En las etapas anteriores se computó la información visual utilizando una representación de Taylor para cada píxel, y calculando la velocidad para un rango de orientaciones. A partir de ellas, en esta etapa se obtienen las medidas de velocidad y velocidad inversa como primitivas de flujo óptico, descompuestas en componentes paralela y ortogonal.

Cálculo final de velocidad en módulo y fase. Finalmente, a partir de las primitivas de velocidad previamente calculadas, se extrae una única medida de movimiento para cada píxel de la imagen de entrada, descompuesto en forma de *módulo* y *dirección* del movimiento.

III. Arquitectura del DSP C6678

El procesador C6678 es un DSP multinúcleo con soporte para aritmética en punto flotante desarrollado por Texas Instruments [12]. Presenta ocho núcleos con arquitectura C66x y una frecuencia de 1 Ghz. La disipación máxima de potencia es de 10W.

El núcleo C66x es la base del procesador C6678. Esta arquitectura VLIW (*Very Long Instruction Word*) explota distintos niveles de paralelismo:

Paralelismo a nivel de instrucción. Cada núcleo posee 8 unidades funcionales dedicadas a distintos tipos de operaciones, llamadas L, M, S y D. Así, esta arquitectura VLIW de 8 vías es capaz de lanzar a ejecución hasta ocho instrucciones en paralelo por ciclo.

Paralelismo a nivel de datos. El repertorio de instrucciones del C66x incluye instrucciones SIMD (*Single Instruction Multiple Data*) capaces de operar con registros de 128 bits. En general, cada núcleo es capaz de ejecutar 8 operaciones de producto-acumulación (MADD) por ciclo en simple precisión, y 2 MADD por ciclo en doble precisión. Al utilizar 8 núcleos, el C6678 arroja un rendimiento pico de 128 GFLOPS (simple precisión) y 32 GFLOPS (doble precisión).

Paralelismo a nivel de hebra. En nuestro caso, diferentes hebras se mapean sobre los distintos núcleos del procesador utilizando OpenMP.

A nivel de *software*, los DSPs de Texas Instruments ejecutan un sistema operativo de tiempo real lla-

mado SYS/BIOS. TI proporciona un compilador de C/C++ que facilita la transición de códigos ya existentes a esta arquitectura. Para mejorar la eficiencia del código generado, el compilador permite al usuario aportar información extra en forma de `#pragmas` e intrínsecas SIMD que permiten explotar de forma eficiente la arquitectura subyacente sin necesidad de utilizar código ensamblador. Además, el compilador soporta OpenMP 3.0 como modelo de programación paralela, lo que facilita el proceso de paralelización.

IV. Implementación sobre el DSP multinúcleo

En esta sección se presentan detalles de implementación y optimización para cada una de las etapas del modelo McGM. Las optimizaciones aplicadas se describen de forma incremental. A la vista de la gran fracción de tiempo dedicado a las tres primeras etapas del modelo (*filtrado temporal*, *filtrado espacial* y *steering*), nos centraremos en dichas etapas, dando únicamente algunos detalles relativos a la implementación y resultados del resto.

Las optimizaciones propuestas son específicas para este tipo de arquitecturas, y se centran en cuatro de sus características más atractivas: extracción de *paralelismo a nivel de instrucción y datos*, flexibilidad de la *jerarquía de memoria* y *uso de DMA* para solapar cálculo y movimiento de datos.

A. Parámetros relevantes para McGM

Evaluar el rendimiento de McGM es una tarea ardua debido a la gran cantidad de parámetros que pueden ser variados para afinar el comportamiento del algoritmo. Muchos de estos parámetros tienen un gran impacto no sólo en la precisión de la solución, sino también en el rendimiento. La Tabla I lista los principales parámetros asociados a las tres primeras etapas de McGM. En la columna etiquetada como *Valores Comunes* se enumeran los valores más típicos para dichos parámetros, aunque éstos pueden ser modificados en función de las necesidades. Además, la Tabla I también proporciona cuatro combinaciones de parámetros que utilizaremos en nuestro estudio experimental.

B. Optimización de McGM sobre el DSP

Los principales detalles de implementación y técnicas de optimización aplicados sobre la implementación del modelo McGM sobre el DSP multinúcleo se detallan a continuación. La mayoría de dichas técnicas de optimización son similares o comunes para todas las etapas, por lo que se detallan de forma conjunta a continuación:

Implementación básica. Estableceremos una implementación básica en lenguaje C que servirá como base para las comparaciones de rendimiento. Esta implementación incluye el uso de *flags* de optimización a nivel de compilador, y técnicas comunes de optimización para explotar la localidad de datos y la jerarquía de memoria. No incluye ninguna optimización específica para DSP.

DMA y optimización de la jerarquía de memoria. Uno de los puntos fuertes del DSP es la posibilidad de gestionar de forma explícita cada uno de los niveles de memoria de la jerarquía. Así, es posible definir *buffers*, asignarlos a distintos niveles de la jerarquía de memoria (usando `#pragma` en el código) y realizar copias de datos entre ellos. Además, estas copias pueden realizarse vía DMA, de forma que efectivamente se solapen comunicaciones y cálculo.

Optimización de bucles. Las arquitecturas VLIW requieren una optimización cuidadosa de los bucles que permita al compilador aplicar de forma efectiva técnicas como *software pipelining*, *loop unrolling* y *data prefetching* [13]. En general, el objetivo es mantener las (8) unidades funcionales de cada core totalmente ocupadas tanto tiempo como sea posible. Para conseguirlo, el programador guía al compilador acerca de factores de desenrollado seguros o específicos para cada bucle, o desambiguación de punteros vía `#pragmas` o palabras reservadas.

Vectorización. Como se mencionó en la Sección III, cada núcleo C66x es capaz de ejecutar operaciones aritméticas y de carga/almacenamiento sobre registros de hasta 128 bits. Naturalmente, esta característica está soportada a nivel de conjunto de instrucciones, y puede ser programada por medio de intrínsecas [13]. En McGM, el paralelismo de datos es masivo, y por tanto el paradigma SIMD se puede aprovechar en un gran número de escenarios. Así, por ejemplo, en las convoluciones, los datos de entrada pueden agruparse y ser procesados usando registros de 128 bits (normalmente llamados *quad registers* para las multiplicaciones, y registros de 64 bits para las sumas/restas. Así, idealmente, cada core puede ejecutar hasta 8 productos-acumulaciones por ciclo (cuatro productos por unidad M y dos por cada unidad L y S).

Paralelización a nivel de bucle. Hasta este punto, todas las optimizaciones se han enfocado hacia la optimización a nivel de un único núcleo. La última fase de optimización consiste en extraer paralelismo a nivel de thread para aprovechar los múltiples núcleos del C6678. En este caso, la paralelización se lleva a cabo a nivel de bucle utilizando OpenMP, recientemente soportado por el compilador de Texas Instruments.

C. Resultados experimentales

C.1 Etapa 1: Filtrado temporal

La Figura 3 muestra los resultados de rendimiento obtenidos tras la implementación y optimización de la primera etapa de filtrado temporal, en términos de fotogramas procesados por segundo, y considerando fotogramas cuadrados de dimensión creciente, así como tamaños de ventana de convolución (L) crecientes. No se muestran resultados para un número variable de filtros, pues 3 es la configuración más común para McGM. En este caso, comparamos el rendimiento obtenido para la implementación básica con aquél para una implementación optimizada sobre un núcleo, y dicha versión paralelizada sobre los 8 núcleos del C6678.

TABLA I
PRINCIPALES PARÁMETROS CONFIGURABLES EN MCGM.

Parámetro	Descripción	Etapa	Valores Comunes	Configuraciones			
frames	Fotogramas de entrada	Todas	-	Conf. 1	Conf. 2	Conf. 3	Conf. 4
N	Número de fotogramas	Todas	Depende de entrada	40			
$nx \times ny$	Dimensiones del fotograma	Todas	Depende de entrada	$32^2 - 128^2$			
L	Ventana (Conv. Temporal)	1	15-23	7	15	19	23
nTemp_filters	Núm. filtros temporales	1-	3	3	3	3	3
T	Ventana (Conv. Espacial)	2	15-23	7	15	19	23
nSpat_filters	Núm. filtros espaciales	2-	6	6	6	6	6
nOrtho_Orders	Núm. órdenes ortogonales	3	3	3	3	3	3
$n\theta s$	Núm. ángulos	3-	6-24	6	12	18	24

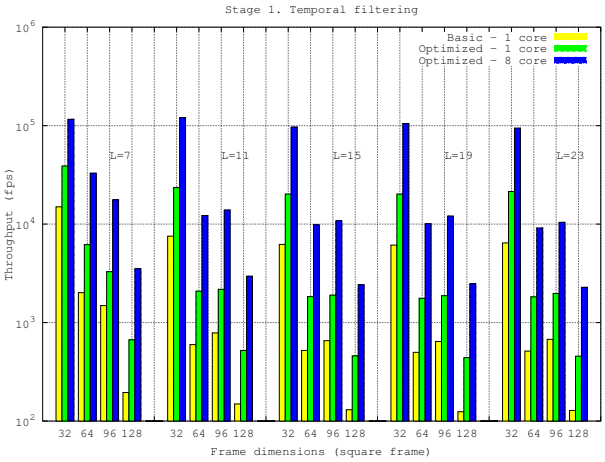


Fig. 3. Rendimiento de la implementación sobre el DSP de la etapa de filtrado temporal para distintas resoluciones y tamaños de ventana.

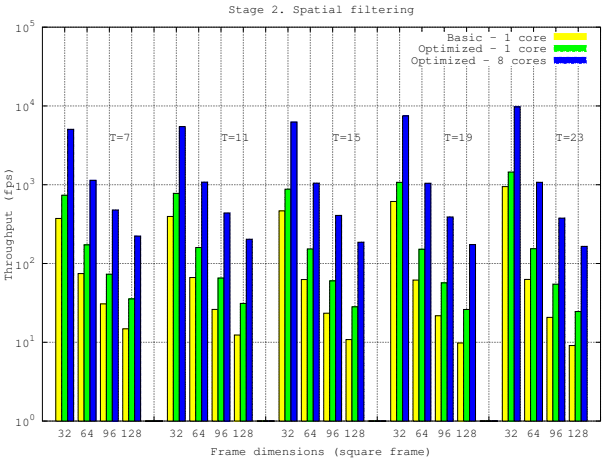


Fig. 4. Rendimiento de la implementación sobre el DSP de la etapa de filtrado espacial para distintas resoluciones y tamaños de ventana.

En esta etapa, los factores críticos con impacto sobre el rendimiento son el tamaño del fotograma ($nx \times ny$) y el tamaño de la ventana de convolución temporal (L). En general, fijando L, el rendimiento decrece a medida que aumenta el tamaño del fotograma. Para un tamaño de fotograma fijo, el impacto del incremento del tamaño de la ventana de convolución también se traduce en un decremento en el rendimiento, aunque en menor medida.

Independientemente de la resolución del fotograma y el tamaño de la ventana, las optimizaciones a nivel de núcleo (uso de DMA, optimizaciones a nivel de bucle y vectorización) arrojan unas mejoras entre 1.5× y 2×, dependiendo de los parámetro elegidos. Al utilizar OpenMP, el rendimiento mejora en un factor entre 5.5× y 7× comparado con la versión secuencial optimizada. Globalmente, el rendimiento tras aplicar el conjunto completo de optimizaciones mejora en un factor entre 7× y 14×.

C.2 Etapa 2: Filtrado espacial

La Figura 4 muestra los resultados obtenidos tras la implementación y optimización de la etapa de filtrado espacial. La figura muestra resultados para distintos tamaños de fotograma y tamaños de ventana de convolución espacial crecientes. Como en el caso anterior, realizamos una comparativa entre la implementación básica e implementaciones optimizadas secuencial y paralela.

En este caso, el tamaño del fotograma ($nx \times ny$) y la dimensión de la ventana de convolución espacial (T) tienen impacto sobre los ratios de rendimiento. Como en la etapa anterior, fijando el valor de (T), el rendimiento decrece a medida que la resolución del fotograma aumenta. En cambio, para una determinada resolución, el impacto de incrementar el tamaño de la ventana de convolución se traduce en un mayor rendimiento. De un análisis más detallado se desprende que nuestra implementación de la convolución bidimensional obtiene mejor rendimiento a medida que se incrementa el tamaño de la ventana.

Las optimizaciones a nivel de núcleo consiguen mejoras de rendimiento entre 1.6× y 2.2×, dependiendo de las dimensiones de fotograma y ventana utilizadas. La paralelización a nivel de hebra se traduce en mejoras entre 5× y 6.5× utilizando los 8 cores disponibles. En general, la mejora obtenida aplicando el conjunto completo de optimizaciones oscila entre 8× y 13×.

C.3 Etapa 3: Steering

La Figura 5 muestra los resultados de rendimiento obtenidos para la etapa de *steering* sobre el DSP. Los resultados se presentan para distintos tamaños de fotograma y número de ángulos (u orientaciones). Como en etapas previas, se compara el rendimiento de la implementación básica con el de versiones optimizadas sobre 1 y 8 núcleos.

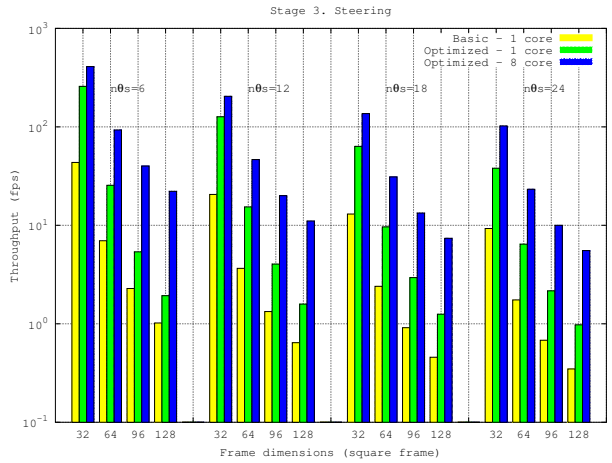


Fig. 5. Rendimiento de la implementación sobre el DSP de la etapa de *steering* para distintas resoluciones y tamaños de ventana.

En esta etapa, los factores que afectan al rendimiento son la resolución del fotograma ($nx \times ny$) y el número de orientaciones ($n\theta$s). Para un número fijo de ángulos, el rendimiento disminuye a medida que la resolución del fotograma aumenta. Para una misma resolución, el incremento en el número de ángulos proporciona mayor rendimiento.

Las optimizaciones a nivel de núcleo son más significativas en este caso, al existir mayor intensidad aritmética en el cuerpo de los bucles. Estas optimizaciones suponen mejoras de rendimiento entre 4× y 5×, dependiendo de las dimensiones de fotograma evaluadas y del número de ángulos considerados. La paralelización utilizando 8 núcleos mejora el rendimiento en factores que oscilan entre 1.5× y 2.5× comparando con la versión secuencial optimizada, con mejores resultados a medida que el número de orientaciones crece. En general, la mejora obtenida varía entre 10× and 12.5× si se aplican todas las optimizaciones propuestas de forma simultánea.

C.4 Etapas finales

Las etapas finales del modelo McGM no se consideran en profundidad desde el punto de vista del rendimiento, ya que suelen requerir una fracción muy reducida del tiempo total de procesamiento. Un análisis detallado del tiempo de ejecución dedicado a cada una de las etapas muestra como, en general, las primeras tres etapas de McGM consumen alrededor del 90 % del tiempo de ejecución total. El tiempo restante se dedica a las etapas compresivas, gestión de memoria y precálculo de filtros previo a la ejecución. Sin embargo, se han observado beneficios similares a los del resto de etapas al aplicar las mismas optimizaciones tanto a nivel de núcleo como a nivel de múltiples núcleos.

D. Consideraciones de rendimiento.

Más allá del rendimiento obtenido para cada etapa de procesamiento, resulta necesario realizar un análisis del rendimiento global obtenido. La Tabla II resume el rendimiento, en términos de fotogramas por segundo (fps) de la implementación completa del model McGM sobre el DSP multinúcleo, considerando únicamente va versión más optimizada de cada etapa. Dada la gran cantidad de parámetros configurables en el algoritmo, hemos elegido cuatro configuraciones representativas, etiquetadas como "Conf. 1" hasta "Conf. 4", cuyos parámetros se detallan en la Tabla I.

TABLA II
RENDIMIENTO DE LA IMPLEMENTACIÓN EN DSP DEL MODELO MCGM PARA DISTINTAS COMBINACIONES DE PARÁMETROS.

	Rendimiento (fps)			
	32^2	64^2	96^2	128^2
Conf. 1	266.86	60.34	25.78	14.18
Conf. 2	343.15	54.83	21.39	11.27
Conf. 3	503.27	54.99	11.77	9.89
Conf. 4	650.71	57.48	19.45	9.79

En general, el rendimiento se reduce a medida que crece la resolución de los fotogramas. Aunque este ratio es elevado para la menor resolución (hasta 650 fps para un tamaño de fotograma 32×32), decrece de forma considerable para fotogramas mayores, hasta un mínimo de 9.74 fps para la mayor resolución mostrada (128×128). Comparando estos resultados con aquellos obtenidos para el resto de etapas (Figuras 3, 4 y 5), resulta evidente que la etapa de *steering* es el factor que limita el rendimiento global de la implementación.

Para poner estos resultados en perspectiva, la Tabla III compara el rendimiento obtenido sobre un conjunto de plataformas representativas de la tecnología multi-núcleo actual. Se han seleccionado tanto procesadores de propósito general y elevado rendimiento (Intel Xeon) como dos arquitecturas de bajo consumo (ARM Cortex A9 e Intel Atom):

- TI DSP C6678 (8 núcleos, 1 GHz).
- Intel Xeon X5570 (8 núcleos, 2.93 GHz).
- Intel Atom D510 (2 núcleos, 1.66 GHz).
- ARM Cortex A9 (2 núcleos, 1 GHz).

La tabla también muestra el TDP (*Thermal Design Power*) para cada arquitectura, dando así una visión del valor máximo de consumo para cada una de ellas. Claramente, la versión multihebra sobre los 8 núcleos del procesador Intel Xeon arroja el mayor rendimiento para cualquier tamaño de fotograma. Para imágenes de entrada de 128×128 píxeles, el rendimiento ronda los 21 fps. Nuestra versión optimizada sobre el DSP C6678 obtiene mejores resultados que la versión secuencial sobre el procesador Intel Xeon (9.74 fps para el mayor tamaño de fotograma propuesto). Si se considera procesamiento en tiempo real aquél que supera los 20 fps, el procesador Intel Xeon puede alcanzar tiempo real para resoluciones de hasta 128×128, mientras que el DSP reduce esta cifra hasta imágenes de dimensión 96×96. Ninguno de los demás procesadores de bajo consumo consigue llegar a cifras que supongan procesamiento en tiempo real para imágenes mayores a 32×32 píxeles.

TABLA III

Rendimiento y eficiencia energética de las implementaciones optimizadas de McGM sobre distintas arquitecturas, usando Conf. 4 y diferentes tamaños de fotograma. Los números en negrita indican procesamiento en tiempo real. Las celdas resaltadas en gris indican la mejor eficiencia energética conseguida para cada tamaño de fotograma.

		Rendimiento (fps)				Eficiencia (kpps/Watt)			
Procesador	TDP	32^2	64^2	96^2	128^2	32^2	64^2	96^2	128^2
Xeon (8c)	190 W	**1390.09**	**136.06**	**45.04**	**20.91**	**7.49**	**2.93**	**2.18**	**1.80**
Xeon (1c)	95 W	**1065.41**	**34.55**	9.70	3.99	**11.48**	**1.49**	0.94	0.68
C6678	10 W	**650.71**	**57.48**	**19.45**	9.79	**66.63**	**23.54**	**17.93**	15.95
Atom	13 W	**182.76**	5.62	3.02	1.23	**14.39**	1.77	2.14	1.55
Cortex A9	1.2 W	**126.87**	6.44	1.79	0.75	**108.26**	21.98	13.76	10.30

E. Consideraciones energéticas.

Estas tasas de rendimiento deben ser consideradas en el contexto de la potencia real disipada por cada plataforma. Para ilustrar la eficiencia en términos energéticos de cada sistema ante una ejecución del algoritmo McGM, la Tabla III proporciona una comparativa de la eficiencia de cada arquitectura en términos de kpps (miles de píxeles procesados por segundo) por vatio. Los mejores ratios de eficiencia aparecen resaltados en gris. Nótese como, aunque la arquitectura ARM es la más eficiente para imágenes de entrada pequeñas (32×32), el DSP es claramente la arquitectura más eficiente para imágenes de entrada mayores. En este sentido, el DSP ofrece una solución de compromiso entre rendimiento y consumo, que puede ser realmente atractiva para aquellas aplicaciones y escenarios en los que el consumo es una restricción, pero el procesamiento en tiempo real es también un requerimiento para imágenes de entrada de tamaño medio/grande.

Los procesadores de propósito general proporcionan tasas de eficiencia energética menores, pero son necesarios siempre que el procesamiento en tiempo real sea deseable sobre imágenes de tamaño considerable. En comparación con las otras dos arquitecturas de bajo consumo (Intel Atom y ARM Cortex A9), el procesamiento en tiempo real sólo se consigue en estos casos para imágenes de baja resolución (hasta 32×32 en ambos casos). Así, nuestra implementación sobre DSP, así como la arquitectura en sí misma, pueden ser consideradas como atractivas no sólo cuando prima la eficiencia energética, sino también cuando el rendimiento es un requisito.

V. Conclusiones

En este artículo se ha propuesto un estudio detallado del rendimiento de una implementación optimizada del modelo de flujo óptico McGM sobre un DSP multinúcleo de bajo consumo, proponiendo optimizaciones *ad-hoc* para este tipo de procesador e incluyendo una comparativa con otro tipo de arquitecturas en términos de rendimiento y eficiencia energética.

A la vista de los resultados obtenidos, el DSP multinúcleo no puede ser considerado como una alternativa a los procesadores de propósito general en cuanto a rendimiento alcanzado, pero sí como una solución muy eficiente en términos de consumo de energía en determinadas aplicaciones y/o escenarios en los que el consumo es un factor determinante, pero es deseable un procesamiento en tiempo real de secuencias de imágenes de resoluciones de tamaño medio.

Agradecimientos

Este trabajo ha sido financiado por los proyectos CICYT-TIN 2008/508 y TIN2012-32180.

Referencias

[1] Hwang-Seok Oh and Heung-Kyu Lee, "Block-matching algorithm based on an adaptive reduction of the search area for motion estimation," *Real-Time Imaging*, vol. 6, pp. 407–414, October 2000.

[2] Chung-Lin Huang and Yng-Tsang Chen, "Motion estimation method using a 3d steerable filter," *Image Vision Comput.*, vol. 13, no. 1, pp. 21–32, 1995.

[3] Bruce D. Lucas and Takeo Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. of 7th Int. Joint Conf. on Artificial Intelligence (IJCAI '81)*, April 1981, pp. 674–679.

[4] Simon Baker, Ralph Gross, and Iain Matthews, "Lucas-kanade 20 years on: A unifying framework: Part 3," *International Journal of Computer Vision*, vol. 56, pp. 221–255, 2002.

[5] C. P. Benton P. W. McOwan and A. Johnston, "Robust velocity computation from a biologically motivated model of motion perception," *Proceedings of the Royal Society B*, vol. 266, pp. 509–518, 1999.

[6] Xuefeng Liang, Peter W. McOwan, and Alan Johnston, "Biologically inspired framework for spatial and spectral velocity estimations," *J. Opt. Soc. Am. A*, vol. 28, no. 4, pp. 713–723, Apr 2011.

[7] Guillermo Botella Juan, Antonio García Ríos, M. Rodriguez-Alvarez, Eduardo Ros Vidal, Uwe Meyer-Bäse, and María C. Molina, "Robust bioinspired architecture for optical-flow computation," *IEEE Trans. VLSI Syst.*, vol. 18, no. 4, pp. 616–629, 2010.

[8] Fermín Ayuso, Guillermo Botella, Carlos Garcia, Manuel Prieto, and Francisco Tirado, "GPU-based Acceleration of Bio-inspired Motion Estimation Model," *Concurrency and Computation: Practice and Experience*, 10/2012 In Press.

[9] R J Snowden and R F Hess, "Temporal frequency filters in the human peripheral visual field.," *Vision Res*, vol. 32, no. 1, pp. 61–72, 1992.

[10] A. Johnston and C. W. Clifford, "A unified account of three apparent motion illusions.," *Vision research*, vol. 35, no. 8, pp. 1109–1123, Apr. 1995.

[11] G Botella, *Robust optical flow implementation in reconfigurable hardware*, Ph.D. thesis, Ph. D. thesis, University of Granada, Granada, Spain, 2007, ISBN 978-84-338-4381-4, 2007.

[12] "TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor," `http://www.ti.com/lit/ds/sprs691c/sprs691c.pdf`, February 2012, Texas Instruments Literature Number: SPRS691C.

[13] "Introduction to TMS320C6000 DSP optimization," `http://www.ti.com/lit/an/sprabf2/sprabf2.pdf`, October 2011, Texas Instruments Literature Number: SPRABF2.

Estudio de diferentes esquemas de planificación para el reparto de consultas en una plataforma heterogénea.

Roberto Uribe-Paredes[1],
Diego Cazorla[2], Enrique Arias[2] y José Luis Sánchez[2]

Resumen— **La búsqueda por similitud aparece en diferentes campos de la ciencia e ingeniería tales como reconocimiento de patrones, recuperación de la información, etc convirtiéndose, hoy en día, en un campo de interés muy activo. En general, en las aplicaciones reales que tratan con un gran volumen de datos, el uso del paralelismo se vuelve esencial para obtener resultados en un tiempo razonable. En este contexto, los dispositivos GPU / MultiGPU son ampliamente utilizados para obtener este razonable tiempo a un precio bajo. En este trabajo, se precisa resolver en un tiempo razonable un conjunto de consultas a una base de datos (cientos de consultas en pocos segundos) utilizando un clúster en el que el recurso final es una plataforma heterogénea que consta de un procesador de 8 núcleos y 4 GPUs. Nuestros algoritmos explotan los recursos computacionales subyacentes al ejecutar el algoritmo de búsqueda en núcleos y en la GPU, simultáneamente.**

Palabras clave— **Búsqueda por similitud, espacios métricos, consultas por rango, plataformas basadas en GPU, plataformas multicore, esquemas de planificación.**

I. Introducción

En la última decada, la búsqueda por similitud en una gran colección de objetos almacenados en una base de datos métrica es un problema que ha adquirido especial interés. Este tipo de búsquedas aparece en diversas aplicaciones de ciencia e ingeniería tales como reconocimiento de voz e imagen, minería de datos, detección de plagio y muchas otras.

La similitud se modeliza generalmente a través de un espacio métrico, y la búsqueda de objetos más similares a través de una búsqueda por rango o de vecinos más cercanos. Un espacio métrico ($\mathbb{X}$,d) es un conjunto $\mathbb{X}$ con una función de distancia $d : \mathbb{X}^2 \rightarrow \mathbb{R}$, tal que $\forall x, y, z \in \mathbb{X}$, se deben cumplir las propiedades de: positividad ($d(x, y) \geq 0 and d(x, y) = 0 ssi x = y$), simetría ($d(x, y) = d(y, x)$) y desigualdad triangular ($d(x, y) + d(y, z) \geq d(x, z)$).

Sobre un espacio métrico $(\mathbb{X}, d)$ y un conjunto de datos finito $\mathbb{Y} \subseteq \mathbb{X}$, se pueden realizar una serie de consultas. La consulta básica es la *consulta por rango*. Sea un objeto $x \in \mathbb{X}$, y un rango $r \in \mathbb{R}$. La consulta por rango alrededor de x con rango r es el conjunto de puntos $y \in \mathbb{Y}$, tal que $d(x, y) \leq r$.

[1]Departamento de Ingeniería en Computación, Universidad de Magallanes, Chile. e-mail: roberto.uribeparedes@gmail.com
[2]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, España. e-mail: {enrique.arias,diego.cazorla,jose.sgarcia}@uclm.es

Un segundo tipo de consulta, que puede construirse usando la consulta por rango, es *los k vecinos más cercanos*. Sea un objeto $x \in \mathbb{X}$ *y un entero* k. Los k vecinos más cercanos a x son un subconjunto $\mathbb{A}$ de objetos de $\mathbb{Y}$, donde $|\mathbb{A}| = k$ y no existe un objeto $y \in \mathbb{A}$ tal que $d(y, x)$ sea menor a la distancia de algún objeto de $\mathbb{A}$ a x.

Por otro lado, la necesidad de almacenar y procesar grandes volúmenes de datos hace necesario aumentar el rendimiento en términos de tiempo de procesamiento. En general, se utilizan para ello diversas plataformas paralelas. En este sentido, nuevas plataformas basadas en tarjetas gráficas (*Graphics Processing Unit, GPU*) permiten un alto nivel de paralelismo a un muy bajo coste.

En los últimos años, grandes clusters y supercomputadores están incorporando nodos híbridos compuestos de una o más CPUs y GPUs. Muchos de estos sistemas están presentes en las primeras posiciones de las últimas listas TOP500 y GREEN500 [1], [2]. Para garantizar un uso apropiado de estas plataformas computacionales, se han de gestionar diferentes retos. Por un lado, y desde el punto de vista del hardware, es muy importante dimensionar correctamente el sistema debido a su alto coste, y por la misma razón, es esencial obtener el mayor ahorro energético posible. En esta dirección alguna propuesta, como rCUDA [3], proporciona contribuciones interesantes.

Desde el punto de vista del software, de cara a explotar toda la potencia computacional de este tipo de plataformas, las aplicaciones deben ser adecuadamente adaptadas a dichos sistemas. El escenario actual es que, en muchas aplicaciones, los datos se distribuyen entre los diferentes nodos del sistema y posteriormente son procesados en un subsistema formado por multicores/multiGPUs. Como consecuencia, las aplicaciones paralelizadas utilizando MPI han de tener en cuenta la heterogeneidad de los recursos hardware subyacentes. Además, las técnicas de planificación resultan un tema esencial debido a que la capacidad computacional de cada uno de los recursos del sistema es diferente.

Si nos centramos en el caso particular de búsquedas por similitud, se pueden encontrar bastantes contribuciones desde el punto de vista del paralelismo aplicado a este problema tanto para memoria distribuida [4], [5], [6], plataformas de

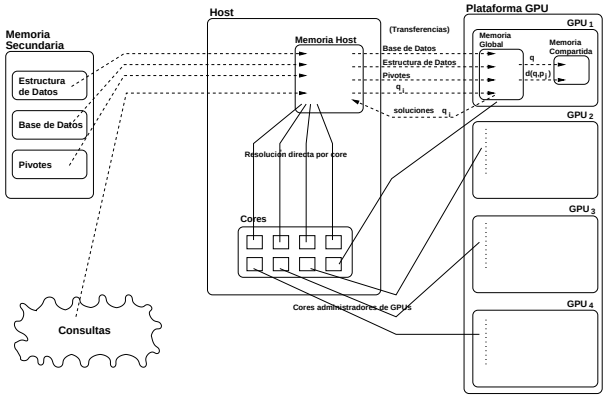


Fig. 1. Plataforma Multicore/multiGPU.

memoria compartida [7], como para GPUs [8], [9], [10], [11], [12].

Sin embargo, no hemos encontrado implementaciones paralelas heterogéneas donde tanto los cores de la CPU como de la GPU están procesando consultas simultáneamente. En este artículo nos centramos en diferentes alternativas de planificación para distribuir las consultas entrantes a todos los elementos de procesamiento en los nodos híbridos de un clúster.

Asumimos que los datos (estructuras de datos y consultas) han sido distribuidos entre los diferentes nodos de un clúster, y nos centramos en los recursos finales formados por un nodo multicore/multiGPU (ver Figura 1). Sin embargo, no es trivial la manera en que las consultas se reparten entre los diferentes cores[1] o GPUs, teniendo un gran impacto en las prestaciones. En este artículo se ha realizado un estudio del impacto que la asignación de consultas a los recursos tiene en las prestaciones, intentando proporcionar algunas guías que ayuden a desarrollar esquemas de planificación eficientes.

El artículo se estructura como sigue: la Sección II introduce brevemente la estructura "Generic Metric structure (GMS)" y la implementación secuencial [13] y la paralela basada en GPU [14] para resolver la búsqueda por similitud, implementaciones desarrolladas por los autores, y que sirve para centrar el resto del trabajo de este artículo puesto que se utilizará en la versión heterogénea que aquí se presenta. En la Sección III se introduce la implementación heterogénea, para posteriormente, en la Sección IV analizar diferentes esquemas de planificación. En la Sección V se describen los casos de estudio, la plataforma donde se llevarán a cabo los experimentos y los resultados experimentales. Adicionalmente, en la Sección VI se lleva a cabo un estudio sobre consumo energético. Finalmente, la Sección VII desgrana las conclusiones derivadas de este trabajo así como el trabajo futuro.

II. Una estructura métrica Genérica

En este trabajo proponemos el uso de una estructura métrica Genérica, que denominamos *GMS*. Si bien esta estructura no supone ninguna aportación en sí misma, sí que supone una reflexión de cara a simplificar las estructuras métricas existentes que proporcionan buenos resultados en un entorno secuencial o de paralelismo clásico, pero no son muy apropiadas para una implementación en GPUs [13]. De hecho, esta estructura métrica consiste en una matriz de datos $N \times P$, siendo N el número de objetos y P el número de pivotes.

Otras estructuras de datos, que también se basan en una estructura bidimensional, son Spaghettis [15] y SSS-Index [16]. La diferencia entre ellas viene dada por la forma de obtener los pivotes o en la manera de almacenar la estructura. Así, *GMS* evita el proceso de ordenación de datos propio de la técnica Spaghettis ya que este proceso es computacionalmente costoso en una GPU. Además, evita la selección de pivotes tal y como lo hace la estructura SSS-Index, y GMS lo hace de manera aleatoria.

Para GMS, el proceso de búsqueda, dada una consulta q y un rango r, se realiza así:

1. A partir de la distancia entre q y los pivotes $p_1, \ldots, p_k$ se obtienen k intervalos de la forma $[a_1, b_1], ..., [a_k, b_k]$, donde $a_i = d(p_i, q)$ - r y $b_i = d(p_i, q) + r$.
2. Los objetos en la intersección de todos los intervalos son candidatos a la consulta q.
3. Para cada objeto candidato y, se calcula la distancia $d(q, y)$, y si $d(q, y) \leq r$, entonces el objeto y es una solución a la consulta.

III. Implementación paralela heterogénea

En una implementación heterogénea se usan cores y GPUs para resolver todas las consultas pero cada consulta se resuelve completamente en un dispositivo (core o GPU). Para hacer esto es necesario asumir ciertas características de la implementación:

- La estructura de datos. Las estructuras de datos principales utilizadas en todas las implementaciones son la base de datos, las consultas, GMS y los pivotes.
- Procesado. Obviamente, todas las operaciones de cómputo esenciales son realizadas por la implementación heterogénea. Sin embargo, para resolver todas las consultas, es necesario considerar operaciones extra, en particular cuando utilizamos la GPU ya que las estructuras y demás datos deben ser transferidos de la memoria de la CPU a la memoria de la GPU.
- Preproceado. En esta fase se construye la estructura GMS y se carga durante la ejecución del programa. En esta fase se definen aleatoriamente los pivotes y se genera la estructura.

Con respecto al proceso de búsqueda, y tal y como hemos comentado anteriormente, cada consulta se resuelve o en el core o en la GPU.

La implementación multicore consiste en ejecutar el programa secuencial una vez que los hilos se han

[1]Se usuará el término cores para hacer referencia a los núcleos de la CPU

creado mediante los pragmas de OpenMP [17], por ejemplo, la directiva **#pragma omp parallel for**. En el caso de la implementación para la GPU, una vez que el hilo que gestiona la GPU se ha creado se realiza una llamada al código de GPU o kernel. El código de GPU se corresponde con las siguientes acciones:

- La consulta q a ser resuelta se transfiere directamente a la memoria caché de la GPU.
- Posteriormente, de cara a resolver la consulta, se lanzan a ejecución tantos hilos como número de objetos en la base de datos. Unos pocos de ellos calculan las distancias entre la consulta q y los pivotes p_i, y la distancia resultante $d(q, p_i)$ se mantiene en la memoria caché de cada multiprocesador ya que serán utilizadas por todos los hilos.
- En el siguiente paso, todos los hilos determinan si los elementos de la base da datos son o no candidatos para la consulta q. Cada hilo procesa sólo un elemento.
- Finalmente, para cada elemento candidato el hilo determina si dicho elemento es una solución válida para la consulta q.

La implementación del kernel toma en consideración buenas prácticas de programación de GPU como el uso eficiente de la jerarquía de memoria para evitar accesos innecesarios a memoria.

Nótese que, independientemente de la bondad de la implementación, el modo en que las consultas se distribuyen en la plataforma heterogénea tiene un gran impacto en las prestaciones.

En la siguiente sección, analizamos algunos esquemas de planificación posibles.

IV. Análisis de diferentes aproximaciones a la planificación de consultas en una plataforma heterogénea

Tal y como hemos comentado en la Sección III, un uso apropiado de los pragmas de OpenMP determina el modo en que las consultas se distribuyen en los recursos de procesamiento (cores o GPU).

Algunos posibles esquemas de planificación soportados por OpenMP son:

1. **Estático I:** Si se utiliza la directiva **#pragma omp parallel for** con planificación **static**, se distribuye la misma cantidad de consultas entre todos los recursos. Claramente, esta planificación no es apropiada para una plataforma heterogénea debido al hecho de que las GPUs son dispositivos más rápidos que los cores. Esto significa que los cores podrían representar un cuello de botella de cara a obtener las mejores prestaciones.
2. **Dinámico I:** Si se utiliza la directiva **#pragma omp parallel for** con una política de planificación **dynamic**, ésta corresponde a una distribución a bloques de consultas a un recurso, y una vez un recurso finaliza cogería el siguiente bloque asignable. En este caso, la carga se balancea, pero las prestaciones dependen del tamaño del bloque. Sin embargo, un análisis en profundidad de como OpenMP funciona internamente, nos muestra que OpenMP asigna un bloque de consultas pero de una en una. Por tanto, parece que el tamaño del bloque no afecta a las prestaciones, y sin embargo el tiempo perdido en las transferencias puede ser muy alto porque hay tantas transferencias a GPU como consultas. Por otro lado, debemos tener en cuenta que el último bloque puede ser asignado a un core y de nuevo este elemento de proceso representa el cuello de botella del sistema completo.
3. **Dinámico II:** Se utiliza la directiva **#pragma omp parallel for** con política de planificación **dynamic**, pero en este caso el programador engaña a OpenMP forzando a calcular y transmitir un bloque de consultas completamente. En este caso el tiempo de transferencia se reduce, pero no se logra evitar el efecto del último bloque asignado.
4. **Estático II:** En lugar de utilizar la directiva de OpenMP **#pragma omp parallel for** que crea los hilos y distribuye las consultas, utilizamos la directiva **#pragma omp parallel**, esto es, sólo creamos los hilos y de manera estática se asigna un porcentaje de consultas a las GPUs y otro porcentaje a los cores. En este caso, sólo se realiza una única transferenciua de datos beneficiando las prestaciones. Además, se evita el efecto perverso del último bloque asignado, ya que el número de consultas asignadas a la GPU es mayor que el número de consultas asignadas a los cores. Una discusión sobre el porcentaje de consultas asignadas a GPU y asignadas a cores se incluye en la Sección V.

En este artículo, vamos a considerar el último esquema de planificación conforme al análisis anterior.

V. Banco de pruebas

En esta sección se describen los casos de estudio y la plataforma utilizada para la obtención de los resultados experimentales, y el posterior análisis.

A. Casos de estudio y plataforma experimental

Como casos de estudio, se han considerado dos bases de datos. La primera es un subconjunto del diccionario de Español (86,061 palabras) utilizando la distancia de edición para cada consulta y rangos de búsqueda entre 1 y 4. Dadas dos palabras, la distancia de edición se define como el mínimo número de inserciones, borrados o sustituciones de caracteres necesarios para que una palabra sea igual a otra. La segunda es un conjunto de $112,682$ histogramas de color (112 vectores) a partir de una base de datos de imágenes. En este caso se utiliza una función de distancia cuadrática, como la distancia Euclidea. El radio utilizado es el que permite recuperar 0.01%, 0.1% y 1% de la base de datos.

Para estas bases de datos creamos las estructuras métricas con el 90% del conjunto de datos elegidos aleatoriamente, y reservamos el 10% restante para las consultas. Seleccionamos 32 pivotes para construir GMS ya que esta cantidad de pivotes proporciona buenos resultados [13]. Los pivotes se seleccionan aleatoriamente. El tiempo invertido en la construcción de la estructura no se ha tenido en cuenta, considerándose como un tiempo de preproceso. Este reducido número de pivotes se ha escogido teniendo en cuenta que la GPU tiene mayores restricciones de memoria. Se considera también que la base de datos se almacena completamente en la GPU, siendo el problema de grandes bases de datos un tema fuera del ámbito de este trabajo, considerado como trabajo futuro.

Para todos los experimentos, el tiempo de ejecución considera sólo el proceso de búsqueda, pero se tiene en cuenta el tiempo empleado en todas las transferencias de datos necesarias, incluidas a GPU. Como se decía anteriormente, la construcción de las estructuras necesarias, los pivotes, etc, se calculan "off-line", y de esta manera todos los métodos se compararán en las mismas condiciones.

La plataforma hardware utlizada consiste en un clúster donde los recursos finales son 2 Quadcore Xeon E5530 a 2.4GHz y 48GB de memoria principal y con 240-core 4 Nvidia Tesla C1060 a 1.3GHz y 4 GB de memoria global, utilizando CUDA SDK v3.2 (http://developer.nvidia.com/). La compilación ha sido realizada utilizando el compilador gcc 4.3.4 y la librería OpenMP. Este trabajo sólo se centra en los recursos finales.

B. Resultados y discusión

Tal y como se comentó al final de la Sección IV, hemos considerado un esquema de planificación estático en el sentido que se asigna un único bloque de consultas a los diferentes dispositivos. Para obtener algunas conclusiones, el porcentaje de consultas asignadas a la GPU varía desde 0% (sólo los cores resuelven las consultas) hasta 100% (sólo las GPUs resuelve las consultas). Las Figuras 2 y 3 muestran los resultados obtenidos para el caso de estudio de diccionario de Español e histogramas de color, respectivamente. El eje X representa el porcentaje de consultas asignadas a las GPUs, mientras que el eje Y representa el tiempo de ejecución en segundos. En el caso de las GPUs se incluye el tiempo de transferencia de datos.

Es importante resaltar que:

- Cuando el número de consultas asignadas a las GPUs es menor que 50%, es más eficiente usar sólo cores. Este resultado significa que el número de consulas asignadas a las GPUs es reducido y por tanto la implementación no saca partido de la potencia computacional de dicho dispositivo y los cores representan el cuello de botella del sistema heterogéneo. Sin embargo, el tiempo de ejecución está lejos de ser el mejor.
- El mejor resultado, en términos de tiempo de ejecución, se obtiene cuando el número de consultas asignadas a las GPUs es mayor del 80%. En este caso, el tiempo de procesamiento empleado por las GPUs y los cores es muy similar, esto es, las GPUs resuelven más consultas que los cores pero empleando aproximadamente el mismo tiempo. En las figuras se puede observar un efecto valle justo en el punto donde se obtienen las mejores prestaciones.
- En los casos de estudio considerados en este trabajo conforme la complejidad del problema crece (por ejemplo, rango 4 para el diccionario de Español o 1% para la recuperación de información en el caso de histograma de color) el mejor resultado se obtiene sólo usando GPUs. Esto es debido a que la contribución en el tiempo de ejecución de la utilización de los cores es muy reducida. Sin embargo, si el número de consultas fuese más elevado que el considerado en este trabajo, de nuevo aparecería el efecto valle en las Figuras 2 y 3.

Hasta el momento, los resultados son coherentes

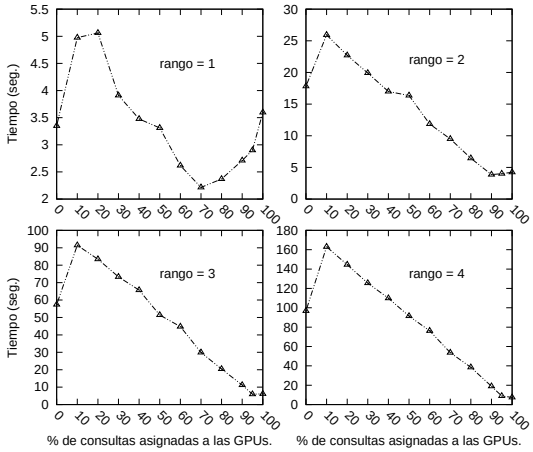


Fig. 2. Tiempo de ejecución para el caso de diccionario de Español para porcentajes de consultas asignadas a la GPU desde 0% a 100%.

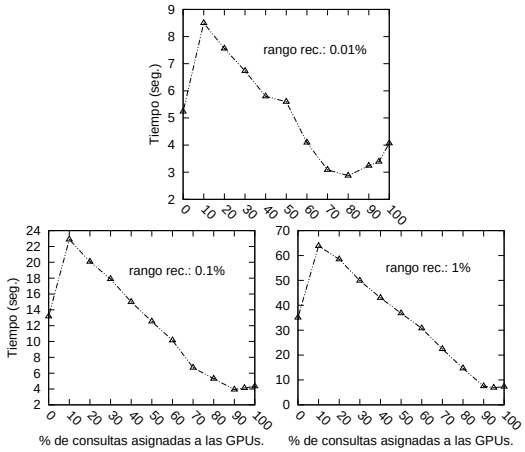
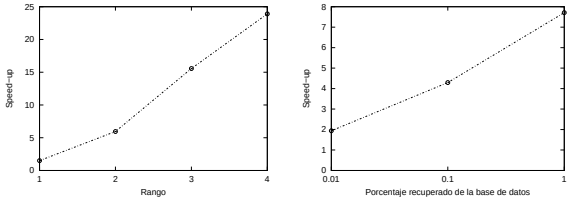
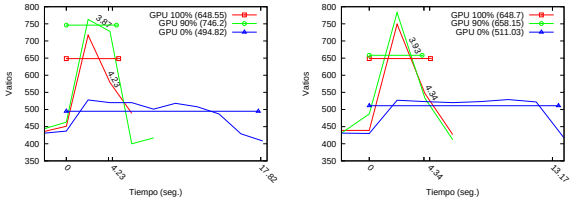


Fig. 3. Tiempo de ejecución para el caso de histogramas de color para porcentajes de consultas asignadas a la GPU desde 0% a 100%.



(a) Diccionario Español. (b) Histogramas de color.

Fig. 4. Speedups para el sistema heterogéneo.



(a) Diccionario Español. (b) Histogramas de color.

Fig. 5. Estudio de potencia consumida en vatios para el caso de diccionario de Español e histogramas de color, considerando 8 cores (GPU 0%), 4 GPUs (GPU 100%) y la plataforma heterogénea (4 cores + 4 GPUs).

pero no hemos determinado el tamaño apropiado de bloque. Para realizar esto, y debido al hecho de que debemos decidir la carga a repartir entre los elementos computacionales muy diferentes, vamos a emplear la relación existente entre resolver el problema de búsqueda por similitud en un core (ejecución secuencial) y en una GPU. De hecho, esta relación viene dada por la ganancia de velocidad o "speedup". La Figura 4 muestra el "speedup" entre la implementación secuencial y la implementación en GPU para el caso de diccionario Español y para el caso de histogramas de color.

Así pues, para el caso de diccionario de Español y rango 1, la ganancia de velocidad es cercana a 2, que significa que debemos asignar al menos el doble de consultas a las GPUS que a los cores. De hecho, el mejor porcentaje de asignación de consultas para el rango 1 es alrededor de 70% para GPUs. Cuando el rango es 2, la ganancia de velocidad está cercana a 6. Por tanto debemos asignar 6 veces más consultas a las GPUs que a los cores. Para este caso, las mejores prestaciones se obtienen para porcentajes de consulas asignadas a GPUs cercanos a 84%. De hecho, los mejores resultados se han ontenido para porcentajes de consultas asignadas alrededor de 90%. Un razonamiento similar se pude hacer para en caso de histogramas de color.

VI. Consumo de energía

Hoy en día hay un gran interés en realizar implementaciones eficientes tanto computacionalmente (menos tiempo de ejecución) como de consumo de energía. De hecho, hay una tendencia global hacia la computación a Exaescala, un nivel de cómputo que será viable gracias al paralelismo y al escalado de la tecnología, pero siempre y cuando se reduzcan los niveles de potencia eléctrica consumida [18].

En este trabajo se ha realizado un estudio para analizar la energía consumida por las implementaciones propuestas, como medida adicional a las medidas de tiempo de ejecución y ganancia de velocidad presentadas previamente. La potencia consumida se mide en términos de vatios (watts) consumidos por la plataforma completa, y en Julios cuando se considera el consumo de energía. Las medidas se ha realizado utilizando el analizador de potencia PZ4000 de YOKOGAWA [19].

Las pruebas se han llevado a cabo considerando las siguientes características:

1. Sería deseable conocer la potencia consumida solo por los cores. En este caso, podemos estudiar el impacto de introducir las GPUs.
2. Además, sería deseable conocer si los cores son o no necesarios, teniendo sólo GPUs.
3. Finalmente, comparar los resultados previos con la plataforma heterogénea. Al realizar este estudio, el número de pruebas se acota ya que hay que buscar unos rangos de valores en las pruebas donde se produzca la anterior casuística. Por tanto, para el caso de diccionario de Español se ha considerado rango 2 que proporciona un mejor resultado cuando las GPUs procesan el 70% de las consultas. Por el mismo razonamiento, para el caso de histogramas de color se utilizará 0.1% de información recuperada.

La Figura 5 muestra la potencia consumida en vatios para los dos casos de estudio, considerando 8 cores (GPU 0%), 4 GPUs (GPU 100%) y la plataforma heterogénea (4 cores + 4 GPUs). En dicha figura se puede observar que introducir una GPU implica un crecimiento en la potencia consumida, pero por otro lado, hay una ligera disminución del tiempo de ejecución.

Las Tablas I y II muestran la potencia consumida (vatios) frente a tiempo de ejecución (segundos), esto es, se presentan los Julios para los dos casos de estudio considerados.

TABLA I
Consumo de energía para el caso de estudio de diccionario de Español

Plataforma	Tiempo	Vatios	Julios
8 cores	17, 82	494, 82	8817, 69
4 GPUs	4, 23	648, 55	2743.37
Heterogénea	3, 87	746, 20	2887, 79

TABLA II
Consumo de energía para el caso de estudio de histograms de color

Plataforma	Tiempo	Vatios	Julios
8 cores	13, 17	511, 03	6730, 26
4 GPUs	4, 34	648, 70	2815, 36
Heterogénea	3, 93	658, 15	2586, 53

En estas tablas podemos observar que los vatios consumidos por las GPUs o la plataforma heterogénea son muy similares, sin embargo necesitan entre 3 y 7 veces menos energía que con respecto al caso de sólo cores. Evidentemente, hay un gran impacto en la potencia consumida cuando se introduce las GPUs, pero también hay que considerar que el problema se resuelve en menos tiempo y que por tanto la energía global consumida se reduce considerablemente.

VII. Conclusiones y trabajo futuro

En este trabajo se presenta y discute sobre un estudio del impacto del esquema de planificación en la distribución de consultas que llegan a un servidor hacia los recursos finales de una plataforma heterogénea.

Conforme a esta discusión, un esquema de planificación estática orientada a bloques ha sido el escogido ya que porporciona los mejores resultados al obtener beneficios de todos los recursos computacionales subyacentes (cores+GPUs). Como conclusión, las GPUs han de realizar más trabajo (más consultas asignadas) y los cores contribuyen a reducir el tiempo de ejecución total (con un pequeño porcentaje de las consultas a ser procesadas) bajo la condición de que el tiempo empleado por las GPUs sea lo más similar posible al empleado por los cores. Para encontrar esta relación se ha realizado un estudio basado en la ganancia de velocidad de ejecutar en un core frente a ejecutar en una GPU.

Desde el punto de vista energético, a pesar de que las GPUs incrementan puntualmente la potencia necesaria, la energía consumida necesaria se reduce drásticamente con respecto al caso de utilizar sólo cores, debido a que resuelve el problema en menos tiempo.

Como trabajo futuro se propone corroborar la discusión sobre los esquemas de planificación implementando las otras alternativas. Adicionalmente, de cara a obtener un mejor rendimiento de la plataforma heterogénea, se incrementará el número de consultas a ser procesadas.

Finalmente, queremos introducir la implementación heterogénea en los recursos finales de un clúster, el cual será alimentado por un servidor de peticiones, tal y como ocurre en la realidad.

Reconocimientos

Este trabajo ha sido parcialmente soportado por el proyecto del Ministerio de Ciencia e Innovación SATSIM (Ref: CGL2010-20787-C02-02), España.

Referencias

[1] URL, "Top 500 list," http://www.top500.org/.

[2] URL, "Green 500 list," http://www.green500.org/.

[3] Federico Silla Rafael Mayo José Duato, Antonio J. Peña and Enrique S. Quintana-Ortí., "rCUDA: reducing the number of gpu-based accelerators in high performance clusters.," in *In Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS 2010)*, Caen, France, June 2010.

[4] Pavel Zezula, Pasquale Savino, Fausto Rabitti, Giuseppe Amato, and Paolo Ciaccia, "Processing m-trees with parallel resources," in *RIDE '98: Proceedings of the Workshop on Research Issues in Database Engineering*, Washington, DC, USA, 1998, pp. 147–, IEEE CS.

[5] Adil Alpkocak, Taner Danisman, and Ulker Tuba, "A parallel similarity search in high dimensional metric space using m-tree," in *Advanced Environments, Tools, and Applications for Cluster Computing*, vol. 2326 of *LNCS*, pp. 247–252. Springer Berlin / Heidelberg, 2002.

[6] Veronica Gil-Costa, Mauricio Marín, and Nora Reyes, "Parallel query processing on distributed clustering indexes," *Journal of Discrete Algorithms*, vol. 7, no. 1, pp. 3–17, 2009.

[7] Veronica Gil-Costa, Ricardo Barrientos, Mauricio Marín, and Carolina Bonacic, "Scheduling metric-space queries processing on multi-core processors," *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, vol. 0, pp. 187–194, 2010.

[8] Quansheng Kuang and Lei Zhao, "A practical GPU based kNN algorithm," *International Symposium on Computer Science and Computational Technology (ISCSCT)*, pp. 151–155, 2009.

[9] Vincent Garcia, Eric Debreuve, and Michel Barlaud, "Fast k nearest neighbor search using GPU," *Computer Vision and Pattern Recognition Workshop*, vol. 0, pp. 1–6, 2008.

[10] Ricardo J. Barrientos, José I. Gómez, Christian Tenllado, Manuel Prieto, and Mauricio Marín, "kNN query processing in metric spaces using GPUs," in *17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011)*, Berlin, Heidelberg, 2011, vol. 6852 of *LNCS*, pp. 380–392, Springer-Verlag.

[11] Ricardo J. Barrientos, José I. Gómez, Christian Tenllado, Manuel Prieto, and Mauricio Marín, "Range query processing in a multi-GPU environment," in *10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2012)*, Madrid, Spain, July 2012, pp. 419–426.

[12] Roberto Uribe-Paredes, Pedro Valero-Lara, Enrique Arias, José Luis Sanchez, and Diego Cazorla, "Similarity search implementations for multi-core and many-core processors," in *International Conference on High Performance Computing and Simulation (HPCS)*, 2011, pp. 656–663.

[13] Roberto Uribe-Paredes, Diego Cazorla, José L. Sánchez, and Enrique Arias, "A comparative study of different metric structures: Thinking on GPU implementations.," in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2012 (WCE 2012)*, London, England, July 2012, pp. 312–317.

[14] Roberto Uribe-Paredes, Enrique Arias, José L. Sánchez, Diego Cazorla, and Pedro Valero-Lara, "Improving the performance for the range search on metric spaces using a multi-GPU platform," in *Database and Expert Systems Applications (DEXA)*, vol. 7447 of *Lecture Notes in Computer Science*, pp. 442–449. Springer Berlin Heidelberg, 2012.

[15] Edgar Chávez, José L. Marroquín, and Ricardo Baeza-Yates, "Spaghettis: An array based algorithm for similarity queries in metric spaces," in *6th International Symposium on String Processing and Information Retrieval (SPIRE'99)*. IEEE CS Press, 1999, pp. 38–46.

[16] Oscar Pedreira and Nieves R. Brisaboa, "Spatial selection of sparse pivots for similarity search in metric spaces," in *33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007)*, Harrachov, Czech Republic, 2007, vol. 4362 of *LNCS*, pp. 434–445, Springer.

[17] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta, *Introduction to Parallel Computing (2nd Edition)*, Addison Wesley, 2 edition, 2003.

[18] Dylan McGrath, "Power consumption hurdles litter path to exascale computing.," In *EETimes: News and Analysis* , 7/11/2012.

[19] "Yokogama pz4000 power analyzer," http://tmi.yokogawa.com/es/.

Transcodificación de vídeo H.264/AVC a HEVC con varios frames de referencia

Antonio Jesus Diaz-Honrubia[1], Jose Luis Martinez[1], Pedro Cuenca[1]

***Resumen*— El nuevo estándar de vídeo High Efficiency Video Coding, ha sido desarrollado recientemente por la ITU-T y la ISO/IEC de forma conjunta para reemplazar a H.264/AVC. Este estándar ha tenido mucho éxito y ha sido ampliamente aceptado durante los últimos años. De esta forma, se hace patente una necesidad en los próximos años de contar con eficientes mecanismos para la conversión entre los estándares H.264/AVC y HEVC. En este artículo, se presenta un mecanismo para acelerar el proceso de la Estimación de Movimiento en el proceso de transcodificación desde H.264/AVC a HEVC. Dado que ambos estándares comparten una estructura de codificación similar, se trata de explotar la información que se recoge de la etapa de decodificación de H.264/AVC. El algoritmo que se propone trata de reducir la complejidad del transcodificador reduciendo, para ello, la complejidad de la Estimación de Movimiento de HEVC. Los resultados experimentales muestran que el algoritmo propuesto puede obtener una buena compensación entre eficiencia de codificación y complejidad.**

***Palabras clave*— H.264/AVC, HEVC, Transcodificación, Frames de referencia**

I. Introducción

LOS últimos avances en la electrónica de los dispositivos para el consumo han implicado un rápido crecimiento de los contenidos multimedia creados por los usuarios, muchos de ellos con altas resoluciones, que pueden llegar hasta a vídeo con una definición de 4K. Todos estos modernos usuarios desean, a su vez, compartir estos contenidos con sus amigos, familiares y/o personas con gustos similares, tendencia que se ha visto impulsada por el uso de las redes sociales. La gestión de estos contenidos, junto con los contenidos comerciales, y todos los dispositivos, protocolos y servicios relacionados (para la grabación, almacenamiento, intercambio y renderizado) son una moda sin ataduras. Además, la forma de adecuar un vídeo para distintos dispositivos, capacidades y anchos de banda variables es otro reto muy importante.

La transcodificación es una de las técnicas más prometedoras, ya que proporciona una adaptación del vídeo en términos de reducción de bitrate, de reducción de la resolución o de conversión de formato para salvaguardar distintos requisitos. Sin embargo, los últimos desarrollos en la tecnología de codificación de vídeo, hacen que la transcodificación sea mucho más complicada. Un nuevo estándar de codificación crea nuevos requisitos para la transcodificación de los formatos ya existentes al nuevo formato para la interoperabilidad de los contenidos visuales.

Actualmente, las comunicaciones en vídeo representan aproximadamente la mitad del tráfico de las redes, con una tendencia al alza. De esta forma, las técnicas que tratan de comprimir el vídeo de forma eficiente son de gran importancia; un ejemplo podría ser cómo evitar la "contracción del espectro" prevista debido a la creciente cantidad de tráfico en redes móviles. Una cantidad sustancial de esfuerzo se ha realizado en esta área durante los últimos años, lo que ha resultado en un estándar de compresión de vídeo de nueva generación, llamado High Efficiency Video Coding (HEVC) [1][2]. Para calidades similares, la arquitectura actual de HEVC sólo consume en su transmisión la mitad de ancho de banda que el estándar anterior H.264/AVC. La capacidad de compresión de HEVC establece un nuevo punto de referencia en la compresión de vídeo. Así, se espera que HEVC sea el sucesor de H.264/AVC. El amplio uso hoy en día de este estándar y la esperada adopción de HEVC hace que aparezca una nueva demanda de un transcodificador de H.264/AVC a HEVC. En la práctica, un transcodificador de vídeo debería mantener una buena compensación entre la complejidad y la eficiencia de la codificación, a la vez que hace uso de la información contenida en el flujo de entrada para generar el de salida.

En este artículo se presenta un novedoso algoritmo de selección de los frames de referencia que serán usados como parte de un transcodificador heterogéneo de baja complejidad entre H.264/AVC y HEVC. El mecanismo de transcodificación propuesto se basa en la reutilización de los frames de referencia recolectados en la etapa de decodificación de H.264/AVC. Además se hace necesario un algoritmo de mapeo entre las particiones de HEVC y el área apropiada de H.264/AVC. Los resultados experimentales muestran que el mecanismo de transcodificación propuesto puede conseguir una buena compensación entre la eficiencia de la codificación y la complejidad del transcodificador.

El resto de este artículo está organizado de esta forma: la Sección II incluye una descripción técnica del nuevo estándar HEVC. La Sección III identifica los trabajos relacionados que se están llevando a cabo en el área. La Sección IV introduce la propuesta realizada en este artículo para la arquitectura del transcodificador. Los resultados experimentales obtenidos se muestran en la Sección V. Finalmente, la sección VI contiene las conclusiones y el trabajo futuro que pueden deducirse de este trabajo.

[1] Instituto de Investigación en Informática de Albacete (I3A). Universidad de Castilla-La Mancha (UCLM). Campus Universitario de Albacete. 02071. Albacete (SPAIN).

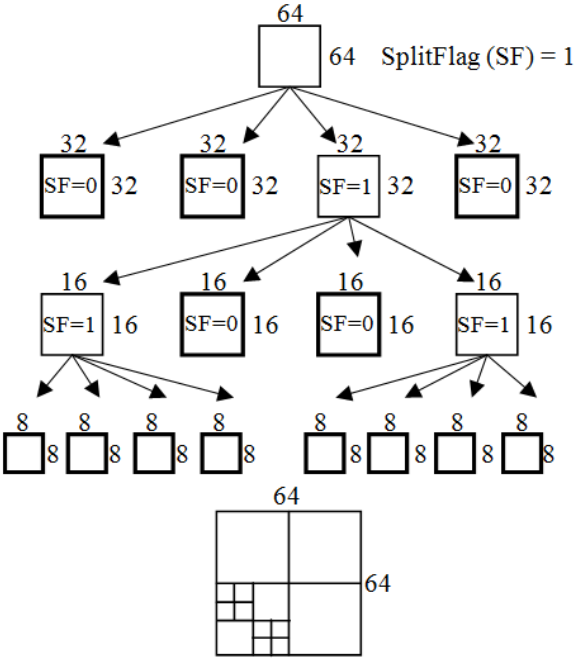


Fig. 1. Particionamiento de las CU en HEVC.

II. Descripción técnica de HEVC

HEVC introduce nuevas herramientas de codificación, y mejora otras ya existentes, con respecto a su predecesor H.264/AVC [1][2]; todo ello hace posible que la eficiencia de codificación se incremente notablemente. Uno de los cambios más notables reside en la forma de particionar las imágenes. HEVC prescinde de los términos de Macrobloque (MB) y bloque en la Estimación de Movimiento (ME) y en la transformada respectivamente e introduce tres nuevos conceptos: Unidad de Codificación (CU), Unidad de Predicción (PU) y Unidad de Transformada (TU). Esta estructura conduce a una codificación más eficiente para ajustarse a las necesidades específicas de cada imagen. Cada frame es originalmente dividido en regiones cuadradas de tamaño variable llamadas CUs, que reemplazan a la estructura de MBs de estándares anteriores. Cada CU puede contener uno o más PUs y TUs y su tamaño puede oscilar entre los 8x8 y los 64x64 píxeles. Para ajustar el tamaño de cada CU, lo primero que el estándar hace es dividir la imagen en áreas de 64x64 píxeles, cada una de ellas llamadas Largest CU (LCU) y, entonces, cada LCU puede ser particionada en cuatro sub-áreas más pequeñas de una cuarta parte del área original. Este particionamiento puede realizarse con cada sub-área de forma recursiva hasta que se alcanza un tamaño mínimo de 8x8 píxeles. De esta forma se usa una estructura de árbol cuaternario, o quad-tree, tal y como puede verse en la figura 1.

HEVC comprueba casi todos estos modos (Inter e Intra) para decidir si particiona o no una CU/PU/TU, escogiendo para ello el caso que mejor relación Rate-Distortion (RD) obtenga. Además, en el caso de la Inter Predicción, se utiliza un algoritmo de ME para cada una de estas particiones; por defecto, el software HM [3] define la Búsqueda Diamond [4]. Este amplio abanico de posibilidades hace de HEVC un estándar mucho más complejo que su predecesor H.264/AVC. Tal y como se puede observar en la tabla 1, el módulo más costoso computacionalmente en el nuevo codificador de HEVC es que se encarga de realizar la Inter Predicción, que ocupa alrededor del 77-81% del tiempo total de codificación [5]. Por esta razón, este es el módulo en el que se centrarán los mayores esfuerzos de aceleración.

Tabla 1. Tiempo consumido por los módulos de HEVC [5].

Módulo	Tiempo consumido (%)
Predicción Intra	1-2%
Predicción Inter	77-81%
Transf. + Cuant.	14-16%
Filtro de Bucle	0,1-0,2%
Codificador Entropía	2-4%

III. Trabajos relacionados

La transcodificación de vídeo es el proceso de convertir un flujo de vídeo comprimido previamente codificado con un determinado formato o características en otro flujo de vídeo codificado con otro códec o características. En el caso de que la secuencia de vídeo sea traducida a otro formato del usado para la codificación del vídeo de entrada, este proceso se conoce como transcodificación heterogénea. El proceso de transcodificación debería llevar a cabo la conversión sin necesidad de completar el proceso de decodificación y recodificación [6]. El proceso de transcodificación ha sido una de las líneas de investigación más candentes durante los últimos años en el marco de los transcodificadores de MPEG-2 a H.264/AVC [7] o de H.263 a H.264/AVC [8], así como entre extensiones de H.264/AVC, como de éste a su versión escalable SVC [9]. También se ha prestado atención a los transcodificadores entre la Codificación Distribuida de Vídeo (DVC) y H.264/AVC [10].

En la actualidad, existen muy pocos casos de transcodificación entre H.264/AVC y el nuevo estándar HEVC [11][12][13]. En [11], los autores proponen la reutilización de los Vectores de Movimiento (MVs), así como una métrica de similaridad para decidir si ciertas particiones de CU en HEVC deben ser usadas. De modo similar, la propuesta presentada en [12], está enfocada a reducir la cantidad de particiones de CUs y de PUs que deben ser comprobadas usando para ello una métrica mejorada de la Optimización RD (RDO).

En [13] puede encontrarse una propuesta más completa, que combina el uso de la paralelización del codificador, aprovechando las estructuras creadas para ello en HEVC, junto con la reutilización de la información extraída en la etapa de decodificación de H.264/AVC. En este caso, usa la resolución del frame para restringir el particionamiento del quadtree y reutiliza los modos de particionamiento para seleccionar los modos de PUs disponibles en cada caso. Además, también reutiliza los MVs codificados para aproximar el nuevo MV de HEVC.

Sin embargo, hasta el momento no se ha encontrado ningún trabajo previo que se base en la reutilización de los frames de referencia para la aceleración de un transcodificador.

IV. Arquitectura propuesta de transcodificador

La arquitectura propuesta para el transcodificador se basa en la reutilización en el codificador de HEVC de los frames de referencia que ya han sido elegidos por H.264/AVC para reducir el tiempo de codificación del transcodificador cuando codifica frames de tipo P.

La motivación de reutilizar estos frames de referencia es que la búsqueda en cada uno de ellos es muy costosa computacionalmente, dado que cada frames tiene muchas LCUs y cada una de estas LCUs es recursivamente particionada en muchas CUs y TUs. Así, cada LCU es dividida en hasta $4^0 + 4^1 + 4^2 + 4^3 = 85$ CUs y, cada una de éstas, es dividida en hasta 17 TUs diferentes, por lo que la ME se realizaría hasta 1.445 veces por cada frame de referencia en cada LCU. El algoritmo propuesto trata de reducir el número de frames de referencia en los que HEVC realiza una búsqueda durante la ME, utilizando tan sólo los finalmente usados por H.264/AVC en el área cubierta en dicho estándar por la partición actual de HEVC.

La solución dada pasa por dividir cada imagen de H.264/AVC en bloques de 4x4 píxeles, ya que éste es el menor tamaño que pueden tener los subMBs en este estándar, y asignarles el frame de referencia usado en cada uno de estos bloques. Entonces, el área correspondiente de HEVC es delimitada con la cantidad correspondiente de estos bloques y cada frame de referencia es almacenado en una lista (sin repetición de elementos). En el ejemplo de la figura 2, la lista final contendrá los frames de referencia 1 y 2. Entonces, para cada frame de referencia de HEVC, la lista es comprobada y sólo se realizará el proceso de la ME con dicho frame si éste está contenido en la lista. En otro caso, el frame de referencia será descartado para la partición actual.

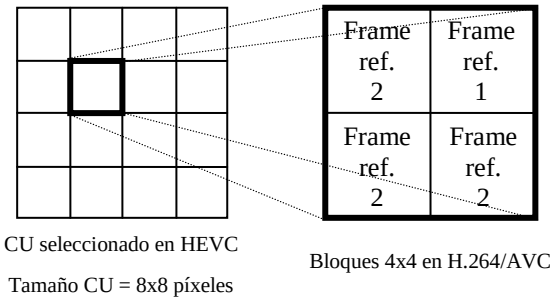


Fig. 2. Mapeo de bloques H.264/AVC para una CU de HEVC.

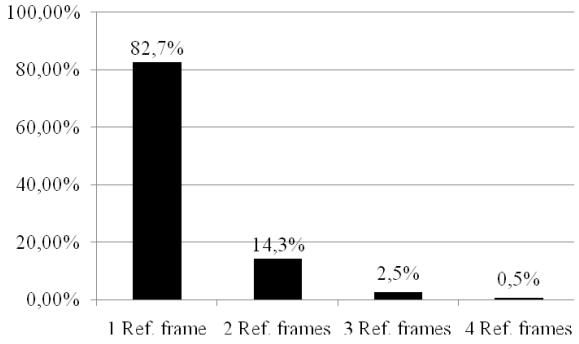


Fig. 3. Ejecuciones usando 1, 2, 3 ó 4 frames de referencia.

Este proceso puede que parezca que elabora una lista que siempre contiene todos los frames de referencia debido a la gran cantidad de bloques 4x4 que corresponden a una CU, por ejemplo, a una CU de 64x64 píxeles, le corresponden (64/4)2 = 256 bloques. Sin embargo, se ha realizado un estudio preliminar del problema y éste muestra que con este algoritmo, en media y usando 4 frames de referencia (cantidad por defecto en HM), el 82.7% de las ejecuciones sólo usan 1 frame de referencia, tal y como se muestra en la figura 3. Este resultado puede explicarse por el hecho de que si un bloque 4x4 usa el frame x como referencia, la redundancia espacial provocará que los bloques adyacentes a él también usen el mismo frame *x* como referencia con alta probabilidad.

V. Evaluación de prestaciones

La evaluación de prestaciones se ha llevado a cabo siguiendo las indicaciones dadas en [14] con los ficheros de configuración provistos por JM y HM. Sin embargo, únicamente se ha utilizado la configuración "Low Delay" con el uso de slices de tipo P, dado que, tal y como se ha dicho previamente, esta propuesta sólo afecta a las particiones de tipo P. De igual forma, sólo se ha usado la configuración "High Efficiency", ya que las configuraciones "HM-like" dadas en JM no incluyen la configuración "Main". Sin embargo, el enfoque que se presenta en este artículo podría ser fácilmente adaptado para su uso con slices de tipo B y con la configuración de "Random Access".

En primer lugar, todas las secuencias han sido codificadas bajo el estándar H.264/AVC mediante el uso de JM 18.4 [15]. Tras ello, los ficheros resultantes han sido transcodificados usando el decodificador de JM 18.4 y el codificador de HM 9.0 [3]. Este proceso se ha realizado con el transcodificador aquí presentado y con el transcodificador de referencia, el cual consiste en decodificar el flujo de H.264/AVC y volverlo a codificar con el codificador de HEVC sin mejora alguna.

En este transcodificador de referencia, los resultados muestras que el módulo de la ME consume, en media, el 77% del tiempo de codificación de HEVC, mientras que el resto de módulos suponen, tan sólo, el 23% de este tiempo. Esto supone que la ME conlleva una gran parte del tiempo de ejecución del codificador de HEVC, tal y como se muestre en la figura 4, lo que viene a reforzar la motivación dada en la Sección II sobre la necesidad de acelerar este módulo.

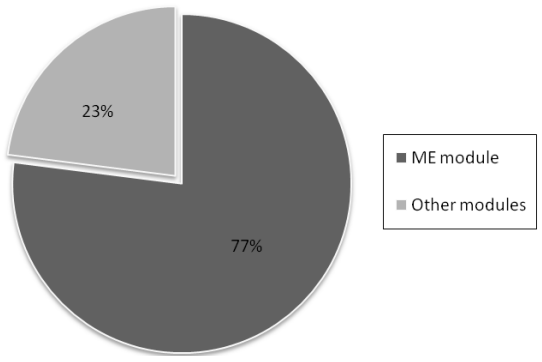


Fig. 4. Tiempo consumido por el módulo de la ME en HEVC.

TABLA 2. BD-RATE Y AHORRO DE TIEMPO DE LA PROPUESTA.

Clase de secuencias	BD-rate Y (%)	BD-rate U (%)	BD-rate V (%)	Ahorro total de tiempo (%)	Ahorro de tiempo en la ME (%)	Ahorro de tiempo en el bucle (%)
B	1.9%	1.8%	1.8%	31.7%	41.6%	60.6%
C	2.0%	1.6%	2.3%	32.1%	44.0%	64.9%
D	3.6%	3.7%	3.7%	30.0%	42.6%	64.7%
E	1.4%	1.5%	1.4%	40.7%	48.4%	69.2%
F	2.2%	2.3%	2.4%	34.2%	43.5%	61.4%
Media	2.2%	2.3%	2.4%	33.3%	43.2%	62.6%

Por otra parte, los resultados, en términos de BD-rate (el incremento medio de bitrate) [16] y ahorro de tiempo de codificación para el *Total* de todos los módulo de HEVC, para el módulo de la *ME* y para el *Bucle* acelerado en el código, con respecto al transcodificador de referencia, pueden verse en la tabla 2. Estos resultados son los valores medios para todas las secuencias de una misma clase (un total de 20 secuencias).

Tal y como puede verse en ellos, el transcodificador propuesto es capaz de reducir el tiempo *Total* de codificación en, aproximadamente, una tercera parte del original, manteniendo la misma calidad que éste mientras que el bitrate se incrementa sólo en un 2.2% para la componente Y, un 2.3% para la componente U y un 2.4% para componente V. De esta forma, se obtiene un incremento medio del (4·2.2 + 2.3 + 2.4) / 6 = 2.25% respecto al codificador de referencia, aumento justificado con la ganancia en el tiempo de codificación.

Los resultados para la clase E son especialmente significativos, ya que la configuración usada es muy adecuada para entornos de videoconferencia (siendo la clase E secuencias de este tipo de entornos). Para esta clase de secuencias, el transcodificador propuesto puede conseguir un ahorro de un 40% del *Total* del tiempo de codificación, incrementando el bitrate en tan sólo un 1.4%.

Además, puede observarse que el tiempo de la ME es reducido, en media, en un 43.2% y que el *Bucle* objetivo es acelerado un 62.6%, lo que representa un valor muy próximo al 75%, que es el valor máximo teórico que se ve reducido al tener que buscar a veces en varios frames de referencia (tal y como se ha mostrado en la Sección III del artículo).

Finalmente, la figura 5 y la figura 6 muestran los resultados graficados en términos de RD para todas las secuencias de video clasificadas por clases. Para cada una de las secuencias pueden observarse dos gráficas, la de línea continua con punto relleno muestra el PSNR para un cierto bitrate en el transcodificador de referencia, mientras que la de línea discontinua con puntos sin rellenar lo muestran para el transcodificador propuesto.

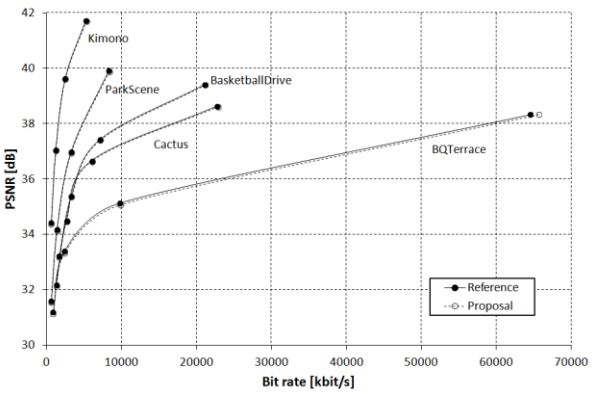
Tal y como puede verse, las curvas RD de el transcodificador de referencia y el propuesto son muy similares. La insignificante pérdida en rendimiento RD se ve más que compensada por la disminución de la complejidad computacional de la propuesta.

VI. CONCLUSIONES Y TRABAJO FUTURO

Este artículo presenta una novedosa arquitectura para la emergente transcodificación entre H.264/AVC y HEVC, basada en la extracción de los índices de los frames de referencia usados por H.264/AVC durante su decodificación y su reutilización en la parte del codificador de HEVC. Los resultados muestran que se ahorra, en media, un 33% del tiempo total de codificación, lo que supone un ahorro del 43% del tiempo en el módulo de la ME (módulo más costoso computacionalmente del codificador) y un ahorro del 62% del tiempo usado por el bucle acelerado. Este ahorro se consigue con un coste insignificante en términos de RD.

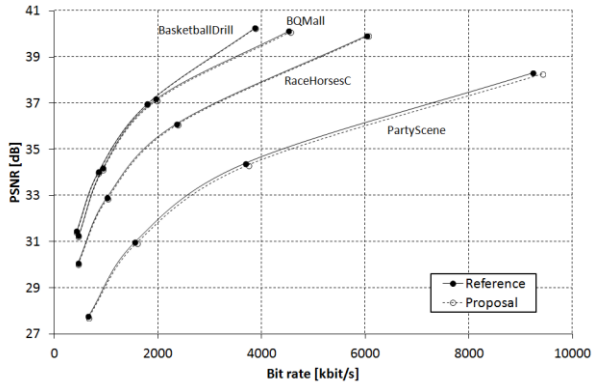
Como trabajo futuro se espera que esta implementación pueda ser extendida a su uso con slices de tipo B y que el transcodificador pueda mejorarse reutilizando más información ya codificada en H.264/AVC, como modos de codificación de los MBs, abriendo las posibilidades de aceleración al particionamiento de las CUs o a los modos de decisión de las TUs.

También puede estudiarse extraer la información de los MVs codificados por H.264/AVC, no para acelerar la obtención de los nuevos MVs en HEVC, ya que el algoritmo por defecto ya es lo suficientemente rápido, sino para mejorar la predicción inicial realizada por el nuevo estándar, ganándole de esta forma en calidad en términos de BD-rate.

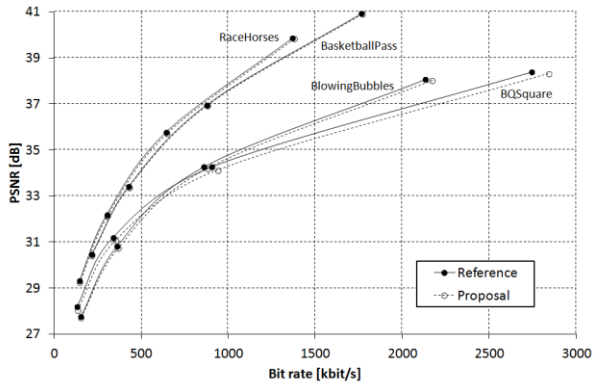


a. Secuencias de la Clase B

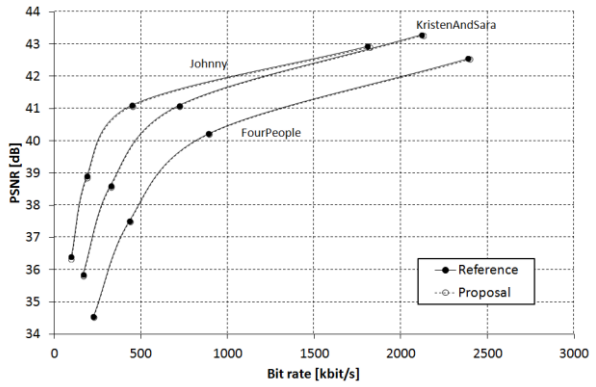
Fig. 5. Prestaciones RD (Referencia vs. Propuesta) I



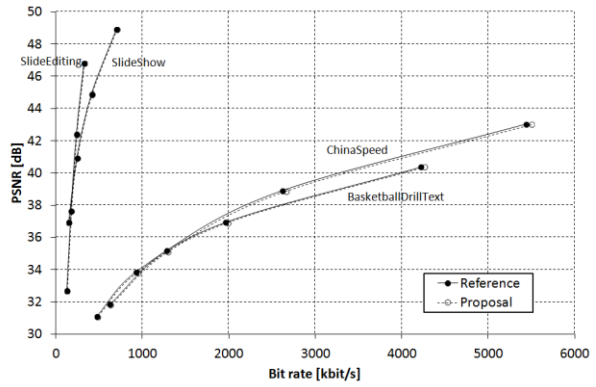
a. Secuencias de la Clase C



b. Secuencias de la Clase D



c. Secuencias de la Clase E



d. Secuencias de la Clase F

Fig. 6. Prestaciones RD (Referencia vs. Propuesta) II

AGRADECIMIENTOS

Este trabajo ha sido financiado conjuntamente por el Ministerio de Economía y Competitividad de España y por la Comisión Europea (fondos FEDER) bajo el proyecto CICYT TIN2012-38341-C04-04.

De igual forma, este trabajo también ha sido financiado por el Ministerio de Educación, Cultura y Deporte bajo la beca FPU12/00994.

REFERENCIAS

[1] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards – Including High Efficiency Video Coding (HEVC)", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, Issue 22, pp. 1669-1684, December 2012.

[2] G. J. Sullivan, J-R Ohm, W-J Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, Issue 22, pp. 1649-1668, December 2012.

[3] I.-K- Kim, K. McCann, K. Sugimoto, B. Bross, W.-J. Han, "HM9: High Efficiency Video Coding (HEVC) Test Model 9 Encoder Description", Joint Collaborative Team on Video Coding (JCT-VC) 11th Meeting: Shanghai, CN, 10-19 October, 2011

[4] S. Zhu and K.-K. Ma, "A New Diamond Search Algorithm for Fast Blockmatching Motion Estimation," IEEE Transaction on Image Processing, Vol. 9, Issue 2. pp.287–290, February 2000

[5] D. Sim. "Complexity and Performance Analysis of HEVC Encoder". Seminar Research Report. Image Processing Systems Laboratory. Kwangwoon University. August 2012.

[6] A. Vetro, C. Christopoulos and H. Sun, "Video Transcoding Architectures and Techniques: an Overview," IEEE Signal Processing Magazine, Vol. 20, Issue 2, pp. 18-29, March 2003.

[7] G. Fernandez-Escribano, H. Kalva, P. Cuenca, L. Orozco-Barbosa and A. Garrido, "A Fast MB Mode Decision Algorithm for MPEG-2 to H.264 P-frame Transcoding", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 18, Issue 2, pp. 172–185, February 2008.

[8] G. Fernandez-Escribano, J. Bialkowski , J.A. Gamez, H. Kalva, P. Cuenca, L. Orozco-Barbosa and A. Kaup, "Low-Complexity Heterogeneous Video Transcoding Using Data Mining", IEEE Transactions on Multimedia, Vol. 10, Issue 2, pp. 286–299, February 2008.

[9] R. Garrido-Cantos, J. De Cock, J. L. Martínez, S. Van Leuven, and P. Cuenca, "Motion-Based Temporal Transcoding from H.264/AVC-to-SVC in Baseline Profile", IEEE Transactions on Consumer Electronics, Vol. 57, Issue. 1, pp. 239-246, February 2011.

[10] A. Corrales-Garcia, J. L. Martinez , G. Fernandez-Escribano and F. J. Quiles, "Variable and Constant bitrate in a DVC to H.264/AVC transcoder", Signal Processing: Image Communication, Vol. 26, Issue 6, pp. 310-323, July 2011

[11] E. Peixoto and E. Izquierdo, "A Complexity-Scalable Transcoder form H.264/AVC to the New HEVC Codec", International Conference on Image Processing (ICIP), Orlando, FL, USA, September 2012

[12] D. Zhang, B. Li, J. Xu and H. Li, "Fast Transcoding From H.264/AVC To High Efficiency Video Coding", IEEE International Conference on Multimedia and Expo (ICME), pp. 651-656, Melbourne, Australia, July 2012.

[13] T. Shen, Y. Lu, Z. Wen, L. Zou, Y. Chen, J. Wen, "Ultra Fast H.264/AVC to HEVC Transcoder", Data Compression Conference (DCC), pp. 241-250, Cliff Lodge, UT, USA, March 2013.

[14] F. Bossen, "Common Test Conditions and Software Reference Configurations", Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 10th Meeting: Stockholm, SE, 11-20, July 2012.

[15] Join Collaborative Team on Video Coding, "Reference Software to Committee Draft, version 18.4," 2012.

[16] G. Bjontegaard, "Improvements of the BD-PSNR model", ITU-T SG16/Q6, 35th VCEG Meeting, Berlin, Germany, 16th - 18th July, 2008, Doc.VCEG-AI11

Paralelización mediante paso de mensajes del código COBRA-TF de simulación nuclear

Enrique Ramos[1], José E. Román[2], Agustín Abarca[3] y Rafael Miró[4]

Resumen— **Los análisis en seguridad nuclear a nivel de subcanal requieren de la utilización de códigos acoplados neutrónico-termohidráulicos con alta resolución espacial. Esta selección requiere tener códigos termohidráulicos que permitan la simulación de grandes dominios físicos con alta resolución y en un tiempo razonable. En este trabajo se presenta una paralelización del código de subcanal COBRA-TF de modo que puedan utilizarse clústeres de computadores. COBRA-TF (Coolant Boiling in Rod Arrays Code-TwoFluid) es un código termohidráulico de subcanal (permite resolver problemas a nivel de varilla de combustible nuclear) que utiliza dos campos y tres fases para modelar el flujo bifásico (agua+vapor). En su origen es un código secuencial caracterizado por unos tiempos de cálculo muy elevados a la hora de realizar las simulaciones y con una gran demanda de memoria, que hace muy complicada su ejecución en secuencial. Por ello, resulta de vital importancia reducir sensiblemente este tiempo para poder simular condiciones del reactor con el menor tiempo de respuesta posible. Este trabajo explica la paralelización que se ha llevado a cabo mediante la incorporación de PETSc, con el fin de reducir, en la medida de lo posible, el tiempo necesario para cada simulación y poder ejecutar trabajos que requieran de una gran cantidad de memoria.**

Palabras clave— **Resolución de Sistemas de Ecuaciones, PETSc, MPI, Simulación Termohidráulica, Reactor Nuclear.**

I. Introducción

En ingeniería nuclear es habitual el uso de códigos de simulación numérica para analizar el comportamiento de los diferentes componentes de las plantas nucleares, tanto en funcionamiento normal del reactor como en situaciones menos frecuentes como pueden ser la recarga de combustible y secuencias accidentales. La elevada exigencia en términos de seguridad hace necesario manejar modelos de gran nivel de detalle, lo que conlleva un alto coste computacional de las simulaciones. La paralelización de los códigos es muy conveniente en este contexto, para hacer más ágil este tipo de estudios.

COBRA-TF (Coolant Boiling in Rod Arrays Code-TwoFluid) es un código termohidráulico de subcanal (permite resolver problemas a nivel de varilla de combustible nuclear) que utiliza dos campos y tres fases para modelar el flujo bifásico (agua+vapor) [1], [2]. Las tres fases son el vapor, el líquido continuo y la fase líquida presente dentro de las burbujas de vapor. Cada fase utiliza un conjunto de tres ecuaciones tridimensionales para la masa, momento y energía, con una excepción: se utiliza una ecuación de la energía común para ambas fases líquidas, la continua y el líquido presente en la fase vapor. La formulación de dos fluidos emplea un conjunto separado de ecuaciones de conservación y relaciones constitutivas para cada fase. Los efectos de una fase en otra se tienen en cuenta en los términos de interacción que aparecen en la correspondiente ecuación de gobierno. Las ecuaciones de conservación tienen la misma forma para cada fase; solamente se diferencian en las relaciones constitutivas y las propiedades físicas.

En este trabajo se presenta la primera fase de la paralelización del código COBRA-TF, la cual ha consistido en la integración con una librería paralela para resolución de sistemas de ecuaciones, concretamente PETSc [4]. Además se ha paralelizado la generación del Jacobiano, de modo que cada procesador se encarga de la generación de los coeficientes asociados a un subconjunto de celdas. En las secciones II y III se describen algunos detalles del código que son relevantes para la paralelización. Las secciones IV,V y VI describen la librería PETSc y la estrategia utilizada, y los resultados preliminares se muestran en la sección VII.

II. Combustible usado en la simulación

Para realizar las simulaciones con el código COBRA-TF se ha modelado un elemento combustible tipo PWR genérico de un reactor nuclear de agua a presión (PWR). Este elemento combustible tiene 3.4 m de longitud activa y consta de 236 varillas de combustible y 20 tubos guía. En la figura 1 podemos ver una representación de un elemento combustible de reactor nuclear PWR.

El modelo termohidráulico realizado mediante COBRA-TF consta de 256 canales termohidráulicos, 236 de los cuales están calefactados ya que presentan combustible nuclear y 20 son tubos guía de las barras de control. Axialmente el modelo se ha dividido en 100 nodos, con lo que en conjunto se obtienen 25600 celdas computacionales por cada elemento combustible. Se han modelado mediante coeficientes de pérdidas de presión las 6 rejillas separadoras situadas en los niveles axiales 11, 26, 41, 55, 70 y 85 del elemento combustible. Además en el primer y último nodo se modelan, también mediante coeficientes de pérdidas de presión, los cabezales del combustible. La disposición radial de varillas de combustible y tubos guía dentro de la matriz de combustible se presenta en la figura 2.

Se realiza la simulación del estado estacionario de

[1]Dpto. de Sistemas Informáticos y Computación, Universitat Politècnica de València e-mail: ramos@dsic.upv.es.
[2]Dpto. de Sistemas Informáticos y Computación, Universitat Politècnica de València e-mail: jroman@dsic.upv.es.
[3]Dpto. de Ingeniería Química y Nuclear, Universitat Politècnica de València e-mail: aabarca@iqn.upv.es.
[4]Dpto. de Ingeniería Química y Nuclear, Universitat Politècnica de València e-mail: rmiro@iqn.upv.es.

Fig. 1. Elemento combustible de reactor PWR.

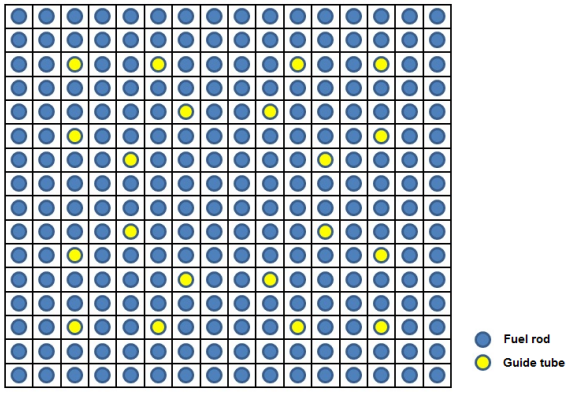


Fig. 2. Disposición de varillas de combustible y tubos guía en la matriz del combustible.

un elemento combustible como el descrito anteriormente. Las condiciones termohidráulicas de contorno son de caudal másico y temperatura a la entrada y presión a la salida. Además, la potencia se mantiene constante a lo largo de toda la simulación.

III. Resolución Numérica

El primer paso de la resolución consiste en resolver las ecuaciones de momento de cada celda por eliminación Gaussiana, cuya forma matricial es la siguiente:

$$\begin{bmatrix} c_1 - 1 & d_1 & 0 \\ c_2 & d_2 - 1 & e_2 \\ 0 & d_3 & e_3 - 1 \end{bmatrix} \begin{Bmatrix} f_l \\ f_v \\ f_e \end{Bmatrix} = \quad (1)$$

$$\begin{Bmatrix} -a_1 - b_1 \Delta P \\ -a_2 - b_2 \Delta P \\ -a_3 - b_3 \Delta P \end{Bmatrix} \quad (2)$$

El segundo paso nos permite calcular las velocidades necesarias para la linealización de las ecuaciones de masa y energía. Mediante el método de Newton-Raphson [3] por bloques se obtiene la variación de las variables independientes que garantizan un error residual nulo, cuya ecuación podemos ver en la figura 3.

$$E_{\mathrm{CV}} = \frac{\left[(\alpha_v \rho_v)_j^n - (\alpha_v \rho_v)_j\right] A_{c_j}}{\Delta t} + \sum_{\mathrm{KA}=1}^{\mathrm{NA}} \frac{\left[(\alpha_v \rho_v)^* \widetilde{U}_{v_j} A_{m_j}\right]_{\mathrm{KA}}}{\Delta x_j} - \sum_{\mathrm{KB}=1}^{\mathrm{NB}} \frac{\left[(\alpha_v \rho_v)^* \widetilde{U}_{v_{j-1}} A_{m_{j-1}}\right]_{\mathrm{KB}}}{\Delta x_j} - \sum_{L=1}^{\mathrm{NKK}} S_L [(\alpha_v \rho_v)^* \widetilde{V}_{v_L}]_j - \frac{\Gamma_j}{\Delta x_j} - \frac{S_{cv_j}}{\Delta x_j}$$

Fig. 3. Ecuación de masa de vapor.

Aplicando Newton-Raphson podemos obtener las ecuaciones matriciales para cada celda que se muestran en la figura 4, en la que podemos ver el Jacobiano del sistema de ecuaciones evaluado para las variables independientes, y compuesto por derivadas analíticas de cada ecuación con respecto a la variación lineal de las variables independientes. También podemos observar el vector solución que contiene las variaciones lineales y el vector de error.

$$\begin{bmatrix} \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CG}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CG}}}{\partial P_j} & \frac{\partial E_{\mathrm{CG}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CG}}}{\partial P_{\mathrm{NCON}}} \\ \frac{\partial E_{\mathrm{CL}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CL}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CL}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CL}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CL}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CL}}}{\partial P_j} & \frac{\partial E_{\mathrm{CL}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CL}}}{\partial P_{\mathrm{NCON}}} \\ \frac{\partial E_{\mathrm{CV}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CV}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CV}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CV}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CV}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CV}}}{\partial P_j} & \frac{\partial E_{\mathrm{CV}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CV}}}{\partial P_{\mathrm{NCON}}} \\ \frac{\partial E_{\mathrm{CE}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CE}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CE}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CE}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CE}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CE}}}{\partial P_j} & \frac{\partial E_{\mathrm{CE}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CE}}}{\partial P_{\mathrm{NCON}}} \\ \frac{\partial E_{\mathrm{EL}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{EL}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{EL}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{EL}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{EL}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{EL}}}{\partial P_j} & \frac{\partial E_{\mathrm{EL}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{EL}}}{\partial P_{\mathrm{NCON}}} \\ \frac{\partial E_{\mathrm{EV}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{EV}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{EV}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{EV}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{EV}}}{\partial P_j} & \frac{\partial E_{\mathrm{EV}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{EV}}}{\partial P_{\mathrm{NCON}}} \end{bmatrix} \cdot \begin{Bmatrix} \delta\alpha_g \\ \delta\alpha_v \\ \delta\alpha_v h_v \\ \delta(1-\alpha_v) h_l \\ \delta\alpha_e \\ \delta P_j \\ \delta P_{i=1} \\ \vdots \\ \delta P_{i=\mathrm{NCON}} \end{Bmatrix} = - \begin{Bmatrix} E_{\mathrm{CG}} \\ E_{\mathrm{CL}} \\ E_{\mathrm{CV}} \\ E_{\mathrm{CE}} \\ E_{\mathrm{EL}} \\ E_{\mathrm{EV}} \end{Bmatrix}$$

Fig. 4. Ecuación matricial.

Una vez resuelta la ecuación se reduce para obtener la solución para las variables independientes, derivando una ecuación para cada celda, que obliga a resolver un sistema de ecuaciones formado por todas las celdas. Si se alcanza la convergencia se pasa al

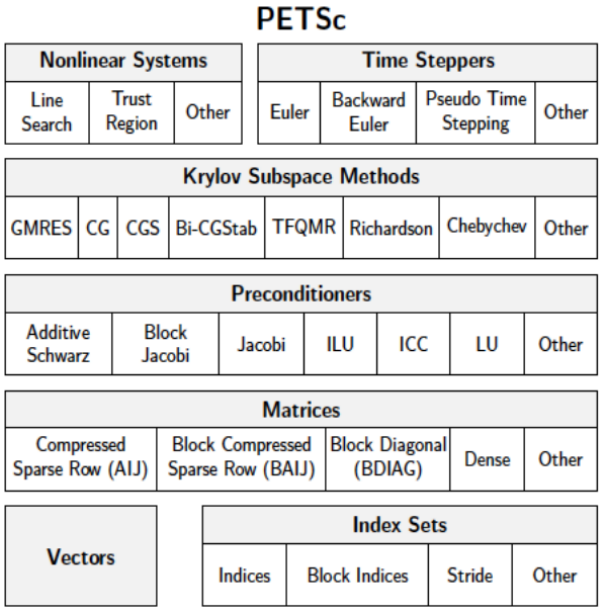


Fig. 5. Componentes de PETSc.

paso de tiempo siguiente, de lo contrario se disminuye el paso de tiempo de la simulación y se repite el proceso.

En resumen, cada iteración externa se compone principalmente de dos partes: el cálculo de los coeficientes y la resolución de un sistema de ecuaciones lineales. La matriz coeficiente de este sistema de ecuaciones es dispersa (el porcentaje de elementos no nulos es muy bajo). Esto se debe al método de discretización utilizado, concretamente se trata del método de diferencias finitas. Es importante aprovechar esta característica para resolver el sistema eficientemente, utilizando métodos iterativos. En este trabajo, se ha abordado la paralelización de la resolución del sistema y del cálculo de los coeficientes mediante el reparto de las celdas del modelo entre varios procesos MPI.

IV. PETSc

PETSc (Portable Extensible Toolkit for Scientific Computation, [4], [5], [6]), es un software numérico orientado a objetos para la resolución de ecuaciones en derivadas parciales, paralelizado mediante paso de mensajes. Está siendo utilizado en muchas aplicaciones científicas en todo el mundo. En PETSc todo el código se construye alrededor de una serie de estructuras de datos y algoritmos que han sido encapsuladas mediante técnicas de orientación a objetos. Su ventaja consiste en trabajar directamente con estos objetos abstrayéndose de la estructura de datos interna, lo que le da una gran potencia al permitir programar los métodos numéricos y aplicaciones habituales sin tener que estar pendiente de multitud de detalles de implementación relativos a las estructuras de datos o la paralelización. En la figura 5 podemos ver sus componentes principales.

Aparte de algunos objetos auxiliares relativos a la gestión de las mallas, los componentes básicos a nivel de datos que incluye PETSc son los vectores y las matrices. Estos objetos son análogos a los vectores y matrices del álgebra lineal, y PETSc ofrece un gran número de operaciones que permiten realizar los cálculos más habituales, facilitando enormemente la labor del programador. Por encima de ellos tenemos la posibilidad de usar varias clases de solvers, tanto lineales como no lineales, diferentes precondicionadores y métodos de resolución de sistemas de ecuaciones entre otras posibilidades, todo ello en el entorno paralelo que el propio PETSc gestiona en función de las características del sistema en el cual se instala.

Las matrices en PETSc tienen una interfaz independiente de la implementación, lo que implica poder utilizar diferentes formatos de almacenamiento de matrices sin necesidad de modificar el código fuente. Esto permite por ejemplo especificar el formato de almacenamiento al ejecutar el programa, pudiendo así comparar las prestaciones de las distintas alternatives. Por defecto las matrices se almacenan en el formato disperso comprido por filas. Otras alternativas disponibles son el formato simétrico (almacenando únicamente la parte triangular superior) o el formato por bloques. Todos ellos son estructuras de datos paralelas, con distribución orientada a bloques de filas contiguas, y la comunicación necesaria para realizar las operaciones en paralelo se gestiona internamente por PETSc. En su uso básico, el programador solo ha de preocuparse de que cada proceso MPI se encargue de rellenar la parte que le ha sido asignada.

PETSc permite instalaciones en modo debugger, para permitir la depuración de errores, y en modo optimizado, con el cual podemos conseguir tiempos de cálculo sensiblemente inferiores, adecuado ya para la fase de producción de resultados.

V. Tareas Computacionales

Las principales tareas computacionales requeridas en cada paso son las siguientes:

1. Actualizar las variables de estado respecto al paso de tiempo actual.
2. Computar la matriz Jacobiana y el vector de residuos de la ecuación no lineal para la iteración de Newton-Raphson.
3. Resolver el sistema lineal de ecuaciones.

Debido al esquema de discretización por diferencias finitas, los dos primeros paso requieren del uso de valores de celdas vecinas, con la necesidad de comunicación que ello conlleva.

Respecto al tercer paso, hay que señalar que la matriz de coeficientes del sistema lineal es dispersa, por lo que los métodos iterativos son los recomedables para la resolución del sistema.

VI. Estrategia de Paralelización

La paralelización permite el uso de varios procesadores (o cores) cooperando en la solución de un problema simple. Esto no sólo reduce el tiempo de

computación sino que también incrementa la cantidad de memoria disponible permitiendo resolver problemas más grandes.

La paralelización efectiva de COBRA-TF debe contemplar dos puntos importantes:

1. Los tres pasos mencionados anteriormente deben ser paralelizados, no sólo la solución del sistema de ecuaciones.
2. La memoria se debe manejar de forma apropiada, repartiendo las variables de estado entre los diferentes procesos (en el caso de paralelización con MPI).

La paralelización de COBRA-TF precisa de una estrategia híbrida, dónde la partición del cómputo se desarrolla en dos niveles:

1. En el nivel superior se ha desarrollado una paralelización basada en la estrategía de descomposición de dominios con el paradigma de paso de mensajes (MPI).
2. En el segundo nivel se prevé realizar en el futuro una paralelización de grano fino asociada a cada subdominio.

Este esquema permite mejor escalabilidad en algunas plataformas de computación, tales como clusters de multicores, o clusters con aceleradores en cada nodo. En este trabajo nos hemos centrado en el nivel superior únicamente.

A. Selección del punto caliente

Para abordar la estrategia correcta a la hora de paralelizar el código COBRA-TF, en primer lugar se realizó un estudio del tiempo y su porcentaje respecto al total, que consumía cada uno de los métodos que se ejecutaban en el proceso. Para ello se utilizó la herramienta **Valgrind** [7].

Analizando los resultados obtenidos con Valgrind se aprecia como el método **outer** acumula el 87.75% del coste, que a su vez llama a **xschem**, quien ejecuta **gssolv**, lugar dónde se resuelve el sistema de ecuaciones propiamente dicho. Viendo que el resto de llamadas se realizan un número muy alto de veces y con coste poco significativo, se tomó este punto como el objetivo de la paralelización. Una vez deducido el punto adecuado para realizar la paralelización, se establece que es necesario eliminar todas las llamadas a las subrutinas que se ejecutan por debajo de **krylovsolv**, de forma que estos ya no serán necesarios en el cálculo ya que implican el uso de la librería **SPARSKIT** para la resolución de los sistemas de ecuaciones.

Con esta estrategía a la hora de elegir el punto caliente, conseguimos realizar la paralelización de la parte del cálculo que absorbe la mayor cantidad de tiempo. Esto implica que tendremos mayores beneficios a medida que aumente el número de veces que hay que resolver el sistema de ecuaciones y sobre todo si aumenta su dimensión, ya que la parte paralela saldrá muy beneficiada frente a realizar el mismo cálculo en secuencial.

B. Integración de PETSc

La parte fundamental de la paralelización de la resolución del sistema de ecuaciones es la sustitución de la librería **SPARSKIT** por **PETSc** como medio para resolver los sistemas de ecuaciones. Para ello se sustituyeron las llamadas a SPARSKIT por las correspondientes a PETSc analizando el lugar correcto para su inicialización, así como la creación y destrucción de todos los objetos (matrices y vectores) necesarios en el cálculo. Todo ello se tuvo que coordinar con las estructuras de datos ya existentes en el código FORTRAN.

Todo ello precedido de la instalación y optimización de PETSc para amoldarse al sistema en el cual se desarrolla la ejecución.

C. Adaptación de PETSc

Una vez resuelto el problema de sustituir SPARSKIT por PETSc, se procedió a analizar las diferentes posibilidades que PETSc ofrece a la hora de gestionar las matrices, vectores y solvers. Para ello se realizó un estudio del patrón de elementos no nulos de la matriz, para poder realizar la reserva de memoria de forma adecuada y en el lugar adecuado. El hecho de reservar correctamente la memoria en los objetos que crea PETSc ofrece importantes ventajas en cuanto a eficiencia a la hora de resolver el sistema de ecuaciones. Además, debido a que el sistema se tiene que resolver muchas veces hasta alcanzar la convergencia, se estudió la posibilidad de reusar los objetos matriz, vector y solver para ganar velocidad, de forma que al mantener su estructura invariable, se consiguió un considerable aumento de velocidad con la correcta reutilización de los mismos.

D. Afinar el PETSc

En la fase de optimización hay que deducir que combinación de precondicionadores y métodos de resolución es la mejor opción para que PETSc realice el cálculo en la menor cantidad de tiempo posible, permitiendo obtener una significativa ganancia de velocidad. No todos los precondicionadores y métodos funcionan igual en el mismo caso, por lo que se realizó una prueba completa probando las diferentes combinaciones existentes en PETSc. Para ello se probaron los siguientes métodos y precondicionadores [8]:

- **Precondicionadores:** Block Jacobi (con ILU(0) como precondicionador local), Jacobi, SOR, Additive Schwarz (con ILU(0) como precondicionador local).
- **Métodos de resolución de sistemas:** GMRES, DGMRES, BiCGstab, BiCGstab(l).

E. Paralelización axial de los nodos del combustible

Una vez paralelizado el sistema de ecuaciones, se procedió a realizar el reparto paralelo de las celdas entre los procesos en su dimensión axial, esto es, cada proceso MPI tiene asignado un espacio continuo de niveles axiales. Habitualmente el número de niveles axiales es limitado, por lo que la escalabilidad

del código paralelo está limitada en esta dimensión. Este problema será compensado en un futuro por la paralelización del segundo nivel comentada anteriormente. Como ya se ha comentado, en muchas ocasiones se necesita información de niveles vecinos, por lo que cada proceso deberá tener espacio para estos niveles "fantasma" y además realizar de forma adecuada la comunicación de los datos implicados. Además siempre se cuenta con dos niveles extras, el inferior y el superior, que tienen que ser actualizados de forma correcta en cada iteración.

Esta etapa conlleva tres partes claramente diferenciadas:

- Reparto entre los procesos de las variables que simulan el comportamiento del combustible.
- Comunicación de los datos de dichas variables que están en la frontera en el reparto entre procesos y que por lo tanto son necesarios en los procesos vecinos.
- Reducción de los cálculos que implican máximos, mínimos y sumatorios al estar la carga repartida entre los procesos, de forma que todos posean la información adecuada a cada iteración.

VII. Resultados

Para el estudio se resolvió de forma completa con los métodos **bcgs**, **bcgsl**, **gmres** y **dgmres** variando con los precondicionadores **asm**, **bjacobi**, **jacobi** y **sor**. Señalar que los métodos gmres y dgmres divergían. En el caso del programa original, el método de resolución usado fue BiCGstab y el precondicionador ILUT (Incomplete LU factorization with dual truncation strategy). El precondicionador ILUT no está disponible en PETSc, dónde sí que está ILU(0), que suele ofrecer peores resultados.

En la tablas I, II y III podemos observar respectivamente el tiempo total de ejecución y el speedup obtenidos con las diferentes combinaciones. Todos los tiempos se obtuvieron en un HP Proliant con 2 procesadores AMD Opteron 6272 con 16 cores a 2.2 GHz con 2M de caché y 96 Gb de memoria RAM. El Sistema Operativo utilizado es CentOS 6.3 y la versión del PETSc es la 3.3-p6. Se utilizó el OpenMPI realizando un estudio de la afinidad entre procesos y unidades de proceso para obtener la mejor opción en cuanto a la ganancia de prestaciones, siendo la opción que mejores resultados proporcionaba la reserva de cores con la opción –cpus-per-proc 4. Los tiempos obtenidos en este ejemplo se corresponden con una simulación de 1 segundo.

En las figuras 6,7 y 8 podemos apreciar los tiempos y el speedup obtenidos con las diferentes combinaciones ejecutadas.

Podemos apreciar todas las combinaciones ofrecen un comportamiento similar no apreciándose ganancia significativa en ninguna de ellas.

VIII. Conclusiones

Las principales conclusiones que podemos obtener de la paralelización com MPI del programa COBRA-TF son las siguientes:

TABLA I
Tiempo total (seg) para el caso de estudio con hasta 8 procesos.

Combinación	1	2	4	8
Original	1421	-	-	-
bcgs-asm	1492	849	523	370
bcgs-bjacobi	1464	843	523	368
bcgs-jacobi	1550	892	551	390
bcgs-sor	1503	872	532	376
bcgsl-asm	1484	855	528	373
bcgsl-bjacobi	1468	861	523	374
bcgsl-jacobi	1564	900	549	391
bcgsl-sor	1507	871	536	375

TABLA II
Tiempo del solver (seg) para el caso de estudio con hasta 8 procesos.

Combinación	1	2	4	8
bcgs-asm	131	72	37	21
bcgs-bjacobi	119	67	33	20
bcgs-jacobi	206	115	61	38
bcgs-sor	153	84	42	26
bcgsl-asm	134	79	39	22
bcgsl-bjacobi	122	86	35	20
bcgsl-jacobi	218	122	64	39
bcgsl-sor	159	93	44	25

- El trabajo es un buen comienzo para la reducción significativa del tiempo de cálculo necesario para realizar las simulaciones. El hecho de haber paralelizado la resolución del sistema de ecuaciones y el reparto de las celdas del modelo nos permite obtener un speedup de 4.02 con 8 procesos.
- Resulta imprescindible paralelizar el combustible a nivel de canal y además probar con sistemas más grandes en los cuales la paralelización del sistema de ecuaciones nos será más beneficiosa. La ganancia frente al tiempo secuencial original es significativa, pero no es escalable a un mayor número de procesos.
- El presente trabajo ha abierto el camino para re-

TABLA III
Speedup para el caso de estudio con hasta 8 procesos.

Combinación	1	2	4	8
bcgs-asm	1	1.76	2.85	4.02
bcgs-bjacobi	1	1.74	2.79	3.98
bcgs-jacobi	1	1.74	2.81	3.97
bcgs-sor	1	1.72	2.83	4.00
bcgsl-asm	1	1.73	2.81	3.98
bcgsl-bjacobi	1	1.70	2.80	3.92
bcgsl-jacobi	1	1.74	2.85	4.00
bcgsl-sor	1	1.73	2.81	4.02

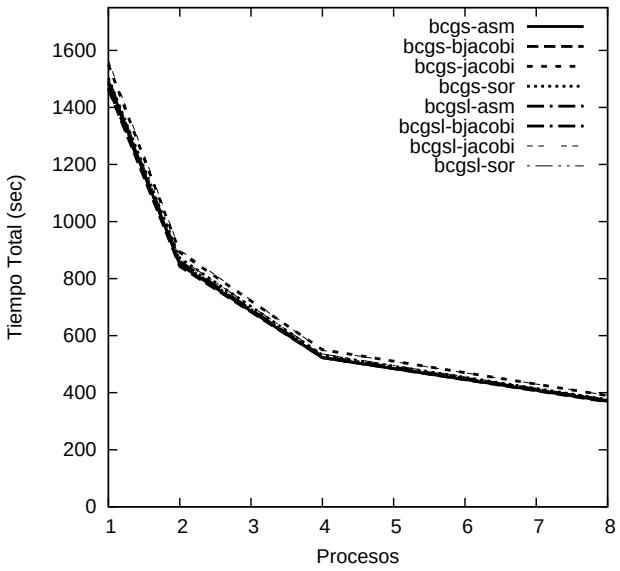


Fig. 6. Tiempo (seg) para el caso de estudio con hasta 8 procesos.

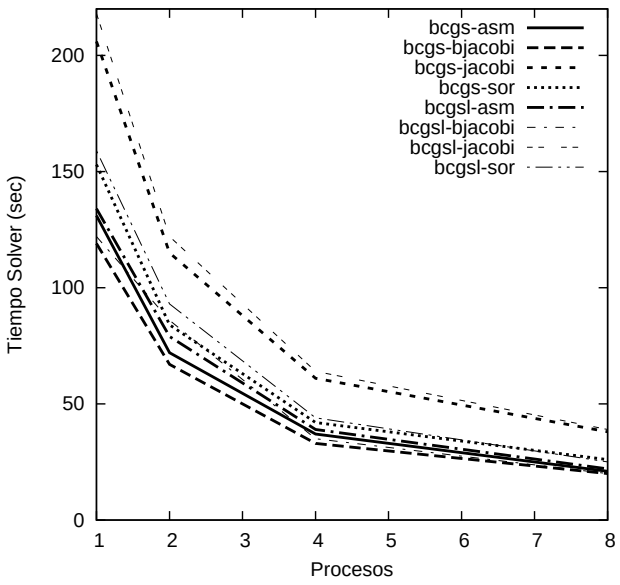


Fig. 7. Tiempo (seg) del solver para el caso de estudio con hasta 8 procesos.

alizar la paralelización completa del código, que por su dimensión implica un enorme esfuerzo de programación.

- La elección correcta de la combinación del método de resolución y el precondicionador nos permite sacar el máximo rendimiento posible a la simulación.

IX. Trabajo Futuro

A la vista de los resultados obtenidos, se presentan una serie de posibilidades de cara a un desarrollo futuro que permitirán mejorar sensiblemente las prestaciones:

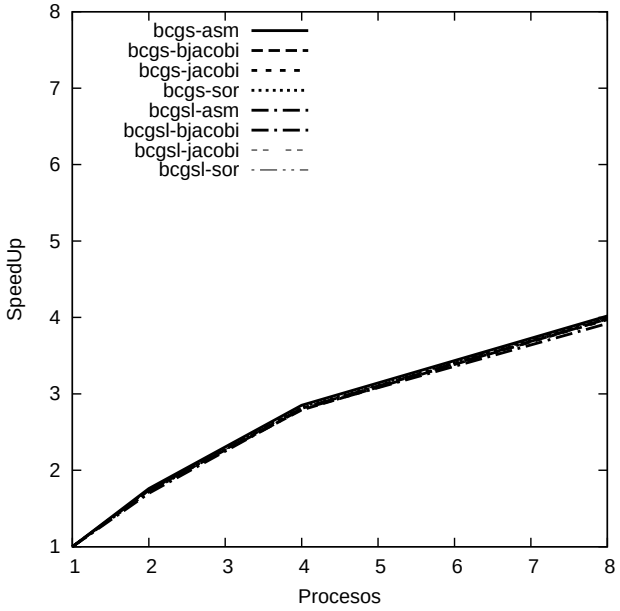


Fig. 8. Speedup para el caso de estudio con hasta 8 procesos.

- Realizar la paralelización completa del combustible en su dimensión radial con memoria compartida usando OpenMP. Esto permitirá rebajar el tiempo total de ejecución y la necesidad de memoria por cada unidad de proceso.
- Realizar ejecuciones en clústeres para poder comparar prestaciones.

Referencias

[1] Diana Cuervo, *Improving the Computation Efficiency of COBRA-TF for LWR Safety Analysis of Large Problems*, PHYSOR, Chicago, USA, 2004.

[2] Diana Cuervo, *Implementation and performance of Krylov Methods for the solution of Two-Fluid Hydrodinamics Equations in the COBRA-TF Code*, M and C, Avignon, France, 2005.

[3] Tjalling J. Ypma, *Historical development of the Newton-Raphson method*, SIAM Review 37 (4), 531-551, 1995.

[4] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, Hong Zhang, *PETSc Web page*, "http://www.mcs.anl.gov/petsc", 2011.

[5] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, Hong Zhang, *PETSc Users Manual, ANL-95/11 - Revision 3.3*, Argonne National Laboratory, 2012.

[6] Satish Balay, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, Barry F. Smith, *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries*, E. Arge and A. M. Bruaset and H. P. Langtangen, 1997.

[7] Cerion Armour-Brown, Jeremy Fitzhardinge, Tom Hughes, Nicholas Nethercote, Paul Mackerras, Dirk Mueller, Julian Seward, Bart Van Assche, Robert Walsh, Josef Weidendorfer, *valgrind.org*, 2012.

[8] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition, Society for Industrial and Applied Mathematics, 2003.

Algoritmo de baja complejidad para la predicción Intra-Frame en HEVC

Damián Ruiz[1], Velibor Adzic[2], Ray Garcia[2], Hari Kalva[2], Gerardo Fernández-Escribano[1], J. Luis Martínez[1] y Pedro Cuenca[1]

Resumen— **High Efficiency Video Coding (HEVC) es el nuevo estándar de codificación de vídeo, recientemente aprobado conjuntamente por el ISO y el ITU a través del Joint Collaborative Team on Video Coding (JCT-VC)**

HEVC permite una reducción de la tasa binaria de codificación superior al 50% con respecto a su predecesor, el estándar H.264/AVC, ofreciendo la misma calidad perceptual. Entre las nuevas herramientas introducidas por el nuevo estándar destaca la nueva predicción Angular Intra-Frame, que permite unas altas eficiencias de compresión en modo "All-Intra", comparado con el actual estado del arte en codificación de imágenes estáticas, como JPEGG2000 y JPEG XR, pero con un elevado coste computacional, que dificulta considerablemente su implementación en tiempo real.

Este artículo analiza la complejidad y eficiencia la nueva etapa de predicción Intra-Frame de HEVC, en especial en la implementación de su algoritmo de RDO (Rate Distortion Optimization). Finalmente, se presentan las investigaciones llevadas a cabo en el diseño de un algoritmo de bajo coste computacional, en la que se aborda el particionado rápido de la nueva unidad de codificación, así como del pre-filtrado de los 33 modos direccionales definidos en el estándar.

Palabras clave— **HEVC, Rate Distortion Optimization, Coding Tree Unit, Intra Prediction, Residual Quad Tree.**

I. INTRODUCCIÓN

En enero de 2013, un nuevo hito en la historia de la compresión de vídeo fue alcanzado con la aprobación del estándar de vídeo HEVC (High Efficiency Video Coding), por parte del grupo de trabajo JCT-VC (Joint Collaborative Team Video Coding); y que será estandarizado formalmente como ITU-T H.265[1], y MPEG-H Part2 (ISO/IEC 23008-2). El nuevo estándar nace como una evolución natural de su predecesor, el H.264/AVC, después de un ciclo de 10 años de indiscutible implantación en todos los segmentos multimedia del mercado domestico y profesional. HEVC fue concebido inicialmente con un doble objetivo, el de ofrecer una solución eficiente a la fuerte demanda de ancho de banda sobre redes fijas y móviles, para el consumo de servicios multimedia con fuertes restricciones de tasas binaria; y el de obtener una alta eficiencia de codificación para formatos de resolución superior a la HD, como los formatos 4k y 8k definidos en el nuevo estándar de Ultra Alta definición (*UHDTV*), que están experimentado un importante crecimiento con la comercialización de nuevas pantallas y sistemas de captación de UHDTV.

La finalización de esta primera fase del estándar concluyó con la aprobación de la versión 1 del mismo [2], la cual soporta formatos de vídeo con submuestreo 4:2:0, y en el que se han definido 3 perfiles, "Main Profile" y "Main10 Profile", que tan solo difieren en sus profundidades de píxel de 8-bits y 10-bits respectivamente, y un perfil "Main Still Profile", para aplicaciones de imágenes estáticas, soportando exclusivamente codificación predictiva Intra-Frame.

Desde las primeras evaluaciones de calidad de vídeo llevadas a cabo durante el desarrollo del estándar, HEVC ha demostrado una eficiencia del compresión superior al 50%, ofreciendo la misma calidad perceptual que H.264/AVC en su "High Profile" [3]. Estos resultados han sido confirmados recientemente por nuevas evaluaciones subjetivas [4][5], donde se ha mostrado como HEVC puede alcanzar un ahorro en ancho de banda del 66% para formatos de UHDTV, con respecto a su predecesor.

A pesar de que tanto H.264/AVC como HEVC son estándares de codificación de secuencias de vídeo, en [6] se muestra como en modo de codificación exclusivamente Intra-Frame (*All-Intra*), HEVC ofrece una mejora del 25% en su eficiencia con respecto a su predecesor. En [7], Nguyen y Marpre presentan los resultados de un estudio de la eficiencia del perfil "Main Still" de HEVC, con respecto a los principales estándares que representan el estado-del-arte en codificación de imágenes, como JPEG2000 [8] y JPEG XR [9] entre otros, revelando como HEVC puede obtener unos ahorros de tasa binaria entre un 22% y un 43% sobre dichos estándares.

Esta mejora se presenta como una gran oportunidad para el nuevo estándar, ampliando su campo de aplicación no solo a los formatos gráficos para soporte documental, sino también para el sector audiovisual profesión en el ámbito de la captura, edición y almacenamiento, donde por motivos de calidad y accesibilidad del contenido, se evita la utilización de esquemas de codificación con predicciones temporales entre frames. Como se describe en la tercera sección del documento, la complejidad de la predicción Intra-Frame definida en HEVC es extremadamente elevada, dificultando su implementación para aplicaciones en tiempo real, muy especialmente para formatos de vídeo de alta resolución como son el HD y el UHDTV.

Este hecho es el que motiva la actual investigación presentada en este trabajo, que cubre el análisis de la complejidad de la etapa de predicción Intra-Frame en HEVC, así como el diseño de un algoritmo de bajo coste computacional, que facilite la aceleración de su modelo de RDO (*Rate Distortion Optimization*).

[1] Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, Albacete, España
[2] Florida Atlantic University, Boca Raton, Florida, United States of America

En la siguiente sección se describen las principales novedades del nuevo estándar de codificación de vídeo HEVC, haciendo especial énfasis en la nueva herramienta de particionado de la unidad de codificación y en el nuevo modelo de predicción Intra-Frame. En la sección III, se describe la implementación del RDO del software de referencia de HEVC, donde se justifican las estrategias de aceleración incluidas. Un análisis y evaluación del coste computacional de la predicción Intra-Frame se muestra en la sección IV. Por último, en la sección V, se introduce el actual modelo bajo desarrollo que permite la aceleración de la etapa de predicción Intra-frame, donde se aborda el particionado rápido de la unidad de codificación y el pre-filtrado de modos direccionales.

II. ARQUITECTURA DE HEVC

Como hemos introducido, HEVC puede considerarse una evolución del actual H.264, ya que éste mantiene el esquema híbrido de codificación de sus antecesores, en los que la secuencia de vídeo es decorrelada temporalmente por medio de la etapa de Estimación y Compensación de Movimiento (ME-MC), y su residuo es compactado energéticamente en el dominio espacial por medio de técnicas transformacionales, utilizando para ello una versión entera de la transformada discreta coseno (DCT).

A nivel de bloques funcionales, HEVC introduce pequeñas mejoras de eficiencia en todos ellos con respecto al H.264/AVC, e incorpora tres novedades que lo distinguen sustancialmente de sus predecesores: la nueva unidad de codificación basada en una nueva estructura jerárquica de bloques denominada CTU (Coding Tree Unit), una nueva predicción angular Intra-Frame, y una nueva herramienta en el bucle de decodificación denominada SAO (Sample Adaptive Offset), que se aplica a las muestras reconstruidas después del deblocking filter, con el objetivo de mejorar perceptualmente la secuencia decodificada. Una descripción más completa de cada una de estas herramientas puede ser encontrada en [10].

El conjunto de pequeñas mejoras y novedades funcionales en la arquitectura de HEVC, le permiten obtener, de modo global, unos altos índices de eficiencia con respecto a otros estándares, pero como contrapartida su complejidad computacional crece considerablemente, siendo preciso abordar algoritmos y modelos que faciliten su reducción. En [11] se presenta un estudio exhaustivo de la complejidad computacional de cada una de las herramientas definidas en el estándar, y de su impacto en el esquema completo de codificación.

En los siguientes apartados se describe la nueva estructura de CTU así como la predicción Angular Intra-Frame, al ser estos los que mayor influencia tienen en la codificación en modo "All-Intra", en términos de eficiencia y complejidad.

A. *Particionado de la unidad de codificación CTU*

HEVC define la estructura de CTU como sustitución de la unidad de codificación basada un bloques no solapados de 16x16 pixeles (MacroBloque), definido en los anteriores entandares de codificación. La CTU es una nueva estructura flexible con un tamaño máximo de 64x64 píxeles, que puede ser iterativamente particionada en cuatro sub-bloques con la mitad de la resolución, hasta alcanzar un tamaño mínimo permitido de 8x8 pixeles.

Como se describe en [12], una CTU se estructura en un árbol jerárquico en el que en cada rama finaliza en una hoja, que define la unidad mínima de codificación denominada CU (Coding Unit). Cada unidad de codificación es a su vez una nueva raíz de dos nuevos árboles, que contienen las unidades de predicción, denominadas PU (Prediction Unit), y las unidades de transformación denominadas TU (Transform Unit). Una PU de resolución 2Nx2N codificada en modo Inter-Frame, puede ser nuevamente particionada en sub-bloques simétricos de resoluciones (2Nx2N, 2NxN, Nx2N, NxN) o en sub-bloques asimétricos (2NxU, 2NxD, nLx2N, nRx2N), mientras que si se codifica en modo Intra-Frame, sólo permite su particionado en un nuevo conjunto de bloques con resolución NxN, si la CU de la cual depende tiene la resolución mínima de 4x4.

El residuo obtenido en cada una de las unidades de predicción, es particionado y transformado aplicando una estructura de árbol denominada RQT (Residual QuadTree) [13], que permite un máximo de tres niveles de descomposición, y donde cada uno de los sub-bloques resultantes son las unidades de trasformación. Las TUs tienen limitado su tamaño de transformación en el rango entre 32x32 y 4x4. En la figura 1, se muestra un ejemplo de particionado de CTU en sus respectivas unidades de codificación, predicción y transformación, codificado en modo Intra-Frame.

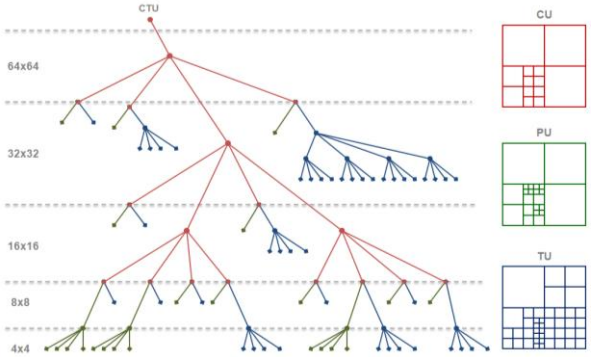


Figura.1. Particionado de CTU en unidades de CU, PU y TU

B. *Predicción Intra-Frame en HEVC*

El modelo predictivo Intra-Frame definido en HEVC, utiliza los mismos fundamentos aplicados en H.264/AVC, y que explotan la alta correlación espacial entre un bloque y sus pixeles vecinos de los bloques superior e izquierdo, utilizando estos pixeles como referencia para la construcción del bloque predictor.

HEVC define dos modos, DC y Planar, similares a sus equivalentes en H.264/AVC (DC y modo Plane), pero amplía el número de predictores direccionales hasta los 33, utilizando un nuevo modelo de cómputo de los predictores definido como "*Angular Intra Prediction*" [14].

Los 33 predictores direccionales son definidos con 16 ángulos con orientación vertical, 16 ángulos de

orientación horizontal, más un ángulo diagonal (±45º), con una mayor concentración en las direcciones dominantes horizontal y vertical (0º y 90º), con el objetivo de explotar la mayor ocurrencia de bloques con gradientes en dichas direcciones, y menor en la dirección diagonal. La predicción de cada pixel se calcula como su proyección sobre las muestras de los pixeles vecinos, en la dirección de su respectivo ángulo de predicción, aplicando una interpolación sobre las muestras más próximas con una precisión no-entera de 1/32 de pixel [15].

La figura 2 muestra el conjunto de predictores direccionales, donde se puede apreciar la variación de la densidad de predictores en función de su proximidad a los ángulos horizontal y vertical. Los pixeles de referencia, así como el residuo obtenido para ciertos tamaños de PU, son filtrados con el objetivo de mejorar la predicción y suavizar el efecto de bordes.

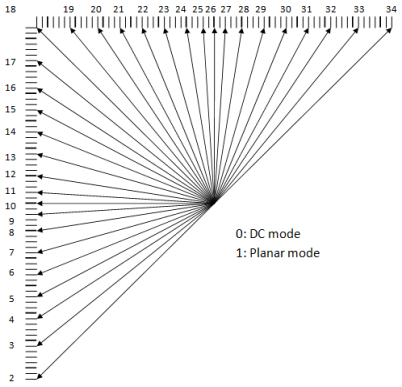


Figura.2. Modos de predicción Intra-frame en HEVC

Este procesado, unido al aumento de los tamaños de las unidades de predicción y del número de ángulos de predicción, permite al estándar obtener una alta eficiencia de codificación en el modo Intra-Frame, pero su complejidad computacional crece extraordinariamente, lo que exige la adopción de algoritmos rápidos de codificación.

III. IMPLEMENTACIÓN DEL RDO EN EL SOFTWARE DE REFERENCIA DE HEVC

Al igual que sus predecesores, HEVC aplica el algoritmo de RDO (Rate Distortion Optimization) con el objetivo de seleccionar el particionado óptimo de la CTU, así como el modo de predicción direccional para cada PU. El RDO está basado en la función de coste de Lagrange, mostrada en la ecuación (1), utilizando la distorsión (D) y tasa binaria (R) obtenida para cada modo de predicción (mode) y parámetros de cuantificación (QP).

$$J(mode) = D(mode, QP) + \lambda(Q_p) \cdot R(mode, QP) \quad (1)$$

La figura 3 muestra el orden de procesado de las CUs llevados a cabo por el RDO, en el que su aplicación exhaustiva implica el computar la función de coste para cada uno de los posibles tamaños de PU permitidos, y para la totalidad de modos de predicción definidos.

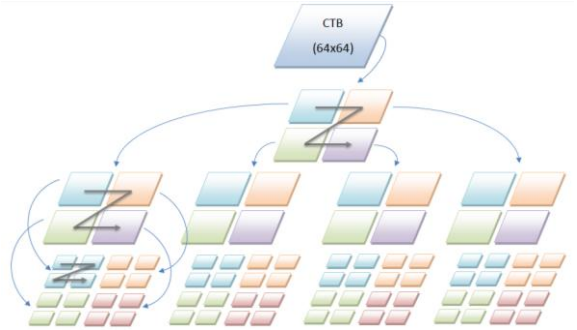


Figura. 3. Orden de procesado de las CU por el RDO.

Con el objetivo de analizar el incremento de complejidad del RDO en HEVC en relación con H.264/AVC, en la tabla I se muestra el número de predictores calculados por cada estándar, en la codificación de un CTU de 64x64 pixeles.

TABLA I

NÚMERO DE PREDICTORES INTRA-FRAME EN HEVC Y H.264

	H.264			HEVC		
CU	Bloques	Predic.	Total	Bloques	Predic.	Total
64x64	-	-	-	1	35	35
32x32	-	-	-	4	35	140
16x16	16	4	64	16	35	560
8x8	64	9	576	64	35	2240
4x4	256	9	2304	256	35	8960
TOTAL			**2944**			**11935**

Como se puede apreciar, un RDO exhaustivo en HEVC exige el cálculo de un número de predictores Intra-Frame cuatro veces superior al requerido en H.264/AVC. Esta elevada complejidad es la que ha motivado la implementación de un RDO rápido en el software de referencia de HEVC (HM) [16], estructurada en dos etapas, como se muestra en la figura 4.

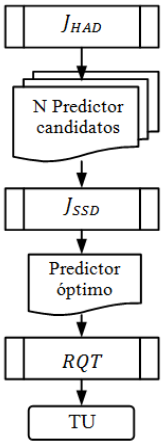


Figura. 4. Orden de procesado de las CU por el RDO.

En primer lugar se calcula una función de coste de baja complejidad (J_{HAD}) para los 35 modos de predicción, donde se utiliza como métrica de distorsión la suma absoluta sobre el residuo transformado, utilizando la transformada de Hadamard. Se seleccionan entre 3 y 8 candidatos en función del tamaño de la PU, a los que se les calcula la función de coste exhaustiva (J_{SSD}), aplicando el proceso completo de codificación con el objeto de conocer la tasa binaria real generada (R), y su

decodificación, con el objetivo de obtener la distorsión obtenida, medida como la diferencia entre el bloque original y el reconstruido por medio de la métrica SSD (Sum of Square Difference). El candidato con el menor valor de coste es seleccionado como el candidato óptimo final.

Por último, y con el objetivo de obtener el tamaño de transformación óptimo, se aplica el RQT sobre el modo de predicción seleccionado, evaluando los tres niveles de particionado de las TU permitido.

Como se puede apreciar, la implementación del software de referencia HM es cuasi-optima, al existir la posibilidad de que el predictor óptimo no sea seleccionado centre los N candidatos en la primera fase del RDO, y al no considerar todos los posibles tamaños de transformación en ninguna de la dos funciones de coste. Por consiguiente, una implementación exhaustiva del RDO podría obtener unas prestaciones de Tasa-Distorsión, superiores a las del software de referencia.

IV. ANÁLISIS DE COMPLEJIDAD DE LA PREDICCIÓN INTRA-FRAME EN HEVC

Vanne et al. presentan en [17] un estudio en el que se revela como, a pesar del algoritmo de aceleración del RDO implementación en el software de referencia de HEVC anteriormente descrito, su complejidad sigue siendo más de tres veces superior a la del software de referencia de H.264 (JM) [18]. Por este motivo, hemos llevado a cabo una serie de simulaciones con el objetivo de analizar las particularidades de la predicción Intra-Frame en HEVC.

En concreto, en los siguientes sub-apartados de esta sección, evaluamos la distribución de la carga computacional del RDO por tamaño de PU, llevaremos a cabo ciertas observaciones con respecto a las predicciones con tamaños de PU 64x64 y analizamos su el impacto en términos de caudal (rate) y distorsión. Por último, evaluaremos la última etapa del RDO, donde se aplica de modo exhaustivo el RQT sobre el modo óptimo seleccionado.

Todos los experimentos se han llevado a cabo con la versión 9 del software de referencia (HM9.0) [16], bajo las condiciones de configuración del codificador y el conjunto de secuencias test definidas por el JCT-VC para el modo “All-Intra” en su perfil “Main”, recogidos ambos en el documento [19]. La librería de pruebas se compone de 24 secuencias de distinta complejidad espacio-temporal, con resoluciones que cubren desde casi la UHDTV (2500x1600), hasta secuencias de muy baja resolución (416x240).

A. Distribución de la carga computacional del RDO por tamaño de PU

A pesar de que todas las PU son evaluadas para los 35 modos de predicción Intra-Frame, el número total de pixeles es el mismo, y que el modelo aplicado para su cálculo de los predictores direccionales es igual para todos ellos, su complejidad varía notablemente por factores como el tamaño de la transformada, o el número de bloques a procesar.

Para llevar a cabo este experimento hemos computado en el método del RDO, el tiempo de ejecución que consume en cada uno de los niveles del árbol mostrado en la figura 3, que determina el tamaño de sus PUs, cubriendo desde la CTU inicial de 64x64, hasta el menor tamaño de PU de 4x4 pixeles. Los resultados obtenidos como valor medio de las distintas simulaciones, se muestran en la figura 5, donde se puede apreciar como para el coste es inversamente proporcional al tamaño de las PU procesadas, y directamente proporcional al número de bloques a procesar. Por ejemplo, para el tamaño de 4x4 se procesan 256 PU, mientras que para 64x64 tan solo una.

Un dato a destacar es el bajo coste computacional de las PU de mayor tamaño, lo que permitiría aliviar el coste computacional del RDO si se dispusiera de un algoritmo que permitiera detectar de modo fiable las PU mayor tamaño, evitando la evaluación de PU de menor tamaño, que son las que realmente requieren un mayor tiempo de computo dentro del bucle del RDO.

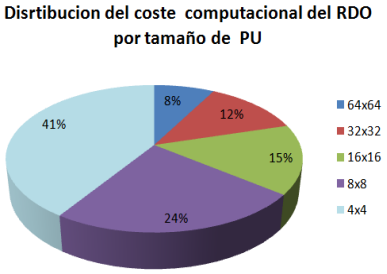


Figura. 5. Distribución del coste computacional del bucle de RDO por tamaño de PU.

Este hecho nos ha motivado a realizar un estudio en detalle de la predicción para tamaños de PU de 64x64, como se describe a continuación.

B. Análisis de eficiencia de predicción Intra-frame en tamaños de PU de 64x64

Como es bien sabido, en H.264/AVC los bloques homogéneos son codificados con tamaños de bloque de 16x16, mientras que los bloques de mayor detalle son codificados bloques de 4x4. Con el objetivo de analizar si la predicción Intra-Frame en HEVC sigue los mismos patrones de comportamiento que en H.264/AVC, se han analizado distintas imágenes, en las que se ha sobreimpuesto el tipo de partición optima seleccionada por el RDO, como puede verse en la figura 6.

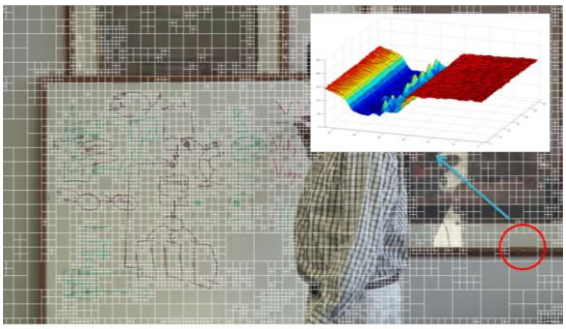


Figura. 6. Primer frame de la secuencia “Vidyo3” con la indicación del particionado optimo de PUs.

En dicha figura (figura 6), se muestra el primer frame de la secuencia "Vidyo3" codificada en modo exclusivamente Intra-Frame, donde se puede apreciar que el número de PUs de 64x64 es muy reducido, incluso existiendo grandes zonas homogéneas en la pizarra y la pared, donde el RDO ha obtenido como partición óptima las PUs de menor tamaño. De igual modo, se puede comprobar visualmente como alguna de las PUs de 64x64 no se corresponden con bloques planos, sino con bloques de cierta complejidad, mostrado en la PU marcada con un circulo y representada su luminancia en la zona 3D. Para dichos bloques parecería más coherente su procesamiento con tamaños menores, para ajustarse a los bordes que incluye el bloque.

Con el objetivo de analizar las prestaciones en términos de tasa binaria y distorsión, del predictor Intra-Frame para tamaños de PU de 64x64, hemos modificado el bucle del RDO anulando el procesamiento de dichas PU (Skip_64) y se han obtenido los resultados de su calidad (PSNR), tasa binaria y tiempo de codificación, con respecto al software de referencia original, para los cuatro QP definidos.

Como métrica, hemos utilizado la BD-Rate (Bjøntegaard Delta - Rate) definida en [19], y recomendada por el JCT-VC en [20], que permite conocer el ahorro o incremento en la tasa binaria entre dos curvas Rate-Distortion. Como convenio, hemos representado en la curva "a" los resultados obtenidos por el RDO original, y en la curva "b" los resultados obtenidos para la configuración "Skip_64", por lo que un valor negativo de BD-Rate (%) implica un ahorro en tasa binaria para el RDO original, para la misma calidad objetiva.

TABLA II

RESULTADOS DE BD-RATE Y TIEMPO DE CODIFICACIÓN PARA LOS MODOS DE CODIFICACIÓN FULL_RDO Y SKIP_64

		BD-Rate (%)			Tiempo (horas)	
Clase	Bloques	Y	U	V	RDO	Skip_64
A	Traffic	-0.1%	-0.6%	-0.9%	5.03	4.44
	People	0.2%	-0.5%	-0.9%	5.13	4.55
B	Kimono	-1.8%	-2.9%	-2.8%	3.63	3.22
	ParkScene	-0.1%	-0.9%	-1.0%	4.24	3.75
	Cactus	-0.3%	-0.1%	1.4%	8.90	7.86
	Basketball	-1.7%	-0.3%	0.0%	8.15	7.28
	BQTerrace	-0.1%	0.6%	1.6%	11.33	10.35
	TOTAL	**-0.4%**	**-0.6%**	**-0.5%**	46.41	41.45
	Tiempo de Codificación (%)				**89%**	

Como se puede apreciar en los resultados recogidos en la tabla II, evitando el modo de predicción de 64x64, la penalización en la tasa binaria es despreciable, 0.5%, mientras que el tiempo de codificación se reduce un 11%. De igual modo, se puede observar como para la secuencia "People" la eficiencia del RDO en modo exhaustivo es ligeramente peor que para el modo "Skip_64" (+0.2%). Como ya se había indicado en la sección III, el algoritmo rápido implementado en el software de referencia provocaba que sus prestaciones fueran sub-optimas.

En la figura 7 se muestran las curvas Rate-Distorsión para la mencionada secuencia "`PeopleOnStrreet", y en donde se puede ver la ligera ganancia del modo "Skip_64" para altas tasas binarias.

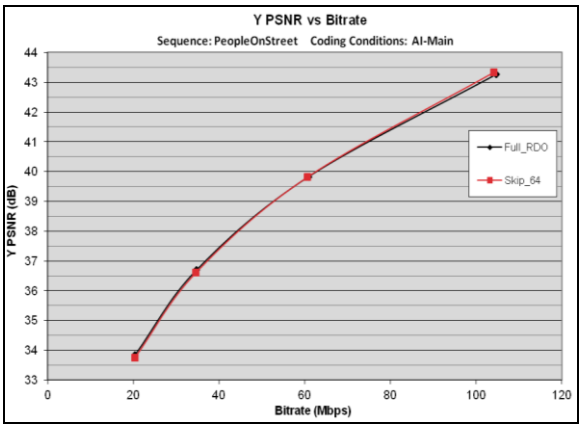


Figura. 7. Resultados Rate-Distorsión para los modos "Full_RDO" vs "Skip_64". Secuencia PeopleOnStreet.

C. Análisis de la eficiencia del RQT en la predicción Intra-frame

De la simple observación de los tamaños de transformada seleccionados como óptimos, en la última etapa del RDO, donde se evalúan para una PU de tamaño 2Nx2N sus tres posibles descomposiciones (tamaños 2Nx2N, NxN y N/2xN/2), se evidencia que, tan solo en ocasiones muy puntuales, el tamaño seleccionado es inferior al tamaño de la PU. Por ello, hemos llevado a cabo simulaciones bajo las condiciones descritas anteriormente, forzando al RQT a utilizar un solo nivel de profundidad (RQT-d1), esto es, no permitiendo al RDO que evalúe las particiones de la TU inferiores al tamaño de la PU. Nuevamente hemos recogido los resultados de las calidades obtenidas, tasa binaria y tiempo de codificación, para su comparación con el modo de RDO exhaustivo (Full_RDO), y se ha computado el BD-Rate, cuyos resultados se recogen en la tabla III.

Los resultados obtenidos ofrecen nuevamente una penalización residual en términos de tasa binaria, inferior al 1% para la componente de luminancia (0.7%), y una reducción de la complejidad computacional del 9%, confirmando que la eficiencia de aplicar la RQT a residuos procedentes de la predicción Intra-Frame no es relevante, y por el contrario exige de un incremento de computo que puede ser excesivo para ciertas aplicaciones.

TABLA III

NÚMERO DE PREDICTORES INTRA-FRAME EN HEVC Y H.264

		BD-Rate (%)			Tiempo (horas)	
Clase	Bloques	Y	U	V	RDO	RQT-d1
A	Traffic	-0.5%	-1.0%	-2.0%	5.03	4.58
	People	-0.4%	0.0%	-0.9%	5.13	4.67
B	Kimono	-2.2%	-3.4%	-3.6%	3.63	3.36
	ParkScene	-0.5%	-0.7%	-1.0%	4.24	3.84
	Cactus	-0.6%	-1.1%	-0.7%	8.90	8.05
	Basketball	-1.7%	-1.4%	-1.0%	8.15	7.42
	BQTerrace	-0.3%	1.3%	1.1%	11.33	10.23
	TOTAL	**-0.7%**	**-0.8%**	**-1.3%**	46.41	42.15
	Tiempo de Codificacion (%)				**91%**	

V. ALGORITMO DE ACELERACIÓN DE LA PREDICCIÓN INTRA-FRAME EN HEVC

Tomando como punto de partida los resultados de los análisis efectuados, y tomando en consideración los principios que se derivan de ellos, en la actualidad se está investigando el diseño de un algoritmo rápido que permita llevar a cabo una reducción de la complejidad computacional del RDO (figura 8), abordando el problema desde una doble perspectiva:

i. Detección rápida del particionado óptimo de la CTU en distintas PU, facilitando la supresión de la evaluación exhaustiva de todo los tamaños de predicción, y que afecta a las dos primeras fases de implementación del RDO.
ii. Prefiltrado del número inicial de modos direccionales de predicción para la primera etapa del RDO, por medio del cálculo del gradiente dominante de la CU a codificar.

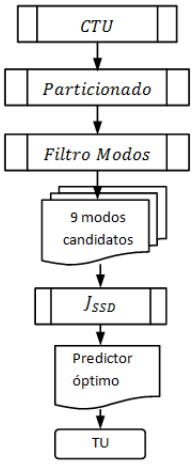


Figura. 8. Modelo propuesto para la aceleración de la predicción Intra-Frame en HEVC.

El actual estado de las investigaciones se centra en el desarrollo del algoritmo rápido de particionado óptimo, y se ha abordado la detección dinámica del primer nodo del RDO para la detección de PUs con tamaño 32x32 con bajo coste computacional. La figura 8 muestra la integración del modelo propuesto bajo desarrollo en el esquema del RDO, donde se suprime la evaluación de la predicción para 64x64 (Skip_64), particionado siempre la CTU en 4 sub-bloques de 32x32. Para cada una de dichas PUs, se computa un conjunto de atributos (F) tanto en el dominio espacial como en el frecuencial, entre ellos la varianza y el número de coeficientes no nulos de la transformada de Hadamard, y se determina si la PU es clasificada como 32x32 o debe evaluarse en tamaños inferiores (16x16, 8x8 y 4x4).

AGRADECIMIENTOS

Este trabajo ha sido financiado parcialmente por el proyecto MICINN y la Comisión Europea (fondos FEDER) en el seno del proyecto TIN2012-38341-C04-04.

REFERENCIAS

[1] New video codec to ease pressure on global Networks. [On-line]. http://www.itu.int/net/pressoffice/press_releases/2013/01.aspx#.USIMtaVJMuf.
[2] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, Y.-K. Wang, T. Wiegand, "High Efficiency Video Coding (HEVC) text specification draft 10 (for FDIS & Consent)", Doc. JCTVC-L1003_v10, Jan. 2013.
[3] F. De Simone et al., Towards high efficiency video coding: Subjective evaluation of potential coding technologies, J. Vis. Commun. (2011), doi:10.1016/j.jvcir.2011.01.008.
[4] Philippe Hanhart ; Martin Rerabek ; Francesca De Simone ; Touradj Ebrahimi; Subjective quality evaluation of the upcoming HEVC video compression standard. Proc. SPIE 8499, Applications of Digital Image Processing XXXV, 84990V (October 15, 2012).
[5] Ohm, J.; Sullivan, G.J.; Schwarz, H.; Thiow Keng Tan; Wiegand, T., "Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC)," Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1669,1684, Dec. 2012.
[6] B. Li, G. J. Sullivan, and J. Xu, "Comparison of Compression Performance of HEVC Working Draft 4 with AVC High Profile," Doc. JCTVC-G399, Nov. 2011.
[7] Tung Nguyen; Marpe, D., "Performance analysis of HEVC-based intra coding for still image compression," Picture Coding Symposium (PCS), 2012 , vol., no., pp.233,236, 7-9 May 2012.
[8] ITU-T Rec. T.800 and ISO/IEC 15444-1, "JPEG2000 Image Coding System: Core Coding System" (JPEG2000 Part 1), 2000.
[9] F. Dufaux, G. J. Sullivan, and T. Ebrahimi, "The JPEG XR image coding standard [Standards in a Nutshell]" IEEE Signal Processing Magazine, Vol. 26, Nov. 2009.
[10] Sullivan, G.J.; Ohm, J.; Woo-Jin Han; Wiegand, T.; Wiegand, T., "Overview of the High Efficiency Video Coding (HEVC) Standard," Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1649,1668, Dec. 2012.
[11] Correa, G.; Assuncao, P.; Agostini, L.; da Silva Cruz, L.A., "Performance and Computational Complexity Assessment of High-Efficiency Video Encoders," Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1899,1909, Dec. 2012.
[12] Han, W.-J.; Min, J.; Kim, I.-K.; Alshina, E.; Alshin, A.; Lee, T.; Chen, J.; Seregin, V.; Lee, S.; Hong, Y. M.; Cheon, M.-S.; Shlyakhov, N.; McCann, K.; Davies, T.; Park, J.-H.,"Improved Video Compression Efficiency Through Flexible Unit Representation and Corresponding Extension of Coding Tools," Circuits and Systems for Video Technology, IEEE Transactions on , vol.20, no.12, pp.1709-1720, Dec. 2010.
[13] K. Panusopone, X. Fang, L. Wang, "Efficient Transform Unit Representation," Doc. JCTVC-D250, Jan. 2011.
[14] J. Min, S. Lee, I. Kim, W.-J. Han, J. Lainema, and K. Ugur, Unification of the Directional Intra Prediction Methods in TMuC, JCTVC-B100, Geneva, Switzerland, Jul. 2010.
[15] Lainema, J.; Bossen, F.; Woo-Jin Han; Junghye Min; Ugur, K., "Intra Coding of the HEVC Standard," Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1792,1801, Dec. 2012.
[16] Joint Collaborative Team on Video Coding Reference Software, ver. HM 10.0 [Online]. Available: https://hevc.hhi.fraunhofer.de/.
[17] Vanne, J.; Viitanen, M.; Hamalainen, T.D.; Hallapuro, A., "Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs," Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1885,1898, Dec. 2012.
[18] Joint Video Team Reference Software, ver. JM 18.0 [Online]. Available: http://iphome.hhi.de/suehring/tml/.
[19] G. Bjøntegaard, "Calculation of average PSNR differences between RD-curves", ITU-T SG16 Q.6 Document, VCEG-M33, Austin, US, Apr. 2001.
[20] F. Bossen, Common Test Conditions and Software Reference Configurations, document JCTVC-J1100, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), San Jose, CA, Feb. 2012.

Incentivación del aprendizaje de la programación en las Ingenierías. Un caso práctico

J. Lladós[1], J. Mateo[1], F. Cores[1], JL. Lérida[1], F. Giné[1]

Resumen— **La dificultad en los procesos de abstracción tan necesarios en las asignaturas de Ingeniería y en especial aquellas con contenidos de programación, dificulta el aprendizaje de sus contenidos y produce falta de motivación en los alumnos. Técnicas como el aprendizaje basado en problemas o el aprendizaje colaborativo aplicadas a la programación de videojuegos o *bots* que se enfrentan en algún tipo de competición, se han mostrado útiles para facilitar el aprendizaje de la programación. No obstante, la implantación eficaz de estas técnicas en su sentido más completo, requiere de la utilización de infraestructuras y herramientas TIC especificas que faciliten tareas tales como la publicación de contenidos, la comunicación, la compartición de información, la supervisión del proceso de aprendizaje, etc. En el presente trabajo se presenta un caso real de implantación de metodologías ABP y aprendizaje colaborativo mediante el uso de una plataforma P2P especialmente diseñada para facilitar y supervisar su implantación. Se describen los servicios de que se ha dotado a la plataforma, la metodología utilizada en un caso real, así como los resultados de aprendizaje.**

Palabras clave— **Aprendizaje basado en problemas, aprendizaje colaborativo, plataformas P2P, Juegos en red.**

I. Introducción

El Espacio Europeo de Educación Superior (EEES) presenta un nuevo marco formativo centrado en el aprendizaje autónomo del alumno y el desarrollo de competencias. Este nuevo escenario requiere la implantación de nuevas metodologías de aprendizaje que permitan al alumno la integración natural de los conocimientos impartidos mediante la resolución de casos prácticos, el trabajo en equipo, la colaboración y cooperación en la búsqueda de soluciones, incorporando además mecanismos de autoevaluación que lo hagan partícipe y consciente de su propio proceso de aprendizaje.

En los últimos años se han implantado con éxito distintas experiencias que plantean al alumno la búsqueda de soluciones a un reto de Ingeniería, a resolver mediante el trabajo en equipo y colaborativo, y una puesta en común mediante competición amistosa la cual proporciona a sus participantes un alto grado de motivación y el feedback necesario para la mejora de sus soluciones. Algunos ejemplos de estas experiencias son los torneos de robótica [1], la programación de videojuegos [2], la programación de *bots* en entornos virtuales como *Robocode* [3] y otros [4][5].

En el presente trabajo presentamos un caso real de puesta en marcha de una de estas experiencias en la asignatura de "Programación Concurrente y Paralela" del Grado de Informática. El objetivo principal es que el alumno adquiera las competencias propias de la asignatura pero de modo natural, sintiéndose motivado, responsable y participe de su propio proceso de aprendizaje. Para ello, es necesario el uso conjunto tanto de metodologías docentes efectivas como de herramientas TIC que permitan al alumno la implantación de las soluciones y al profesor el seguimiento y supervisión del proceso de aprendizaje. Por ello, en este trabajo se describe con detalle tanto las metodologías docentes utilizadas como la plataforma y el servicio web implementados para dar soporte a esta implantación, así como los resultados de aprendizaje y las conclusiones a las que hemos llegado tras analizar nuestro caso de implantación.

El presente artículo se estructura en los siguientes apartados. Los conceptos previos sobre las tecnologías utilizadas como base en nuestro caso de implantación se describen en la Sección II. Las metodologías docentes aplicadas así como los requerimientos del servicio web para dar soporte a la implantación se detallan en la Sección III. En la Sección IV se muestran los resultados de aprendizaje y en la Sección V se describen las principales conclusiones del caso en estudio.

II. Conceptos previos

RoboDist es la plataforma utilizada para dar soporte a la implantación de nuevas metodologías docentes en el caso de estudio presentado en este trabajo. Esta plataforma presentada en [6], se compone por dos capas (Fig. 1): la primera integra una herramienta para la programación de *bots* y su puesta en común mediante competiciones amistosas denominada *Robocode* [3], juntamente con una arquitectura P2P de cómputo distribuido denominada *CoDiP2P* [7], que interacciona con la red y permite distribuir la ejecución de batallas entre los peers que integran el sistema y la segunda, la capa de datos que incluye la base de datos. Además se dispone de una lógica del juego, la cual ofrece una interfaz para poder interactuar con el servidor.

La finalidad de esta estructura es gestionar un juego distribuido a través de internet, ofreciendo la posibilidad de entablar múltiples partidas con otros jugadores mediante distintas instancias del juego de forma distribuida. Se aprovecha la capacidad de cómputo que ofrece *CoDiP2P* para resolver las batallas mediante la ejecución de tareas *Robocode*.

[1] Dpto. de Informática e Ingeniería Industrial, Universitat de Lleida. e-mail: jordi.llados@udl.cat, jmateo@alumnes.udl.cat, {fcores, jlerida, sisco}@diei.udl.cat

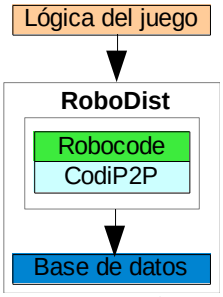


Fig. 1. Estructura de la plataforma *RoboDist*

A. *Plataforma RoboDist*

La ventaja más importante de la plataforma es la abstracción que ofrece, facilitando el poder desarrollar distintas lógicas de juego. Para lograrlo se ha establecido un modelo cliente – servidor con comunicación de sockets java vía TCP, mediante el cual el servidor ofrece un conjunto de servicios:

- Gestión de usuarios: Los estudiantes se conectan al sistema haciendo login contra el servidor de entrada.
- Gestión de salas: Una vez autenticados, los estudiantes pueden definir nuevas salas de juego o unirse a otras existentes para iniciar una partida.
- Iniciar batalla: Cuando un determinado número de estudiantes se encuentran reunidos en una sala, estos pueden iniciar una batalla.

Una vez configurada una batalla se asignan determinados roles a los peers (clientes) que forman parte de la partida. Un cliente puede actuar como Servidor de Partida (SP), Servidor Replicado de Partida (SRP) o bien ejecutar partidas como nodo de cómputo. Una vez establecidos los roles el SP gestiona la ejecución de la partida en la plataforma. Así pues, como se muestra en la Fig. 2, podemos identificar en la plataforma dos componentes claramente diferenciados: Servidor de Entrada y los Clientes (jugadores).

El servidor de entrada dota al sistema de una base de datos de jugadores que permite además almacenar su progreso, también actúa como punto de entrada al sistema P2P y gestiona las comunicaciones con los clientes de forma segura. A su vez el cliente se une como peer para ofrecer cómputo a las partidas en curso.

B. *Robocode*

Robocode es un juego de programación que tiene por objetivo principal despertar el interés de estudiantes y del público en general por la programación, la informática y los estudios de Ingeniería en general.

Este objetivo lo logra *Robocode* mediante un juego que simula combates entre *bots* virtuales, de modo que el jugador controla el comportamiento de su *bot* mediante un pequeño programa java. Este programa aplica algoritmos de diferentes campos (física, programación estructurada, inteligencia artificial, etc.) con el objeto de detectar y derrotar a los objetivos enemigos, así como realizar las acciones necesarias para poder esquivar/escapar de los ataques enemigos.

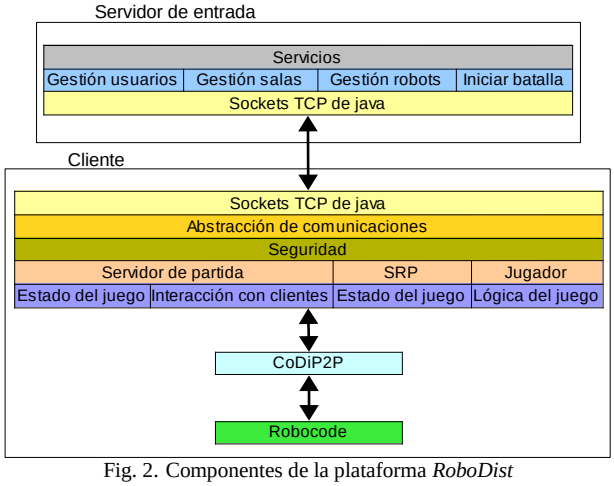


Fig. 2. Componentes de la plataforma *RoboDist*

En *Robocode*, los *bots* cuentan con una cuota de energía limitada (que puede utilizar o no) y un radar para detectar al enemigo. Dicha energía se reduce cada vez que se dispara, cuando se recibe un disparo o un golpe.

La Fig. 3 muestra un ejemplo de una batalla *Robocode* en donde intervienen ocho *bots*, con su correspondiente campo de visión y los disparos de los mismos.

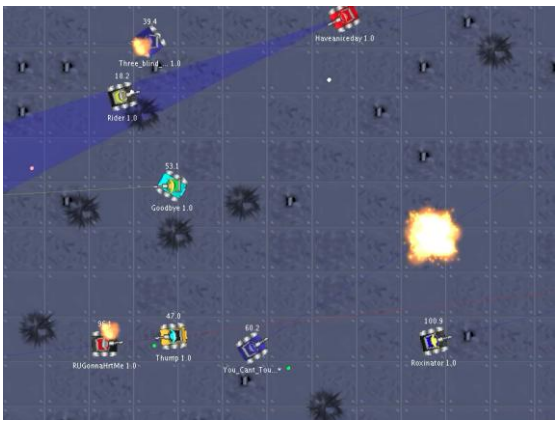


Fig. 3. Ejemplo ejecución de *Robocode*

C. *CoDiP2P*

CoDiP2P es una arquitectura P2P de cómputo distribuido implementada por el grupo GCD (Grupo de Computación Distribuida) de la Universidad de Lleida, junto con la empresa Indra.

La topología de este entorno es jerárquica, en forma de árbol *B-ario*, donde cada nodo tiene *B* hijos. Cabe destacar que su topología se divide en niveles y a su vez, cada nivel en áreas, que son agrupaciones lógicas de peers.

Cada peer puede realizar tres tipos de roles: *Manager*, *Worker y Master*. El rol *Manager* controla y gestiona a los peers conectados inmediatamente en el nivel inferior mediante un subárbol. El rol de *Worker* es el encargado de ejecutar tareas y enviar información sobre el estado de sus recursos a su Manager. Se denomina *Master* al peer que lanza una aplicación al sistema y que se encarga de gestionar su ejecución.

Un *área* es un espacio lógico formado por un *Manager* más sus *Workers*. Un área puede ejecutar localmente aplicaciones intensivas de cómputo y si es necesario distribuir las tareas que no puede asumir entre las áreas vecinas.

Cabe destacar que la manera de usar la plataforma es completamente transparente al usuario, solo hay que enviar contra *CoDiP2P* los cálculos que se quieran realizar. La plataforma automáticamente mandará a una serie de *Workers* que procesen estas tareas y cuando se hayan completado, la plataforma devolverá el resultado al máster solicitante.

III. Metodología docente

En el proceso de adaptación a un aprendizaje basado en competencias, se ha dotado a las asignaturas en las titulaciones de Grado y Máster en Ingeniería Informática de una componente práctica muy importante, con el objetivo de facilitar la adquisición de estos conocimientos por parte del estudiante. Gracias a ello el estudiante se convierte en participante activo en su propio proceso de aprendizaje, utilizando técnicas como la resolución de casos prácticos, fomentando el trabajo en equipo, la colaboración y cooperación entre ellos. Existen diversas metodologías de aprendizaje activo que permiten articular este propósito, entre las que podemos destacar el trabajo en equipo [4] (*Team Based Learning*), el aprendizaje basado en problemas (*Problem Based Learning*) [5] y el aprendizaje basado en Proyectos [8].

A pesar del esfuerzo realizado en la aplicación de las nuevas metodologías, éstas no son efectivas al cien por cien a la hora de solventar los problemas de desmotivación asociados con el aprendizaje y la puesta en práctica de conocimientos con un elevado grado de abstracción. En el caso de las titulaciones de Informática, estos problemas se están empezando a poner de manifiesto principalmente en el aprendizaje de las competencias de programación, tanto en los niveles básicos (fundamentos de programación) como en niveles más avanzados (programación paralela y concurrente, ingeniería del software, etc.). El nexo común entre todas las asignaturas que ejercitan esta competencia, es el alto grado de desmotivación de los estudiantes a la hora de aplicar dichos conocimientos.

Para revertir esta tendencia, planteamos integrar las estrategias clásicas basadas en el trabajo en equipo y la resolución de un proyecto o reto común, en un entorno lúdico/didáctico que motive en los estudiantes en la adquisición de dichos conocimientos. De esta forma, nos hemos centrado en encontrar una plataforma que combine el aprendizaje de la programación junto con un aspecto más lúdico que sirva de incentivo al estudiante. Nos interesa un entorno que permita una sana competencia entre los estudiantes y que fomenten el espíritu de superación y la reflexión crítica sobre su progreso y conocimientos adquiridos y el trabajo colaborativo y cooperativo. Algunos casos de éxito en este sentido son los torneos de programación de robots Lego [1], la programación de videojuegos [2], o incluso la programación de comportamientos dentro de entornos de competición virtual como el caso de *Robocode* y otras experiencias [3][9].

En nuestro caso de estudio nos hemos decantado por la utilización de *Robocode* como plataforma de desarrollo y testeo de las soluciones de programación. Esta plataforma permite definir distintos objetivos a alcanzar con múltiples grados de dificultad, así como testear de forma rápida, sencilla y visual las distintas soluciones aportadas. El alumno puede contrastar sus soluciones con otras ofrecidas por Internet o con las de otros compañeros, lo que facilita la interacción con el resto de compañeros, la compartición de información y fomenta la colaboración tanto a nivel de grupo como entre los distintos grupos de trabajo. Todo esto hace al alumno más participe, más consciente de su evolución, lo motiva en el aprendizaje de los objetivos y además fomenta su espíritu de superación y el interés por el aprendizaje. Por último, la plataforma se puede adaptar los objetivos pedagógicos de diferentes asignaturas, la colaboración entre múltiples asignaturas del mismo nivel o de diferentes niveles de forma transversal. Este aspecto es fundamental para fomentar la cohesión y la consolidación de los conocimientos y potenciando al mismo tiempo el trabajo de competencias transversales dentro de la propia titulación [8].

IV. Implantación Metodología

La metodología propuesta se ha desplegado en la asignatura "Programación Concurrente y Paralela" de 3er curso del grado de Informática. Esta asignatura tiene por objetivo introducir los conceptos y las técnicas de programación concurrente y paralela.

En concreto, se ha propuesto a los estudiantes la realización de una actividad práctica basada en la plataforma *RoboDist*. El objetivo es que los estudiantes puedan aplicar de forma didáctica y amena las técnicas y métodos vistos en las clases de teoría mediante la programación de *bots* virtuales.

RoboCode soporta diferentes variantes de competición: individual (*bot* vs. *bot*), mele (todos contra todos) y en equipo (*N* equipos de *M bots* luchan entre ellos). En la experiencia presentada en este estudio se ha escogido la modalidad por equipos dado que la coordinación de las acciones entre los *bots* de un equipo requiere de la implantación de mecanismos de comunicación mediante paso de mensajes y el uso de primitivas de sincronización.

Teniendo en cuenta la temática de la asignatura y las características de la plataforma utilizada esta actividad se ha enfocado hacia la adquisición de las siguientes competencias:

- **Programación dirigida por eventos**. El modelo de programación utilizado por *RoboCode* se basa en el paradigma de programación dirigido por eventos. En *RoboCode* la estructura y el flujo de ejecución del programa está determinado por una serie de eventos o mensajes generados por el propio sistema u otros programas/robots. El estudiante necesita definir una serie de gestores de eventos, los cuales se dispararán cuando un determinado evento ocurre.
- **Comunicación mediante paso de mensajes**. El objetivo es que apliquen los métodos de comunicación punto a punto y broadcast de *RoboCode*, para el intercambio de información entre los robots. La utilización efectiva de esta información permitirá el incremento de la inteligencia de los robots y un mejor desempeño en el campo de batalla.
- **Sincronización de procesos**. Los estudiantes deben utilizar los mecanismos de sincronización

vistos en teoría para coordinar las acciones que tiene que realizar cada uno de los robots del equipo.

A. Planificación de la actividad

La planificación de la actividad debe permitir la implicación del estudiante, de forma que una vez se haya captado su interés, utilicemos dicha dinámica para obtener los máximos beneficios pedagógicos. Por ello el desarrollo y evaluación de los robots se ha dividido en dos etapas. Una primera etapa que sirva como punto de contacto, para que la mayoría de los estudiantes puedan alcanzar los requisitos mínimos y una segunda que tiene que permitir a los estudiantes profundizar en su aprendizaje y la mejora de sus soluciones, a partir del feedback obtenido mediante el intercambio de información y el contraste de soluciones con el resto de compañeros. Es por ello, que la actividad no se completa hasta que los estudiantes hayan presentado y debatido su diseño e implementación con el resto de sus compañeros.

Como consecuencia la planificación de la actividad consta de los siguientes hitos:

1. Presentación de la actividad.
2. Definición de los equipos.
3. Desarrollo de la primera versión del equipo de los *bots*.
4. Torneo de validación.
5. Refinamiento del equipo de los *bots*.
6. Torneo de evaluación.
7. Publicación resultados, debate y discusión sobre los diseños de los *bots*.
8. Evaluación.

B. Procedimiento de Evaluación

A la hora de plantear la evaluación se ha buscado tres objetivos principales: 1) fomentar el espíritu de esfuerzo y superación, 2) incentivar el interés por la mejora de la calidad de las soluciones y 3) garantizar que todo estudiante que cumpla con unos criterios mínimos pueda superar la actividad. Estos criterios pueden resultar contradictorios entre si, al menos que se defina cuidadosamente el mecanismo de evaluación.

Para fomentar el espíritu de superación y la mejora de la calidad de las soluciones, un porcentaje importante de la evaluación se obtiene a partir de un torneo entre todos los equipos de estudiantes. Cada equipo se enfrentará al resto de compañeros y como resultado se obtendrá una clasificación en función del número de combates ganados por cada uno de ellos y de otros parámetros que modelarán la efectividad del diseño/desarrollo realizado. En una primera instancia, a partir de esta clasificación se distribuyen las notas de forma proporcional a la posición que ocupa el equipo. El ganador absoluto obtendría un 10 y a partir de ahí, el 5% mejor clasificados un 9.5, los siguientes 5% un 9 y así sucesivamente. Este modelo de competición garantiza la competitividad entre los estudiantes y la búsqueda de la excelencia.

Sin embargo, este esquema de puntuación tiene una desventaja importante, se trata de una medida de calificación relativa. Solo permite definir si un estudiante lo ha hecho mejor o peor que otro, pero no valora el grado de cumplimiento de los objetivos pedagógicos definidos en la actividad. Otro inconveniente es que dicho esquema garantiza de forma estricta un 45% de suspensos.

Por lo tanto, es necesario definir unos requisitos mínimos que todos los estudiantes deben cumplir para poder superar la actividad. Al mismo tiempo se debe garantizar que todos aquellos estudiantes que cumplan dichos criterios mínimos van a poder superar la actividad independientemente de la posición que ocupen en la clasificación. Para lograr este objetivo, se ha implementado un equipo de robots de referencia que en su programación incorpora los conocimientos y competencias mínimas que debe adquirir los estudiantes. Este equipo define el umbral entre superar o suspender la actividad. Independientemente de su posición en la clasificación del torneo, todos los equipos que logren vencer al equipo de referencia, automáticamente tienen aprobada la actividad y los que no logren vencerlo la tienen suspendida. Se ha utilizado la misma aproximación para definir la nota máxima, proporcionando un equipo establezca los criterios de excelencia. Todos aquellos estudiantes, cuyos robots superen en la liguilla a este robot, obtendrán la máxima nota. El archivo *jar* de ambos robots (*BasicTeam* y *ProfeTeam*) es proporcionado junto con los materiales de la actividad y los estudiantes lo pueden utilizar para valorar su progresión.

C. Herramientas de evaluación y seguimiento

Con el fin de que los estudiantes puedan probar sus soluciones y compartirlas con el resto de equipos, se ha implementado un portal web (Fig. 4) que ofrece al estudiante la posibilidad de subir su solución, realizar competiciones con *bots* de otros equipos, así como el acceso a un histórico de batallas, que posibilita a los alumnos ver la progresión y evolución de sus *bots*.

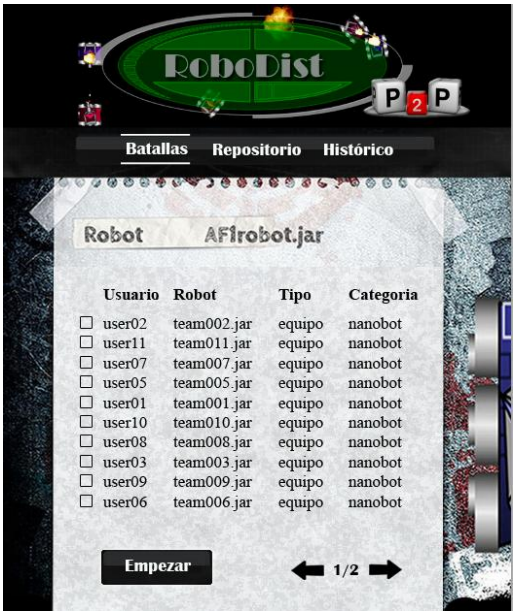


Fig. 4. Interfaz del Web –Service.

El portal web ofrece acceso a un conjunto de servicios implementados mediante una arquitectura SOA. Esta arquitectura define una capa intermedia entre la plataforma *RoboDist* descrita en la Fig. 1 y la capa de presentación con la que interactúa el usuario. Las

ventajas de la adopción de este diseño son principalmente la reusabilidad y flexibilidad del sistema.

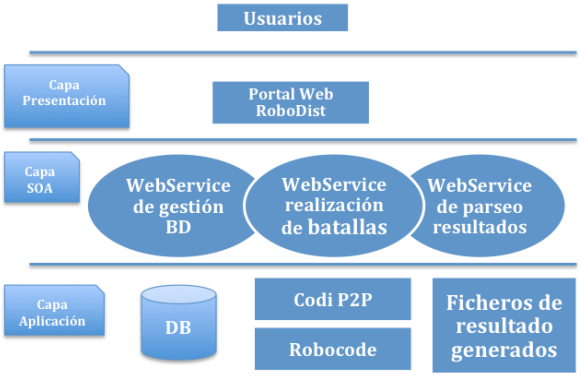


Fig. 5. Arquitectura SOA Web-Service

En la Fig. 5 se muestra la estructura de capas de que se compone la herramienta final y concretamente en la capa SOA se describen el conjunto de servicios principales que se proporcionan al usuario profesor o alumno. En la primera capa encontramos el Portal web encargado de facilitar el acceso del usuario a los servicios implementados según su rol (alumno o profesor). Este portal presenta una interfaz gráfica visualmente atractiva y usable que utiliza la potencia de las tecnologías más avanzadas de diseño web.

Con el fin de garantizar la seguridad entre los diferentes roles se ha diseñado un módulo de autentificación utilizando los protocolos de seguridad TLS i SSL que proporciona acceso remoto al espacio personal de cada usuario y funcionalidades propias de cada rol. En el caso de los alumnos, este módulo ofrece servicios para subir sus *bots*, probar sus *bots* contra otros, así como observar su progreso. En cuanto a los profesores, estos pueden acceder a información relativa al histórico de partidas de cada alumno, resumen de los resultados por equipo, estadísticas de utilización del servicio por usuario, creación de torneos, etc.

Las principales funcionalidades del perfil alumno están relacionadas con la realización de batallas y la visualización de resultados. Para ello, se hace uso y se implementa un módulo de gestión de base de datos, a través del cual donde el alumno puede registrar sus bots, además de las pruebas realizadas y los resultados obtenidos. El alumno puede obtener información sobre los *bots* disponibles de otros usuarios o bien seleccionar con que *bots* quiere entrenar. Además podrá consultar un resumen de sus resultados y las partidas realizadas para obtener feedback y analizar las mejoras. Todo esto se ofrece mediante una interfaz sencilla que da acceso a toda esta información. Una futura implementación en este portal es el acceso a contenidos de la asignatura y permitir el intercambio de mensajes entre usuarios.

El profesorado tiene acceso a la información sobre los resultados de los equipos en las distintas batallas, así como a las estadísticas de utilización de los servicios por parte de los alumnos. De esta forma el profesor puede hacer un seguimiento del proceso de aprendizaje y progreso de los alumnos. También se puede crear distintas modalidades de torneos, donde intervengan todos o parte de los equipos de de robots.

Con el fin de utilizar toda la información que nos proporciona Robocode sobre las distintas partidas y teniendo en mano la potencia de las soluciones SOA, se ha implementado un servició web que parsea y transforma la información de los ficheros de combate en tablas y gráficos permitiendo al usuario tener una idea rápida sobre todo lo sucedido. Este servició web es utilizado en la implementación de la sección de resultados e histórico de nuestro portal. Los resultados de estos torneos quedan almacenados en la BD. De esta forma se pueden recuperar posteriormente tanto las tablas de resultados resumen como el combate gráfico. Esto permite compartir los resultados con los alumnos y servir de base para una posterior evaluación.

V. Resultados

Inicialmente se estableció una fecha límite para que los alumnos subieran su última versión del robot. Gracias a la plataforma web, los alumnos podían subir sus robots e ir haciendo batallas de prueba contra sus oponentes para ver su evolución y finalmente lanzar el torneo de validación/evaluación.

Realizar un torneo es un proceso sencillo pero costoso en cuando a cómputo. Un torneo entre *N* robots, implica realizar *N* combinaciones sin repetición de dos elementos, es decir, lanzar $(N \cdot (N-1))/2$ tareas de *Robocode*. Además cada tarea implica realizar 10 rondas, reduciendo así el efecto aleatorio introducido por el posicionamiento en el campo de batalla. Aquí es donde la plataforma *RoboDist* ofrece las mayores ventajas, siendo capaz de gestionar un gran número de batallas, repartir la carga de ejecución entre distintos peers y reducir enormemente el tiempo de ejecución de los torneos.

En nuestro caso de estudio, el torneo implantado se compone de 13 *bots*, esto implica que se deben realizar 78 batallas distintas para cubrir todas las posibilidades. El tiempo medio de una batalla (10 rondas) es aproximadamente de 15 segundos. Teniendo en cuenta estos datos la Tabla 1 muestra el efecto de la inclusión de más peers en el tiempo total del torneo. El sistema genera una cola con las batallas, con lo cual cuando una batalla se termina entra la siguiente.

TABLA I
TIEMPO DE EJECUCIÓN DEL TORNEO

#Peers	#Task	#Time avg.
1	78	1173s
2	78	597s
3	78	402s
4	78	329s
6	78	221s

Como se puede observar el tiempo de ejecución es aproximadamente 5 veces menor con la utilización de 6 peers en el sistema, que con uno solo.

A. Resultados Evaluación

Inicialmente se realizo un torneo de validación, con el cual los alumnos recibieron feedback para poder refinar sus *bots* y así estar preparados para el torneo de evaluación.

En la Tabla 2 se muestra el resultado del torneo de evaluación. Se muestra el nombre del *bot* (team), las victorias/derrotas (Wins/Losses) obtenidas y la nota

final de la práctica (Score). También se puede observar la posición de los *bots* usados como umbrales ***SCPBasicCommTeam_0.0.jar*** y ***SCPProfeTeam_1.0.jar*** para establecer las notas mínimas y máximas de los alumnos.

TABLA II
RESULTADOS TORNEO DE EVALUACIÓN

Team	Wins	Losses	Mark
Team003.jar	11	2	10
Team007.jar	11	2	10
SCPProfeTeam_1.0.jar	***11***	***2***	
Team005.jar	10	3	9
Team004.jar	9	4	8
Team011.jar	8	5	7
Team008.jar	6	7	5
Team001.jar	5	8	5
SCPBasicCommTeam_0.0.jar	***7***	***6***	
Team010.jar	3	10	3
Team009.jar	2	11	2
Team002.jar	2	11	2
Team006.jar	0	13	0

B. *Resultados Encuestas*

Al finalizar la actividad se realizaron un conjunto de encuestas para conocer la opinión de los estudiantes ante la metodología utilizada.

La Tabla 3 muestra las preguntas realizadas a los alumnos en el proceso de encuesta.

TABLA III
ENCUESTA DE EVALUACIÓN

Nº	Cuestión
1	*Consideras que la metodología aplicada en la última práctica de esta asignatura:*
1.1	¿Permite una mejor comprensión de los conceptos explicados?
1.2	¿Mejora tu motivación para la realización de la práctica?
1.3	¿Incrementa tu dedicación?
1.4	¿Mejora tu motivación a la hora de afrontar la solución de la práctica?
1.5	¿Te ayuda a alcanzar más eficazmente los objetivos de aprendizaje?
2	*Crees que sería útil utilizar/aplicar esta metodología:*
2.1	¿En otras prácticas de la asignatura?
2.2	¿En otras asignaturas de la titulación?
3	*Solo en cuanto a la última práctica realizada:*
3.1	¿Estás de acuerdo con el método de evaluación utilizado?
3.2	¿Crees que esta metodología ayuda a mejorar el trabajo en grupo?
3.3	¿Crees que esta metodología ayuda a mejorar la interacción entre los grupos?
3.4	¿Crees que esta metodología ayuda a mejorar la calidad de las soluciones?

En la Fig. 6 se muestran los boxplots con el resultado de las respuestas a cada pregunta, siendo el rango de 1 a 5, donde 1 indica totalmente en desacuerdo y un 5 totalmente de acuerdo.

En general, se observa como los valores se concentran entre [3 - 4] y con bastantes tendiendo a 5. Esto implica que los alumnos en general han quedado satisfechos con la actividad propuesta. En concreto, las respuestas a las preguntas 1.2, 1.3 y 1.4 denotan que se ha logrado el objetivo de motivar a los estudiantes, incrementando su dedicación a la actividad e incentivándoles para lograr una mejor solución. Los estudiantes también han coincidido en que les gustaría utilizar esta metodología en otras asignaturas de la titulación.

Por último, se puede corroborar con los resultados de la pregunta 3.2, que el factor competitivo ha preponderado sobre el trabajo en grupo. Sin embargo, al mismo tiempo, se constata que dicha competencia no disuade que los grupos puedan interactuar entre ellos y compartir información. Este es un aspecto, se puede mejorar de cara a los siguientes cursos, posiblemente permitiendo que los grupos compartan el diseñó (no la implementación) de algunas de sus soluciones.

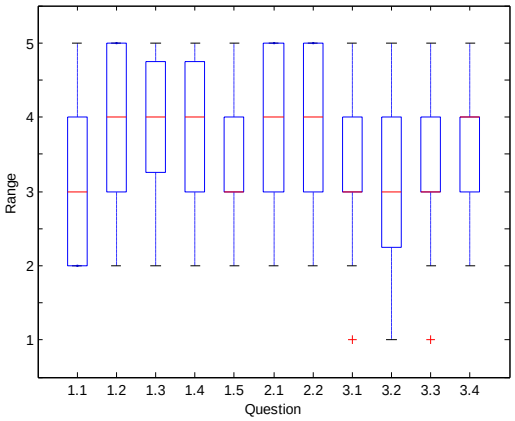


Fig. 6. Boxplot de valoración de las respuestas

VI. CONCLUSIONES

En el presente artículo se describe una metodología basada en la plataforma en Red *RoboDist*, cuyo objetivo es incentivar las competencias de programación entre los estudiantes de Ingeniería. Dicha metodología se ha implantado en una asignatura de 3º del grado de informática.

Los resultados, tanto cuantitativos como cualitativos, han demostrado que dicha metodología logra incrementar la motivación de los estudiantes en las actividades de programación.

AGRADECIMIENTOS

El presente trabajo ha sido financiado mediante el proyecto CICYT TIN2011-28689-C02-02 y el proyecto de innovación docente de la UdL PID-0821.

REFERENCIAS

[1] Cliburn, D.C., "Experiences with the LEGO Mindstorms throughout the Undergraduate Computer Science Curriculum", 36th Annual Frontiers in Education Conference, pages 1-6, 2006.
[2] Kelvin Sung, "Computer games and traditional CS courses", Communications ACM, 52 (12), pages 74-78, 2009.
[3] Jackie O'Kelly and J. Paul Gibson. "RoboCode & problem-based learning: a non-prescriptive approach to teaching programming", SIGCSE Bull. 38 (3), 217-221, 2006.
[4] L. K. Michaelsen and M. Sweet, "The essential elements of team-based learning", New Directions for Teaching and Learning, pages 7-27, 2008.
[5] Esko Nuutila, Seppo Törmä & Lauri Malmi, "PBL and Computer Programming - The Seven Steps Method with Adaptations", Computer Science Education 15:2, pages 123-142, 2005.
[6] Lladós J., Arroyo I., Cores F., Lerida JL., Gine F.. "RoboDist: Una plataforma de juego P2P para incentivar la programación en los grados de Ingeniería". XXIII Jornadas de Paralelismo 2012, pp: 501-506, 2012.
[7] Solé R., Acín J.I., Agraz A., Garcia M., Josa I., Sentís J.M., Cores F., "Computación Distribuida de altas prestaciones en entornos Peer-to-Peer", XXI Jornadas Paralelismo, 2010.
[8] Samuel B. Fee & Amanda M. Holland-Minkley, "Teaching computer science through problems, not solutions", Computer Science Education 20:2, pages 129-144, 2010.
[9] Chen-Chung Liu, Yuan-Bang Cheng, Chia-Wen Huang, "The effect of simulation games on the learning of computational problem solving" Computers & Education 57:3, pages 1907-1918, 2011.

Paralelización del Algoritmo de Descomposición Cluster Benders

Jordi Mateo[1], Jordi Lladós[1], Josep Ll. Lérida[1], Lluís M. Plà[2], Francesc Solsona[1]

Resumen—**El método Benders es un método de descomposición que se puede utilizar para resolver los problemas lineales estocásticos multietapa mediante el análisis de escenarios. El algoritmo se compone de tres pasos: el primero resuelve el problema master, el segundo obtiene los cortes de factibilidad y el tercero los cortes de optimalidad. Estos cortes se añaden secuencialmente al problema master hasta que el procedimiento iterativo llega a la solución óptima. Dado que los cortes de factibilidad se obtienen mediante la resolución de subproblemas por cada escenario, independientes entre ellos, ésto facilita la paralelización del algoritmo. El algoritmo CBD (Cluster Benders Decomposition) es una modificación del de Benders original, dónde se obtienen los cortes de factibilidad mediante la agrupación de escenarios permitiendo ajustar la carga computacional al tamaño del subproblema. En este trabajo se presenta la paralelización del algoritmo CBD. Los resultados muestran como la paralelización del algoritmo CBD obtiene ganancias significativas de rendimiento con respecto a CBD y Benders**.

Palabras clave—**Problema Lineal Estocástico de dos Etapas, Descomposición Cluster Benders, Paralelización.**

I. Introducción

Los problemas lineales estocásticos de dos etapas proporcionan un marco adecuado para el modelado de problemas complejos de decisión en condiciones de incertidumbre aplicables en diversos campos. Además de las variables de primera etapa que, en un contexto de incertidumbre representan las decisiones tomadas aquí y ahora, el modelo tiene en cuenta las decisiones de segunda etapa (wait and see), a saber, que se pueden tomar una vez observados los parámetros aleatorios de una realización concreta. Para más información de los modelos de programación estocásticos de dos etapas y de los métodos de resolución basados en el análisis de escenarios, ver [5] y [6].

Muchas aplicaciones requieren un gran número de escenarios aumentando la complejidad de los modelos. Debido a ello, los métodos que ignoran la estructura particular de los programas lineales estocásticos pueden llegar a ser bastante ineficientes, incluso irresolubles. El método de descomposición llamado Dantzig-Wolfe ([7]) resuelve la dualidad del problema. Otra descomposición, conocida como el método de Benders [1], resuelve el problema primal.

El método Benders ha sido ampliamente utilizado en programación estocástica en la generación de cortes de factibilidad, entre el que se encuentra el conocido método "*L-shaped*", cuyo algoritmo denominaremos en este artículo mediante TBD (Traditional Benders Decomposition) [3].

Cuando el número de escenarios es finito y el problema es de recurso completo, el método de descomposición de Benders converge a una solución óptima en un número finito de iteraciones. El método *L-shaped* se compone de tres pasos: el primero resuelve el llamado problema master (principal), el segundo busca los cortes de factibilidad y, por último, el tercer paso obtiene los cortes de optimalidad.

La estructura del método *L-shaped* permite modificar fácilmente la fase de generación de cortes. Este método consiste en añadir al problema master los cortes de factibilidad y optimalidad que se van obteniendo en el segundo y tercer paso. El método permite agrupar las iteraciones en un cluster de escenarios, obteniendo cortes más precisos.

Un cluster de escenarios se define como un subconjunto de escenarios en los que se consideran las restricciones de No-AntiCipatividad (NAC) [9]. Aranburu et al. propuso en [3] una descomposición en clusters de escenarios para obtener cortes de factibilidad más restrictivos aplicados al método de descomposición de Benders para la resolución de problemas con variables continuas. El método se denomina "*Cluster Benders Decomposition*" (CBD). CBD es un método de descomposición de agrupación de escenarios en clusters para solucionar el problema de factibilidad del método Benders aplicado a problemas estocásticos lineales de dos etapas a gran escala. El método está basado en la modificación de la descomposición de Benders con el fin de generar cortes factibles agrupando recursos computacionales en clusters. De esta manera, los clusters de recursos computacionales pueden ser adaptados inteligentemente a diferentes tamaños de submodelos con el fin de lograr una mayor eficiencia computacional en la resolución de problemas complejos. Se comparó el rendimiento entre CBD y la descomposición de Benders variando el número de clusters así como el número de escenarios por cluster. CBD obtuvo mejores resultados tanto para problemas de media como gran escala, pero no para tamaños pequeños de cluster. La carga computacional añadida, mayoritariamente concentrada en la inicialización, debe ser compensada con un mayor rendimiento del método CBD.

Como los cortes de factibilidad y optimalidad se obtienen mediante la resolución de submodelos auxiliares derivados de los escenarios y éstos son independientes entre ellos, la paralelización de este paso del algoritmo TBD es inmediato. Además de la paralelización del algoritmo CBD de la segunda y tercera etapa (en la obtención de los cortes de factibilidad y optimalidad), en el presente trabajo se propone la adaptación de los tamaños de los clusters de

[1]Dept. d'Informàtica i Enginyeria Industrial, Av. Jaume II 69, 25001 Lleida, Universidad de Lleida (UdL), España. e-mails: {jmateo@alumnes.udl.cat, jordi.llados@udl.cat, jlerida@diei.udl.cat, francesc@diei.udl.cat}.
[2] Dept. Matmàtica, Av. Jaume II 73, 25001 Lleida, Universidad de Lleida (UdL), España. e-mail: lmpla@matematica.udl.cat

los submodelos de acuerdo con los recursos computacionales del host en donde se mapea, lo que aumenta la eficiencia computacional en la resolución del problema, y muy especialmente de los complejos.

Los resultados obtenidos muestran como la paralelización del algoritmo CBD siempre obtiene mejores resultados, excepto para el caso en que los problemas son lo suficientemente pequeños, en donde la sobrecarga computacional, así como la sincronización y comunicación de la aplicación paralela no compensan las ganancias. Esto demuestra la bondad de nuestra propuesta.

El resto del trabajo se organiza de la siguiente manera. La Sección II describe el método de Benders así como el algoritmo "*Traditional Benders Decomposition*" (TBD). En la siguiente sección, la III, se explica el método "*Cluster Benders Decomposition*" (CBD), basado en la clusterización de escenarios. En la Sección IV presentamos PCBD, nuestra propuesta de paralelización del método CBD. En la Sección V se comparan y analizan los resultados obtenidos mediante los tres métodos objeto de estudio de este trabajo, TBD, CBD y PCBD. En las secciones VI y VII se presentan las conclusiones y el trabajo futuro.

II. Descomposición de Benders

Los problemas lineales bietapa "*Two-Stage Linear Problems*" (*TSLP*) se pueden representar matemáticamente mediante la forma compacta siguiente:

$$\begin{aligned} \text{(TSLP)} \quad & min \;\; c^T x + \textstyle\sum_{k=1}^{K} p_k q_k^T y_k \\ & \text{s.t.:} \;\; Ax = b \\ & \quad T_k x + W_k y_k = h_k \qquad \forall k \in \Omega \\ & \quad x, y_k \geq 0 \end{aligned}$$

donde x es un vector de dimensión n_x, el cual representa las variables de primera etapa, e y es un vector de dimensión n_y, de variables de segunda etapa del escenario $k \in \Omega$ (conjunto de escenarios a considerar), c es un vector conocido de los coeficientes de la función objetivo de las variables de la primera etapa, b es un vector de escalares resultado de la restricción de primera etapa, A es la matriz de constantes de la misma restricción, p_k es la probabilidad del escenario k, h es el vector de escalares resultado de la restricción de segunda etapa, q_k^T es el vector de coeficientes de las variables de segunda etapa de la función objetivo, y finalmente T_k y W_k son respectivamente la matriz tecnológica y de recursos (o restricciones) del escenario k, $\forall k \in \Omega$. Además, $|\Omega|$ representa el número de escenarios.

La **Fig. 1** muestra un ejemplo de un árbol de $|\Omega|$=8 escenarios. Según el principio de No-AntiCipatividad (NAC) [9], para todos los escenarios, las variables de la primera etapa deben tener el mismo valor en el problema bietapa. La **Fig. 1** representa implícitamente el principio NAC en el modelo compacto del problema lineal bietapa (TSLP).

Un problema lineal bietapa (TSLP) se puede descomponer en un problema master-worker. Esto permite obtener la solución óptima de forma iterativa en el programa master. Este método requiere la obtención de los cortes de factibilidad y optimalidad del problema de optimización de dos subproblemas auxiliares/workers [4], denominados subproblemas de Factibilidad (**FWorker**) y Optimalidad (**OWorker**). A medida que se van obteniendo los cortes de factibilidad y optimalidad se van añadiendo al programa master de forma iterativa.

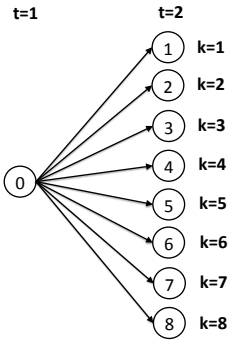


Fig. 1. Árbol de escenarios. Representación Compacta.

El método "Traditional Benders Decomposition" (TBD), presentado en [5] es el algoritmo de descomposición principal en el que se basa el presente trabajo para obtener la solución final del problema lineal bietapa (TSLP).

A. Algoritmo TBD

El Algoritmo TBD (*L-shaped*) está formado por tres pasos: el primero resuelve el problema master, el segundo obtiene los cortes de factibilidad y finalmente, el tercero es el responsable de encontrar los cortes de optimalidad. Los cortes se van añadiendo iterativamente al problema master hasta que el Master encuentra la solución óptima.

Paso 0. Set $s = t = v = 0$.

Paso 1. Set $v = v + 1$. Resolver el programa lineal Master:

$$\begin{aligned} \textbf{(Master)} \quad & min \;\; c^T x + \theta \\ & \text{s.t.:} \;\; Ax = b & (1) \\ & \quad D_l x \geq d_l \qquad l = 1..s & (2) \\ & \quad E_i x + \theta \geq e_i \quad i = 1..t & (3) \\ & \quad x \geq 0;\; \theta \in R \end{aligned}$$

Sea (x^v, θ^v) una solución óptima. Si no existe la restricción (3), θ se actualiza a $-\infty$ y se ignora.

Paso 2. Para todo escenario $k = 1..|\Omega|$, se resuelve el subproblema lineal "*Factibilidad Worker*" (**FWorker**) hasta que, para todas las k, el valor óptimo $w = 0$:

$$\begin{aligned} \textbf{(FWorker)} \quad & min \;\; w = ev^+ + ev^- \\ & \text{s.t.:} \;\; W_k y_k + Iv^+ + Iv^- = h_k - T_k x^v \\ & \quad y_k, v^+, v^- \geq 0 \end{aligned}$$

donde $e = (1,..,1)$. Sean σ^s los multiplicadores simplex asociados, obtenidos de la previa resolución del problema **FWorker**. Definimos $D_{s+1} = \sigma^s T_k$ y $d_{s+1} = \sigma^s h_k$ para generar un corte de factibilidad del tipo (2). Actualizar $s = s + 1$ y volver al ***Paso 1***. Si $\forall k \in \Omega$, $w = 0$, ir al ***Paso 3***.

Paso 3. Para todos los escenarios $k = 1..|\Omega|$, resolver el subproblema lineal "*Optimalidad Worker*" (**OWorker**):

(**OWorker**) $min\ w = q_k y$
s.t.: $Wy = h_k - T_k x^v$
$y \geq 0$

Sea π_k^t los multiplicadores simplex asociados a una solución óptima del problema **OWorker**. Definimos $E_{t+1} = \sum_{k=1}^{K} p_k \pi_k^t T_k$ y $e_{t+1} = \sum_{k=1}^{K} p_k \pi_k^t h_k$. Sea $w = e_{t+1} - E_{t+1} x^v$. Si $\theta^v \geq w$ stop, x^v es una solución óptima. Sino, añadir corte de optimalidad del tipo (3), establecer $t = t + 1$ y volver al ***Paso 1***.

III. ALGORITMO CBD

El algoritmo de descomposición CBD se basa en el anteriormente presentado algoritmo TBD. En Aramburu et al. [3] se propuso la partición de escenarios en clusters de escenarios. Definimos Ω^p como el conjunto de escenarios del cluster p, para $p = 1,..,\hat{p}$.

La **Fig. 2** muestra un ejemplo con tres particiones de clusters distintas del ejemplo de la **Fig. 1**. Cada árbol representa explícitamente el principio NAC (i.e., imponiendo la igualdad) en las variables de primera etapa x^p, esto es, $x^p = x^{p'}$ para $p, p' = 1,..,\hat{p}$, siendo $\hat{p}$ el número de clusters. En el árbol de la izquierda hay dos clusters, $\Omega^1 = \{1,2,3,4\}$ y $\Omega^2 = \{5,6,7,8\}$. En el árbol del medio hay cuatro clusters de escenarios, $\Omega^1 = \{1,2\}$, $\Omega^2 = \{3,4\}$, $\Omega^3 = \{5,6\}$ y $\Omega^4 = \{7,8\}$. Finalmente, el árbol de la derecha está formado por ocho clusters $\Omega^1 = \{1\}$, $\Omega^2 = \{2\}$, $\Omega^3 = \{3\}$, $\Omega^4 = \{4\}$, $\Omega^5 = \{5\}$, $\Omega^6 = \{6\}$, $\Omega^7 = \{7\}$ y $\Omega^8 = \{8\}$.

Cabe resaltar que el árbol de la derecha corresponde a la representación de la versión puramente paralela de un problema lineal bietapa (TSLP). Por simplicidad y sin pérdida de generalidad, seleccionamos el número de clusters de escenarios, $\hat{p}$, como un divisor del número de escenarios $|\Omega| = 8$.

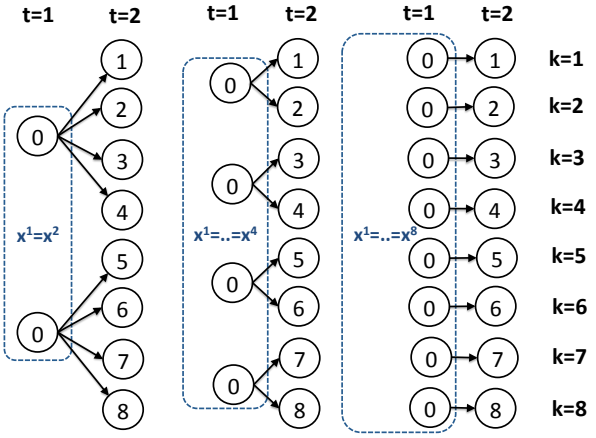


Fig. 2. Ejemplos de particionado de clusters.

Los coeficientes de la función objetivo **FWorker**, en el ***Paso 2*** del algoritmo TBD no dependen de ningún escenario en particular, por lo que podemos considerar un conjunto de escenarios en lugar de un solo escenario.

Básicamente, el algoritmo CBD consiste en modificar el **FWorker** de TBD con el fin de procesar clusters de escenarios en cada iteración, en lugar de sólo uno de los escenarios.

La función objetivo **FWorker** depende del conjunto de variables artificiales $v+$, $v-$, luego, este modelo de factibilidad para un grupo de escenarios puede formularse como un problema de minimización de las variables v^{+k}, v^{-k}, para $k \in \Omega^p$.

Luego podemos redefinir el modelo de cluster de escenarios de factibilidad, "Feasibility Worker Cluster Benders" (FWCBD) como:

(**FWCBD**) $min \quad w = ev^{+k} + ev^{-k}$
s.t.: $W_k y_k + Iv^+ + Iv^- = h_k - T_k x^v \ \forall k \in \Omega^p$
$y_k, v^{+k}, v^{-k} \geq 0$

Por lo tanto TBD debe cambiar en consecuencia. El ***Paso 2*** del Algoritmo CBD se define como (***Paso 1*** y ***Paso 3*** no cambian):

Paso 2. Para el cluster escenario $p = 1..\hat{p}$, resolver el programa Feasibility Worker Cluster Benders (FWCBD) hasta que, para alguna k, el valor óptimo $w > 0$:

(**FWCBD**) $min \quad w = ev^{+k} + ev^{-k}$
s.t.: $W_k y_k + Iv^+ + Iv^- = h_k - T_k x^v \ \forall k \in \Omega^p$
$y_k, v^{+k}, v^{-k} \geq 0$

donde $e = (1, \ldots, 1)$. Sea σ^s los multiplicadores asociados. Definimos $D_{s+1} = \sigma^s T_p$ y $d_{s+1} = \sigma^s h_p$ para generar un corte de factibilidad del tipo (2). Luego se incrementa s en una unidad ($s = s + 1$) y se regresa al ***Paso 1***. Si $\forall k \in \Omega^p$, $w = 0$, ir al ***Paso 3***.

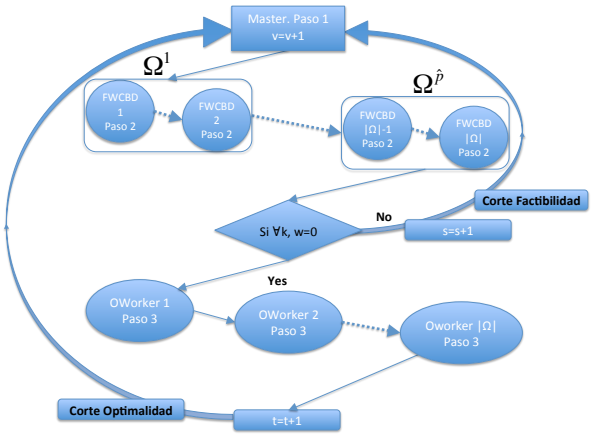


Fig. 3. Diagrama de flujo de CBD.

La **Fig. 3** muestra el diagrama de flujo del algoritmo CBD. Se puede apreciar como las ejecuciones de los diferentes clusters p, en el rango $p = 1,..,\hat{p}$ se ejecutan de forma serie. Además, en el ***Paso 2***, todos los $k = |\Omega^p|$ escenarios de cada cluster de escenarios p, esto es $\forall k \in \Omega^p$, se ejecutan también de forma serie. La diferencia con el algoritmo TBD radica en que en este último no se agrupan los escenarios en clusters en el ***Paso 2***.

IV. PCBD. PARALLEL CLUSTER BENDERS DECOMPOSITION

En esta sección, presentamos una versión paralela del algoritmo CBD, explicado en la Sección III,denominado PCBD (Parallel Cluster Benders Decomposition).

Aprovechando la estructura del algoritmo CBD, la versión paralela de dicho algoritmo, PCBD, también sigue el paradigma master-worker. Esto es, PCBD también se compone de dos partes: el proceso master y los procesos workers.

El proceso master comienza realizando el ***Paso 1*** del algoritmo CBD. Después, el master ordena a los procesos worker que inicien la ejecución (***Paso 2***). El master esperará entoces la recepción de una señal de sincronización (i.e. mensaje) enviada por cada uno de los worker.

Los workers se encargan de la ejecución del ***Paso 2*** mientras $\exists k \in \Omega, w \neq 0$. Durante este bucle se van generando los cortes de factibilidad y se van añadiendo al master. Existe 2 posibles arquitecturas alternativas para desplegar de forma concurrente el ***Paso 2***.

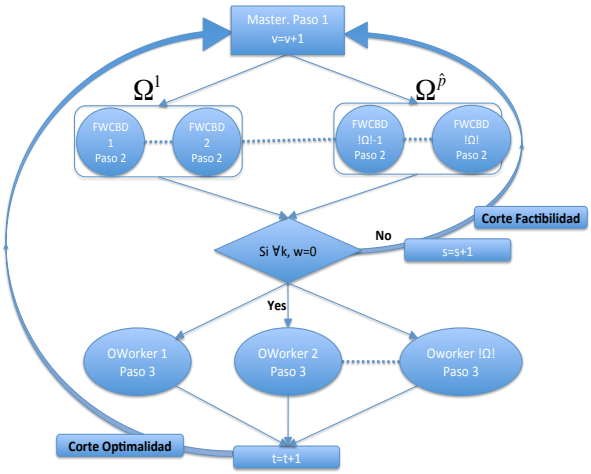


Fig. 4. Diagrama de flujo de PCBD.

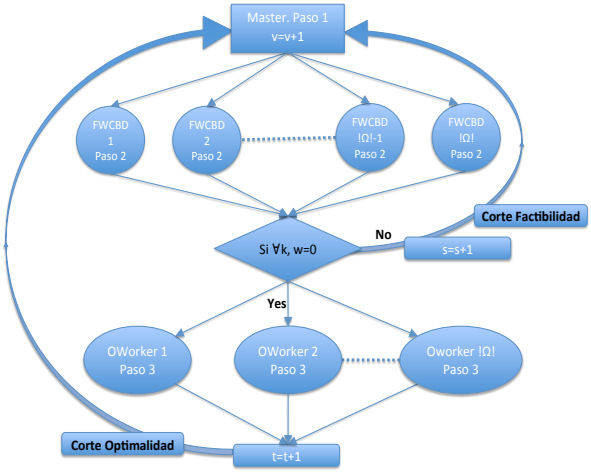


Fig. 5. Diagrama de flujo alternativo de PCBD

Tal y como se puede ver en las Figuras **Fig. 4** y **Fig. 5**, hay muchos escenarios posibles ya que existe un gran número y tamaño de clusters de escenarios, pudiendo variar desde un solo cluster de $|\Omega|$ escenarios hasta $|\Omega|$ clusters con un solo escenario. Finalmente, cuando $\forall k \in \Omega, w = 0$, el ***Paso 3*** es ejecutado. Este paso, a diferencia de CBD y TBD, se realiza también en paralelo. Se crea un worker por cada escenario, que a su vez ejecuta el Paso 3 en busca de cortes de optimalidad (tipo 3). Finalmente, una vez obtenidos todos los cortes de optimalidad, al igual que en el algoritmo CBD, si $\theta^v \geq w$ stop, x^v es una solución óptima, añadir todos los cortes de optimalidad del tipo (3), establecer $t = t + 1$ y volver al ***Paso 1***.

En el diseño del algoritmo PCBD, la cuestión clave es la identificación de las partes del código que corresponden al master y cuáles a los worker y qué política seguirá su creación/destrucción. Además, se debería desarrollar políticas de asignación y planificación del proceso master así como de los procesos worker. Esto también depende del modelo de memoria, el cual determina el paradigma de sincronización/comunicación.

Se han realizado dos implementaciones diferentes del algoritmo PCBD, la versión de memorica compartida (C-PCBD) y la versión distribuida (D-PCBD). Las diferencias entre las dos versiones está en el modelo de memoria utilizado.

A. PCBD de memoria compartida (C-PCBD)

En un sistema de memoria compartida, todos los threads/procesos se ejecutan en la misma máquina. Las aplicaciones paralelas utilizan memoria compartida como paradigma de comunicación/sincronización entre threads/procesos que componen las aplicaciones paralelas. En nuestro caso utilizaremos threads por ser más eficientes en nuestro propósito (básicamente son más ligeros) que los procesos.

Estamos interesados en conseguir el número apropiado de threads que deben crearse y ser ejecutados en paralelo, de forma que se obtenga un tiempo de ejecución óptimo. Concretamente, estamos interesados en entender la relación entre los recursos hardware disponibles (principalmente memoria principal y threads físicos) y el número de threads lógicos que podemos crear a la vez con el fin de identificar la relación $R_{optimal}$ entre las dos cantidades que optimice el tiempo de ejecución de la aplicación paralela.

Obteniendo esta relación nos permitirá saber los límites de escalabilidad de la aplicación en una máquina concreta. En nuestro caso, el escalado de C-PCBD depende además de muchos factores: el sistema operativo, las características del servidor, la propia aplicación C-PCBD y la política de planificación de los threads.

No existe un método sistemático para definir $R_{optimal}$. Siempre existirá un margen de mejora que depende de las características propias del entorno en que se ejecute la aplicación paralela.

B. PCBD de memoria distribuida (D-PCBD)

En este caso, la memoria está distribuida entre los nodos que componen el sistema. La aplicación utiliza el paso de mensajes como paradigma de comunicación y sincronización entre procesos o threads, ubicados en diferentes nodos de un entorno de memoria distribuida en red, como por ejemplo un sistema Cluster/Grid. Uno de los protocolos de comunicación y sincronización comúnmente utilizados para implementar aplicaciones paralelas en entornos distribuidos es MPI[1] (Message

[1] MPI. http://www.mpi-forum.org/

Passing Interface). Uno de los modelos de programación soportados por MPI es el denominado SMPD (Simple Program Multiple Data), en el que cada núcleo ejecuta el mismo programa. En nuestro caso, hemos elegido este modelo distribuido de programación para implementar D-PCBD en un entorno distribuido del tipo Cluster, donde la red de interconexión entre los nodos es una LAN (Local Area Network).

MPI es un modelo API (Appication Programming Interface) estándar de programación paralela en sistemas de memoria distribuida entre procesos/threads. En nuestro caso, se ha escogido el proceso como unidad de ejecución y nuestro estudio se ha centrado en los mecanismos utilizados en la comunicación y sincronización entre procesos.

En este caso, la relación $R_{optimal}$ depende de la relación entre los nodos disponibles (threads físicos por nodo) y el número de procesos lógicos a ser distribuidos a través del cluster. Al igual que en el caso C-PCBD, el escalado de D-PCBD depende de muchos otros factores: la capacidad de cálculo de los nodos que forman el Cluster, bandwidth y latencia de la red de interconexión (LAN), política de scheduling .

A diferencia del caso C-PCBD, donde simplemente escribiendo el nombre del programa es suficiente para ejecutar dicho programa, en un sistema de memoria distribuida (i.e. cluster), el procedimiento para hacerlo es un poco más complejo. En primer lugar, el trabajo se envía al sistema de colas que se encuentra en el (nodo) front-end, y que es responsable de la recepción, planificación y ejecución de trabajos en el cluster. El sistema de colas (en nuestro caso OGE, Oracle Grid Engine [2]) selecciona el trabajo D-PCBD y asigna el master y los workers a los nodos que forman el cluster mediante una política de planificación previamente especificada. La política de scheduling establecida en el presente trabajo es Round Robin, es decir, el planificador mapea tareas en los nodos del cluster de forma circular.

Al igual que en el caso de C-PCBD, no existe un método sistemático para definir $R_{optimal}$, y por lo tanto, la propuesta original puede mejorarse de acuerdo a ajustes específicos del entorno en donde se ejecute D-PCBD.

V. Experimentación

Para realizar la experimentación se han utilizado 2 tipos de arquitecturas hardware distintos, uno de memoria compartida y el otro de memoria distribuida. Además, para poder demostrar que nuestra propuesta escala convenientemente, en cada sistema se han utilizado máquinas con capacidades computacionales diferentes.

En el sistema de memoria compartida se han utilizado 4 máquinas con diferentes prestaciones, OS2 (OpenStack con 2 cores), OS4 (OpenStack con 4 cores) y OS12 (OpenStack con 12 cores) y Turing. OS12, OS4 y OS2 son máquinas virtual de la plataforma de código abierto OpenStack [11] (entorno para la creación y gestión de clouds), que constan respectivamente de 16, 4 y 2 cores de 64 bits a 2.1 GHz y 80 GB de Memoria principal. Turing consta de 4 cores de 64 bits a 3.1 GHz y 16 GB de Memoria.

[2] OGE. Oracle Grid Engine. http://www.oracle.com

En el sistema de memoria distribuida se ha utilizado dos clusters, Loras y Maracas. Loras está compuesto de 16 nodos con 2 cores de 32 bits a 3 GHz y 1GB de Memoria RAM. Maracas consta de 24 nodos con 4 cores de 64 bits a 2.4 GHz y 4 GB de Memoria.

El benchmark utilizado para realizar las comparativas entre las diferentes arquitecturas y algoritmos es P1 [3]. Su elección ha facilitado la comparación de rendimiento entre TBD, CBD y PCBD. Nuestra propuesta ha sido implementada en C++. Se ha utilizado el solver de libre distribución lp_solve 5.5 [10].

La TABLA I muestra las dimensiones de la representación compacta del problema P1. El significado de los acrónimos de las cabeceras es el siguiente: *nc*, número de restricciones obtenidas mediante la ecuación $(m+n_y)\Omega$; n_x número de variables de la primera etapa; n_y número de variables de la segunda etapa; *nel* número de coeficientes no nulos de la matriz Π, definida más abajo; *dens* densidad de la matriz de restricciones; y $|\Omega|$ indica el número de escenarios.

$$\Pi = \begin{pmatrix} W_0 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & W_{|\Omega|} \end{pmatrix}$$

Donde $W_k, k = 1..|\Omega|$, son las matrices de restricciones.

TABLA I

PARÁMETROS DEL BENCHMARK P1.

nc	*nv*	n_x	n_y	*nel*	*dens*	$\|\Omega\|$
1200	660	60	60	72,600	0.0916	10

En la TABLA II se muestran los resultados obtenidos en la ejecución de P1 mediante la utilización del solver lp_solve en los diferentes entornos. A continuación se describen los acrónimos utilizados. S: ejecución Serie, Xc: ejecución de X escenarios por cluster; MC: ejecución del modelo de memoria compartida C-PCBD, mediante threads. MD ejecución del modelo D-PCBD en un cluster utilizando MPI.

TABLA II

TIEMPO DE RETORNO DEL BENCHMARK P1.

		CBD				
	S	1c	5c	10c	MC	MD
OS12	143.7	145.8	586.2	998.2	16.2	12.6
OS4	146.2	146.5	582.6	998.3	52.3	46.8
OS2	143.2	143.2	594.9	988	58.6	48.2
Turing	107.8	108.3	417.6	698.8	10.52	10.1
Loras	136.2	137	569.3	947.0	13.8	16.9
Maracas	110.6	112.1	442.7	569.3	11.2	15.3

En la TABLA II se puede observar como las implementaciones paralelas MC y MD obtienen mejor rendimiento en todos los casos. Podemos decir que en media se ha obtenido unas ganancias que han alcanzado en algunos casos hasta el 91% con respecto a la implementación serie.

Los resultados obtenidos en le método CBD, para el caso 1c, prueban su correcta implementación. Como puede observarse, los valores son ligeramente superiores a los obtenidos en la versión serie. Sus variaciones son debidas al tiempo adicional de arranque de las aplicaciones CBD. Para los casos 5c y 10c en cambio, se observa un gran incremento en los tiempos debido a

la forma de implementar y resolver el ***Paso 2*** del algoritmo. Ello es debido a que la complejidad del subproblema del ***Paso 2*** (**FWorker**) incrementa de forma exponencial con la cantidad de escenarios por cluster.

El peor rendimiento para el caso MC que se observa entre las máquinas virtualizadas (OS2, OS4, OS12) con respecto a las reales (Turing, Loras, Maracas) es debido a su menor frecuencia de procesamiento.

Se puede apreciar también el buen comportamiento del entorno MPI. Aunque el nivel de paralelización es a nivel de proceso, la correcta distribución de los procesos entre los nodos y sus eficientes librerías de comunicación/sincronización hacen que incluso mejore la versión con memoria compartida.

Sin embargo, en entornos cluster, la eficiencia de MPI no es suficiente para contrarrestar las latencias de comunicación entre los nodos de la LAN. Es por este motivo que se obtienen peores resultados para el caso de memoria distribuida.

VI. Conclusiones

En este trabajo se ha presentado PCBD (Parallel Cluster Benders Decomposition), algoritmo de paralelización del algoritmo CBD (Cluster Benders Decomposition), que propone la agrupación en clusters de escenarios en la búsqueda de los cortes de factibilidad del método de Benders, aplicado en el algoritmo de descomposición TBD. Nuestra propuesta no solo se ha centrado en optimizar la obtención de los cortes de factibilidad mediante su paralelización, sino que también se ha paralelizado la obtención de los cortes de optimalidad. Se han propuesto dos alternativas, una basada en entornos de memoria compartida, C-PCBD, y la otra en un entorno Cluster de memoria distribuida, D-PCBD.

Las dos propuestas de paralelización han mejorado tanto el rendimiento de TBD como de CBD. La diferencia de rendimiento entre las dos propuestas, C-PCBD y D-PCBD, depende de las características intrínsecas de los entornos utilizados.

Serian necesarios realizar muchos mas experimentos para encontrar $R_{optimal}$. En todo caso, se ha comprobado que el rendimiento depende en gran medida de la aplicación, así como del entorno utilizado y por consiguiente es muy difícil obtener una fórmula cerrada de $R_{optimal}$.

VII. Trabajo Futuro

Actualmente estamos diseñando nuevos algoritmos de planificación para asignar de forma más eficiente las tareas entre los threads físicos de los sistemas de memoria compartida. A continuación, vamos a diseñar un planificador con el mismo propósito pero pensado para sistemas distribuidos, i.e. clusters.

Es necesario realizar una experimentación más exhaustiva para generalizar aún más nuestras propuestas y poder encontrar fórmulas más cerradas que determinen con mayor precisión el ratio $R_{optimal}$.

Los retos futuros irán en la dirección de ampliar el método de paralelización para contemplar problemas lineales multietapa. Además se quiere construir un servidor web para dar servicio a la comunidad científica en la resolución de problemas lineales estocásticos utilizando nuestros modelos paralelos. Esto será posible siempre y cuando se consigan recursos de cómputo adicionales (sistemas multithreading, cluster o grid) para poder garantizar niveles adecuados de calidad de servicio.

Además se quiere ampliar la experimentación utilizando un solver privado y ampliamente utilizado, CPLEX.

Agradecimientos

El presente trabajo ha sido financiado por el MICINN-Spain, mediante el contrato TIN2011-28689-C02-02.

Referencias

[1] J. Benders. Partitioning procedures for solving mixed variables programming problems. Numerische Mathematik 4:238-252. 1962.

[2] J.R. Birge. Decomposition and partitioning methods for multi-stage stochastic linear programs. Operations Research 33:989-1007. 1985.

[3] Larraitz Aranburu, Laureano Escudero, Araceli Garín and Gloria Pérez. A so-called Cluster Benders Decomposition approach for solving two-stage stochastic linear problems. TOP doi:10.1007/s11750-0242-4. 2012.

[4] R. Van Slyke and R. J-B Wets. L-shaped linear programs with applications to optimal control and stochastic programming. SIAM Journal on Applied Mathematics 17:638-663. 1969.

[5] J. R. Birge and F. Louveaux. Introduction to Stochastic Programming. Springer, New York. 1997.

[6] P. Kall and S. W. Wallace. Stochastic Program-ming. John Wiley and Sons. 1994.

[7] G.B. Dantzig; P. Wolfe. Decomposition Principle for Linear Programs. Operations Research 8:101–111. 1960.

[8] R. Van Slyke and R. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. SIAM Journal on Applied Mathematics 17:639-663. 1969.

[9] S.D. Falm. Nonanticipativity in stochastic programming. Journal of Optimization Theory and Applications 46(1): 23-30. 1985

[10] lp_solve 5.5. lpslve.sourceforge.net/5.5. 2013.

[11] OpenStack. http://www.openstack.org/. 2013.

[12] CPLEX. http://www-01.ibm.com/software/ commerce/optimization/cplex-optimizer/. 2013.

Aspectos computacionales en la bisección de un n-simplex regular

G. Aparicio, [1] L.G. Casado, [2] I. García, E.M.T. Hendrix [3] y B. G.-Tóth [4]

Resumen— **En el ámbito de la optimización global basada en técnicas de ramificación y acotación, cuando el espacio de búsqueda es un n-simplex regular es habitual utilizar como regla de división la bisección por el lado mayor. Este modo de división evita que los subproblemas generados tengan una forma degenerada o poco redondeada y además da lugar a un muestreo más uniforme del espacio de búsqueda ya que la función objetivo es normalmente evaluada en los vértices de los sub-problemas o símplices. En este trabajo se muestra como la división por el lado mayor puede afectar a parámetros tales como el número total de sub-problemas generados, el número de formas similares que estos pueden tener o el grado de redondez de los sub-problemas. La dificultad de determinar estos parámetros se incrementa con el valor de n. En este trabajo se presentan los resultados de los estudios realizados para $n \leq 3$, es decir, hasta un espacio 4-dimensional.**

Debido al crecimiento exponencial del árbol binario de búsqueda generado, se hace necesario el uso de computación paralela cuando se usan criterios de terminación más precisos y/o n-símplices con $n \geq 3$. Aquí se presenta un modelo paralelo que hace uso de las posibilidades de paralelización de MATLAB.

Palabras clave— **Simplex, Lado más largo, Bisección, Formas Similares, Paralelismo.**

I. Introducción

LOS algoritmos de ramificación y acotación (del inglés Branch-and-Bound, B&B) se aplican a la resolución de problemas de optimización global mediante la realización de una búsqueda exhaustiva del mínimo de una función objetivo en un dominio dado. En este trabajo estamos interesados en problemas donde el espacio de búsqueda está definido por un n-simplex regular, definido en un espacio $(n+1)$-dimensional [1]. Los algoritmos de B&B están caracterizados por las reglas de Acotación, Selección, División, Rechazo y Terminación. Se pretende estudiar las características del árbol de búsqueda generado cuando solo se tienen en cuenta las reglas de División y Terminación, es decir, ningún nodo del árbol es eliminado. Se usará como regla de división la bisección del lado mayor (BLM) [2], [3] y como regla de terminación la longitud del lado mayor de un subproblema o nodo del árbol.

En este trabajo investigamos el efecto de la BLM sobre el número de símplices generados, el número de formas similares de los subsímplices y su nivel de redondez para un determinado valor de la regla de terminación. El objetivo de estos estudios es la determinación del limite superior del número de símplices que se pueden generar a partir de uno dado, lo que permitirá estimar el trabajo pendiente a partir de un nodo del árbol de búsqueda. Esta información permitiría mejorar los métodos de balanceo dinámico de la carga en los correspondientes algoritmos paralelos de optimización global [4], [5].

En este articulo se presentan resultados para un 1-simplex, un 2-simplex y un 3-simplex. Debido al crecimiento exponencial del árbol binario, se hace necesario el uso de computación paralela cuando se usan criterios de terminación más precisos y/o n-símplices con $n \geq 3$. Por ejemplo, solo se han podido alcanzar 20 niveles del árbol de búsqueda para un 3-simplex usando un algoritmo secuencial en un computador de sobremesa.

En la Sección II se describen los Algoritmos de B&B utilizados para solucionar los problemas de Optimización Global que generan este tipo de árboles de búsqueda. En la Sección III se muestran las características específicas de los Algoritmos de B&B que realizan la búsqueda en un dominio definido por un simplex. En la Sección IV se presentan las métricas utilizadas para valorar las características del árbol que se genera, que son evaluadas en la Sección V. En la Sección VI se plantea el diseño de una versión paralela que haga uso de la ToolBox de computación paralela disponible en MATLAB. Finalmente en la Sección VII se muestran las conclusiones y trabajos futuros.

II. Optimización Global y los Algoritmos de B&B

La resolución de un problema de optimización global consiste en encontrar el valor mínimo o máximo de una función objetivo f, satisfaciendo ciertas restricciones en el espacio de búsqueda. Para problemas de minimización, cuando la solución está contenida en un simplex regular S, el problema de optimización global puede escribirse como:

$$f^* = \min_{x \in S} f(x), \text{ con } f(x^*) = f^*. \quad (1)$$

Cuando se requiere una búsqueda exhaustiva del mínimo global, se suelen utilizar algoritmos B&B. Un algoritmo B&B realiza una búsqueda exhaustiva basada en la descomposición sucesiva del problema inicial en sub-problemas de menor tamaño hasta alcanzar la(s) solución(es) final(es) o una aproximación a ella(s). Esta aproximación esta determinada por

[1]Grupo de Investigación TIC146: Supercomputación-Algoritmos, Universidad de Almería, España, e-mail: guillermoaparicio@ual.es

[2]Departamento de Informática, Universidad de Almería, España, e-mail: leo@ual.es

[3]Departamento de Arquitectura de Computadores, Universidad de Málaga, España, e-mail: igarciaf@uma.es Eligius.Hendrix@wur.nl

[4]Departamento de Ecuaciones Diferenciales. Universidad de Tecnología y Economía de Budapest, Hungría, e-mail: bog@math.bme.hu

Algoritmo 1 Ramificación y Acotación

1. $\Lambda := \{S_1 = S\}$ *Conjunto de trabajo*
2. $\Omega := \{\}$ *Conjunto final*
3. $ns := 1$ *Número de símplices*
4. **while** $\Lambda \neq \emptyset$
5. Selecciona un simplex S_i de Λ *Regla de selección*
6. Evalúa S_i *Regla de acotación: no usada aquí.*
7. **if** S_i no puede ser eliminado *Regla de rechazo: no usada aquí.*
8. **if** S_i satisface el criterio de terminación *Regla de terminación: $w(S_i) \leq \epsilon$*
9. Almacena S_i in Ω
10. **else**
11. $\{S_{2i}, S_{2i+1}\}$:=BLM(S_i) *Regla de división: Bisección del Lado Mayor.*
12. Almacena S_{2i}, S_{2i+1} en Λ
13. $ns := ns + 2$
14. **return** Ω y $f(x^*)$ *No usado aquí.*

la precisión requerida para la solución. El algoritmo crea un árbol de búsqueda en el cual se puede podar una rama cuando se determina que un sub-problema no contiene una solución global. Para realizar la poda o rechazo de un sub-problema se suelen calcular cotas de los valores de la función objetivo para cada sub-problema. El Algoritmo 1 muestra un ejemplo básico. El comportamiento y por tanto la eficiencia de un algoritmo B&B se puede caracterizar por cinco reglas: Selección, Acotación, Rechazo, División y Terminación. En este estudio no se van a aplicar las reglas de Acotación, Rechazo y Selección ya que estamos interesados solo en la eficiencia de la regla de División. Al no existir poda del árbol de búsqueda, éste se generará completamente.

III. Algoritmos B&B para símplices

A. El simplex regular

En este estudio, el espacio inicial está determinado por un simplex regular. Un n-simplex está definido en un espacio de $n+1$ dimensiones. Un simplex es el politopo más sencillo posible en un espacio dado. Un 1-simplex es un segmento de una línea; un 2-simplex un triángulo; un 3-simplex es un tetraedro. La Figura 1 muestra gráficamente los ejemplos anteriores.

Estrictamente, un simplex se define como la envoltura convexa de un conjunto de $(n + 1)$ puntos independientes afines en un espacio euclídeo de dimensión mayor o igual que n, es decir, el conjunto de puntos tal que ningún m-plano contiene más de $(m + 1)$ de estos puntos con $m \leq n$.

El n-simplex unidad, o n-simplex estándar, es un subconjunto de $\mathbb{R}^{n+1}$ tal que:

$$T = \left\{ x \in \mathbb{R}^{n+1} \mid \sum_{j=1}^{n+1} x_j = 1;\ x_j \geq 0 \right\} \quad (2)$$

Por simplicidad y sin perdida de generalidad, en este estudio se va a hacer uso de un simplex regular

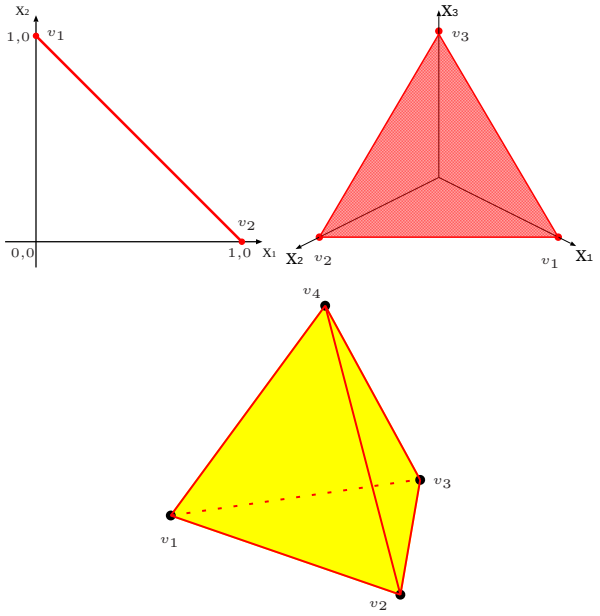


Fig. 1. Símplices de dimesiones 1, 2 y 3.

con longitud de lado 1, que se define como:

$$S = \left\{ x \in \mathbb{R}^{n+1} \mid \sum_{j=1}^{n+1} x_j = \frac{\sqrt{2}}{2};\ x_j \geq 0 \right\}, \quad (3)$$

Las caracteristicas principales de un n-simplex son las siguientes

- Un n-simplex tiene $n + 1$ vértices y $n(n + 1)/2$ lados.
- A la envoltura convexa de un conjunto de $(m + 1)$ puntos independientes afines de un n-simplex $(m < n)$ se denomina m-face.
- A la $(n - 1)$-face se le denomina facet.
- Se define la anchura de un simplex, $\omega(S)$, como la longitud del lado mayor del simplex.

B. Bisección del Lado Mayor (BLM)

La BLM es uno de los métodos de refinamiento más populares ya que es muy simple y puede ser aplicado a problemas de cualquier dimensión [6].

La Figura 2 muestra los símplices generados después de tres bisecciones usando BLM sobre un 2-simplex regular.

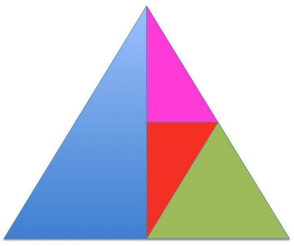


Fig. 2. Primeras BLM en un 2-simplex regular.

El método de BLM puede implementarse como describe en el Algoritmo 2. La Figura 3 muestra los símplices en cada nivel del árbol binario que se ha

Algoritmo 2 BLM(S_i)

$S_i.\{v_1, \ldots, v_{n+1}\}$ *Conjunto de vértices de* S_i

1. $\{v_j, v_k\}$:= SeleccionaLadoMayor(S_i)
2. $v_{new} = \frac{v_j + v_k}{2}$
3. $S_{2i} := S_{2i+1} := S_i$ *Se heredan las características*
4. $S_{2i}.v_j = v_{new}$ *Se actualiza el nuevo vértice*
5. $S_{2i+1}.v_k = v_{new}$
6. **return** S_{2i}, S_{2i+1}

generado mediante el uso de BLM para un 2-simplex regular.

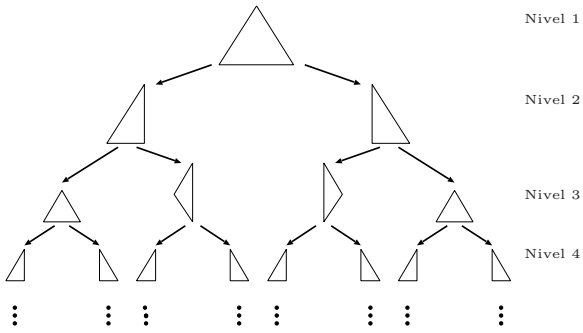


Fig. 3. Niveles del árbol binario obtenido mediante BLM de un 2-simplex regular.

Si el simplex es irregular solo existe un lado mayor cuando $n \leq 2$. Para un 3-simplex, después de la primera BLM aparecen dos símplices irregulares con varias opciones para elegir el lado mayor. La Figura 4 muestra gráficamente este caso.

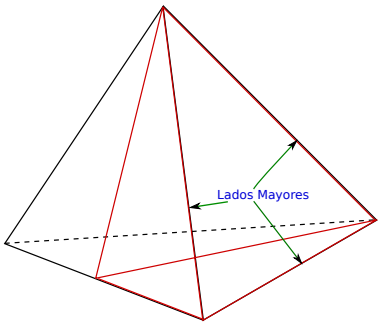


Fig. 4. Lados mayores tras la primera bisección de un 3-simplex regular.

IV. Caracterización del árbol de búsqueda.

Un método de división debería contribuir a:

- Realizar un muestreo uniforme del espacio de búsqueda.
- Garantizar la convergencia del algoritmo.
- Reducir el tamaño del árbol de búsqueda.

La BLM garantiza la convergencia del algoritmo ya que se ha establecido como regla de terminación la longitud del lado mayor. En cuanto a la realización de un muestreo uniforme, se pretende que la forma de los subproblemas sea lo más redondeada posible. Además, estamos interesados en que el número de formas similares generadas sea la menor posible para hacer más fácil la estimación del tamaño de un subárbol a partir de un nodo.

A continuación se presentan estás métricas, las cuales dependerán del lado mayor seleccionado, en los casos en los que existan varios y el simplex no sea regular.

A. Tamaño del árbol de búsqueda

El tamaño del árbol dependerá de como disminuye el tamaño de los sub-problemas conforme se aplica la regla de División. Kearfott presenta un límite superior del tamaño de los sub-problemas cuando se usa BLE en el siguiente teorema [7]:

Teorema 1: Sea S_0 un n-simplex, sea p un entero positivo cualquiera y sea S_p cualquier n-simplex que se ha producido tras p BLM divisiones de S_0. Entonces, el diámetro (o lado mayor) de S_p, $\omega(S_p)$, nunca será mayor que $(\frac{\sqrt{3}}{2})^{\lfloor \frac{p}{n} \rfloor}$ veces el diámetro de S_0, $\omega(S_0)$, donde $\lfloor \frac{p}{n} \rfloor$ es el mayor entero menor o igual que $\frac{p}{n}$.

Kearfott también mostró que, para un simplex S y $p > n$, los diámetros se reducían un factor 2 cada n iteraciones.

Stynes [8] mejoró la cota de Kearfott para un 2-simplex:

$$\omega(S_{2p}) \leq \frac{\sqrt{3}}{2} \left(\frac{1}{2}\right)^{\frac{p}{2}-1}. \tag{4}$$

Por ejemplo, para p=6, la cota de Kearfott es

$$\omega(S_6) \leq \frac{3\sqrt{3}}{2^3}, \tag{5}$$

mientras que la cota de Stynes es

$$\omega(S_6) \leq \frac{\sqrt{3}}{2^3}, \tag{6}$$

Nuestro propósito es hallar el valor exacto del diámetro de los símplices, después de $p > n$ iteraciones, en vez de una cota superior. Para ello nos basaremos en las siguientes definiciones.

Definición 1: Un nivel l del árbol binario se dice completo si el número de elementos en el nivel es 2^{l-1}.

Definición 2: El último nivel completo L_c de un árbol binario es máx$\{l,$ con 2^{l-1} elementos$\}$.

A continuación presentamos los resultados obtenidos para símplices con $n \leq 3$.

A.1 1-simplex

En el caso más simple, usando la BLM en un 1-simplex, el diámetro $\omega(X)$ de los símplices X en el nivel l es:

$$\omega(X) = \frac{\omega(S)}{2^l} \tag{7}$$

Para generar símplices con $\omega(X) \leq \epsilon$ se tiene que:

$$\epsilon \geq \frac{\omega(S)}{2^L}, \tag{8}$$

y el último nivel L del árbol es:

$$L = \lceil \log_2 \frac{\omega(S)}{\epsilon} \rceil \tag{9}$$

A.2 2-simplex

Para el caso de un 2-simplex se pueden obtener las siguientes conclusiones:

Proposición 1: Sea $\epsilon > 0$ el criterio de terminación. Dado un 2-simplex regular inicial S con $\omega(S) = 1$ y $k \in \mathbb{N}^+$. El último nivel del árbol binario L es el último nivel completo L_c si ϵ está en el intervalo

$$\frac{1}{2^{k-1}} \leq \epsilon < \frac{\sqrt{3}}{2^k}, \tag{10}$$

con

$$L = L_c = 2k + 1, \tag{11}$$

Proposición 2: Bajo las mismas condiciones de la Proposición 1, el último nivel del árbol L no estará completo cuando: $\frac{\sqrt{3}}{2^k} \leq \epsilon < \frac{1}{2^{k+1}}$, siendo el número de nodos en el árbol igual a $2^{L-1} - 1 + \frac{2^{L-1}}{2}$. En este caso, el nivel alcanzado en el árbol binario será:

$$L = 2k + 2 \tag{12}$$

pero tan solo la mitad de los símplices de ese nivel habrán sido generados y por tanto el número total de nodos en el árbol será:

$$N_{simplex} = 2^{L-1} + 2^{L-2} - 1 \tag{13}$$

A.3 3-simplex

A partir de n=3 pueden existir varias opciones en la selección del lado mayor [9]. En este estudio se usarán los siguientes métodos:

- *BLM_1:* Es el método normalmente usado y se basa en elegir el primer lado mayor en términos de los índices de los vértices del simplex.
- *BLM_α:* Selecciona el lado mayor, cuyos ángulos con los otros lados son los menores. De esta manera se reducen los ángulos mayores.

Usando BLM_α se pueden deducir los intervalos en los que se encontraría ϵ según el valor de k ya que es más fácil determinar el tamaño del lado mayor de los símplices:

$$\frac{\sqrt{3}}{2^k} \leq \epsilon_1 < \frac{\sqrt{2}}{2^k} \leq \epsilon_2 < \frac{1}{2^k} \leq \epsilon_3 < \frac{\sqrt{3}}{2^{k+1}} \tag{14}$$

Por otro lado, en la ecuación (15) se muestran los posibles valores de k:

$$k = \frac{L - P}{3} \begin{cases} P = 4 \text{ si } \epsilon = \epsilon_1 \\ P = 5 \text{ si } \epsilon = \epsilon_2 \\ P = 6 \text{ si } \epsilon = \epsilon_3 \end{cases} \tag{15}$$

El valor de L puede obtenerse como:

$$L = \lceil (P - 3) + 3 \cdot log_2(\frac{Z}{\epsilon}) \rceil, \text{con}$$
$$\begin{cases} P = 4 \ y \ Z = \sqrt{3} \text{ si } \epsilon = \epsilon_1 \\ P = 5 \ y \ Z = \sqrt{2} \text{ si } \epsilon = \epsilon_2 \\ P = 6 \ y \ Z = 1 \text{ si } \epsilon = \epsilon_3 \end{cases} \tag{16}$$

B. Número de clases de símplices

Definición 3: Dos símplices pertenecen a la misma clase si tienen formas similares, es decir, si aplicando transformaciones de escala, rotación y desplazamiento a un simplex se puede obtener la identidad con el otro simplex.

La Figura 5 muestra el proceso para determinar si dos símplices pertenecen a la misma clase.

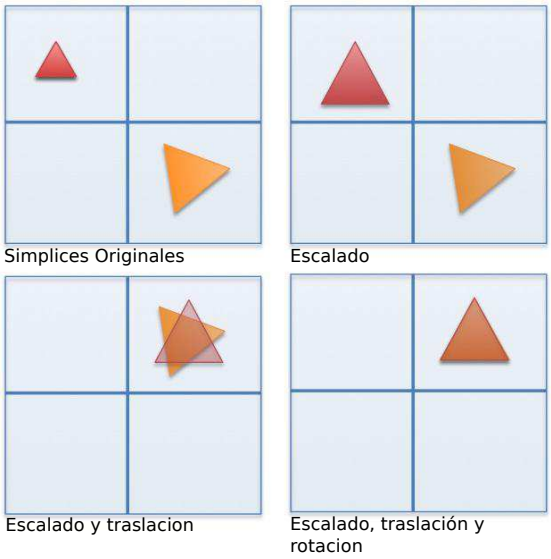


Fig. 5. Escalado, traslación y rotación de 2-símplices.

Los requerimientos computacionales del proceso de determinación del número de clases crece con el tamaño del árbol de búsqueda ya que hay que comprobar si un nuevo simplex pertenece a una clase existente o crea una nueva clase.

C. Índice de redondez de un simplex

En la literatura existen diversos índices que informan sobre la redondez de un simplex. En [10], [11] se presentan varias medidas de calidad para un simplex y se demuestra que la mayoría son equivalentes.

Aquí se muestra un nuevo índice de redondez o calidad de un simplex al que denotaremos por SQ (Simplex Quality) basado en la relación entre la media geométrica y la aritmética de los ángulos existentes entre los lados coincidentes en los vértices de un simplex [12]. Sea m el número total de ángulos que posee un n-simplex, se define SQ como:

$$SQ(S) = n \cdot \frac{\sqrt[n]{\alpha_1^2 \cdot \alpha_2^2 \cdot ... \cdot \alpha_m^2}}{\alpha_1^2 + \alpha_2^2 + ...\alpha_m^2} \tag{17}$$

Este índice genera valores elevados para símplices regulares y valores pequeños para símplices con forma de aguja.

Además, para obtener una versión normalizada de este índice, se define NSQ (Normalized Simplex Quality) como:

$$NSQ(S) = \frac{SQ(S)}{SQ(S_1)} \in (0, 1], \tag{18}$$

V. Resultados de la versión secuencial

El algoritmo secuencial se ha desarrollado en MATLAB y se ha ejecutado sobre un procesador Intel(R) Xeon(R) CPU E5-2620 con 8 cores, una velocidad de 2.00GHz, 20Mb de caché L3 y 64GB de memoria RAM.

La Tabla I muestra el número de símplices generados N, el nivel máximo del árbol de búsqueda L, el tiempo de ejecución cuando se desactiva el cálculo de clases en el algoritmo (T_{nc}) y el tiempo de ejecución cuando sí se realiza el cálculo de clases (T), para un 2-simplex y distintos valores ϵ.

TABLA I
Resultados para un 2-simplex

ϵ	N	L	T_{nc}	T
0.5	11	4	0.24s	0.30s
0.25	47	6	0.26s	0.32s
0.125	191	8	0.36s	0.48s
0.0625	767	10	0.75s	1.14s
0.03125	3071	12	2.29s	3.74s
0.015625	12287	14	8.48s	14.34s

La Tabla II muestra la misma información para un 3-simplex usando BLM_α.

TABLA II
Resultados usando BLM_α para un 3-simplex

ϵ	N	L	T_{nc}	T
0.5	47	6	0.28s	1.46s
0.25	351	9	0.59s	10.93s
0.125	2751	12	3.01s	1.5m
0.0625	21887	15	22.37s	12.16m
0.03125	174847	18	2.95m	1.65h
0.015625	1398271	21	23.78m	13.20h

Puede observarse como el tamaño del árbol binario de búsqueda generado y el tiempo de ejecución aumenta conforme disminuye ϵ y aumenta n. Por este motivo se hace necesario el uso de computación paralela cuando se usan criterios de terminación más precisos y/o n-símplices con $n \geq 3$.

La Tabla III muestra el número máximo de clases NMC, los valores de NSQ y el número de símplices en cada clase, para un 2-Simplex y un 3-Simplex. Para el 3-simplex se usa BLM_α. Como puede observarse, BLM_α permite obtener un número de clases fijo para un 3-simplex.

La Tabla IV compara el número de símplices, el número total de clases, el valor medio de NSQ y el nivel alcanzado en el árbol binario obtenido usando BLM_1 y BLM_α sobre un 3-Simplex con $\epsilon \leq 0{,}03125$.

Para BLM_1 el número de clases crece conforme decrece ϵ [13]. BLM_α genera menos símplices, con formas más redondeadas en media y el número de clases es mucho menor y fijo.

TABLA III
NMC y NSQ para un 2-Simplex y un 3-Simplex

n	NMC	NSQ	Nº Simpl.
2	3	1	683
		0.6429	1706
		0.3333	682
3	8	1	1
		0.7504	12400
		0.6509	24800
		0.6177	25050
		0.4994	56176
		0.4414	6324
		0.4081	25048
		0.3729	25048

TABLA IV
Comparación de BLM_1 y BLM_α sobre un 3-Simplex. $\epsilon \leq 0{,}03125$.

	BLM_1	BLM_α
Nº símplices	259701	174847
Nº Clases	300	8
NSQ Medio	0.4117	0.5926
L	20	18

VI. Versión paralela

Como se ha mostrado, este tipo de experimentos tienen grandes requerimientos de procesamiento y memoria. Además, distintas ramas del árbol pueden generarse de forma independiente, lo que hace a este tipo de algoritmos buenos candidatos para su paralelización.

Para la ejecución de la versión paralela se utilizó un nodo de cómputo de un servidor Bullx compuesto por dos procesadores Intel(R) Xeon(R) CPU E5-2620 de 8 cores cada uno a una velocidad de 2.00GHz, 20Mb de caché L3 y 64GB de memoria RAM. Se ha hecho uso de la Parallel Computing ToolBox que MATLAB posee para tareas específicas de computación paralela.

El modelo de paralelismo se basa en que cada proceso se encargue de generar una rama del árbol. El Algoritmo 3 muestra el pseudo-código.

Algoritmo 3 Paralelización del Código

1. NP *Número de procesos paralelos*
2. NS_h *Número de símplices hoja*
3. Se ejecuta el Algoritmo 1 hasta que $NS_h = NP$
4. Se crean NP procesos
5. Cada proceso ejecuta el Algoritmo 1 sobre uno de los simplices creados.
6. Se combina la información obtenida en los pasos 3 y 5
7. **return** Estadísticas globales de la ejecución.

La Figura 6 muestra los resultados del Algoritmo 3 para NP= 1, 2, 4, 8 y 12, utilizando tanto BLM_α

como BLM_1 con distintos valores de ϵ.

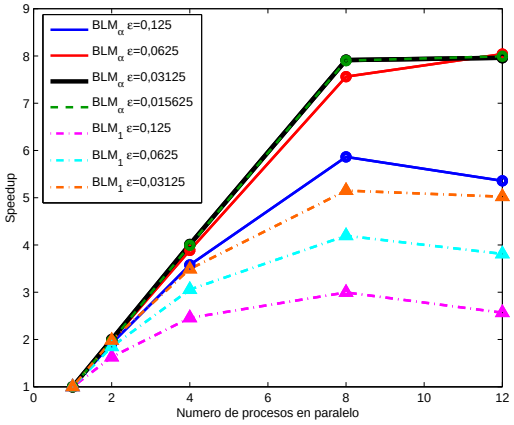


Fig. 6. Speedup para distintos vales de ϵ utilizando BLM_α y BLM_1

Como puede observarse, para BLM_α el speed-up crece de manera lineal hasta llegar a 8 procesos, valor que coincide con el número de cores que posee cada procesador. Sin embargo, el speed-up para BLM_1 es menor. Esto es debido a que BLM_α genera solo 8 clases de símplices, mientras que cuando se usa BLM_1 el número de clases aumenta conforme disminuye ϵ. Cada proceso paralelo genera un número de clases parecido o igual al de la versión secuencual, por lo que el trabajo de determinar si un simlex pertenece a una de las clases existentes o genera una nueva tiene una carga computacional similar a la versión secuencial y no se realiza en paralelo. Además, el paso 6 del Algoritmo 3 se realiza de forma secuencial y debido al número de clases generadas, el número de combinaciones de información es mayor en BLM_1 que en BLM_α.

Hay que destacar que la versión paralela ha permitido la obtención de resultados para $\epsilon = 0{,}015625$ usando BLM_α. El tiempo de ejecución usando BLM_1 para ese valor de ϵ, es de más de un día.

Aunque la ToolBox de programación paralela de MATLAB permite definir hasta 12 ejecuciones en paralelo y disponer de dos procesadores por nodo de cómputo, los resultados obtenidos hacen pensar que el uso de los cores del segundo procesador reduce la tendencia lineal de la ganancia en velocidad obtenida en un solo procesador.

VII. Conclusiones y trabajo futuro

Los algoritmos de B&B aplicados a problemas optimización global, suelen exigir una alta carga computacional cuando se buscan soluciones precisas. Este trabajo presenta un estudio sobre el comportamiento de estos algoritmos cuando el espacio de búsqueda es un simplex. Se presentan estudios sobre el número de elementos del árbol, su nivel de calidad y el número de clases similares cuando se usa una bisección del lado mayor como regla de división. Se presenta un nuevo método de selección del lado mayor a dividir cuando existen varias alternativas. Estos algoritmos son buenos candidatos para su paralelización. Se muestran los resultados obtenidos con una versión paralela desarrollada en MATLAB obteniéndose un speed-up casi lineal hasta llegar a 8 núcleos usando el método de bisección propuesto BLM_α.

Como trabajos futuros se pretende extender los métodos desarrollados a problemas con $n > 3$. Para ello será necesario desarrollar algoritmos que permitan el uso de un número mucho mayor de cores.

Agradecimientos

Este trabajo ha sido subvencionado por el Ministerio de Ciencia e Innovación (TIN2008-01117, TIN2012-37483-C03-01 y TIN2012-37483-C03-03) y la Junta de Andalucía (P011-TIC7176) y financiado en parte por el Fondo Europeo de Desarrollo Regional (ERDF).

Referencias

[1] L.G. Casado, I. García, B.G. Tóth, and E.M.T. Hendrix, "On determining the cover of a simplex by spheres centered at its vertices," *Journal of Global Optimization*, vol. 50, no. 4, pp. 645–655, 2011, doi:10.1007/s10898-010-9524-x.

[2] Andrew Adler, "On the Bisection Method for Triangles," *Mathematics of Computation*, vol. 40, no. 162, pp. 571–574, 1983, doi:10.1090/S0025-5718-1983-0689473-5.

[3] R. Horst, "On generalized bisection of n-simplices," *Mathematics of Computation*, vol. 66, no. 218, pp. 691–698, 1997.

[4] José L. Berenguel, L.G. Casado, I. García, and E.M.T. Hendrix, "On estimating workload in interval branch-and-bound global optimization algorithms," *Journal of Global Optimization*, In Press, doi:10.1007/s10898-011-9771-5.

[5] J.F. Sanjuan-Estrada, L.G. Casado, and I. García, "Adaptive parallel interval branch and bound algorithms based on their performance for multicore architectures," *Journal of Supercomputing*, vol. 58, pp. 376–384, 2011, doi: 10.1007/s11227-011-0594-4.

[6] A. Hannukainen, S. Korotov, and M. Krízek, "On numerical regularity of the longest-edge bisection algorithm," *BcaMath*, 2013.

[7] Baker Kearfott, "A proof of convergence and an error bound for the method of bisection in $\mathbf{R}^n$," *Journal of Mathematical Computing*, vol. 32, no. 144, pp. 1147–1153, 1978.

[8] M. Stynes, "On the faster convergence of the bisection method for all triangles," *Math. Comp.*, vol. 35, pp. 1195–1201, 1980.

[9] Anwei Liu and Barry Joe, "On the shape of tetrahedra from bisection," *Math. Comput.*, vol. 63, no. 207, pp. 141–154, July 1994, doi: 10.2307/2153566.

[10] V.N. Parthasarathy, C.M. Graichen, and A.F. Hathaway, "A comparison of tetrahedron quality measures," *Finite Elements in Analysis and Design*, vol. 15, no. 3, pp. 255 – 261, 1994.

[11] A. Plaza y G. F. Carey, "Explicación geométrica de una medida de forma de símplices n-dimensional.," *Métodos numéricos para cálculo y diseño en ingeniería*, vol. 18, no. 4, pp. 475–480, 2011.

[12] Ivo G. Rosenberg and Frank Stenger, "A lower bound on the angles of triangles constructed by bisecting the longest side," *Mathematics of Computation*, vol. 29, no. 130, pp. 390–395, 1975, doi:10.1090/S0025-5718-1975-0375068-5.

[13] Joseph M. Maubach, "The efficient location of neighbors for locally refined n-simplicial grids," in *In 5th Int. Meshing Roundable*, 1996, pp. 137–153.

Performance and scalability in distributed cluster-based individual-oriented fish school simulations

Francisco Borges, Roberto Solar, Remo Suppi, Emilio Luque [1]

Abstract— **With increasing computational power and parallel/distributed computing it is possible to create more complex models to achieve simulation results closest to reality. More realistic results allow scientists to reach conclusions and gain knowledge about the system under study. However this realistic solutions implies increasing the complexity of model, consequently it's required more communication. The problem is that communication has negative impact in parallel distributed simulations. Two of the main challenge in distributed simulation is how to distribute individuals efficiently through distributed architecture and the efficiency of communication strategies. In this work we compare our cluster-based partitioning approach based on Voronoi diagrams and covering radius criterion by using asynchronous and synchronous communication strategy for distributed individual-oriented fish school simulation. The main contributions are: we showed that cluster-based partitioning approach is scalable in both communication strategies and show that asynchronous communication strategy has best speedup in comparison to synchronous communication strategy.**

Keywords— **parallel distributed simulation, individual-oriented models, data clustering, fish schooling, high performance distributed simulation;**

I. Introduction

ECOLOGISTS have been using discrete models to study global patterns that emerge through local interaction by computational simulation. Individual-oriented modeling is a powerful tool to simulate complex systems in which system-level properties emerge from individual-level interaction. Individual-level interaction can be seen in many research areas such as: ecology and biology [1],[2],[3],[4],[5], military strategies [6], sociology [7],[8],[9], physics [10],[11], health care [12], fire suppression strategies [13], etc. With increasing computational power and parallel/distributed computing it is possible to create more complex models to achieve simulation results closest to reality. Although many researchers think these results are not enough and claim by more realistic simulations.

The realistic simulations aim is achieving a similar interaction level such as the real environment. In fish schooling simulations, for example, more realistic simulations means more fish in simulation, more fish iterations, more realistic schooling parameters, different schooling movements, varied form and structures schools, natural obstacles, predators. This type of simulations are usually fairly complex thus requiring compute and communication. If the simulator is scalable then we just need to increase resource quantity so that total time execution decreases. However it's not enough, because the enlarge processors quantity will impact communication, which is a bottleneck in parallel distributed program. The communication has been a problem in our fish schooling simulator because it is a communication-consumer parallel. In previous instrumentation of code, we verified that almost 41% of total execution time is spent with MPI_Bcast function. It happens because there are migration of individuals between the nodes. Then it is necessary to exchange the individual's informations between the nodes. In spite of our simulator be scalable, we have to improve the efficiency and propose solutions to decrease the communication time.

Thereby is necessary that the simulator also be efficient in reducing time communication and consequently the total time execution simulation. Thus more realistic behavior can be used in simulation.

In this context, we have been researching the individual-oriented model in high performance time-driven simulation with the purpose of decreasing total time simulation and consequently providing more realistic models. We implemented a robust cluster-based partitioning method by means of covering radius criterion and voronoi diagrams. This partitioning method decreased the sequential algorithm complexity from $O(n^2)$ to $O(nm)$. Furthermore, this method reduced the interaction between individuals that are far away from each other, consequently decreasing the compute. In this work we compare our cluster-based partitioning approach based on Voronoi diagrams and covering radius criterion by using asynchronous and synchronous communication strategy for distributed individual-oriented fish school simulation. As we presented before the communication is a problem, therefore we should understand and analyze communication in time-driven simulations. In this direction, our main goal is verify the behaviour of cluster-based partitioning algorithm over two differents communication strategy. In addition verify which communication strategy is more efficient.

In section II we present an overview of individual-oriented models. A brief resume about the fish school model and the simulator is done in III. The distributed time-driven simulation is discussed in IV. Our partitioning algorithm, the data structure to store and manipulate individuals and its construc-

[1]Dept. of Computer Architecture & Operating Systems Universitat Autonoma de Barcelona, e-mail: francisco.borges@caos.uab.es, rsolar@caos.uab.es, remo.suppi@uab.es, emilio.luque@uab.es.

tion are explained in section V. The experimental results and analysis are presented in section VI. Finally, in section VII the conclusions and future work are proposed.

II. Individual-oriented Models

An individual-oriented approach can provide techniques and tools for analyzing complex organizations and social phenomena. These models allow us to understand global consequences from local interactions of members of a population. These individuals might represent: *fish* in *schools* [1], [4], [3], *people* in *crowds* [7], [8], *birds* in *flocks* [2], etc. These type of models typically consist of: an *enviroment* in which interactions occur and a fixed set of *individuals* defined in terms of their behavioral rules and characteristic attributes.

Some individual-oriented models are also *spatially-explicit* which means individuals are associated with a location in geometrical space. Furthermore, some *spatially-explicit* individual-oriented models can exhibit *motion patterns* which means individuals can change their position in geometrical space as the simulation progresses.

Individual-oriented models also allows us: to include different types of individuals within the same model, to define individuals with two levels of heterogeneity (behavioral rules and attributes values), to model interactions between individuals and its environment, and to represent learning mechanisms.

III. Fish school model and the simulator

Our distributed cluster-based individual-oriented fish schooling simulator is based on the biological model described in [14] and [1]. Fish Schools are one of the most frequent social groups in the animal world [14]. This social aggregation shows complex emergent properties, such as: strong group cohesion and high level of synchronization. In order to describe the fish behavior the model considers that each fish changes its position and orientation in discrete step of time (time-driven simulation) and the new fish's position and orientation depends on the position and orientation of a fixed number of nearest-neighbors. The neighbor's influence on a single fish depends on its space-time position. In order to select appropriately the neighbors's influence the model identifies three vision areas: attraction, repulsion and parallel orientation, see Fig. 1.

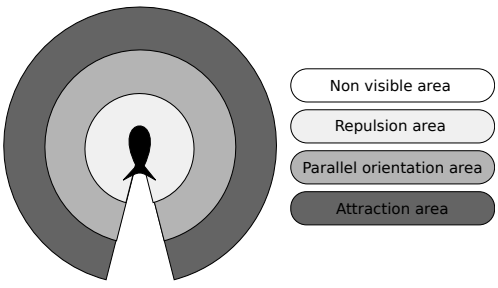


Fig. 1. Neighbor influences areas.

Depending on the spatial-temporal position of its neighbors, the fish chooses between three behavior patterns: repulsion - collision avoidance between fish of the same group (turning the fish's orientation minimum-angle rotation so the fish's orientation and its neighbor's orientation are perpendicular, Fig. 3; parallel orientation - the group moves in the same direction (matching the fish's orientation with its neighbor's orientation, Fig. 4, and attraction - to maintain the cohesion of the group (steering the fish's orientation towards its neighbor's position, Fig. 2).

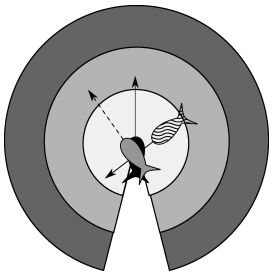


Fig. 2. Repulsion behavior pattern.

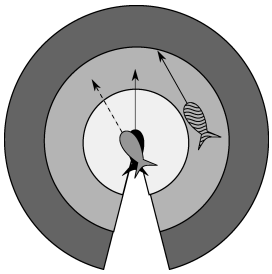


Fig. 3. Parallel orientation behavior pattern.

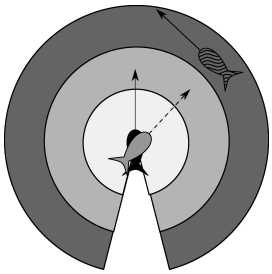


Fig. 4. Attraction behavior pattern.

IV. Distributed time-driven simulation

An important issue in discrete simulation is the mechanism in which the state variables changes as simulation time advance. We can distinguish two time-advance approaches: *time-driven* and *event-driven*. In a *time-driven* approach, the simulation time is subdivided into a sequence of equal-sized steps, and the simulation advances from one time step to the next [15]. In the *event-driven* mechanism the simulation time is computed when a event is processed adding the event time to the execution time. Furthermore, is necessary to consider the time management protocol because this will ensuring that

the execution of the distributed simulation is properly synchronized. Time management algorithms can be classified as *conservative* or *optimistic* [15]. In *conservative* algorithms, the process is blocked until all execution conditions are satisfied. In *optimistic* algorithms, the process will continues even if some execution condition is not fulfilled but this algorithm includes mechanisms in order recover from causality issues. We implement a time-driven time-advance mechanism because the individual-oriented model used in this work describes the motion of a fish school in discrete time steps, i.e. each fish moves at the same time. Furthermore, we implement a conservative time-management protocol because each logical process requires exchanging information (migration or neighboring request) from adjacent logical processes before starting the next simulation step [15].

A. Model distribution

After building our *list of clusters*, we proceed to distribute it through the distributed architecture. The model distribution used in this work is based on proximity concept. We distribute the list header together a fixed set of different clusters to each node. This fixed set of clusters is determined from the maximum number of individuals that each node can have ($\frac{N_{individuals}}{N_{processors}}$), and how close are to each other. The main issue our model distribution is assigning of contiguous groups of clusters to each node in order to decrease communication and computing involved selecting data to transfer. Every node independently executes the simulation with local data making data exchange through three type of messages: *neighbors exchange*, *centroids exchange* and *migration*. The *neighbors exchange* process consists intersecting local clusters with non-local clusters in order to find adjacency.

In this computation we use the cluster's distance data and the covering radius in order to intersect them, obtaining the data set to transfer distance data and the covering radius in order to intersect them, obtaining the data set to transfer. Next, centroids' *neighbors selection* is made, and its position and orientation is updated. After this operation, we execute the centroids exchange process. This process consists of send local centroids to each node. Once updated the list header, we proceed to neighbors selection and updating the position and orientation of each individual belonging to each bucket. Finally, we reinsert each bucket element within the list header. After this step some individuals can be stored within a non-local cluster, and in this case, these individuals are reasigned to the correct node.

V. Partitioning method

In this section we explain the partitioning method used in this work. We use an hybrid partitioning method based on *voronoi diagrams* and *covering radius* criterion. Voronoi diagram is one of the most fundamental data structures in computational geometry. Given some number of objects in the space, their Voronoi diagram divides the space according to the *nearest-neighbor rule*: each object is associated with the area closest to it [16]. Covering radius criterion consist of trying to bound the area $[c_i]$ by considering a sphere centered at c_i that contains all the objects of the problem domain that lie in the area [17]. A metric space is a particular type of vectorial space where a distance between objects is defined. Given a set of objects $\mathbb{X}$ subset of the universe of valid objects $\mathbb{U}$ and a distance function $d : \mathbb{X}^{\not\in} \to \mathbb{R}$, so $\forall x, y, z \in \mathbb{X}$, must be met the following conditions:

Positiveness: $d(x, y) \geq 0, d(x, y) = 0 \Rightarrow x = y$
Symmetry: $d(x, y) = d(y, x)$
Triangular inequality: $d(x, y) + d(y, z) \geq d(x, z)$

Since individuals are associated with a position in three-dimensional euclidean space, we assume that these individuals together with the euclidean distance (which determines the visibility of individuals or objects in environment), generating a metric space. This allow us introducing concepts of similarity or proximity within the distributed simulation.

The partitioning method proposed consist of two stages: First, the *centroids selection* by means of covering radius criterion, which ensures us a set of centroids far away enough the others. Second, the *space decomposition* by means of voronoi diagrams, which allow us define similar size areas with similar number of individuals (as long as individuals are uniformly distributed in space). Both criteria are applied to construct the data structure described in the next section.

A. Data Structure

The metric data structure used to store individuals is called fixed-radius *list of clusters* [17], specifically, we use an hybrid *voronoi diagram/covering radius* building criterion for a fixed-radius *list of clusters* [18]. The radius is fixed in function of the maximum fish vision area ($r \geq R_{ATTRACTION}$). This allow us defining areas in which individuals can interact only with individuals belonging to adjacent areas, reducing computing involved in the *neighbors selection* process.

The structure is formed by a linked list of *clusters*. Each cluster consists of several data such as: a *centroid* - which is the most representative element of the cluster, the *processor identifier* (PID) - indicating in which node each cluster is stored, a *bucket* - in which individuals belonging to the cluster are stored, the *cluster identifier* (CID) - indicating the cluster position in the list, and some information about distances to other clusters (Fig. 5).

B. Construction

The construction consists of iterative insertion of individuals within the data structure. First, if the list is empty, the first individual as center is selected. Next, for each individual, the distance to each center in order to find the minimum distance is calcu-

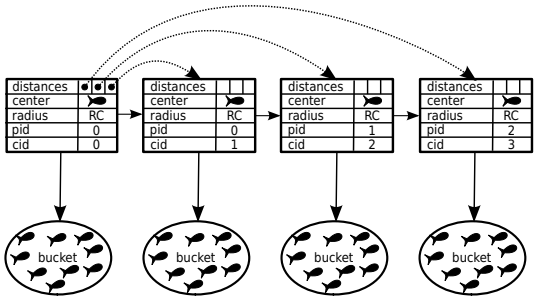


Fig. 5. List of clusters.

lated. If the minimum distance is greater than the covering radius, the individual as center is selected, otherwise, the individual within the closest cluster is inserted. The construction algorithm complexity is $O(nk)$, where n is the number of individuals and k the number of clusters ($k < n$).

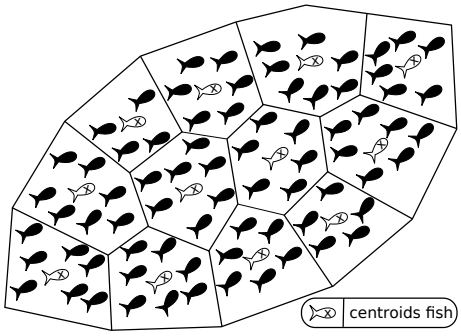


Fig. 6. Covering radius/Voronoi diagram partitioning.

In Fig. 6 a two-dimensional example of our cluster-based partitioning method is shown. One of the main improvements of our proposed is reducing the execution time of sequential algorithm. Sequential algorithm performs exhaustive computing ($n \times n$) in order to find the most influential neighbors. In our case, individuals only perform computing of their nearest-neighbors individuals belonging to adjacent clusters, reducing computing involved in obtaining the spatial-temporal position of their neighbors.

VI. Experimental results

The execution environment used for testing has the following characteristics: Cluster IBM - 32 nodes 2 x dual-core Intel(R) Xeon(R) CPU 5160@ 3.00GHz, 12GB fully buffered DIMM, Integrated dual Gigabit Ethernet and hot-swap SAS controller Disk. Experimental results were obtained from the average of 250 simulation steps. The simulator was developed using C++ (gcc 4.3.2), STL (c++ standard template library) and MPI namespace (openmpi 1.4.1).

The simulation experiments were carried out using 1, 2, 4, 8, 16, 32 and 64 processors. The data input were formed by three files containing 131.072, 262.144 and 524.288 individuals uniformly distributed. Each processor and data input was executed with a load balance factor of 10%.

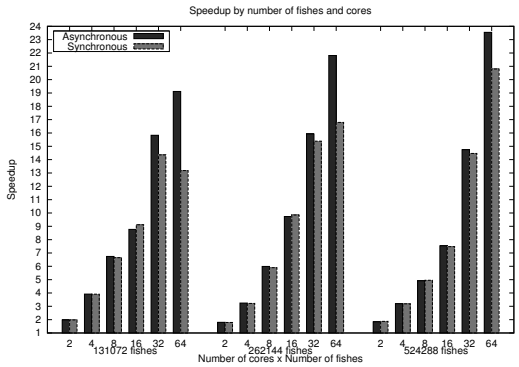


Fig. 7. Speedup.

The Fig. 7 show the speedup of each communication strategies by using 1, 2, 4, 8, 16, 32 and 64 cores with workloads of 131.072, 262.144 and 524.288 individuals and load imbalance factor of 10%. The cluster-based approach is scalable at both strategies. This scalability can be observed when increasing the number of individuals as well as with increasing of processors. We can observe that the speedup in both strategies are similares of 2 to 16 processors. The differences between speedups are more evident at 32 and 64 processors. Because there are more communication between the LPs, as result it increase the amount of communication in simulation. The asynchronous communication strategy has better speedup with more processors than synchronous communication strategy. The figures follows help us understand best this behaviour.

The Fig. 8, 9, 10, 11, 12 and 13 show the results of experiment when it was executed with 524.288 fishes. Those figures, enable us analyze the relation between computing and communication of each communication strategies. The behaviour observed for this workload also can be seen at 131.072, 262.144 individuals.

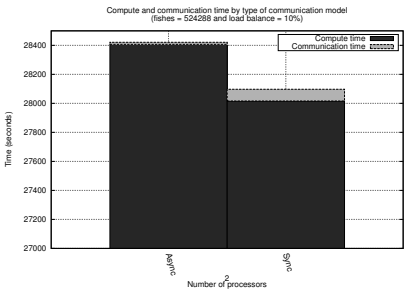


Fig. 8. Comparison between communication and computing by using 524288 fishes executed at 2 processors.

The experiment demonstrate that the asynchronous communication strategy has best communication times than synchronous communication strategies. However this strategy spend more time doing computing than synchronous communication strategy.

In summary, the cluster-basead approach by using the asynchronous communication strategy has a

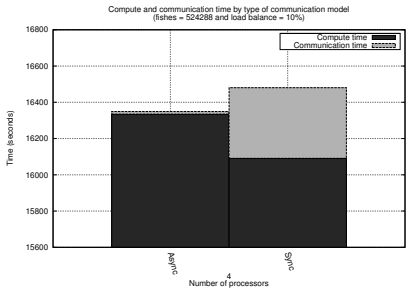


Fig. 9. Comparison between communication and computing by using 524288 fishes executed at 4 processors.

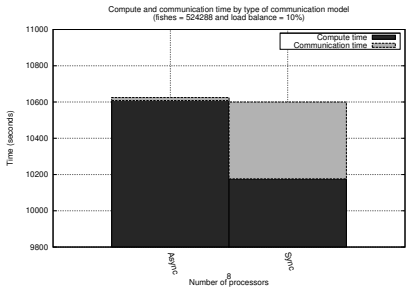


Fig. 10. Comparison between communication and computing by using 524288 fishes executed at 8 processors.

better performance in terms of total execution times in comparison to synchronous communication strategies. The synchronous communication strategy has a better performance in terms of computing times, but higher communication times in comparison to asynchronous communication strategy. Therefore, what it is gained in compute is lost in communication.

The synchronous communication strategy would be an attractive strategy if it is possible fine adjust the load balancing algorithm. In this way the time taken in barrier communication will be less. Consequently, the communication time would less too.

As theoretical interpretation of the results, we believe that the synchronous communication strategy in this solution has one drawback: two level-barrier. The first level-barrier is the synchronous routine communication. The receive has to waiting until receive all package. The other level-barrier is come from conservative time-management algorithm that blocked the process until all execution conditions are satisfied.

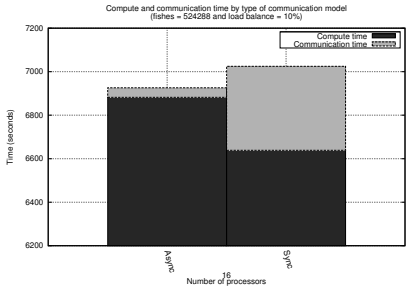


Fig. 11. Comparison between communication and computing by using 524288 fishes executed at 16 processors.

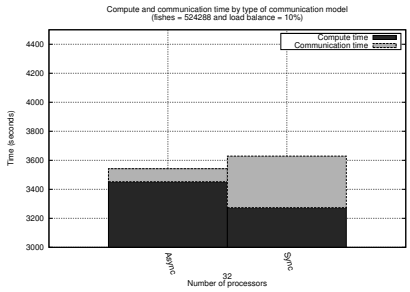


Fig. 12. Comparison between communication and computing by using 524288 fishes executed at 32 processors.

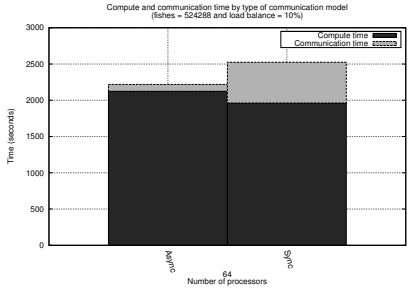


Fig. 13. Comparison between communication and computing by using 524288 fishes executed at 64 processors.

VII. Conclusions and Future work

In this work, we have analyzed the performance of our cluster-based partitioning approach over asynchronous and synchronous communication strategies. The main conclusions that can be extracted are:

- We showed that cluster-based partitioning approach is scalable in both communication strategies.
- The results show that asynchronous communication strategy has best speedup in comparison to synchronous communication strategy.

The main objectives for future work are:

- Compare the bulk-synchronous parallel communication strategy with the synchronous and asynchronous message passing communications strategies. By reviewing the literature [19] [20] we have verified that the BSP computing model can be an alternative to improve performance of parallel applications.
- Increasing performance of our simulator using hybrid programing. A lot of fishes are in the same processors but the communication between them occurs via MPI. In this case OpenMP operations would decrease communication time and computing time.
- Explore the process affinity with purpose of decreasing the communication time.

Acknowledgements

This research has been supported by the MICINN Spain under contract TIN2007-64974 and the MINECO (MICINN) Spain under contract TIN2011-24384.

Referencias

[1] Andreas Huth and Christian Wissel, "The simulation of fish schools in comparison with experimental data," *Ecological Modelling*, vol. 75-76, pp. 135 – 146, 1994, State-of-the-Art in Ecological Modelling proceedings of ISEM's 8th International Conference.

[2] Craig W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *SIGGRAPH Comput. Graph.*, vol. 21, pp. 25–34, August 1987.

[3] Roberto Solar, Remo Suppi, and Emilio Luque, "High performance individual-oriented simulation using complex models," *Procedia Computer Science*, vol. 1, no. 1, pp. 447 – 456, 2010, ICCS 2010.

[4] Remo Suppi, Pere Munt, and Emilio Luque, "Using pdes to simulate individual-oriented models in ecology: A case study," in *Proceedings of the International Conference on Computational Science-Part I*, London, UK, 2002, ICCS '02, pp. 107–116, Springer-Verlag.

[5] Bo Zhou and Suiping Zhou, "Parallel simulation of group behaviors," in *Proceedings of the 36th conference on Winter simulation*. 2004, WSC '04, pp. 364–370, Winter Simulation Conference.

[6] Joshua J. Corner and Gary B. Lamont, "Parallel simulation of uav swarm scenarios," in *Proceedings of the 36th conference on Winter simulation*. 2004, WSC '04, pp. 355–363, Winter Simulation Conference.

[7] Guillermo Vigueras, Miguel Lozano, and Juan Ordu1a, "Workload balancing in distributed crowd simulations: the partitioning method," *The Journal of Supercomputing*, pp. 1–9, 2009.

[8] Yongwei Wang, M. Lees, Wentong Cai, Suiping Zhou, and M.Y.H. Low, "Cluster based partitioning for agent-based crowd simulations," in *Winter Simulation Conference (WSC), Proceedings of the 2009*, 2009, pp. 1047 –1058.

[9] Michael J. Quinn, Ronald A. Metoyer, and Katharine Hunter-zaworski, "Parallel implementation of the social forces model," in *Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, 2003, pp. 63–74.

[10] M. A. Stijnman, R. H. Bisseling, and G. T. Barkema, "Partitioning 3d space for parallel many-particle simulations," *Computer Physics Comm*, vol. 149, pp. 121–134, 2003.

[11] Youssef M. Marzouk and Ahmed F. Ghoniem, "K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical n-body simulations," *J. Comput. Phys.*, vol. 207, pp. 493–528, August 2005.

[12] Dhananjai M. Rao and Alexander Chernyakhovsky, "Parallel simulation of the global epidemiology of avian influenza," in *Proceedings of the 40th Conference on Winter Simulation*. 2008, WSC '08, pp. 1583–1591, Winter Simulation Conference.

[13] Xiaolin Hu and Yi Sun, "Agent-based modeling and simulation of wildland fire suppression," in *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, Piscataway, NJ, USA, 2007, WSC '07, pp. 1275–1283, IEEE Press.

[14] Andreas Huth and Christian Wissel, "The simulation of the movement of fish schools," *Journal of Theoretical Biology*, vol. 156, no. 3, pp. 365 – 385, 1992.

[15] Richard M. Fujimoto, *Parallel and Distribution Simulation Systems*, John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999.

[16] Franz Aurenhammer, "Voronoi diagrams - a survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, pp. 345–405, September 1991.

[17] E. Chavez and G. Navarro, "An effective clustering algorithm to index high dimensional metric spaces," in *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, Washington, DC, USA, 2000, pp. 75–, IEEE Computer Society.

[18] Roberto Uribe Paredes, Claudio Marquez, and Roberto Solar, "Construction strategies on metric structures for similarity search," *CLEI Electron. J.*, vol. 12, no. 3, 2009.

[19] David B. Skillicorn, Jonathan M. D. Hill, and William F. McColl, "Questions and answers about bsp," in *Scientific Programming*, 1997, vol. 6, pp. 249–274.

[20] Jonathan M. D. Hill, Bill McColl, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B. Rao, Torsten Suel, Thanasis Tsantilas, and Rob H. Bisseling, "Bsplib: The bsp programming library," *Parallel Comput.*, vol. 24, no. 14, pp. 1947–1980, Dec. 1998.

Solución de múltiples sistemas lineales en GPUs

J. M. Molero, [1] E. M. Garzón[2] I. García[3] E.S. Quintana-Ortí[4] A. Plaza[5]

Resumen— **Este trabajo se centra en el cálculo, de forma concurrente, de múltiples sistemas lineales definidos por matrices densas de una dimensión media. Se considera una solución basada en la factorización de Cholesky y su implementación sobre plataformas GPUs. Este tipo de problema algebraico forma parte de distintos modelos de procesamiento de señal, y además exhibe distintos niveles de paralelismo que pueden ser explotados muy eficientemente por las GPUs. En este trabajo, como aplicación del problema, nos centramos en la detección de anomalías sobre imágenes hiperespectrales, lo que supone una importante tarea en la explotación de este tipo de imágenes obtenidas en la observación de la Tierra. Concretamente, consideramos la versión local del ampliamente usado algoritmo RX (Reed-Xiaoli), denotado como LRX, en el que una de sus etapas más costosas consiste en la solución, para cada pixel de la imagen, de un sistema lineal cuyas dimensiones coinciden con el número de bandas de la imagen. En este contexto se describe y evalúa la solución múltiple de sistemas lineales en GPUs, y se comprueba el factor de aceleración obtenido con la implementación propuesta en este trabajo.**

Palabras clave— **Factorización de Cholesky, Computación GPU, Detección de anomalías, Imagenes hiperespectrales , Algoritmo RX.**

I. Introducción

La factorización de Cholesky es un método algebraico ampliamente usado para la resolución numérica de sistemas lineales de ecuaciones cuya matriz es simétrica y definida positiva (SPD) [1, 2]. Varias implementaciones por bloques de este método están disponible en gran variedad de librerías algebraicas de uso tan extendido como LAPACK [1], MKL [2] y recientemente MAGMA[3] que incluye una implementación híbrida CPU-GPU de esta factorización. Cabe destacar que todas estas implementaciones están diseñadas para acelerar el cálculo de una única factorización aplicada a matrices de grandes dimensiones [3].

Por otra parte, existen aplicaciones que requieren un esquema de cálculo diferente, donde es necesario resolver muchos sistemas lineales independientes de dimensión pequeña o media. Las operaciones de álgebra lineal relacionadas con este esquema particular, se identifican como *batched* en el contexto de la computación en GPU [4]. Dos niveles de paralelismo pueden ser explotados en este tipo de soluciones: (1) el paralelismo de la solución estándar de cada sistema que está limitado por las dependencias de datos y el tamaño del problema; y (2) la concurrencia intrínseca a la solución de múltiples sistemas lineales. Existen algunas referencias que analizan la solución múltiple de sistemas en GPUs [5]. Cabe destacar que las plataformas GPUs permiten explotar eficazmente ambos tipos de paralelismo si las dimensiones de los sistemas no son excesivas. De esta forma, si las dimensiones de las matrices son pequeñas cada sistema podrá ser resuelto por un hilo de la GPU, si la dimensión es media será necesario que los hilos de un bloque colaboren en la solución de cada sistema y si la dimensión es grande entonces será necesario que todos los hilos colaboren en la solución de un único sistema y por tanto el segundo nivel de paralelismo mencionado no podrá ser explotado en la GPU.

En este trabajo, se analiza una versión CUDA de la solución de múltiples sistemas lineales de dimensión media basada en la factorización de Cholesky (en lo sucesivo, `CuBCholS`) para GPUs de NVIDIA [6, 7]. Para lo cual, como caso de estudio nos centramos en una aplicación relacionada con el campo del procesado de imágenes hiperespectrales. Este campo se caracteriza por trabajar con imágenes de teledetección compuestas por cientos de canales diferentes (correspondientes a diferentes longitudes de onda) para una área en la superficie de la Tierra. Estos datos suponen enormes posibilidades desde el punto de vista de la observación de la Tierra, pero su procesamiento demanda altos requerimientos computacionales. Para hacer frente a estos problemas con una alta carga computacional, se hace necesario el uso y el aprovechamiento eficiente de arquitecturas de alto rendimiento, tales como las actuales plataformas de procesamiento gráfico o GPUs.

En la próxima sección se analizan los aspectos más importantes de la implementación de `CuBCholS` en GPUs. A continuación, la Sección III se centra en el algoritmo RX local para la detección de anomalías en imágenes hiperespectrales, usado como un prototipo de aplicación de la librería que se propone en este trabajo. La Sección IV analiza y describe los principales detalles de la implementación y evaluación experimental con una tarjeta GPU NVIDIA GeForce GTX 680, resumiendo la evaluación del rendimiento alcanzado por `CuBCholS` en el contexto de la aplicación descrito en la sección previa. Para finalizar, la Sección V destaca las principales conclusiones y trabajos futuros.

[1]Dpto. de Informática y CEIA3, Univ. Almería, e-mail: jmp384@ual.es
[2]Dpto. de Informática y CEIA3, Univ. Almería, e-mail: gmartin@ual.es
[3]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: igarciaf@uma.es
[4]Dpto. de Ingeniería y Ciencia de los Computadores, Univ. Jaime I, e-mail: quintana@icc.uji.es
[5]Laboratorio de Computación Hiperespectral, Univ. Extremadura, e-mail: aplaza@unex.es

[1]http://www.netlib.org/lapack/
[2]http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation
[3]http://icl.cs.utk.edu/magma/

II. Solución de múltiples sistemas en GPUs basado en la factorización de Cholesky

Dada una matriz $A \in \mathbb{R}^{n \times n}$ simétrica y definida positiva, la factorización de Cholesky calcula la descomposición $A = LL^T$, donde $L \in \mathbb{R}^{n \times n}$ es triangular inferior, por tanto permite expresar la solución del sistema lineal $Ax = b$ en la solución de dos sistemas triangulares [1].

El siguiente algoritmo calcula la factorización de Cholesky [8]:

```
for (i=1; i≤n; i++) {
   for (sum=A[i][j], j=i; j≤n; j++) {
      for (k=i-1; k≥1; k- -)
         sum -= A[i][k]*A[j][k];
      if (i = j) {
         if (sum ≤ 0.0) error;
         p[i]=sqrt(sum);
      } else A[j][i]=sum/p[i];
   }
}
```

Una vez que la matriz ha sido factorizada, la matriz triangular L calculada se almacena en dos estructuras, la triangular de A y los elementos diagonales en el vector p. Con estos elementos, para completar la solución es necesario resolver sucesivamente dos sistemas triangulares. Se puede resolver el sistema lineal por sustitución de la siguiente manera:

```
for (i=1; i≤n; i++) {
   for (sum=b[i],k=i-1; k≥1; k- -){
      sum-=a[i][k]*x[k];
   }
   x[i]=sum/p[i];
}
for (i=n; i≥1; i- -) {
   for (sum=x[i],k=i+1; k≤n; k++){
      sum-=a[k][i]*x[k];
      }
   x[i]=sum/p[i];
   }
}
```

En este trabajo, nuestro interés se centra en escenarios en los que es necesario resolver cientos o miles de sistemas lineales densos (SPD) de dimensión media (concretamente, $50 \leq n \leq 500$) utilizando la factorización de Cholesky. Por lo tanto, este tipo de problemas tiene una alta carga computacional, por lo que se hace necesaria la explotación de plataformas de alto rendimiento.

En este trabajo, se propone una implementación *batched* para resolver estos sistemas de forma simultánea en una GPU, por lo que se propone crear una nueva rutina denominada (`CuBCholS`). Por esto, se desarrolla este algoritmo que aprovecha los recursos de computación de la GPU para realizar el cálculo concurrentemente de cientos de sistemas lineales independientes.

La implementación de la rutina `CuBCholS` en GPU explota dos niveles de paralelismo: (1) internamente, cada sistema se puede descomponer en una serie de tareas con dependencias entre ellas, y (2) externamente, el cálculo de los sistemas lineales se puede realizar de forma independiente.

De acuerdo con este esquema, en nuestra implementación GPU de la resolución de sistemas, cada bloque CUDA calcula una factorización de Cholesky y resuelve el sistema correspondiente. De esta forma, cada bloque de CUDA tiene un tamaño igual al de las matrices de los sistemas, con lo que cada hilo realizará las operaciones correspondientes a un elemento de la columna de dichos sistemas, repitiendo el proceso para cada fila de los mismos con el objetivo de satisfacer las dependencias de datos existentes.

Cabe destacar que en este contexto, con el objetivo de optimizar la intensidad aritmética del algoritmo [5], se ha expresado la solución de los sistemas integrando la factorización de Cholesky con la solución del primer sistema triangular, de esta forma, se puede explotar de manera eficiente el uso de la memoria compartida de la GPU. Para terminar, se resuelve el segundo de los sistemas obteniendo como resultado el vector x con la solución.

A continuación, se analiza un caso de estudio en el campo del procesado de imágenes hiperespectrales ya que una de las fases que consume mas recursos computacionales corresponde a la solución múltiple de sistemas de ecuaciones. Es en este contexto donde se motiva y evalúa nuestra rutina `CuBCholS`.

III. RX Local basado en `CuBCholS`

Una imagen hiperespectral es obtenida por un dispositivo de teledetección o sensor hiperespectral y sus pixeles se caracterizan por incluir información de un conjunto de bandas o longitudes de onda. A esta información se le denomina *la firma espectral de cada pixel*, la cual caracteriza de forma única al elemento caracterizado por ese píxel. Nuestro interés se centra en la detección de anomalías en este tipo de imágenes, lo que trata la identificación, sin conocimiento *a priori* sobre la imagen, de un conjunto reducido de pixeles cuya firma espectral es anómala cuando se compara con la de los pixeles que le rodean. Este tipo de procesado es importante, por ejemplo para la detección de incendios o puntos termales [9,10]. La imágenes hiperespectrales están caracterizadas por las dimensiones espaciales en número de pixeles ($l \times c$ donde l representa el número de líneas de la imagen y c el número de pixeles que componen cada línea) y el número de bandas de la firma espectral (b).

El algoritmo RX, desarrollado por Reed y Xiaoli, es un detector de anomalías ampliamente usado que ha demostrado ser eficaz en imágenes hiperespectrales [11, 14]. El algoritmo RX compara la firma de cada píxel con la del resto de pixeles de la imagen mediante el cálculo de la distancia de Mahalanobis, [12]. De esta forma, para cada pixel se asocia un

valor, $\delta(\mathbf{x})$, que representa una valoración de la discrepancia entre la firma espectral del pixel y de las firmas del resto de pixeles de la imagen.

En este trabajo consideramos la versión local del algoritmo RX denominada RX local o LRX, descrita en [13,15,16] que es especialmente adecuado para la localización de anomalías de pequeñas dimensiones o sub-píxel. El algoritmo LRX define una ventana de tamaño $\kappa \times \kappa$ centrada en cada pixel $\mathbf{x}$, y evalúa la distancia de Mahalanobis para el píxel central (píxel bajo estudio) con el resto de la ventana mediante la siguiente expresión:

$$\delta_{\kappa}^{LRX}(\mathbf{x}) = \mathbf{x}^T \mathbf{R}_{\kappa\times\kappa}(\mathbf{x})^{-1}\mathbf{x}, \qquad (1)$$

donde $\mathbf{R}$ es la matriz de correlación, $\mathbf{x}$ es la firma espectral del pixel (es decir un vector de dimensión igual al número de bandas de la imagen, b).

Teniendo en cuenta esta descripción, el algoritmo LRX se puede considerar como un proceso iterativo de tres etapas aplicado a cada píxel de la imagen ($\mathbf{x}$):

1. Cálculo de las matrices de correlación $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$ [18];
2. Computación del vector intermedio $y(\mathbf{x}) = \mathbf{R}^{-1}\mathbf{x}$, con el objetivo de evitar el cálculo de la inversa de forma explícita, esta etapa puede es expresada en términos de la solución de múltiples sistemas de ecuaciones lineales, ya que para cada pixel se debe resolver un sistema de dimensión b. Las matrices de correlación $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$ son simétricas y definidas positivas, por tanto la solución basada en la descomposición de Cholesky es la más ventajosa en términos de coste computacional [1,8,17].
3. Cálculo del filtro de salida $\delta(\mathbf{x}) = \mathbf{x}^T\mathbf{y}$ que asocia un valor positivo a cada píxel cuya magnitud es proporcional a la probabilidad de que el píxel forme parte de una anomalía en la imagen. Esta fase consiste en calcular un producto escalar de vectores de dimensión b para cada pixel.

Las etapas (1) y (2) requieren recursos computacionales muy elevados, ya que por cada pixel de la imagen se necesita reservar espacio en memoria para almacenar una matriz densa de dimensión b y computar cada matriz de correlación, más la solución del sistema correspondiente. Teniendo en cuenta las dimensiones típicas de las imágenes hiperespectrales que se procesan actualmente ($64 \lesssim c \lesssim 4096$), la arquitectura de las modernas GPUs ofrecen suficientes recursos para procesar concurrentemente el algoritmo LRX para los pixeles de una o varias líneas de la imagen. El kernel GPU de la primera etapa se estructura de forma que los hilos de cada bloque colaboran para calcular la matriz $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$, de esta manera, se activan tantos bloques como pixeles se procesan concurrentemente. Una vez que se obtienen las correspondientes matrices de correlación de cada píxel, se ejecuta el kernel `CuBCholS` para resolver los múltiples sistemas con esquema similar, es decir, cada bloque de hilos resuelve cada sistema, activando de nuevo el mismo número de bloques. Cabe destacar que en esta aplicación de la rutina $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$, no es necesario incluir procesos de comunicación GPU-CPU que tanto penalizan el rendimiento de este tipo de plataformas, ya que las matrices se calculan en la GPU y se almacenan en su memoria global.

IV. Evaluación Experimental

La evaluación de `CuBCholS` se ha llevado a cabo en una plataforma equipada con un procesador multicore Intel Xeon E5640 (2.67 GHz) con 12 GB de memoria RAM. La GPU que se conecta al sistema es una tarjeta NVIDIA GeForce GTX 680 (con la arquitectura GK104 también conocida como *Kepler*) [19], con 8 multiprocesadores y 192 núcleos por multiprocesador (dispone de un total de 1536 cores), frecuencia de reloj 1.06 GHz, y 2 GB de memoria global.

Para evaluar el rendimiento de `CuBCholS`, hemos considerado una imagen real adquirida por el sensor AVIRIS, operado por el *Jet Propulsion Laboratory* de NASA. La imagen fue tomada el 16 de septiembre de 2001, justo 5 días después de los ataques terroristas sobre el *World Trade Center*. La imagen original está descrita en [20]. Consideramos unas dimensiones espaciales de 512×512 pixeles y 224 bandas, por tanto, de acuerdo con la notación que hemos introducido tenemos $l = 512$, $c = 512$ y $b = 224$. Además, a partir de esta imagen también hemos construido otros tests variando el número de bandas y el número de columnas con el objetivo de evaluar nuestra rutina `CuBCholS`.

La Tabla I muestra los resultados obtenidos al evaluar la implementación de nuestra rutina `CuBCholS` ejecutada en la GPU, frente a una implementación en secuencial de la misma función usando un core de la CPU. Para esto y como estudio preliminar, partiendo de la imagen descrita anteriormente, la rutina ha sido evaluada para el procesado de una línea de la imagen, no solo teniendo en cuenta todas las columnas y todas las bandas de la imagen, sino también han sido consideradas versiones reducidas de las imagenes test con distintos valores de columnas (c, número de sistemas a resolver) y de bandas (b, dimensión de los sistemas).

Aunque los tiempos de ejecución son pequeños, los resultados de la Tabla I nos muestran que se ha reducido el tiempo de cómputo con un factor de aceleración que va desde 10 a 13,9. Tengase en cuenta que las imagenes reales están compuestas por cientos o miles de lineas (l), y por tanto el tiempo de procesamiento para la segunda etapa de LRX es de varios ordenes de magnitud superior al mostrado en la Tabla I. Por tanto, `CuBCholS` supone una mejora significativa para el algoritmo LRX ya que resuelve la etapa más costosa dentro del mismo, lo que demuestra los beneficios del uso de la rutina `CuBCholS` sobre una GPU, y justifica su desarrollo e implementación.

V. Conclusiones

Este trabajo describe y estudia una nueva implementación de solución de múltiples sistemas lineales

TABLA I

Tiempo de cómputo (en milisegundos) para las etapas más importantes de LRX, en el procesado de una línea de la imagen hiperespectral real WTC, usando diferente número de bandas (b) y de columnas (c).

columnas	bandas	`CuBCholS`(GPU)	Solución Cholesky (CPU)	SpeedUp
512	128	36,5	508,3	13,9 ×
	160	66,1	685,2	10,3 ×
	192	121,9	1501,4	12,3 ×
	224	155,6	2030,2	13,1 ×
256	128	18,2	254,4	13,9 ×
	160	32,4	342,8	10,5 ×
	192	61,2	751,3	12,3 ×
	224	84,6	1015,2	12,1 ×
128	128	10,8	127,9	11,8 ×
	160	16,1	171,4	10,1 ×
	192	31,4	375,6	11,9 ×
	224	42,9	507,7	11,8 ×

de dimensiones medias, cuya solución se basa en la factorización de Cholesky. Esta implementación, denominada `CuBCholS` , es especialmente apropiada para resolver cientos de sistemas lineales definidos positivos de tamaño medio. En nuestra implementación, asignamos cada una de las factorizaciones de Cholesky junto con su correspondiente resolución del sistema lineal, a un bloque CUDA de la GPU. `CuBCholS` es aplicada y evaluada en el contexto de la detección de anomalías sobre una imagen hiperespectral real, basada en el algoritmo LRX. Los resultados preliminares muestran la ventaja de usar esta rutina en términos de rendimiento, incluso con imágenes de reducidas dimensiones. En este caso, hemos desarrollado una implementación a medida, que cumple con los requisitos específicos que requiere el procesado de imágenes hiperespectrales, que ha sido el objeto de este trabajo. Como linea de trabajo futura, se pretende generalizar esta rutina para que sea válida para otras aplicaciones que muestren como requisitos la solución de múltiples sistemas independientes. En esta linea, se pretende flexibilizar la rutina para que se pueda adaptar a distintos tamaños de problemas, de acuerdo con la carga computacional de la aplicación y los recursos computacionales disponibles en la plataforma GPU especifica.

Acknowledgements

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (TIN2008-01117, TIN2011-23283, TIN2012-37483-C03-01 and AYA2011-29334-C02-02), Junta de Andalucía (P10-TIC-6002 y P011-TIC7176), Junta de Extremadura (PRI09A110 y GR10035), CAPAP-H4 (TIN2011-15734-E) y por los fondos FEDER.

Referencias

[1] G. Golub and C. V. Loan, *Matrix Computations Third Edition*, The Johns Hopkins University Press, 1996.

[2] A. Quarteroni, R. Sacco and F. Saleri, *Numerical Mathematics*. Springer, ISBN 0-387-98959-5, 2000.

[3] *Matrix Algebra on GPU and Multicore Architectures*, http://icl.cs.utk.edu/magma

[4] NVIDIA Cublas manual, *http://docs.nvidia.com/cuda/cublas/index.html*

[5] M. J. Anderson, D. Sheffield and K. Keutzer, *A Predictive Model for Solving Small Linear Algebra Problems in GPU Registers*, IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS). (2012) p 2–13.

[6] NVIDIA CUDA C Programming Guide, *http://developer.nvidia.com/category/cuda-zone*, October, 2012.

[7] R. Farber, *CUDA, Application Design and Development*, Morgan Kaufman, 2010.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C (2nd ed.): the art of scientific computing*, Cambridge University Press, 1992.

[9] G. Shaw and D. Manolakis, "Signal processing for hyperspectral image exploitation," *IEEE Signal Processing Magazine*, vol. 19, pp. 12–16, 2002.

[10] A. Plaza and C.-I. Chang, *High performance computing in remote sensing*, Boca Raton: CRC Press, 2007.

[11] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*, Norwell, MA: Kluwer. (2003).

[12] J. Richards and X. Jia, *Remote Sensing Digital Image Analysis: An Introduction*. Springer, 2006.

[13] J. M. Molero, E. M. Garzón, I. García and A. Plaza, *Analysis and Optimizations of Global and Local Versions of the RX Algorithm for Anomaly Detection in Hyperspectral Data*, IEEE JSTARS. vol. 6, no. 2, pp. 801-8014, April 2013.

[14] I. S. Reed and X. Yu, *Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution.*, IEEE Trans. Acoustics Speech and Signal Processing. 138 (1990) 1760–1770.

[15] Y. P. Taitano, B. A. Geier, and K. W. B. Jr, *A locally adaptable iterative RX detector*, EURASIP Journal on Advances in Signal Processing. **10** (2010).

[16] S. Matteoli, M. Diani, and G. Corsini, *A tutorial overview of anomaly detection in hyperspectral images*, IEEE Aerospace and Electronic Systems Magazine. 25 (2010) 5–28.

[17] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. V. D. Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*. Society for Industrial and Applied Mathematics Philadelphia, 1990.

[18] J. M. Molero, E. M. Garzón, I. García, E. S. Quintana and A. Plaza, *Accelerating the KRX Algorithm for Anomaly Detection in Hyperspectral Data on GPUs*, Proc. of the CMMSE 2012.

[19] NVIDIA GeForce GTX 680 Whitepaper, *http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf* (2012).

[20] R. Green, *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment*, vol. 65, no. 3, pp. 227–248, 1998.

Arquitectura paralela para la eliminación del wandering y el ruido en señales ECG fetales

Luis Parrilla[1], Encarnación Castillo [2], Diego P. Morales [3], Víctor Unai [4], Francisca S. Molina [5], Jesús Florido [6] y Antonio García [7]

Resumen— **El procesamiento de señales electrocardiográficas presenta dificultades derivadas de las múltiples fuentes de ruido, especialmente en el caso de electrocardiogramas fetales (FECG). El uso de varios electrodos para tomar las muestras es una de las posibilidades empleadas para reducir el número de falsos positivos y negativos en la detección de los distintos parámetros del FECG. En el caso de utilizar instrumentación portátil en consulta, el uso de una procesador único dificulta el procesamiento de las señales en tiempo real. En este artículo de propone una arquitectura paralela que permite la gestión en tiempo real de FECGs tomados a partir de varios sensores para instrumentación portátil.**

Palabras clave— **ECG, ECG fetal, Wavelets, DWT, Paralelización.**

I. Introducción

La adquisición de señales electrocardiográficas (ECG) a través de la piel en humanos presenta varias dificultades relacionadas con las distintas fuentes de ruido que producen una contaminación de las señales ECG. Estos ruidos pueden tener un origen instrumental y fisiológico [1], y se ven agravados por la necesidad de amplificadores de instrumentación de elevada ganancia, especialmente en el caso de la medida de ECGs fetales adquiridas a través del abdomen materno [2]. De estos ruidos, los más significativos son los producidos por la fuente de alimentación y el *wandering* de la línea base (BW). El resto de ruidos pueden presentar un ancho de banda apreciable y provenir de procesos estocásticos complejos, generando una distorsión de la señal ECG. El BW y el resto de ruidos con gran ancho de banda presentan dificultades para su supresión utilizando hardware analógico, siendo mucho más eficaces esquemas de procesamiento software. Sin embargo, las plataformas hardware deben de ser el objetivo para conseguir sistemas portátiles de adquisición y procesamiento de ECGs. Así, en este artículo se estudian los procedimientos para la implementación hardware de sistemas digitales que permitan una supresión sencilla de los ruidos de gran ancho de banda mencionados. Una de las posibilidades utilizadas en monitorización de FECGs para mejorar la adquisición de las señales es utilizar varios canales, de manera que se disponga de varias muestras en cada instante de tiempo que faciliten el reconstruir todos los eventos que hayan ido sucediendo aunque el ruido los haya enmascarado en un determinado canal. Sin embargo, el procedimiento estándard consiste en estudiar a posteriori las señales obtenidas en los distintos canales y realizar una inspección visual de las mismas. En el caso de realizar una monitorización en tiempo real en una consulta, debe tenerse una señal lo más fiable posible de lo que está sucediendo en ese momento, y debe ser el instrumento el que realice el procesamiento de las señales de los distintos canales y decida qué debe presentarse en pantalla para que sea de la mayor utilidad posible. Esta cantidad de procesamiento es difícil de mantener en tiempo real si se utiliza una única unidad de procesamiento, que además debe de ser portátil. En este artículo se propone utilizar una arquitectura hardware específica que permita realizar un procesamiento paralelo de las señales suministradas por cada canal.

II. Eliminación de ruido en FECGS

La Transformada Wavelet [3] ha demostrado su utilidad en la eliminación del ruido de baja frecuencia [4]. Teniendo en cuenta que el BW presenta frecuencias entre 0.15 y 0.3Hz, un procedimiento para la supresión del BW consiste en la aplicación de la descomposición *wavelet* hasta un nivel de resolución en el que la secuencia de aproximación puede capturar el BW, restar esta parte de la señal ECG sin procesar, y calcular la reconstrucción *wavelet*. En estas bandas de frecuencia, el nivel de resolución debe ser tal que la secuencia de aproximación capture las componentes del ECG para frecuencias menores que 1Hz. Por otra parte, la supresión de ruido mediante *wavelets* está mostrando una gran efectividad sin que sea preciso realizar una tratamiento complejo de la señal con ruido [5], al localizar las principales propiedades espaciales y de frecuencia de la señal en un número limitado de coeficientes *wavelet*. Además, debido a la ortogonalidad de la transformada, el ruido blanco se distribuye uniformemente entre todos los coeficientes de detalle, garantizando que la contracción *wavelet* produce una reducción del ruido, a la vez que se conservan las características más pronunciadas (picos del complejo QRS).

Debido a la similitud de las estructuras *wavelet*

[1]Dpto. Electrónica y Tecnología de Computadores, Univ. de Granada, e-mail: lparrilla@ditec.ugr.es.
[2]Dpto. Electrónica y Tecnología de Computadores, Univ. de Granada, e-mail: encas@ditec.ugr.es.
[3]Dpto. Electrónica y Tecnología de Computadores, Univ. de Granada, e-mail: diegopm@ugr.es.
[4]Dpto. Electrónica y Tecnología de Computadores, Univ. de Granada, e-mail: vunai@correo.ugr.es.
[5]Hospital Universitario San Cecilio, Granada, e-mail: fsoniamolina@gmail.com.
[6]Dpto. de Obstetricia y Ginecología, Univ. de Granada, e-mail: jflorido@ugr.es.
[7]Dpto. Electrónica y Tecnología de Computadores, Univ. de Granada, e-mail: agarcia@ditec.ugr.es.

para la eliminación del BW y del ruido, en este trabajo se propone la aplicación simultánea de las dos técnicas, realizando todo el procesamiento en un solo paso. Con este esquema de procesamiento se consigue un ahorro importante en los recursos y el tiempo de procesamiento, facilitando la posterior implementación hardware. Los pasos a seguir para la aplicación de esta propuesta son los siguientes:

1. *Descomposición*: Se aplica la la descomposición *wavelet* hasta un cierto nivel L, con el fin de generar los coeficientes de aproximación $a_n^{(L)}$ para la captura del BW.
2. *Puesta a cero de las aproximaciones*: La secuencia de aproximación $a_n^{(L)}$ se reemplaza por un vector con todas sus componentes a cero [4].
3. *Definición de los umbrales en los detalles*: Se selecciona el nivel M (con $M < L$) que permita distinguir apropiadamente la presencia de descargas parciales en los detalles con ruido. Además, se aplica un umbral según unas terminadas reglas en cada nivel desde $i = 1$ hasta M, a los coeficientes de detalle $d_n^{(i)}$ para conseguir una supresión óptima del ruido [5].
4. *Reconstrucción*: Se calcula la reconstrucción *wavelet* basada en la puesta a cero de las aproximaciones en el nivel L, los detalles modificados en los niveles 1 a M, y los detalles originales de los niveles $M + 1$ a L, obteniendo la señal con el BW corregido y limpia de ruido.

Así, mediante una selección adecuada de los parámetros, es posible obtener una supresión simultánea del BW y del ruido, utilizando la técnica *wavelet* presentada. La selección de la *familia wavelet* debe basarse en la similitudes entre la estructura básica de un ECG y las funciones *wavelet* empleadas, y el tipo de procesamiento a realizar. Según se comentó anteriormente, para eliminar el BW es necesario elegir un nivel de resolución tal que la aproximación correspondiente capture las componentes del ECG menores que 1 Hz. Teniendo en cuenta que en cada nivel de de descomposición, la banda de frecuencia de la aproximación se divide por 2, el nivel de descomposición para la eliminación del BW se puede calcular como:

$$L = \lceil log_2(F_0) \rceil \tag{1}$$

donde F_0 es la máxima componente en frecuencia de la señal. Por otra parte, el máximo nivel para el umbral de detalle, M, depende varios factores como el SNR de la señal original, o la frecuencia de muestreo. El nivel de ruido es significativo en las subbandas de alta frecuencia del detalle, mientras que la mayoría del espectro de energía se encuentra en las subbandas de baja frecuencia [5]. Por tanto, si se desea evitar la pérdida de componentes clínicamente relevantes de la señal, tales como morfologías del PQRST, sólo deben tratarse las subbandas de alta frecuencia del detalle para la supresión del ruido.

Existen varios métodos para determinar el umbral límite [5], pero aquí se considerarán aquellos que sean más apropiados para su implementación hardware por su complejidad de cálculo, recursos necesarios, retardo, etc. Concretamente, algunos umbrales que pueden resultar aplicables son los siguientes:

- *Umbral universal*

$$Th_{un} = \sqrt{2 \cdot logN} \tag{2}$$

- *Umbral exponencial*

$$Th_{exp} = 2^{\left(\frac{i-M}{2}\right)} \sqrt{2 \cdot logN} \tag{3}$$

- *Umbral minimax* [6]

$$Th_{minimax} = 0.3936 + 0.1829 \cdot \left(\frac{log(n_i)}{log(2)}\right) \tag{4}$$

donde N es la longitud de la señal, y n_i representa el número de coeficientes en cada nivel $i = 1, ..., M$. En principio, el cálculo de estos umbrales requiere de operaciones como la raiz cuadrada o el logaritmo, que pueden necesitar de algoritmos específicos en punto fijo para su implementación hardware. Sin embargo, para una longitud prefijada de la señal, N, es posible precalcular el número de coeficientes en cada nivel, n_i. Así, en nuestro modelo se utilizan umbrales precalculados desde N y M, sin que exista ninguna otra dependencia de la señal a limpiar. La siguiente expresión para estimar la varianza del ruido:

$$\sigma_i = media\left(\left|d_n^{(i)}\right|\right)/0.6745 \tag{5}$$

permite elegir entre reescalado *simple* o *multiple* [7]. El reescalado no se puede precalcular porque depende de los detalles, pero las operaciones a realizar presentan mucha menor complejidad que otras aproximaciones de la varianza.

III. Modelado en punto fijo para implementación harware

Utilizando las técnicas presentadas en la sección anterior, se han desarrollado modelos software para los procedimientos de supresión de BW y ruido en señales ECG basados en la DWT. Los modelos software permiten realizar un análisis y ajuste rápido de los parámetros, así como una evaluación de las técnicas desarrolladas, mientras que la traslación de estos modelos a aritmética de punto fijo permite replicar el funcionamiento de las implementaciones hardware y analizar parámetros tales como el truncado, numero de muestras de la ventana de datos, frecuencias de muestreo, etc. Para el modelado en punto fijo, la tarea más importante a realizar es la determinación de las longitudes de palabra a utilizar en las distintas variables involucradas en el sistema de procesamiento. Por otra parte, se necesitan dos módulos específicos para el cálculo de la DWT y la transformada inversa (IDWT), que deben diseñarse para obtener las mejores prestaciones con la *wavelet* concreta que se vaya a utilizar. Los recursos que se precisan para el modelado hardware, son básicamente una lógica de control y una serie de registros. Los filtros paso-baja y paso-alta necesarios

se modelan utilizando bancos de filtros FIR y estructura directa, mientras que el número de etapas queda determinado por la función *wavelet*.

IV. Parámetros de entrada

A continuación se detallan brevemente los parámetros de entrada para el modelo de supresión simultánea de BW y ruido:

- *Función Wavelet*: El modelo de punto fijo propuesto permite la utilización de diversas familias *wavelet* como son: Daubechies, Coiflets, Symlets, Biorthogonal y Reverse Biorthogonal.
- *Niveles de descomposición L y M*: Nuestro modelo calcula automáticamente los niveles de descomposición para la eliminación del BW según la ecuación (1). Sin embargo, defido a la dificultad de establecer a priori el nivel máximo óptimo M para aplicar los umbrales de detalle, el modelo permite elegir este parámetro.
- *Umbrales y reescalado*: Sólo se han considerado los umbrales con bajo costo computacional. El modelo propuesto incluye la posibilidad de elegir uno de los tres umbrales definidos anteriormente: *universal*, *exponencial* o *minimax*.
- *Reglas de aplicación de los umbrales*: El modelo permite seleccionar si se utilizan umbrales *soft* o *hard*. Las operaciones necesarias de pueden implementar fácilmente en aritmética de punto fijo.

V. Implementación hardware

El modelo de punto fijo propuesto se ha implementado usando VHDL sobre FPGAs de Xilinx. El diagrama de bloques general de la arquitectura propuesta puede verse en la Fig. 1. La arquitectura se ha diseñado para la realización de aplicaciones en tiempo real basadas en la adquisición y procesamiento de ventanas de datos ECG de n-muestras. La implementación consta de los siguientes bloques:

- *Buffer de entrada-RAM de descomposición.* Este bloque se utiliza para la adquisición y procesamiento de cada una de las ventanas de datos ECG. Consta de dos memorias, llamadas *buffer de entrada* y *RAM de descomposición*. El buffer de entrada almacena la ventana de datos muestreados en una pila. Esta ventana de datos se copia en la RAM de descomposición, que se va refrescando con los coeficientes obtenidos a partir del bloque de descomposición *wavelet*. Se evitan así conflictos en los instantes de tiempo en los que la ventana de datos se está procesando y a la vez se produce la adquisición de los datos de una nueva ventana de datos.
- *Procesador wavelet.* Este módulo constituye un bloque IP que implementa las operaciones de descomposición y reconstrucción para la DWT. El bloque consiste en tres unidades de control, dos bloques de procesamiento y una FIFO. Las primeras dos unidades de control permiten configurar los bloques de descomposición y reconstrucción, mientras que la tercera unidad de control permite la sincronización de los dos bloques de procesamiento y el control de los accesos a memoria.

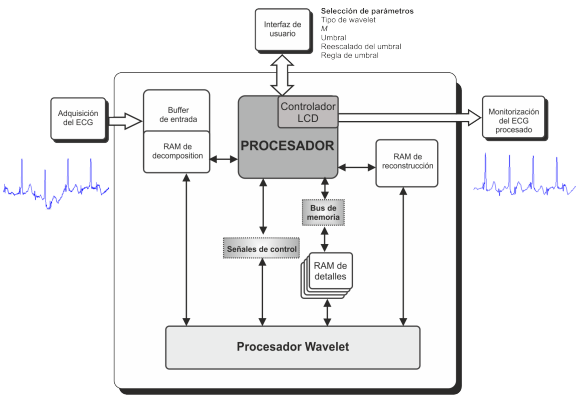


Fig. 1. Diagrama de bloques de la arquitectura propuesta

- *RAM de detalle.* Cada una de estas memorias se utiliza para almacenar los coeficientes de detalle de cada uno de los niveles de descomposición. Puesto que se necesitan L niveles de descomposición *wavelet*, se precisan de L RAMs de detalle. El acceso a estas memorias se producirá durante el proceso de reconstrucción para acceder a las L secuencias de detalle.
- *Memoria de reconstrucción.* La señal reconstruida en cada nivel se almacena en esta memoria para ir siendo procesada por el bloque de reconstrucción. Cuando el proceso de reconstrucción termina, se almacena la señal procesada en esta memoria.
- *Bloque procesador.* El procesador permite implementar las operaciones de control necesarias, seleccionar los parámetros involucrados en la supresión del BW y el ruido, calcular los detalles modificados, aplicar el procesamiento propuesto para la eliminación del BW y el ruido y gestionar la visualización de la señal procesada. Este bloque integra un sistema compuesto por un microcontrolador, una memoria ROM, una memoria RAM, un controlador LCD para la representación en tiempo real, puertos de entrada/salida, interfaz para los accesos de descomposición y reconstrucción y la lógica requerida para la conexión entre estos módulos.

VI. Resultados para el algoritmo de eliminación de ruido

Para estudiar la supresión BW junto con la eliminación del ruido de forma simultánea, el mejor procedimiento es la inspección visual ante la dificultad de definir parámetros cuantitativos. Se utilizaron las bases de datos *DaIsy Dataset* [8] y *Physionet Dataset* [9] para evaluar el procedimiento de supresión simultánea de BW y ruido. Las monitorizaciones de *Physionet Dataset* son de la base de datos *Non-Invasive Fetal ECG Database* incluyendo dos canales torácicos y cuatro abdominales muestreados a 1kmps y una longitud total de 60 segun-

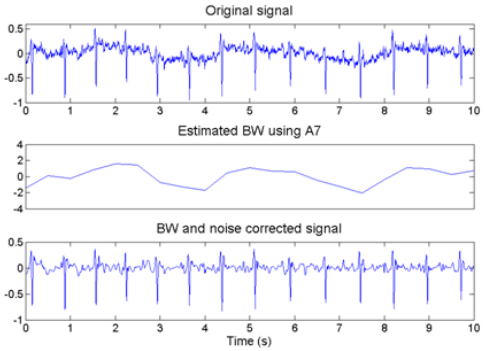


Fig. 2. Canal 4 de la base de datos *DaIsy*, BW estimado y señal con BW y ruido corregidos

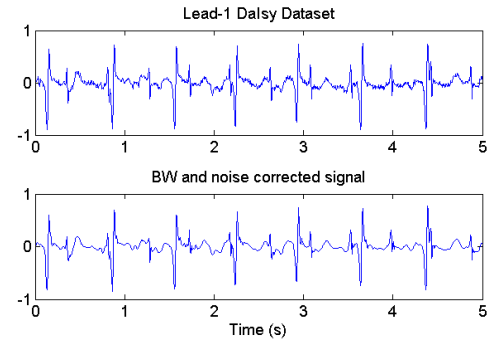
dos. Para estas señales, los parámetros seleccionados fueron: función wavelet db6, $M = 3$, *umbral universal*, *regla soft*, *reescalamiento simple* para la *DaIsy dataset*, y *reescalado m´ultiple* para la *Physionet Dataset*. La Fig. 2 incluye los resultados obtenidos para el canal 4, donde se muestra la señal original el BW estimado, y la señal corregida en BW y ruido. En la Fig. 3 se incluyen resultados para el canal 1 (a) y el canal 2 (b). Finalmente, la Fig. 4 se muestra un ejemplo de resultados para la señal ecgca 746 de la base de datos *Physionet Dataset* incluyendo el detalle de uno de los complejos QRS fetales antes y después de su procesamiento. Estas figuras muestran que en todas estas señales se ha corregido el BW y se ha eliminado el ruido, reteniendo las principales características de la señal ECG abdominal, así como los complejos QRS fetales, de gran importancia para futuras extracciones de parámetros de utilidad diagnóstica [10].

VII. Paralelización del procesado para ECG

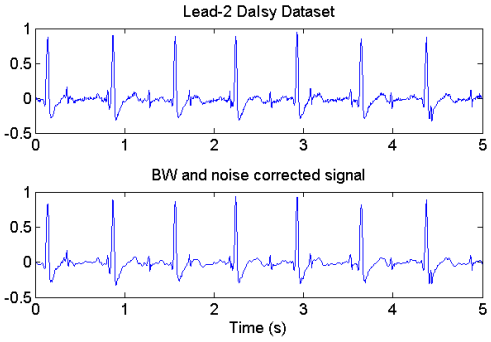
Una de las técnicas utilizadas para mejorar la detección de los parámetros en electrocardiogramas consiste en usar varios sensores para capturas las señales, disponiendo asì de varios canales. A partir de las señales capturadas en los diversos canales, el médico puede seleccionar en cada instante de tiempo el canal que presenta menos problemas de ruido para extraer los parámetros característicos. En todo caso, este estudio se realiza *a posteriori*, analizando las señales capturadas. El objetivo planteado en este artículo es que sea el propio instrumento el que seleccione la mejor señal en cada momento y presente en pantalla el electrocardiograma en las mejores condiciones posibles para poder efectuar un diagnóstico en consulta y en tiempo real. Para ello se propone procesar de forma paralela las señales de cada uno de los canales, y a partir de las señales ya limpias de BW y de ruido, decidir qué se presenta en pantalla para una mejor ayuda al diagnóstico.

A. Parámetros para la paralelización

El tamaño de la ventana de datos, seleccionado en función del número de niveles necesarios en la descomposición wavelet, es el que va a determinar el

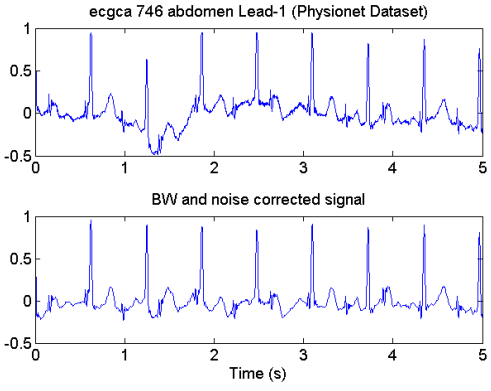


(a)

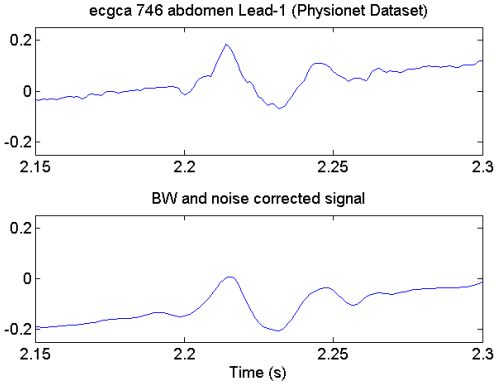


(b)

Fig. 3. Señales de los canales 1 y 2 de la base de datos DaIsy y señal con BW y ruido corregidos



(a)



(b)

Fig. 4. Canal 1 abdominal, señal ecgca 746 de la base de datos Physionet, BW y ruido corregidos (a) y detalle de la señal (b)

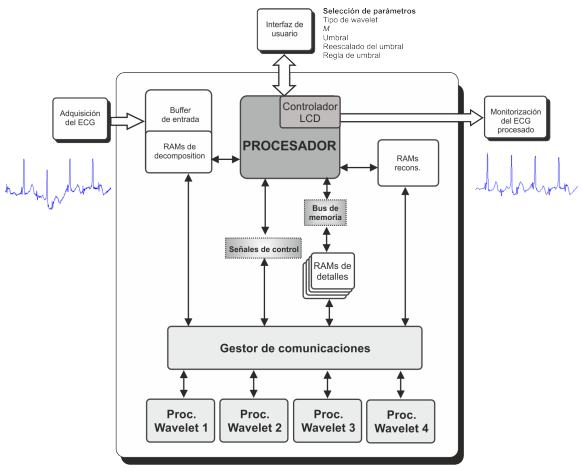


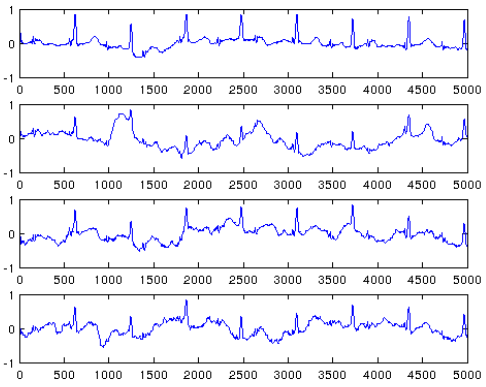
Fig. 5. Diagrama de bloques de la arquitectura para procesamiento paralelo de los canales

tamaño de grano para el procesamiento paralelo. De esta forma, en la arquitectura paralela propuesta, se va a tener un procesador *wavelet* por cada uno de los canales, que realizará las tareas de supresión del BW y del ruido para la ventana de datos que se esté procesando en ese momento. Una vez completada esta tarea, el gestor de comunicaciones almacenará la ventana de datos, ya procesada, en la memoria RAM correspondiente a cada uno de los procesadores *wavelet*. A continuación el nodo central (procesador) calculará para cada uno de los puntos a presentar y/o registrar, qué valores son los más adecuados, según el algoritmo de decisión que se propone en la sección siguiente. La Figura 5 muestra la arquitectura propuesta.

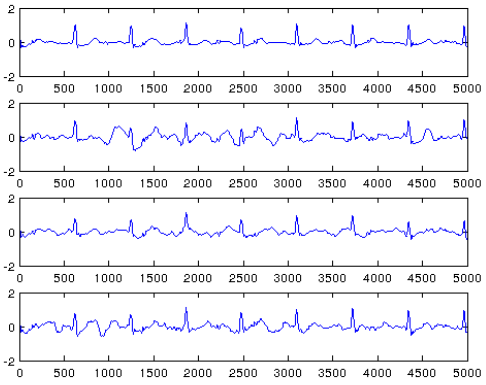
B. Algoritmo de decisión

El algoritmo de decisión debe decidir en cada instante de tiempo cual es el valor más adecuado para ir presentando en pantalla y registrando como resumen del ECG fetal obtenido. Se proponen dos algoritmos para ello:

- *Cálculo de la media.* Como primera posibilidad, se puede utilizar la media de los valores obtenidos para cada canal, después de haber procesado y normalizado las señales adquiridas.
- *Agrupación por desviación típica.* En principio la media puede dar buenos resultados, especialmente si el número de canales es grande. Sin embargo, si hay pocos canales, y se producen artefactos de gran magnitud en alguno de ellos, la media puede no ser el mejor estadístico. EN estos casos se propone un algoritmo sencillo que no requiere de elevados tiempos de ejecución, que a su vez es paralelizable, y que además se puede implementar en hardware. La idea es calcular las medias y las desviaciones típicas dos a dos de los distintos canales. Suponiendo que los valores más próximos entre sí deben de ser los correctos, se selecciona la pareja de canales con menor desviación típica y se obtiene como representación la media de ambos. Como sólo interesa ordenar las medias por desviación típica, se pueden calcular las varianzas en su lugar, requiriendo así menos recursos de cálculo



(a)



(b)

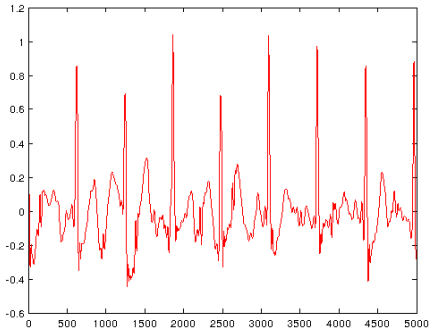
Fig. 6. Canales abdominales 1 a 4, señal ecgca 746 de la base de datos Physionet, originales (a) y después de su procesamiento y escalado (b)

En la siguiente sección se muestran los resultados obtenidos al aplicar estos dos algoritmos a distintas señales.

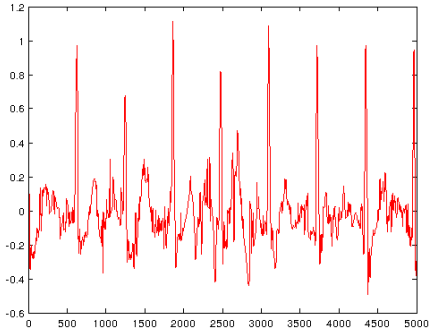
VIII. Resultados para el procesamiento paralelo multicanal

Para comparar los resultados obtenidos por los dos algoritmos se han utilizado las señales ecgca 746 y 840 de la base de datos *Physionet Dataset*. En la Fig. 6 (a) se muestran las señales sin procesar de los cuatro canales torácicos de la ecgca 746, y en la (b) esos cuatro canales procesados. La Fig 7 (a) presenta la señal obtenida a partir de la media de los cuatro canales, y la (b) el resultado usando el algoritmo de agrupación por desviación típica. En la Fig. 8 (a) y (b) se muestran estos mismos resultados para la señal ecgca 840.

En ambos casos se observa que el uso de la media proporciona una señal más suavizada que resulta de mayor utilidad para la observación en tiempo real. No obstante, la señal obtenida a partir de la media de los datos con menor desviación típica, conserva más picos y características de la señal que pueden proporcionar más información en un análisis poste-

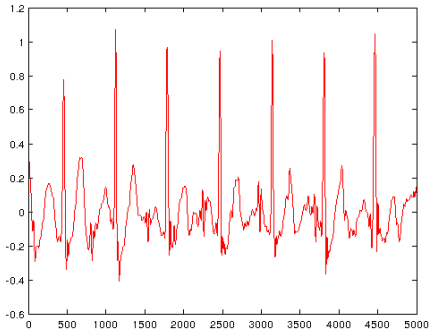


(a)

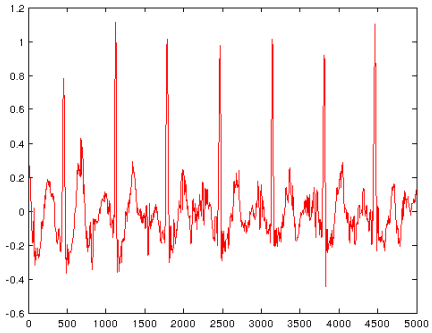


(b)

Fig. 7. Procesamiento conjunto de los cuatro canales de la ecgca 746 usando la media (a) y el agrupamiento por desviación típica (b)



(a)



(b)

Fig. 8. Procesamiento conjunto de los cuatro canales de la ecgca 840 usando la media (a) y el agrupamiento por desviación típica (b)

rior de los datos registrados. En todo caso se obtiene un resumen instantáneo de la señal obtenida por los cuatro canales, reuniendo la información que proporcionan todos ellos en una sola gráfica, sin que se pierda ninguno de los picos correspondientes a las señales electrocardiográficas.

IX. Conclusiones

En este artículo se ha presentado un sistema para la eliminación del ruido en señales ECG que introduce la novedad de realizar de forma simultánea mediante procesamiento *wavelet*, la eliminación del BW y del ruido, con una considerable reducción de los recursos hardware a utilizar. El modelo permite su ajuste utilizando diferentes parámetros permiten adaptarlo a las características específicas de la señal ECG a estudiar. Los resultados presentados para señales ECG sintéticas validan el procedimiento. Asimismo, se ha propuesto una arquitectura paralela que permite el procesamiento en tiempo de real de varios canales simultáneamente, obteniendo una gráfica resumen de todos los canales, útil para el desarrollo de aplicaciones ECG portátiles para usar en consulta.

Agradecimientos

Este trabajo ha sido financiado parcialmente por CEI-BIOTIC Granada a través del proyecto 2013/81.

Referencias

[1] J.G. Webster (Ed.), *Medical Instrumentation, Application and Design*, John Wiley & Sons,Inc., 1995

[2] D.P. Morales, A. García, et al., "Flexible ECG acquisition system based on analog and digital reconfigurable devices," Sensors & Actuators: A. Physical, Vol. 165, No. 2, pp. 261–270, 2011.

[3] S.G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation", IEEE Trans. On Pattern Recongition and Machine Intelligence, Vol. 11, No. 7, pp. 674–693, 1989.

[4] E.C. Karvounis, C. Papaloukas, D.I. Fotiadis et al.: "An automated methodology for fetal heart rate extraction from the abdominal electrocardiogram", IEEE Trans. Information Technology in Biomedicine, Vol. 11, No. 6, pp. 628–638, 2006.

[5] L.N. Sharma, S. Dandapat, A. Mahanta: "ECG signal denoising using higher order statistics in Wavelet subbands", Biomedical Signal Processing and Control, Vol. 5, No. 3, pp. 14–22, 2010.

[6] S. Sardy: "Minimax threshold for denoising complex signals with waveshrink, IEEE Trans. Signal Processing", Vol. 48, No.4, pp. 1023–1028, 2000.

[7] H.G.R. Tan, A.C. Tan, P.Y. Khong, and V.H. Mok: "Best Wavelet Function Identification System for ECG signal denoise applications", Inter. Conf. on Intelligent and Advanced Systems, pp. 631–634, 2007.

[8] B. De Moor: "Database for the identification of systems (DaISy)". [Online]. `http://homes.esat.kuleuven.be/~smc/daisy/`, 2010.

[9] A.L. Goldberger, L.A.N. Amaral, L. Glass, J.M. Hausdorff, PCh Ivanov,R.G. Mark, J.E.Mietus, G.B. Moody,C-K Peng, H.E. Stanley: "PhysioBank PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals.," Circulation, Vol. 12, No. 101, e215-e220, [Circulation Electronic Pages; `http://circ.ahajournals.org/cgi/content/full/101/23/e215`; `http://physionet.org/physiobank/database/nifecgdb/`. 2000

[10] R. Sameni and D.C. Gari, "A Review of Fetal ECG Signal Processing; Issues and Promising Directions," The Open Pacing, Electrophysiology & Therapy Journal, Vol. 3, No. 1, pp. 4–20, 2010.

Autonomous image segmentation method using a Spartan 3 and a Spartan 6 FPGAs

J. M. Montañana, [1] D. Sanchez-Benítez, [2] and J. M. de la Cruz [3]

Resumen— **In this paper, we describe an implementation of a segmentation method. Also results when implementing it on two different generations of Xilinx Spartan FPGA are shown.**

The design is based on processing only with hardware without using any processor with a sequential program. Such purpose was based on provide for future works a small portable autonomous system, which will work without need any communication with a base station, but at same time process video with a low power consumption.

Palabras clave— **vision application, segmentation, FPGA, parallel processing, reconfigurable logic.**

I. Introduction

THE image processing objective in this paper is a segmentation method. Typically, a segmentation process, used by many the vision application, consists of extracting features or regions from an image. In particular, we will use it to slice the image into regions based on the luminance on each pixels.

Figure 1 shows an example of the use of segmentation, which can be useful for locating an object in the image.

There are been published a large number of image segmentation algorithms, for instance [18], [19], [20], and different algorithm libraries that can be found on software like MATLAB, PhotoShop or OpenCV. However, we require a fast method. But, we also require it be simple because we need that it fits inside a single FPGA with other modules that we are developing.

In particular, We are interested to build a segmentation process for identifying a particular logo designed in our research department, and distinguish it among any other possible objects in the image. Such logo can be shown in the figure 2 which is currently in process of being patented.

[1]Dpto. de Arquitectura de Computadores, Univ. Complutense de Madrid, e-mail: jmontanana@fdi.ucm.es
[2]Dpto. de Arquitectura de Computadores, Univ. Complutense de Madrid, e-mail: davisanc@fis.ucm.es
[3]Dpto. de Arquitectura de Computadores, Univ. Complutense de Madrid, e-mail: jmcruz@fis.ucm.es

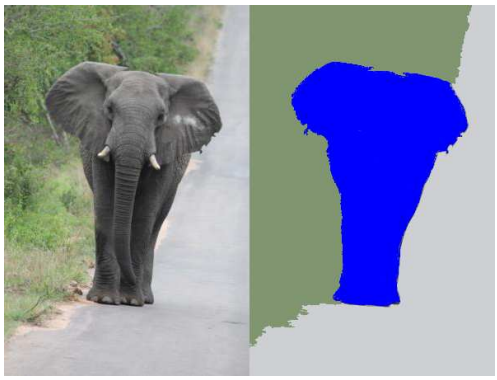


Fig. 1. Example of segmentation of an image.

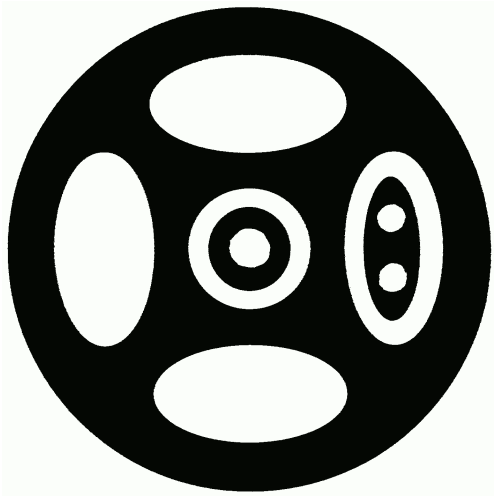


Fig. 2. Specific logo to be segmented, which patent is pending to be solved.

The simplicity on colours of the figure 2 allows to implement a simple and fast segmentation process based on binarizing previously the image captured from the camera. The binarization process, that we applied consists in this paper, on assign a value to all image pixels which brightness exceed a critical value, and assign every other pixel to the other value.

II. Related work

Shape optimization is at the core of several computer vision problems, and image segmentation is part of it. Algorithms usually used when facing the problem are level set methods [24], [27], graph cuts [25] and convex relaxation [23]. When dealing with scene recognition and analysis, a possible approach is the interactive motion segmentation. [26] takes this approach and propose to use spatially varying affine motion model parameter distributions combined with minimal guidance via user drawn scribbles. Several algorithms and generalized techniques have been developed as part of image segmentation, but the simplest might be the threshold method, based on a selected threshold to gray-scaled images before being binarized. Obviously, the value chosen as the threshold is a key point in the success of the technique.

In this paper we are interested in the recognition of an specific pattern. Pattern recognition techniques are concerned with the theory and algorithms

of putting abstract objects, e.g., measurements made on physical objects, into categories.

III. System design and architecture

In this section we describe the implemented system. Notice that the video will be processed entirely in the autonomous system, without requiring to use a computer process.

The components used on the system are shown in the figure 3. Where we can see a low cost camera, which will provide the video to be processed. The video signal will be sent to a processing element, in our case will be done on a spartan FPGA. The last element will be a screen, where we can see the acquired video, and its status at each processing stage.

The complete system of camera, and the FPGA board looks like 4.(a) and 4.(b) when using the spartan3 board, and the spartan6 board, respectively.

The camera contains a image sensor, which consists on an array of sensors cells on a chip. Figure 6 shows an example of the structure of a CMOS camera sensor. Where each pixel of the image is captured at same time on the sensor cells on the array. The data for each image pixel is provided in a sequential way due to the limited number of pins on the sensor chip and the large number of pixels in the image.

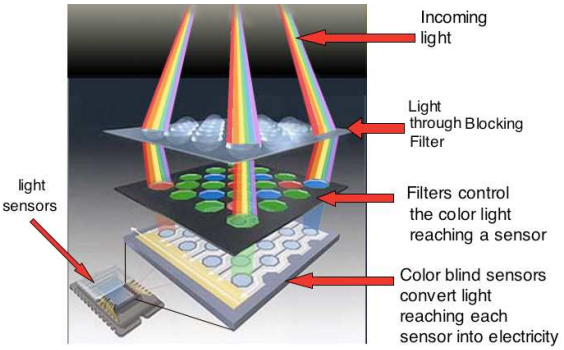


Fig. 6. Structure of a CMOS image sensor.

The second actor is the processing element is the FPGA. The reason because we choose to use a FPGA, is because its architecture allows to implement hardware parallel designs defined by the user. But also flexibility in the development of image processing algorithms, since it can be modified an unbounded number of times[1]. Actually, many researching and commercial applications used them for video processing.

[1]Unbounded number of load times in the FPGA from a computer or from the board flash memory, however the board flash memory could be updated up to 100.000 times.

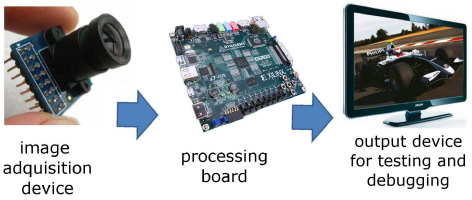
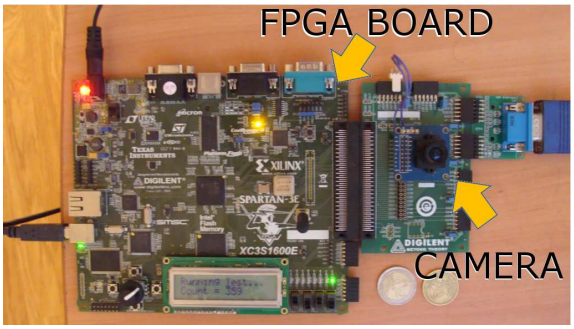
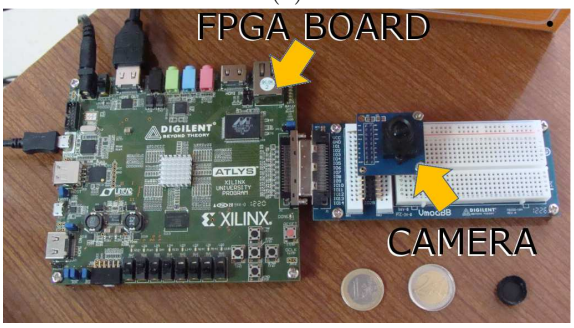


Fig. 3. Different elements in the final system are sensors, a processing element and some response elements.



(a)



(b)

Fig. 4. Spartan boards,(a) with Spartan 3, and (b) with Spartan6.

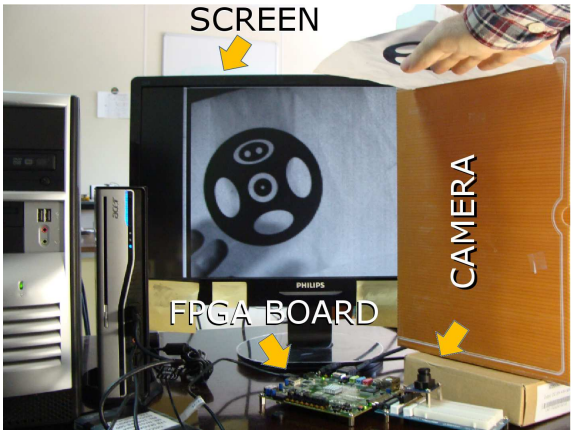


Fig. 5. The complete experimental system, composed by the screen, process board and camera.

In the figure 7 shows a simplified structure of a Spartan FPGA[21]. The structure consists of a programmable bidimensional network for interconnection components such input-output (IOBs), configurable logic (CLBs), and RAM blocks.

The CLBs, In the case of the Spartan FPGAs, contains two slices which contains each one four logic-function generators (LUTs) and 8 Flip-Flops (FFs), some may also contain shift registers, multiplexers, distributed RAM and/or some additional logic [21]. (Vendors provide information about the amount of CLBs and its characteristics on the FPGA chip datasheets).

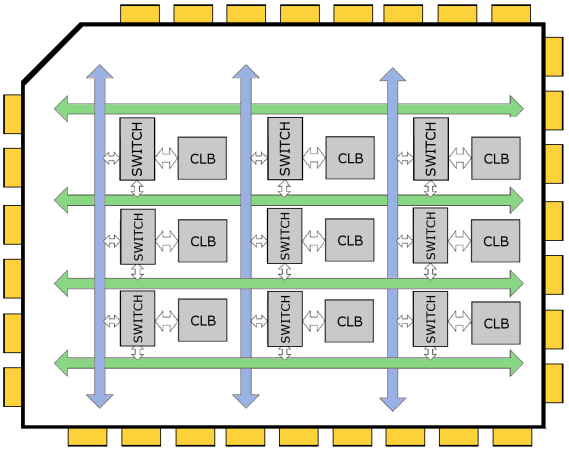


Fig. 7. Simplified structure of a Spartan FPGA.

Figure 8 shows the interconnection structure among some modules in the design. The figure shows the data path from the camera trough a sequence of modules up to be stored in the RAM memory. At same time the graphic controller will show on the screen the processed image.

The system can store the image on RAM memory after be processed on any of the modules. The purpose to make possible to show the intermediate results on the screen, for confirming the correct function of each module. In particular, such intermediate results are the original image, the result of its binarization, and final result of the segmentation.

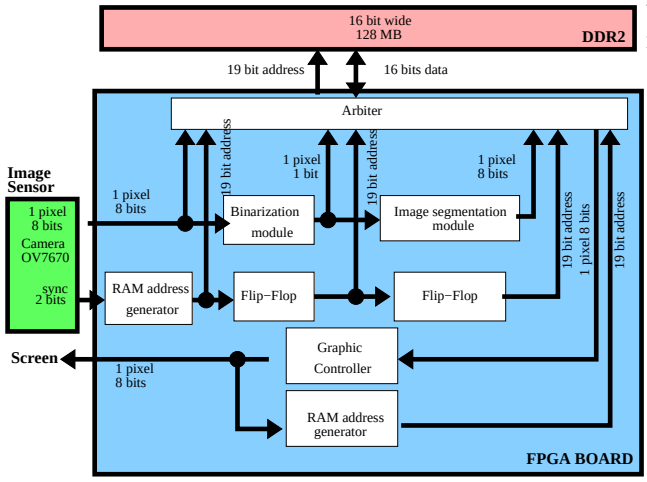


Fig. 8. System structure build inside the FPGA.

The novelty on this document, is the development of a fast segmentation module. The module has low memory requirements, just only store 2 values for each pixel line, and one additional value for each possible area. The first value is one bit, which corresponds to the *binarized* image (referred as bit *Array B* on figure 9). The second of those values are a intermediate *Code* (referred as *Array C* in the figure 9), and the last value is the final result colour value of the *segmentation* (referred as *Array S* in figure 9) for each area code.

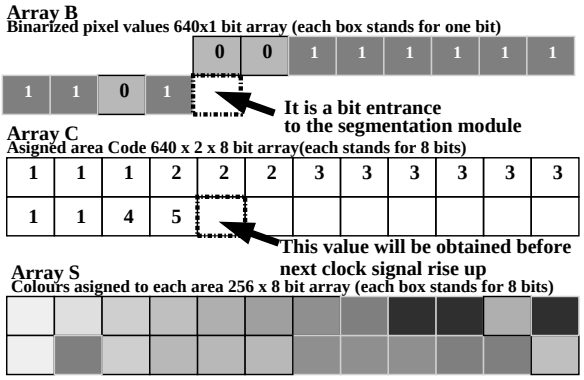


Fig. 9. System structure build inside the FPGA.

UPDATE PROCESS OF THE ARRAY C:
The segmentation process compares the binarized pixel value with the previous pixel in the same line. Some entries on the arrays will be updated depending on the case of their relation from the next list:

- the pixel is equal to the previous one: We assign the same code value on the Array C as the respective value for the previous pixel,
- in other case: We assign the code value on the Array C for the previous pixel PLUS ONE,

UPDATE PROCESS OF THE ARRAY S:
The Array S will be updated in parallel to the update process of the arrays B and C. The new values will depend on the binarized values of the pixel, the previous pixel in the same line, and the upper pixel on the previous line. Only one entry is updated on the array S depending on the first possible case of next list:

- the pixel is equal to the upper one in the previous line: We assign the same value on the Array S as the respective value for the upper pixel in the previous line, for the area code to be assigned for this pixel in the array C,
- when the pixel is different to the upper one in the previous line, but equal the previous one in the same line: We not update any entry on the array S.
- in other case: We assign a new area code not used before in the image to the new code area that we assign for this pixel in the ARRAY C.

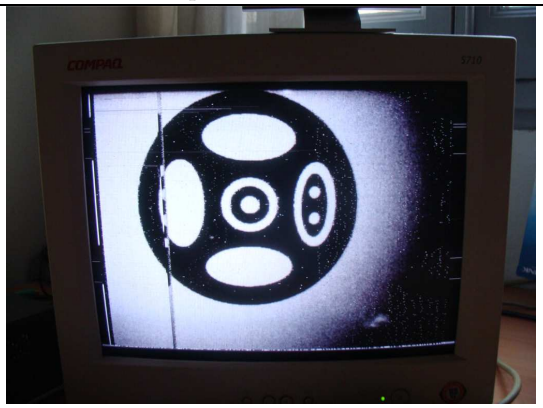
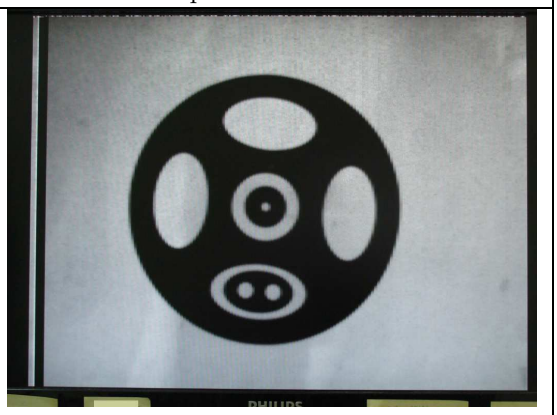
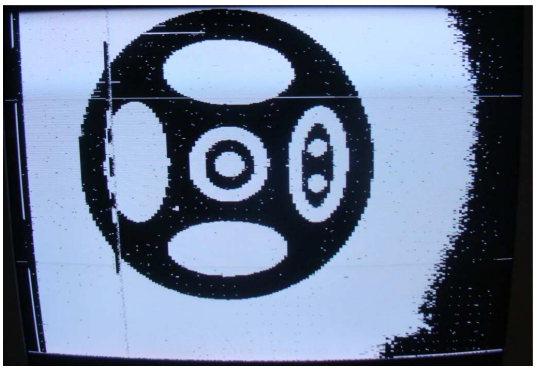
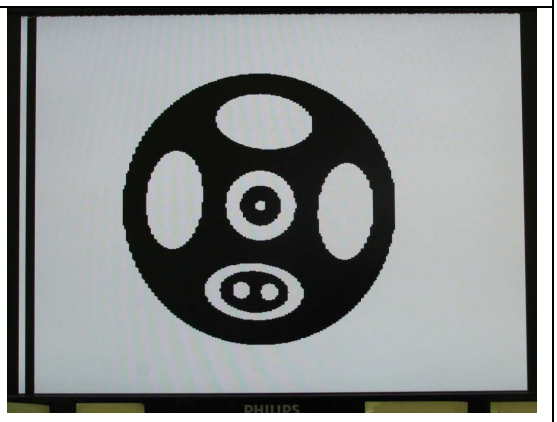
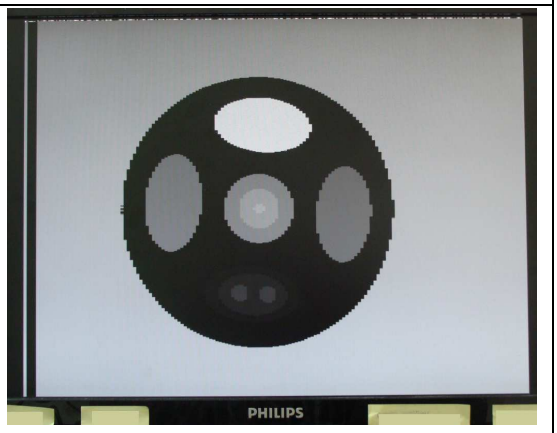
Notice, that the number of pixels in the screen is big even in low resolution sensors, this will make the code area could reach big values. Therefore, increasing the number of bits for the code area values will make the size of Array C increase too.

Notice also that we not need to access any other data previous to the last line.

Then, a nice solution consist on use odd code values for the odd lines, and even code values for the even lines. In such way, start from the smaller even or odd value each time that will start to process a even or odd line, respectively. This guarantee that never be needed more code areas then twice the number pixels in one line (all those values only used in the worst case, when all pixels are different to every

TABLE I

EXPERIMENTAL RESULTS OF THE PROPOSED METHOD ON A SPARTAN3 AND A SPARTAN6 BOARDS.

	spartan 3 board	spartan 6 board
Captured Image		
Binarized Image		
Segmented Image		

of their neighbours).

The last memory array, referred as array S in figure 9, contains the final colour value for each area code.

IV. EXPERIMENTAL RESULTS

The Figure I shows results after each process module for one experiment on each board. The figures on the left side, shows results obtained when using the spartan3 board, and the results on the right side when using the spartan 6 board. The three images for each board shows an instant captured image, the respective binarized result, and its respective segmentation result.

The results on the spartan3 board, at this moment, shows some noise over the image. Such noise makes that the segmentation of the image not be successfully done. It is because the noise in the capture image divides the areas into different smaller areas.

We suppose, after many experiences, that those noises may be from the physical connection from the camera to the FPGA Board. We detect, on the practical experiences, that amount of noise changes when plugging again the camera to board.

The spartan6 board offered perfect and clean results at every stage of the system. We never detect any noise from the camera signal at any experiment when using the spartan 6 board. Thus, we should recommend to use board connectors such in the spartan6 board rather than the connector present in the spartan3 board.

V. CONCLUSIONS

The developed design works successfully on the Atlys spartan6 board, however it needs some im-

provements, related to time constrains, for make it working correctly in the spartan3 board.

Focusing on the design loaded on the spartan6 board, shows a a delay of only one pixel line delay from the capture device to the final processed output, regardless of the camera resolution. Such processing time is faster than any of our requirements, and allows a real time processing required in our projects. In particular, the camera was providing a resolution of 640x480, and the working frequency on the camera was about 20 MHz, which means that one line delay is 32 μ seconds.

In case of require faster processing, we can replace the camera by a faster one, since the bottleneck is on this component. We should refer that the logic in the FPGA can be able to process on few nano seconds.

Alternatively, we can consider to compose a larger image sensor joining multiple cameras like the used in this project.

Notice that this cameras allow to provide externally their pixel capturing frequency from a working range, as we already done on this work. And specify the starting time to capture of the images, which easy to compose such multiple sensor camera and the parallel processing of the images from them.

Acknowledgements

This work have been supported by Spanish government with Grant DPI2009-14552-C02-01. And Grant TIN 2012-32180.

Referencias

[1] Jain, Anil K. (1989). *Fundamentals of Digital Image Processing*, Prentice-Hall, Inc.

[2] Zhengyang Guo, Wenbo Xu?Zhilei Chai Image Edge detection based on FPGA 2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering

[3] Gonzalez, Rafael C. and Woods, Richard E. (2002). Digital Image Processing, Pearson Education, Inc.

[4] Pratt, W. K. (2004). Digital Image Processing, John Wiley and Sons, Inc.

[5] R.C. Gonzalez, Richard E. Woods. Digital Image Processing (2nd Edition) Prentice Hall, 2nd edition (January 15, 2002)

[6] D. T.Saegusa, T.Maruyama, Y.Yamaguchi, How fast is an FPGA in image processing?, IEICE Technical Report, Vol.108. No.48, 2008, pp.8388

[7] Yangli,Yangbing. Study of FPGA based Parallel Processing of Sobel Operator AI Modern Electronics Technique 2005.J.

[8] SHEN fengting WEI hong An Improved Thread Edge Detection Method Based On Sobel Algorithm. Control and Automation 2008.

[9] Steve Kilts, Advanced FPGA Design: Arichitecture, Implementation, and Optimization, John Tiley and Sons.

[10] Arrigo Benedetti, Andrea Prati, Nello Scarabottolo. Image convolution on FPGAs: the implementation of a multi-FPGA FIFO structure. Euromicro Conference, 1998.

[11] Spartan FPGA Complete Data Sheet Xilinx Inc.

[12] P. Athanas and A. Abbott. Real-time image processing on acustom computing platform. In IEEE Computer, Feb. 1995

[13] Bellon, O.R.P. Silva, L. New improvements to range image segmentation by edge detection Signal Processing Letters, IEEE Volume 9, Issue 2, Feb. 2002 Page(s):43 -45

[14] Gevers, T. Robust segmentation and tracking of color objects in video, Circuits and Systems for Video Technology IEEE Transactions on, Volume 14, Issue 6, Jun 2004. pp:776 - 781.

[15] Shanthi K J, Ashok L R, Anandu A S and Gokul Das BFPGA Implementation of Image Segmentation Processor. Second International Conference on Emerging Trends in Engineering and Technology, ICETET-09

[16] T.Morimoto,Y. Harada, T. Koide and H J Mattauschs, A Pixel Parallel digital CMOS implementation of image segmentation by region growing, IEEE Proceedings Circuits Devices and Systems, Vol 152,No.6 Dec 2005.

[17] K Yamohaka, T. Morimoto, H Adachi, T. Koide, and H.J. Mattausch Image Segmentation and Pattern Matching Based FPGA/ASIC Implementation Architecture of Real Time Object Tracking Design Automation, 2006. Asia and South Pacific Conference on 2006, IEEE

[18] Jun, T.: A color image segmentation algorithm based on region growing. In: 2010 2nd International Conference on Computer Engineering and Technology (ICCET), April 16-18, vol. 6, pp. V6-634-V6-637 (2010)

[19] Farmer, M.E., Jain, A.K.: A wrapper-based approach to image segmentation and classification. IEEE Transactions on Image Processing 14(12), 2060-2072 (2005)

[20] Lan, Y., Li, C., Zhang, Y., Zhao, X.: A novel image segmentation method based on random walk. In: Asia-Pacific Conference on Computational Intelligence and Industrial Applications, PACIIA 2009, November 28-29, vol. 1, pp. 207-210 (2009)

[21] http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

[22] http://www.moglik.com/foro/electronica-audio-y-video/sensor-cmos/

[23] M. Klodt, and D. Cremers. "A Convex Framework for Image Segmentation with Moment Constraints". In IEEE International Conference on Computer Vision (ICCV), 2011.

[24] S. J. Osher and J. A. Sethian. "Fronts propagation with curvature dependent speed: Algorithms based on HamiltonJacobi formulations". J. of Comp. Phys., 79:1249, 1988.

[25] D. M. Greig, B. T. Porteous, and A. H. Seheult. "Exact maximum a posteriori estimation for binary images". J. Roy. Statist. Soc., Ser. B., 51(2):271279, 1989. 1

[26] C. Nieuwenhuis, B. Berkels, M. Rumpf, D. Cremers "Interactive Motion Segmentation", In Pattern Recognition (Proc. DAGM), Springer, volume 6376, 2010.

[27] D. Cremers, O. Fluck, M. Rousson, S. Aharon). "A probabilistic level set formulation for interactive organ segmentation", In Proc. of the SPIE Medical Imaging, 2007.

[28] Visual Control of a Remote Vehicle. David Sanchez-Benitez,Jesús Manuel de la Cruz, Gonzalo Pajares, Dawei Gu. ICIRA p. 579-588. Achen, Germany, 2011.

[29] Vertical Rotor for the implementation of control laws. Sanchez, David 9th IFAC Symposium Advances in Control Education .Page Numbers: 224-229, 2012.

Performance Analysis of the Multi-pass Transformation for Complex 3d-Stencils on GPUs

S. Tabik [1], L. F. Romero [1], E. L. Zapata [1]

Abstract— **Complex iterative 3d stencils based on a series of multiple simpler stencils with different computation intensities cannot be handled properly using standard techniques on the GPU. This work demonstrates that decomposing these kind of stencils into a sequence of up to a specific number of simpler stencils and further optimizing each individual kernel provides the best overall performance. We focus on the family of PDE-based denoising methods, which can be reformulated as sequence of multiple stencils-based tasks. The performance results and analysis show that there exists an optimal level of splitting-coalescence of these stencils-based tasks that reaches the best compromise between better use of fast-memories and higher concurrency.**

Palabras clave— **complex stencils, decomposing-coalescing tasks, GPU accelerators, multi-pass optimization.**

I. Motivation and Related Works

THERE exists a large number of works that focuses on optimizing naive 2d and 3d stencil kernels on multi-core and many-core systems. These works can be divided into two classes. The first class focuses on applying transformations such as ghost-zone, time-tiling, or a mix of both [9], [7], [5]. While the second category focuses on building auto-tuning environments which determine the subset of optimizations that improves the performance of the stencil on each specific architecture [3].

Using ghost-zone optimization, which implies enlarging the size of the halo and performing redundant computation, reduces communications between tiles on the GPU but only for iterative 2d stencils with low computation intensity [7], [9]. Time tiling, which consists of tiling the matrix along the time dimension, enhances data locality but again only for simple memory-bound iterative 2-d stencil codes [5], [9].

All these standard optimizations are essential to improve the performance of very simple stencils but they do not require or simply are not applicable to complex stencils, such as iterative compute-intensive 3d-stencils.

On the other hand, the multi-pass optimization, which consists of decomposing the CUDA-kernel into multiple CUDA-kernels, has not been analyzed in a wider context. There exist few works in the literature that employed this optimization to improve a specific application. Indeed, none of them has analyzed at what extent it is beneficial to the performance for a whole family of algorithms. For instance, Lee et al. claimed that applying two-passes approach to a memory-bound neuroimaging algorithm increases concurrency. While, Abdelfattah et al. [1] used two-passes optimization to separate two stages with different computation patterns in the symmetric matrix-vector product code.

The main contributions of this work are: 1) Understanding when the multi-pass optimization can be beneficial to complex stencils representative of PDE-based denoising methods and 2) presenting an interesting pattern of PDE-denoising methods that substantially helps optimizing them for both many and multi-cores.

[1]Dpto. de Arquitectura de Computadores, Univ. Malaga, e-mail: stabik@uma.es

II. PDE-denoising methods: Complex Stencils

Denoising algorithms are essential tools in image processing in many bioimaging modalities. The most sophisticated and powerful methods are those that solve Partial Differential Equations (PDE). This family of algorithms, like for instance the ones described in [2], [4], [6], apply an iterative sequence of mathematical kernels, formulated as stencils with different computation intensity, where the input image is partially or entirely read and updated several times during each denoising iteration till obtaining the final filtered 3d-image after a total number of about 60 to 100 iterations.

A. A Case Study: Anisotropic Nonlinear Diffusion Algorithm

For simplicity, let's focus on a particular case study, the Anisotropic Nonlinear Diffusion (AND) algorithm to understand the features of the considered family of methods. AND-algorithm can be formulated as a stream of four stages, where the noisy image travels through these four methods of different data dependencies, data access patterns and arithmetic intensity. That is, each single iteration consists of:

- First a Gaussian smoothing is applied to the initial 3d-image. Each pixel is updated using a convolution along the X-, Y- and Z-axes, i.e., using the six nearest neighbors. The update of each pixel needs 12 flops (6 reads+1 write). Which is comparable to a 7-points stencil read from and wrote to the 3d-image as shown in Figure 1(a).
- Then, the structure tensor, ST, of the obtained 3d smoothed image is calculated. Each pixel of ST, which consists of a structure of 6 floats,

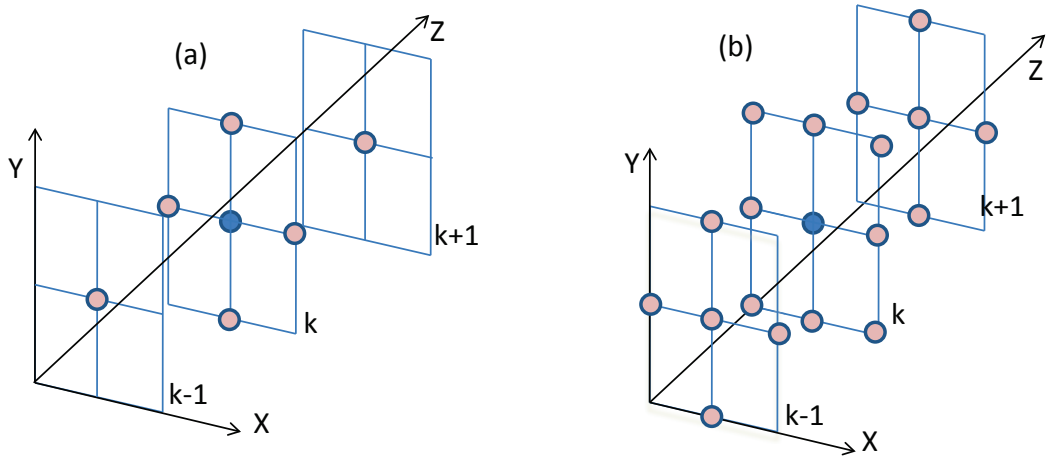


Fig. 1. (a) 7-point stencil, i.e., each pixel needs 7 nearest neighbors including itself to be updated. (b) 19-point stencil, i.e., each pixel needs 19 nearest neighbors to be updated.

is calculated using its corresponding six nearest neighbors from the smoothed image. The data access pattern is a 7-points stencil read from the 3d-image of size $N_x \cdot N_y \cdot N_z$ and wrote to ST of size $N_x \cdot N_y \cdot N_z \cdot 6$.

- Afterwards, the Diffusion Tensor, DT, is calculated. Each pixel of DT is calculated by calculating the eigenvalues and eigenvectors of the corresponding symmetric 3x3-matrix using the iterative Jacobi method. The elements of the symmetric 3x3-matrix are initialized using the values of the corresponding ST pixel. This is comparable to 1-point stencil read and wrote to ST. This stage is clearly compute intensive.
- Finally, each pixel of the 3d-image is updated using 7 neighbor points from ST and 19 neighbor points from the 3d-image. Which is a multiple 7-points and 19-points stencil that reads data from the 3d-image and ST and writes the result in the 3d-image. Updating each pixel of the 3d-image needs 26 reads, one write and 68 flops.

III. Splitting-Coalescence of Stencils on GPU: Multi-pass Approach

Commonly, parallel computing PDE-based methods on clusters of muli-processors employs the SPMD model, where each processor denoises its own 3d-block of the image during the whole iterative process. At the end of each iteration processors that work on neighbor blocks intercommunicate their halos [10].

An intuitive CUDA implementation of this family of methods can consider either encapsulating the whole complex stencil into one pass or splitting it into multiple passes. The performance of the one kernel version can be delimited by the smaller size of faster memory resources during the kernel execution while the multiple kernels version may be penalized by higher scheduling overheads and traffic between off-chip and on-ship memories, which is necessary for inter-kernels communication. However, more combinations can be also considered by merging different successive stencils, 2'-passes and 3'-passes. One of the objectives of this work is to analyze to what extent one should split or coalesce these stencils to obtain the best overall performance on GPU accelerators.

We developed six implementations of AND-method by spiting it into one-, two-, three- and four passes. Applying two- and three-passes optimization has two possible versions. Stencil computation is performed along the z-dimension using space-tiling. First, the 2D tiles and necessary halos are loaded into the shared memory, where each threadblock computes its corresponding output 2D tile, similar to the strategy employed in [8]. Here is a brief description of each implementation.

- The 1-pass approach, encapsulates the whole method, Gauss smoothing, structure tensor calculation, diffusion tensor calculation and, the PDE solution, i.e., Gauss+ST+DT+PDE, into one pass. Updating one 2D tile of the output 3d-image needs to upload to shared memory 4 tiles from the original image (i.e., the corresponding input tile + 3 halos) and 3 tiles of ST (i.e., the corresponding ST tile to be calculated + 2 ST-tile-halos). Recall that three tiles of the original 3d-image are required to compute one ST tile. The 2d-tiles with $Z = 0$ and $Z = Nz$ of the output 3D-image and ST are calculated using a mirroring operation of the 2D-plan with $Z = 2$ and $Z = Nz - 2$ respectively.
- The 2-passes implements Gauss+ST+DT in one pass and PDE in a second pass. The first pass updates each ST-tile using 1 tile of the input 3d-image; the halo pixels are uploaded into registers. The PDE pass needs one ST-tiles and 3 tiles of the input 3d-image; the halo pixels are uploaded into registers.
- The 2'-passes implements Gauss+ST in one pass and DT+PDE in a second pass. The first pass updates each ST tile using one tile of the input 3d-image; the halo pixels are uploaded into registers. The DT+PDE pass needs 3 ST-tiles and 4 tiles of the input 3d-image.

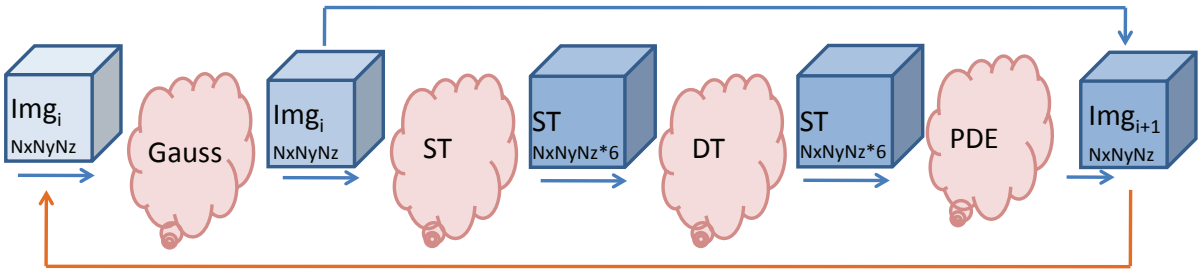


Fig. 2. Pattern of one AND-method iteration: A sequence of four stencil-based stages (in pink color). Gauss() smooths the image, ST() computes the structure tensor, DT() computes the diffusion tensor and PDE() update the image

- The 3-passes version implements Gauss+ST in a first pass, DT in a second pass and, PDE in a third pass. The first pass needs to upload into shared memory one 2D-tile of the input image and one tile of its corresponding ST tile; the halo pixels are uploaded into registers.
- The 3'-passes approach implements Gauss in a first pass, ST+DT in a second pass and, PDE in a third pass. The ST+DT pass needs one tile of 3d-image and one ST-tile.
- Finally, the 4-passes version implements each single stage into one single kernel. Gauss pass needs one 2D-tile of the input image. ST pass needs one tile of ST and one tile of the 3D-image. DT pass needs only to update one ST tile.

IV. Experimental Results and Discussion

This section provides and discusses the performance results of the AND-complex stencil when decomposed into one, two, three and four simpler stencils, each pass is encapsulated into one CUDA kernel, and its comparison with the Pthreads-implementation described in [10]. The evaluations of the CUDA-implementations, written in CUDA C v4.0, were carried out on a single Fermi C2070, with 14 multiprocessors, 448 cores, 48 Kb of shared memory, 32768 registers per block and 1.25 GB of DRAM. While the Pthreads-implementation of the code was executed on 8-core Intel X7550 Beckton using 8threads. The Pthreads-implementation is used only for comparison purposes .

Figure 3 (a) shows the speedup of the different CUDA-passes implementations (i.e., 1-, 2-, 2', 3-, 3'- and 4-passes codes) with respect to the Pthreads-implementation for three 3d-images of sizes $128 \times 128 \times 128$, $256 \times 256 \times 256$ and, $512 \times 512 \times 512$. The found optimal thread-block sizes for the CUDA-codes are 4×32, 8×32, 16×64, 16×16, 16×64 and, 16×64 respectively.

Analyzing the speedup from the 1-pass to the 4-pass versions shows that the 3-passes version, which implements Gauss+ST, DT and PDE into one pass each one, is the fastest version over all the implementations providing a speedup with respect to Pthreads-implementation of up to 290 for the smaller size and 55 for the larger image. In addition, Figure 4(a),(b),(c) shows that the 3-passes version overcomes all the passes versions for the three image sizes by up to 160× and 40× for the smaller and larger image respectively.

A further analysis of registers and shared memory utilization and performance profiling of each pass of the six codes are provided in Table 1 and 2 respectively. Examining the results of the 3-passes code and comparing it with the other versions shows that implementing DT into an independent kernel allows this pass reaching an optimal utilization of on-ship and off-ship memories. In particular, it has a better registers and shared memory utilization, i.e., it applies less stress on this fast memories, from 63 (which is the maximum number of registers that can be used per threads) to 42, in comparison with the pass that merges ST+DT from the 3'-passes version and the one that merges DT+PDE in the 2'-passes version. In addition, implementing DT into one pass reaches a better use of off-ship memories by showing better L1 and L2 hit rates. Moreover, in this 3-passes version DT reaches a higher concurrency, i.e., higher number of active warps per cycle from 15 to 22 when comparing 3'-passes version versus the 3-passes one from Table 2. Notice that DT represents more than 40% of the total runtime of all the implementations and therefore increasing its performance affects the overall performance.

Notice from Table 1 and 2 that merging ST with GAUSS does not affect the concurrency of ST pass. This means that the overhead produced by separating ST and GAUSS into two different passes is higher than the fact of sharing the same fast memories.

On the other hand, the performance of the PDE pass is limited by registers as can be seen from Table 1 and it provides better performance when it does not have to share fast memories.

In summary and from a programming point of view, an effective use of the multi-pass approach as we learnt from the considered case study consists of 1) implementing stencils limited by registers into one pass, 2) merging stencils that have the same data access pattern and use the same data structures into one pass and 3) implementing the most time consuming stencils with high concurrency grad into one pass because merging it with passes with different data utilization penalizes its concurrency.

TABLE I

Registers and shmem utilization per thread (estimated using nvcc 4.0) for 256×256×256 3D-image

	1-pass	2-passes	2'-passes	3-passes	3'-passes	4-passes
Reg count shmem (bytes)	63 29736	63,59 41616,32376	39,63 9080,29728	39,42,63 9080,7776,11664	26,61,63 1304,9072,11664	26,38,42,63 4632,32368,27744,41616

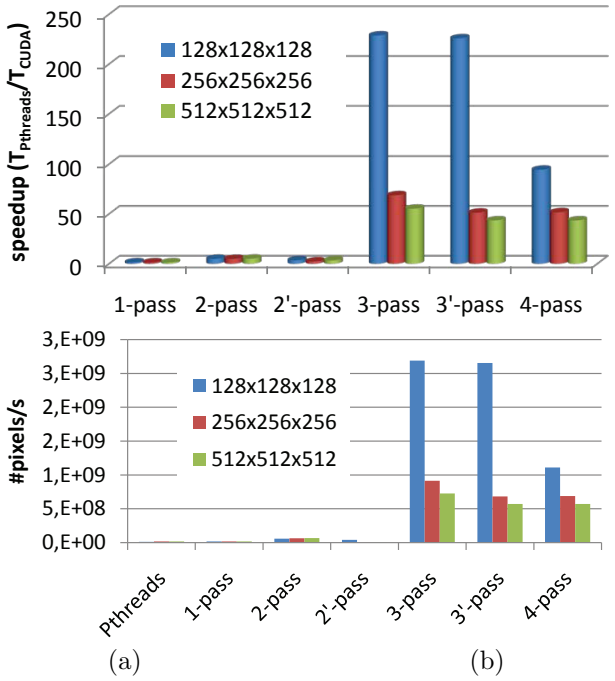


Fig. 3. (a) Speedup of the 1-, 2-, 2'- 3-, 3'- and 4- passes CUDA-versions (on Tesla C5070) with respect to the Pthreads-version (on a 8-core Intel X7550 Beckton) for three 3d-image sizes, $128 \times 128 \times 128$, $256 \times 256 \times 256$ and, $512 \times 512 \times 512$. (b) The number of processed pixels per second by the Pthreads- and CUDA-, 1-, 2-, 2'- 3-, 3'- and 4- passes versions respectively.

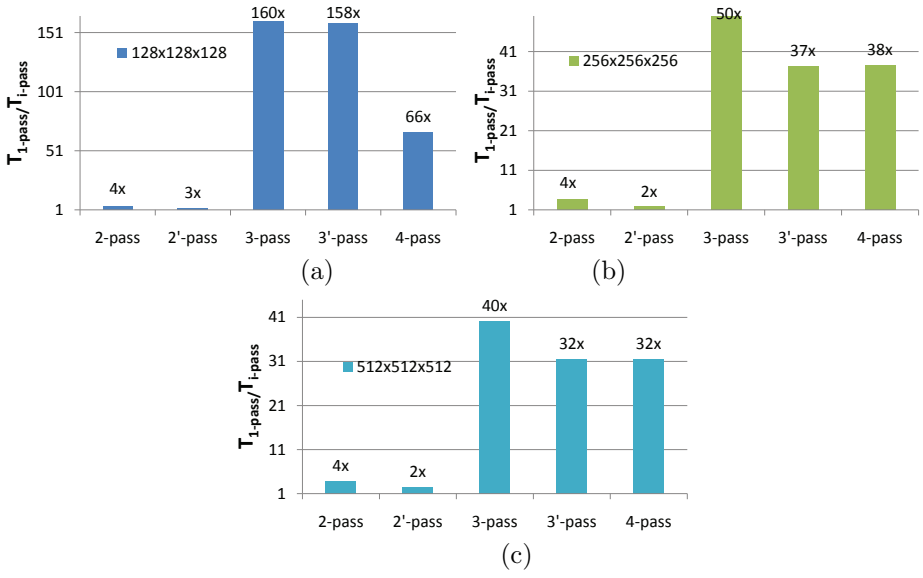


Fig. 4. (a) Speedup of 2-, 3-, 3'- and 4- passes versions with respect to the 1-pass implementation, on Tesla C5070, for three 3d-image sizes, (a) $128 \times 128 \times 128$, (b) $256 \times 256 \times 256$ and, (c)$512 \times 512 \times 512$.

V. Conclusions

This paper characterizes PDE-denoising images to make their CUDA-parallel implementation as efficient as possible with less programming efforts. We showed that 1) implementing register limited stencils into one single pass, 2) merging stencils that use the same data structure and data access into one pass and 3) implementing time consuming stencils with high concurrency into one pass improves substantially the performance of complex stencils that are initially bounded by registers and shared mem-

TABLE II
ONE(GAUSS+ST+DT+PDE), TWO(GAUSS+ST+DT PDE), THREE'(GAUSS+ST, DT, PDE) AND (GAUSS, ST+DT, PDE) AND, FOUR-PASSES(GAUSS, ST, DT, PDE) FOR A 256×256×256 3D-IMAGE AND A FIX THREAD-BLOCK 16×16

	time (%)	active warps/ active cycles	L1 gld hit rate (%)	IPC	inst/ byte	L2 glb mem read throughput (Gb/s)	L2 glb mem write throughput (Gb/s)
1-pass: Gauss+ST+DT+PDE	83,71	7,98	85,28	0,47	3,52	16,4	16,47
2-passess: **Gauss+ST+DT**	**44,64**	**16,02**	**46,24**	**0,84**	**1,34**	**67,69**	**84,3**
PDE	7,95	16,46	86,39	1,06	4,51	47,16	9,79
2'-passess: Gauss+ST	2,56	16,79	66,18	0,36	1,53	9,20	52,78
DT+PDE	**80,11**	**4,21**	**84,36**	**0,55**	**13,81**	**5,24**	**5,25**
3-passes: Gauss+ST	9,8	23,74	69,82	0,37	1,52	9,18	53,42
DT	**47,3**	**22,04**	**72,46**	**0,67**	**1,23**	**70,36**	**67,75**
PDE	6.08	14,89	86,42	1	4,65	45,61	9,81
3'-passes: Gauss	1,44	31,81	69,47	1,15	2,06	84,11	52,87
ST+DT	**43,76**	**15,92**	**40,92**	**0,83**	**1,33**	**68,17**	**84,66**
PDE	7,76	15,58	86,26	1,05	4,64	45,74	9,85
4-passes: Gauss	1,06	30,34	68,29	1,13	2,03	83,60	52,66
ST	9,49	23,70	67,26	0,33	1,43	8,91	51,87
DT	**44,5**	**22,59**	**72,59**	**0,67**	**1,22**	**70,54**	**67,81**
PDE	5,7	16,51	85,69	1,13	4,89	45,64	9,83

ory. We also proposed a way of systematizing the application of this optimization to diminish the programming time and effort.

REFERENCIAS

[1] Dongarra J. Keyes D. Abdelfattah, A. and H. Ltaief. Optimizing memory-bound numerical kernels on gpu hardware accelerators, 10th international meeting on high-performance computing for computational science (vecpar 2012), riken advanced institute for computational science (aics), kobe, japan, july 17th-20th. 2012.

[2] D. Barash. Fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2002.

[3] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, 2008.

[4] J.J. Fernandez. Tomobflow: feature-preserving noise filtering for electron tomography. doi:10.1186/1471-2105-10-178. 2009.

[5] J. Holewinski, L. Pouchet, and P. P. Sadayappan. High-performance code generation for stencil computations on gpu architectures. In *Proceedings of the 26th ACM international conference on Supercomputing*, ICS '12, pages 311–320. ACM, 2012.

[6] S. Li J.J. Fernandez. Anisotropic nonlinear filtering of cellular structures in cryo-electron tomography. *Computing in Science & Engineering*, 7(5):5461, 2005.

[7] J. Meng and K. Skadron. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 256–265, New York, NY, USA, 2009. ACM.

[8] P. Micikevicius. 3d finite difference computation on gpus using cuda. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, pages 79–84, 2009.

[9] G. Rivera and Chau-Wen Tseng. Tiling optimizations for 3d scientific computations. In *Supercomputing, ACM/IEEE 2000 Conference*, page 32, 2000.

[10] S. Tabik, E. M. Garzón, I. García, and J. J. Fernández. High performance noise reduction for biomedical multidimensional data. *Digit. Signal Process.*, 17(4):724–736, July 2007.

Índice de autores