# Realistic Evaluation of Interconnection Networks Using Synthetic Traffic

Javier Navaridas and Jose Miguel-Alonso
*Department of Computer Architecture and Technology, The University of the Basque Country*
*P.O. Box 649, 20080 San Sebastian, Spain. E-mail: {javier.navaridas, j.miguel}@ehu.es*

## Abstract

*Evaluation of high performance parallel systems is a delicate issue, due to the difficulty of generating workloads that represent, those that will run on actual systems. We overview the most usual workloads for performance evaluation purposes, in the scope of interconnection networks simulation. Aiming to fill the gap between purely synthetic and application-driven workloads, we present a set of synthetic communication micro-kernels that enhance regular synthetic traffic by adding point-to-point causality. They are conceived to stress the interconnection architecture. As an example of the proposed methodology, we use these micro-kernels to evaluate a topological improvement of* k-*ary* n-*cubes.*

## 1. Introduction

The interconnection network (IN) is a characteristic component in any parallel computer. Its performance has a definite impact on the overall execution time of applications, especially for those that are fine-grained and communication intensive. Thus, we should not decide lightly the interconnection infrastructure that links compute nodes in a high performance computing site. The evaluation of INs is a complex task that requires a complete model of the network technology we want to assess. Once we have the model of the system, we ought to measure its performance, and some important questions arise: How should we evaluate the IN? Should we measure only raw performance? Is it a better idea to fine-tune the system for a given set of applications? There is not just a valid answer to these questions. Often, the most important performance figure lies simply in running Linpack, whose measured performance is the sorting-key for the top500 list. Other places focus evaluations on the execution speed achievable by the applications currently in use. Alternatively, for a networking technology engineer, the most important evaluation concern is the raw performance of the product, *i.e.* a design that performs acceptably well in most scenarios.

In this paper we propose a set of synthetic workload generators specifically designed to stress the IN. These workloads are to be used in simulation context as performance measurement micro-kernels. They do not take advantage of locality in communications, and emulate different models of contention for the use of network resources. They are parameterizable, allowing the evaluation of INs using workloads with different number of communicating tasks, and different levels of task coupling. This represents a great advantage compared to traces, where these characteristics are fixed—scaling communications in a trace is not trivial. Finally, one of the most important advantages of this approach is that evaluating a system with such micro-kernels is orders of magnitude faster than running a large set of applications.

This paper is arranged as follows. Section 2 discusses methodologies used to evaluate parallel computing systems pointing out their capabilities and limitations and motivates the use of the proposed micro-kernels. Section 3 introduces and justifies the proposed workloads and gives some clues of how they stress the IN. In Section 4 this methodology is used to compare two different direct topologies: a torus and a twisted torus. Section 5 closes this paper with conclusions and future work.

## 2. Related Work

*Synthetic* traffic patterns from *independent sources* [6] provide a good first approach to evaluate an IN because they allow us to rapidly assess raw performance. Often, *random* traffic is used to evaluate systems: uniform, hot region and hot spot traffic patterns have been used in many studies [3, 4, 5, 10]. Other commonly used patterns are those that send packets from each source node to a destination one as indicated by a certain *permutation*. Some examples of these permutations are gathered in [6]. However actual applications do not use uncoordinated communication patterns like these. We can state that synthetic traffic patterns do not accurately mimic the behavior of applications [12].

*Trace-driven* simulation is often used to perform a more realistic evaluation of a system [6]. Feeding a simulator with a trace is not an easy task. To evaluate only the IN of a parallel system we could implement a dummy model of the processing node, allowing it to inject messages as fast as it can, ignoring the causality of messages and the computation intervals. This approach is a stress test, because of the contention generated by all nodes injecting at the maximum pace. It would be more realistic to maintain the causal relationship between all the messages in the trace [7]. To further improve the accuracy of the simulation, compute intervals should be taken into account, maybe applying a CPU scaling factor.

A hybrid between the utilization of synthetic traffic patterns and traces is the spatial, inter-generation intervals and message lengths *probability distributions estimation*, feeding some distribution-fitting software with the traces. We can generate random traffic following distributions that resemble those of the traced application. For example, the spatial distributions of some base parallel applications (namely “13 dwarves”) are plotted in [2].

However use of traces has some problems that we should not ignore: the information within the trace can be inexact due to the logging mechanism and may reflect some of the characteristics of the system in which they were captured. Finally, traces from actual applications running in a large set of processors—those of interest in our performance studies—are hard to obtain, store and manage.

The *a priori* most accurate methodology to evaluate a parallel computer would be running a detailed *full-system simulation* that includes IN, CPUs, the OS, and the applications running on them. This is a very complex and error-prone task, as discussed in [9]. It is also a high resource-consuming methodology that may need a system similar in dimension to the one to evaluate.

In order to introduce causality in the simulation and narrow the gap between application-driven and synthetic traffic from independent sources, a *bursty* traffic model was evaluated in [12]. This model uses synthetic traffic patterns and emulates application causality using a coarse-grained approach. The message generation process passes through a certain number of bursts. During a burst each node is able to inject only a given number of packets (*b*) before stalling until the burst is finished, *i.e.* all the packets are injected and received. Short bursts emulate tightly-coupled applications and long bursts emulate loosely-coupled applications. A primitive synchronization model is included (roughly a barrier every *b* packets); but fine-grained dependencies among messages/tasks are not considered.

In [8] we proposed a set of *micro-kernels* aimed to fulfill the gap between purely-synthetic and application-driven workloads which we focus on virtual topologies and collective implementations. Within this paper we will increase our library of micro-kernels with a slightly different focus. The proposed workloads are devised as network-stressing benchmarks as they test different scenarios that make the IN to suffer contention and lack of locality.

## 3. Proposed Workloads

This section describes and discusses the proposed micro-kernels. They are described algorithmically and, when possible, graphically. All the proposed workloads require the specification of a couple of parameters: number of communicating tasks (*N*) and length of the messages (*S*). We identified tasks from 0 to *N*-1. `Send(from, to, length)` and `Wait(from, to, length)` functions, in the algorithmic definitions of the patterns, do what their names suggest: send a message to a destination or wait until a message from the desired task arrives. `Rnd(N)` returns a random value uniformly distributed in the range [0, *N*). The `Store(from, to, length)` function stores sent messages in order to insert the corresponding receptions at the end of a wave. Regarding graphical description of the patterns, grey arrows represent top-down arranged tasks, *i.e.* the one at the top figure represents task 0 and the one at the bottom represents task *N*-1. Small black arrows represent messages: the rounded end represents a send, and the arrowhead represents a reception.

Most scientific parallel applications use collectives to implement parts of their functionality: from scattering information to collecting results, or just to synchronize a group of processes. For example, EP, IS and FT of the NAS Parallel Benchmarks rely completely on collective operations. The remaining applications in this suite also make use of them but only for initialization and result gathering purposes. In a previous work [8], we proposed optimized implementations of MPI collectives to be used as micro-kernels. As the aiming of this paper is to stress the network, non-optimized collectives are proposed.

The one-to-all pattern (**O2A**) is composed by a wave of messages sent from a *root* task to the rest of the tasks in the group. This generates contention at injection-level. The spatial and causal pattern is defined algorithmically and graphically in Fig. 1. In the all-to-one pattern (**A2O**) all the tasks in the group send a message to a single root task, a situation that generates contention at consumption level. Note that this contention may spread through the network, leading to highly congested scenarios. The spatial and causal patterns of this workload are defined algorithmically and graphically in Fig. 2. In the all-to-all pattern (**A2A**) all the tasks have to communicate with the rest of the tasks in the group. In order to reduce contention at consumption, each tasks *n* sends messages in order starting from its next task, *i.e.* to *n*+1, then to *n*+2, and so on. This pattern may generate a high level of contention

```
one-to-all (N, S):
  for d in [1, N)
    Send(0, d, S)
    Wait(d, 0, S)
  endfor
```

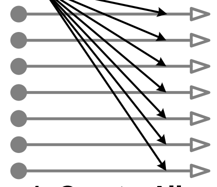

**Figure 1. One-to-All pattern**

```
all-to-one (N, S):
  for s in [1, N)
    Send(0, s, S)
    Wait(s, 0, S)
  endfor
```

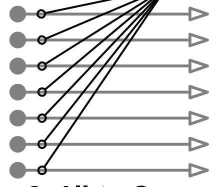

**Figure 2. All-to-One pattern**

```
all-to-all (N, S):
  for t in [0, N)
    for u in [1, N)
      Send(t, (t+u) mod N, S)
    end for
  end for
  for t in [0, N)
    for u in [1, N)
      Wait((t+u) mod N, t, S)
    endfor
  endfor
```

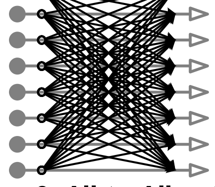

**Figure 3. All-to-All pattern**

```
sync-rnd(N, S, M, W):
  for t in [0, M)
    src=Rnd(N)
    dst=Rnd(N)
    Send(src, dst, S)
    Store(src, dst, S)
    if W messages in Stored
      for m in Stored
        Wait(m.src, m.dst. m.S)
      endfor
      for n in [0, N)
        Computation(n)
      endfor
    endif
  endfor
```

**Figure 4. Synchronized Random**

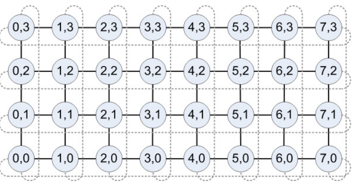

**Figure 5. Regular Torus 8×4**

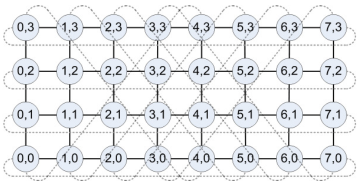

**Figure 6. Twisted Torus 8×4 with twist 4**

for the use of resources. Algorithmic and graphical definitions of this pattern are shown in Fig. 3.

As discussed in Section 2, random uniform traffic from independent sources is a widely accepted workload to evaluate INs. In this section we discuss how to enhance this model by adding support to point-to-point synchronization at task level. The synchronized, random workload (**SR**) is algorithmically defined in Fig. 4. It accepts two extra parameters: total number of messages ($M$) and number of messages per wave ($W$). In short, this workload generator creates waves of messages, with causal dependencies among them. A *wave* is defined as a set of message emissions that can be performed by tasks before waiting for the corresponding receptions. After every communication phase we may include a computation phase. The source and destination of the messages are selected uniformly in [0, $N$). If $S$ and $W$ are large enough, this workload may lead to highly congested states. If we reach the extreme scenario in which $W$ and $M$ are equal, all messages are sent in a single wave, and therefore no synchronization is involved. Then, we are emulating the RandomAccess benchmark, *a.k.a.* giga-update-per-second (**GUPS**) from the High Performance Computing Challenge Benchmark Suite.

We want to remark that uniform traffic stresses the network because it does not take advantage of communication locality; but in contrast, it is also a best-case scenario because, as traffic is evenly distributed through the network, it does not introduce bottlenecks.

## 4. Example of the Proposed Methodology

As an example of the proposed methodology we performed an evaluation of a topological enhancement for cube-like topologies. In [5] we proposed and discussed the *twisted torus*, which improves the topological characteristics of the regular torus: increased throughput, reduced distance-related figures and balanced utilization of the channels in X and Y dimensions. However that study only used purely synthetic traffic: uniform and permutations. This paper evaluates small-size networks with 32 nodes arranged in an 8×4 mesh with the proposed workloads using INSEE [11]. Depictions of the two topologies are shown in Fig. 5 and Fig. 6. Note that these topologies are greatly reduced versions of those used in current MPPs such as IBM's BlueGene/L [4] and Cray's XT4 [1]. We used the three non-optimized collectives all-to-all (**A2A**), all-to-one (**A2O**) and one-to-all (**O2A**) with a fixed message length of 10KB. Regarding the random communications, we generated 10000 messages of 1KB length each, and selected four different wave sizes: 10 (**SR10**), 100 (**SR100**), 1000 (**SR1000**) and 10000 that, as explained before resembles the giga-update-per-second benchmark (**GUPS**). In order to obtain trustworthy results we repeated the experiments 10 times for each of the four workloads, using different sets of messages and plotted the average execution time of the ten repetitions, showing the 99% confidence intervals.

We measured the execution times needed by each topology to deliver all the traffic in the workloads and plotted them in Fig. 7. As running times for each workload are very different, we normalized them to those obtained by the torus. Reader can see that, as the analytical study in our previous paper supports, the twisted torus delivers all the workloads faster than the regular torus. The only exception is **O2A**. In this workload, injection is serialized at the (single) injection port of the source node, which becomes a bottleneck for both topologies. This situation results in almost identical execution times. Reader should note that **A2A** workload is the one in which differences in execution time are more evident. This is because this workload exerts great pressure over the X axis as its rings are longer than those in the Y axis. The twist in the Y axis of the twisted torus helps to alleviate this pressure by balancing the use of the two dimensions. This results in a measured time that is near 20% smaller than the time required by the regular torus topology.

Focusing on the synchronized random workloads, we can see that the larger the number of messages of a wave, the larger is the performance improvement obtained by twisting wrap-around Y links. This is because when waves are short the twisted torus is not able to show its benefits on bandwidth, and the execution time depends on average distance, that is smaller in the twisted torus. However as we are handling with small networks the reduction in average distance is close to negligible compared with the packet length. To further explore this issue, we made an exhaustive analysis of their behavior with different wave sizes.

Fig. 8 shows the execution time of both torus and twisted torus when being fed by a workload composed by 16384 messages and a wide range of values of the wave size: $W=2^k$ : $1\leq k\leq 14$. To easily compare them we also plotted the relative difference. It is clear that the larger the wave size is, the less time is required to deliver the workload, and the more noticeable are the benefits of the twisted torus topology.

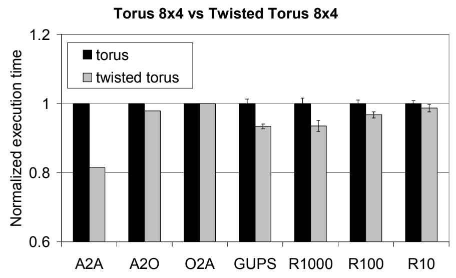

**Figure 7. Normalized execution time**

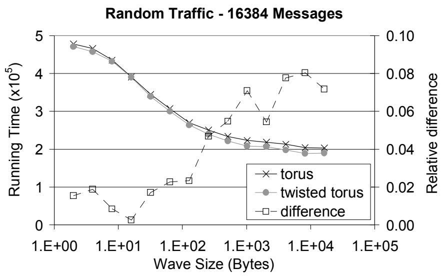

**Figure 8. Simulated time**

To summarize, we want to remind that results presented here are supported by the evaluation carried out in [5] and remark that the main objective of this paper is *not* to do a thorough comparison of topologies. This paper is aimed to introduce the most recent additions to our library of realistic synthetic workloads. This library allows us to evaluate INs in a realistic way, but several orders of magnitude faster and without the issues observed by our group in trace-driven and execution-driven evaluations.

Table 1 closes this section showing the actual execution time, in a 3 GHz Pentium IV desktop PC, needed to simulate the two slowest micro-kernels (**SR10** and **A2A**) and the Class A of the CG benchmark from the NAS parallel Benchmarks, all of them for 32 asks. The reader should note that simulations with our proposal are roughly 30 times faster than with traces.

**Table 1. Actual time to run simulation**

| Workload | Torus | Twisted torus |
|---|---|---|
| SR10 | 48 s. | 46 s. |
| A2A | 35 s. | 28 s. |
| CG class A | 1459 s. | 1412 s. |

## 5. Conclusions and Future Work

In this paper we have discussed methodologies to evaluate high-performance parallel systems, focusing on the workloads used in evaluations performed by means of simulation. These workloads can be purely synthetic or based on actual applications. Also, they can use causal or independent traffic sources. We have described the pros and cons of generating and using each one of these paradigms to evaluate INs. Furthermore, we proposed new synthetically-generated workloads that allow us to evaluate INs in a realistic way.

We have characterized and justified several pseudo-synthetic network-stressing workloads, organized in two groups. The first group includes emulations of message interchanges aimed to implement collective operations in a non-optimized way. They stress the IN because create contention at the consumption, at the injection or inside the network. The second group comprises random communications that involve task-level point-to-point synchronization. These workloads increase our library of communication micro-kernels, described and discussed in [8]. The main benefits of using the proposed workloads are that they are completely parameterizable in terms of number of communicating tasks and application coupling. They also allow performing simulations faster than using application-driven workloads which can lead to a wider exploration of architecture designs. Moreover, as an example of how these synthetic workloads can be used in performance-related studies, we have done a comparison of two direct topologies: a regular torus and a twisted torus, a topological enhancement of the torus.

## Acknowledgment

This work was supported by the Spanish Ministry of Education and Science, grant TIN2007-68023-C02-02, and by the Basque Government, grant IT-242-07. Mr. Javier Navaridas is supported by a doctoral grant of the University of the Basque Country.

## References

[1] SR Alam et al., “Cray XT4: An Early Evaluation for Petascale Scientific Simulation”, Proceedings of ACM/IEEE conference on Supercomputing, Nov 2007, Reno, Nevada.
[2] K Asanovic et al. "The Landscape of Parallel Computing Research: A View from Berkeley". EECS Department. University of California, Berkeley. TR UCB/EECS-2006-183.
[3] E. Baydal, P. Lopez and J. Duato. “A Family of Mechanisms for Congestion Control in Wormhole Networks” IEEE Trans. on Parallel and Distributed Systems, V. 16, N. 9, Sept. 2005, pp 772-784.
[4] M Blumrich, et al. "Design and Analysis of the BlueGene/L Torus Interconnection Network" IBM Research Report RC23025 Dec. 2003.
[5] JM Cámara et al. “Mixed-radix Twisted Torus Interconnection Networks”. Parallel and Distributed Processing Symposium, 2007. IPDPS 2007.
[6] WJ Dally, B Towles. "Principles and Practices of Interconnection Networks". Morgan-Kaufmann, 2004. ISBN: 0122007514, 9780122007514
[7] J Miguel-Alonso, J Navaridas, FJ Ridruejo. "Interconnection network simulation using traces of MPI applications". International Journal of Parallel Programming. In Press.
[8] J Navaridas, J Miguel-Alonso, FJ Ridruejo. “On synthesizing workloads emulating MPI applications”. 9th International Workshop on Parallel and Distributed Scientific and Engineering Computing, Miami, April 18, 2008.
[9] V Puente, C Izu, R Beivide, JA Gregorio, F Vallejo, JM Prellezo "The Adaptive Bubble router", Journal on Parallel and Distributed Computing, vol 61, Sept. 2001.
[10] FJ Ridruejo, J Miguel-Alonso. "INSEE: an Interconnection Network Simulation and Evaluation Environment". Lecture Notes in Computer Science, Volume 3648 / 2005.
[11] FJ Ridruejo, J Miguel-Alonso, J Navaridas. “Full-System Simulation of Distributed Memory Multicomputers”. Cluster Computing. In Press. DOI: 10.1007/s10586-009-0086-y
[12] FJ Ridruejo et al, "Realistic Evaluation of Interconnection Network Performance", 8th Intl Conference on Parallel and Distributed Computing Applications and Technologies, PDCAT 2007. December 3-6 2007, Adelaide, Australia.