Optimization-based Mapping Framework for Parallel Applications

Jose A. Pascual[1], Jose Miguel-Alonso[1] and Jose A. Lozano[2]

[1]Department of Computer Architecture and Technology

[2]Department of Computer Science and Artificial Intelligence

The University of the Basque Country

San Sebastian, Spain 20018

{joseantonio.pascual, j.miguel, ja.lozano}@ehu.es

Technical Report EHU-KAT-IK-02-10

Abstract

The optimal mapping of tasks of a parallel program onto nodes of a parallel computing system has a remarkable impact on application performance. In this paper we propose an optimization framework to solve the mapping problem, which takes into account the communication matrix of the application and a cost matrix that depends on the topology of the parallel system. This cost function can be a distance matrix (the classic approach), or can take into account other considerations. We propose a novel definition of the cost criterion, applicable to torus networks, that tries to distribute traffic evenly over the different axes: the Traffic Distribution criterion. As the mapping problem is a particular instance of the Quadratic Assignment Problem, we can apply any QAP solver to the mapping. In particular, in this work we use GRASP. Using simulation, we test the performance of the optimization-based mappings, and compare it with that of trivial mappings (consecutive, random), in two different environments: static (a single applications uses all system resources all the time) and dynamic (applications are managed by a scheduler, and use and free resources as needed), always using a system with a 2D topology and real application traffic. In both environments optimization-based mappings with the TD criterion provides excellent performance.

1 Introduction

Parallel applications are composed of a collection of tasks that interchange information and synchronize among them using different communication patterns. These applications are often designed and developed with a parallel communications architecture in mind, and arrange the interchanges of data blocks using some schema that tries to efficiently use the underlying network. The way tasks are arranged to perform communications is called the virtual topology.

These applications, once programmed, can be executed in a wide variety of parallel architectures, with different physical interconnection networks. These architectures vary from on-chip multiprocessors with small-sized networks-on-chip to clusters with simple LAN-based networks, and even to supercomputers with custom-made networks organized as meshes, tori, trees, etc.

Given this variety of topologies, it is not uncommon to find a mismatch between the network architecture (machine-dependent) and the virtual topology (application-dependent). This may result in an inefficient use of the network, that materializes in large delays and bandwidth bottlenecks that, in turn, results in a general loss of application performance. The way tasks are mapped onto processors (network nodes) has a remarkable impact on the performance of parallel applications, and on the performance of the parallel system as a whole [1].

Current supercomputers generally share their resources between different users and applications. This means that the system can be running simultaneously several parallel jobs. A system scheduler is in charge of allocating resources to jobs, applying different policies to select the collection of nodes (partition) on which a given job (application) will run. The scheduler manages one or several job queues. When a job finishes, released resources are allocated to the next job (or jobs) in the queue. In this dynamic environment, it is difficult to find a match between the physical topology of the allocated partition and the application's virtual topology. Actually, many schedulers totally ignore the machine's topology, considering a flat network. Some others search for contiguous partitions, a policy that can result in improvements in applications' performance due to the efficient exploitation of communication locality [26] [29]. In any case, a mapping strategy is required to allocate job's tasks onto the assigned partition's nodes. Ideally, this mapping should allow the tasks to make an efficient usage of the assigned resources.

In smaller contexts in which a scheduler is not required, mapping strategies are also relevant. Consider, for example, an embedded system where a single application runs permanently on an on-chip multiprocessor. An optimal mapping of tasks to nodes is critical to keep under control the time to complete tasks' duties, even to comply with real-time constraints. This mapping can be computed off-line, before application deployment.

The mapping problem has been studied by many research groups during the last years. In the context of massively parallel computers, in [26] [29], the authors show, using some simple mappings, the great influence mappings have on application performance. Achievable improvements depend on the particularities of the application, as shown in [4] [5], where some ad-hoc mapping strategies were developed for NAMD, a classical molecular dynamics application. Due to the increase in the use of multi-core architectures, many researchers are studying the mapping problem in this field, developing mapping strategies that focus on the volume of data interchanged by tasks [18] [24] or on the reduction of power consumption [14] [20].

Ideally, in dynamic environments, topology and application-aware allocation and mapping policies should be included in the system scheduler, in order to increase its performance. However, in practice, most schedulers managing current supercomputers include very simplistic allocation/mapping policies that are application and network agnostic. This fact results in random assignments of tasks to nodes: applications are not located in contiguous nodes, and mapping is done without considering their implications on the utilization of network resources [1]. Therefore, communications are inefficient due to inter- and intra-application interferences (as studied in [29]). To our knowledge, only the BlueGene family of supercomputers provides a way to assign to a job a file containing a mapping for its tasks [3]. To illustrate the importance of this issue, in [5], the authors perform a study with three applications in two supercomputers arranged as 3D tori: a BlueGene/L [3] and a Cray XT3 [2]. They conclude that, by co-locating communicating tasks on nearby processors, the distance travelled by messages, and hence the communication traffic, can be minimized, reducing communication latency and network contention. They achieve this goal by manually defining partition and mapping procedures for the target machines. In [7], the authors evaluate the effect that contention has on packets' latencies,

[1] As a notable exception, SLURM [17] includes topology-aware scheduling for BlueGene, Cray XT, Sun Constellation and tree-based hierarchical networks.

concluding that developers should take into account the network topology when mapping tasks to nodes.

In this paper we propose an optimization-based framework to deal with the mapping problem. This framework allows us to easily select different criteria when searching for good mappings. These criteria need to be captured in a cost matrix that finally translates in a function to optimize. In addition to the commonly used distance criterion, that tries to minimize the average distance traversed by messages [27] [35], we propose and evaluate another criterion, which we call *Traffic Distribution* (TD). The motivation to propose the TD criterion is that the reduction of the average distance (*classic* criterion) can create network contention due to the locality of communications, resulting in a degradation of the applications' performance. The new criterion tries to reduce contention, taking into account not only the communication patterns of the application tasks, but also the topology of the interconnection network.

The mapping problem [8] can be seen as a specific instance of the Quadratic Assignment Problem (QAP, see Section 3.1). Consequently, we can tackle it using any method developed to solve the QAP. This problem has NP-hard complexity [13] and, therefore, we use heuristic approaches to find good solutions for it. In particular, we have selected the Greedy Randomized Adaptive Search Procedure (GRASP) [11] [31]. This algorithm, with the optimization criterion expressed as a matrix cost, will minimize the target function and provide as solution a mapping vector that associates each task with a processing node. In terms of optimization, a given mapping is better than another one if the value of the target function is smaller. However, the target function is only a *model* of what happens in reality, and as such it only considers part of the effects that the mapping will have on the application's execution. Therefore we have to *validate* the mappings, proving that a reduction in the target function actually makes the application run faster.

In order to carry out this validation we use simulation, due to its inherent flexibility for testing different system configurations. The simulation workbench used is INSEE [33] which, given a trace of an application, a target network, and a mapping, can provide an estimation of the application's execution time. As target applications we use (traces of) a subset of the NAS Parallel Benchmarks (NPB) [25]. For the interconnection networks, we have restricted our study to the well-known 2D tori. We have carried out two different sets of experiments to evaluate mappings for static and

dynamic environments. For simplicity, in the *static environment* we only consider configurations in which the number of tasks (belonging to a single parallel application) equals the number of nodes. Regarding mappings, we have tested some trivial ones (consecutive, random) and those provided by the optimization procedure, considering both the classic and the TD criteria. Results show that, in static environments, the consecutive mapping is the best option when the application's virtual topology matches the network's physical topology. In these situations the solutions obtained by means of optimization-based strategies cannot provide any improvement. For applications in which this match does not take place the TD approach provides excellent results. Only for a few, pathological cases, the best results are obtained using random mappings.

For *dynamic environments* (systems managed by a scheduler and shared by many concurrent parallel jobs), we have measured the performance of applications that fill a quarter of system's nodes, introducing some "background noise" in the form of synthetic, random traffic between the remaining nodes (those not allocated to the application). The objective of this background traffic is to disturb the application's inter-task communications, trying to emulate real environments in which network resources are shared with other running jobs. Regarding the allocation of system partitions to the target application, three partitioning schemas have been tested: contiguous forming a square, contiguous forming a band, and random (see Figure 3 for the details). We have tested different mappings for each partitioning schema, using as in the static environment the trivial ones (consecutive, random) and those provided by the optimization procedure using both criteria (classic and TD). In this environment, results show that the mismatch between virtual and physical topology results in a bad behaviour for the consecutive mapping, revealing the great potential of mappings based on the TD criterion, which performs better than the classic one.

The rest of the paper is organized as follows. Section 2 is devoted to the definition of the Quadratic Assignment Problem, and to present some approaches developed to solve it. In Section 3 we define formally the mapping problem and its expression as a particular instance of the QAP studying in detail the GRASP algorithm used to solve it. In Section 4 we detail the criteria (classic and the proposed TD) used to solve the mapping problem. Section 5 presents the experimental workbench, following in Sections 6 and 7 with the results of the experiments for both static and dynamic environments, carried out to validate the proposed mapping criteria. Finally, the paper ends in Section 8 with some

conclusions and future lines of research.

2 The Quadratic Assignment Problem

The mapping problem consists of finding an assignment for the set of tasks belonging to a parallel job onto the set of nodes in the partition assigned to run the job; in what we call static environments, the partition can be the whole system. We can do this assignment randomly, or can follow some criterion trying to impact positively on the execution speed of the application. In order to formulate this as an optimization problem, we need to define a criterion to minimize, which is expressed as a cost matrix. This representation of the mapping problem allows us to identify it as an instance of the Quadratic Assignment Problem (QAP), one of the fundamental combinatorial optimization problems in the branch of optimization in mathematics [28]. Consequently, we can take advantage of all the work already done for the QAP, including the different techniques developed to solve it. In the next sections we explore the QAP; first we provide its definition and formal representation, and we follow with a brief review of some approaches developed to solve it.

2.1 Definition of the QAP

The problem of how to assign a certain number of facilities to a certain number of locations with the minimum cost is the well-known QAP problem [28]. Given a weight w representing the flux between each pair of facilities and a distance d between each pair of locations, the problem is to assign each facility to a different location with the goal of minimizing the sum of distances multiplied by the corresponding weight.

One way of formally expressing the QAP, known as the matrix form, is as follows: given F and L, two equal size sets representing facilities and locations, $W = [w_{i,j}]_{i,j \in F}$ representing the weight matrix, and $D = [d_{i,j}]_{i,j \in L}$ representing the distance matrix between location pairs, find the bijection $\pi : F \longrightarrow L$ that minimizes the following objective function:

$$\sum_{i,j \in F} w_{i,j} \cdot d_{\pi(i),\pi(j)} \tag{1}$$

2.2 Approaches to solve the QAP

The QAP is a NP-Hard problem [34]. Due to its computational complexity, extensive research has been carried out to propose effective methods to solve it. We can roughly classify these methods into two groups: exact methods that guarantee finding an optimal solution, and heuristic methods that, while providing good solutions, cannot give optimality guarantees.

- Exact methods: The two main approaches in this group include dynamic programming algorithms [21], and branch and bound techniques [30]. In general, problems with size larger than 16 are hard to solve with these methods, and only certain instances of the problem with particular structures have been solved for larger sizes [19].

- Heuristic methods: These include improvement methods such as local search and tabu search [23], simulated annealing [15] and genetic algorithms [12]. Another heuristic widely used to solve the QAP is the Greedy Randomized Adaptive Search Procedures (GRASP) [32]. Besides producing good quality results, in terms of execution time and solutions, it requires minimal parameter tuning and is relatively easy to implement. See Section 3.2 for a description of this method.

3 A Framework for Solving the Mapping Problem

In this section we explain in more detail our proposal of an optimization-based framework to find mappings vectors. First we formalize the definition of the mapping problem, and then we describe the use of the GRASP algorithm to generate the mappings. The framework we will define is valid for both static and dynamic environments. In a static environment, it is assumed that a single application runs on a target architecture for a long period of time, and the main concern is the quality of the solution, even if the process of generating it is slow. In contrast, in dynamic environments several applications share a machine, and the partition temporally assigned to a particular job can have an arbitrary topology – given by the decision of the scheduler. In this case the solution still has to be good, but there are time restrictions: it has to be obtained fast to keep scheduling overheads low. For the shake of homogeneity in the process of analysing the performance of our proposals, we have set

the same time restrictions (iterations of the GRASP algorithm) for both environments.

3.1 The Mapping Problem

The mapping problem can be formally defined as follows: given a set of tasks belonging to a parallel job $T = \{t_1, ..., t_n\}$ and a set of processing nodes of a parallel computing system $P = \{p_1, ..., p_n\}$ find a mapping function π that assigns a task t to a node p

$$\begin{aligned} \pi: \quad T &\longrightarrow \quad P \\ t &\longrightarrow \quad \pi(t) = p \end{aligned}$$

trying to optimize a given objective function. Note that we use (network) node and processor interchangeably.

The mapping problem can be expressed easily as a QAP in the matrix form: given T and P two equal size sets representing parallel application tasks and processors respectively, matrix $W = [w_{i,j}]_{i,j \in T}$ representing the number of bytes interchanged between each pair of tasks, and matrix $C = [c_{i,j}]_{i,j \in P}$ representing some cost characteristic involving pairs of network nodes, find the bijection $\pi : T \longrightarrow P$ that minimizes:

$$\sum_{i,j \in P} w_{i,j} \cdot c_{\pi(i),\pi(j)} \tag{2}$$

The formulation shown in Equation 2, that states the mapping problem as an instance of the QAP, allows us to leverage the strategies developed for the QAP to find solutions for the mapping problem. We need two data structures: the communication matrix W representing the amount of information interchanged by the tasks, and the cost matrix C which models the criterion selected to distribute the traffic across the network. Note that the first depends solely on the characteristics of the application, while the other depends on the characteristics of the network: topology and routing function. In order to fill W for a particular application, we use traces containing all the interchanges of messages, counting the messages interchanged between each task pair multiplied by their lengths (in bytes). The procedure to fill C depends on the criterion selected to create mappings represented as a function to be optimized.

3.2 Mapping Algorithm

In our experiments we used the GRASP solver, because we found in the literature that this is the algorithm that provides the best results known when solving the generic QAP [9]. We have adapted it slightly in order to deal with the data structures of the mapping problem. GRASP is a constructive and a multi-step algorithm that iterates over the following steps.

First, an initial solution is created by means of a greedy algorithm. The solution is constructed iteratively, adding one solution element (one entry of the mapping vector) at each step, using a greedy function that measures the cost of adding each element into the solution being constructed. The element to be incorporated into the partial solution is selected randomly (uniformly) from the unassigned elements, considering only the $\alpha\%$ of those with lowest cost, being α a parameter provided to the algorithm. Once the selected element is incorporated into the partial solution, the list of remaining elements is updated and the incremental costs are re-evaluated. The greedy function depends on the mapping criterion selected to construct the solutions. The objective of this step is to generate a good quality solution that helps reducing the time required by the local search (next step) to converge to a local optimum.

In a second stage, once an initial mapping is provided (using the previously explained procedure), a local search is carried out. The local search algorithm works iteratively, by successively replacing the current solution with a better one from those in the neighbourhood. In our case the neighbourhood of a solution (a mapping vector) is given by all solutions that can be reached performing a two element swap. The search in the neighbourhood is performed using a best improving strategy: all neighbours are evaluated and the best one replaces the current solution. This process is repeated until no improvement is found.

The previous two steps construct a solution that is a local optimum. The algorithm iterates over these steps selecting, at each iteration, the solution with the best objective function value. The use of problem instances of sizes up to 64, as well as the time restrictions imposed by the scheduling process, forced us to find a trade-off between the quality of solutions and the number of iterations performed. In Section 5.2 we will state the value of the parameters (number of iterations and α) provided to the algorithm.

4 Mapping Criteria

The archetypal context in which the QAP has been applied is in location problems, where the cost matrix represents the distance between facilities, as shown in Equation 1. Often the mapping problem has been seen as a location problem [8], searching a mapping vector that minimizes the average distance traversed by application messages. This is what we call the *classic* (optimization-based) strategy. The result of using this strategy is a mapping that locates in neighbouring nodes those tasks that interchange large data volumes. Apparently, this is a good policy, because it exploits communication locality, reducing the utilization of network resources. However, experiments show that a reduction in average distance does not always result in a reduction of application execution time, because it may create contention hot-spots inside the network [1] [6]. For this reason, we propose a different cost matrix to be used instead of the distance matrix. With this matrix the criterion of minimizing average distance is relaxed, in order to favour communication paths that distribute traffic through the network, avoiding bottlenecks. We have called this the *Traffic Distribution* (TD) criterion.

The definition of both criteria, that materialized in cost matrices, depends on topological characteristics of the target network in which applications will run. In this work we consider just one network topology: 2D tori of $N = n \times n$ nodes, as represented in Figure 1. In these networks each node has an identifier i in the range $0 \ldots N-1$. We will consider routing functions that make messages advance using shortest-path routes between source and a destination. The simplest form of routing is Dimension Order Routing (DOR) [10], in which messages traverse first all the necessary hops in the X axis and then switches to the Y axis, but we do not make any assumption regarding the routing algorithm, except that it must use minimal paths.

4.1 Classic Distance Criterion

In a square 2D torus, given two nodes with identifiers i and j, the distance between them is given by Equation 3.

$$dist(i,j) = dist_x(i,j) + dist_y(i,j) \tag{3}$$

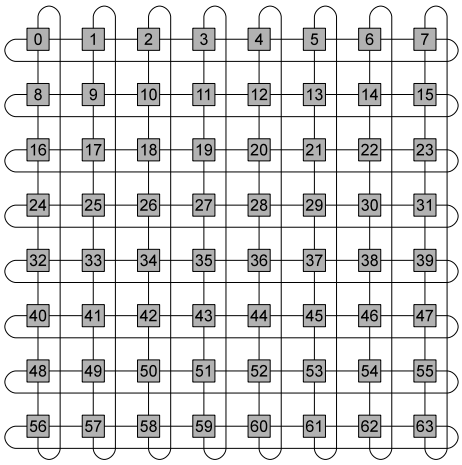

Figure 1: Example of the topology used in static environments: square 2D torus of 64 nodes.

where the first term of the sum is the number of hops through the X axis a message must traverse when going from i towards j, and the second term is the number of hops through the Y axis. Equations 4 and 5 show terms $dist_x$ and $dist_y$.

$$dist_x(i,j) = |\lfloor j/n \rfloor - \lfloor i/n \rfloor| \bmod \lfloor n/2 \rfloor \tag{4}$$

$$dist_y(i,j) = |j \bmod n - i \bmod n| \bmod \lfloor n/2 \rfloor \tag{5}$$

Therefore, the classic criterion to tackle the mapping problem identifies C as the distance matrix, whose components can be filled using Equation 3 for each pair of source-destination nodes.

$$C = [c_{i,j}]_{i,j=1...N} \text{ where } c_{i,j} = dist(i,j) \tag{6}$$

4.2 Traffic Distribution Criterion

The rationale behind use of the classic criterion is clear: minimizing the distance should result in lower latencies (because latency depends on the number of hops) and in lower utilization of network resources, because the number of involved links will be minimized. However, a preliminary collection

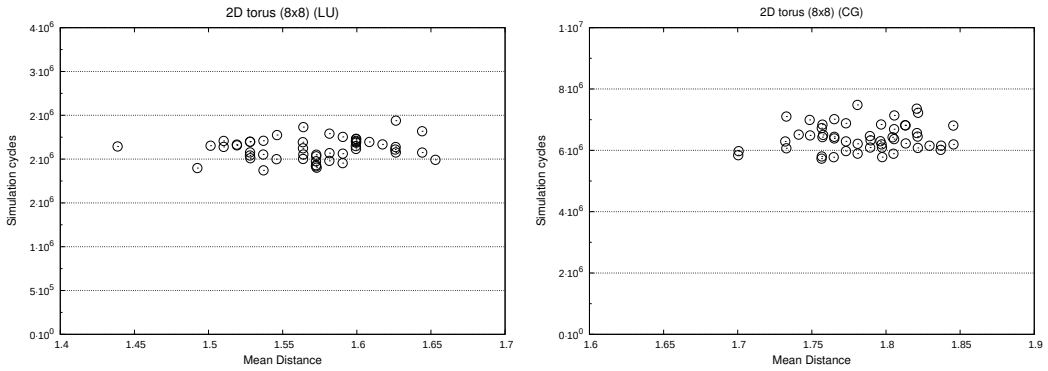

Figure 2: Relationship between the simulation time (in cycles) and the communications mean distance in a 8×8 torus for NPB applications LU and CG, class A, size 64.

of experiments using the INSEE simulator shows that this is not always true. We have generated 10 random mappings for two 64-node NPB applications (LU and CG) running on a 8×8 torus, and measured the mean distance traversed by packets. We have also measured the execution time of the applications as reported by INSEE. All this data has been plotted in Figure 2, using the distance as the ordering criterion. We would expect a correlation between distance and execution time, but the figure does not show it.

The distance criterion hides a fundamental characteristic of interconnection networks: resources (routers, links) have to be shared between many simultaneous communications. It may happen that a given mapping that is optimal in terms of distance, results in multiple communications contending for the use of a single link (or a small collection of links). The contention generated for the shared resources results in increased latencies.

We propose in this paper a new criterion, expressed as an alternative cost matrix, that allows us to find mappings that, while not optimal in terms of distance, helps avoiding network bottlenecks. In particular, we pay attention to the number of hops per dimension that messages have to travel. A 6-hop route in which all the hops are in the X axis is worse than another 6-hop route with 3 hops per dimension, because the latter imposes less pressure in the X axis, spreading the utilization of network resources evenly among the two axes. Formalizing this, we favour those routes in which the number of hops per dimension is equalized – or, in other words, we penalize those routes in which the number

of hops per dimension is not equal. The level of penalization depends on the level of asymmetry, as represented in Equation 7.

$$cost(i,j) = dist(i,j) + |dist_x(i,j) - dist_y(i,j)| \tag{7}$$

Hence, when using the TD criterion for the resolution of the mapping problem using our framework, the cost matrix is defined as:

$$C = [c_{i,j}]_{i,j=1...N} \text{ where } c_{i,j} = cost(i,j) \tag{8}$$

5 Experimental Set-up

In this section we detail the collection of experiments that we carried out to evaluate the effectiveness of the two approaches to the mapping problem described in the previous section. We will use networks of three different sizes: 16 (4×4), 64 (8×8) and 256 (16×16) nodes, in order to test the impact of network size on that effectiveness. Now we describe the experimental workbench used for the experimentation.

5.1 Experimental Workbench

The experimental workbench comprises three key pieces:

1. **NPB traces [25]:** For each application and size we need to generate the traffic matrix W. We have done so for particular instances of the benchmarks included in the NPB, class W size 16, and class A size 64. These applications were run in a real parallel machine, in which traces of all the messages interchanged by tasks were captured. These traces contain the necessary information to fill W, and are also valid to be used within the INSEE simulation environment.

2. **QAP Solver:** This is a program that implements the GRASP algorithm. It accepts as parameters matrix W (the communication matrix of the application trace to evaluate, expressed in bytes), matrix C (the cost matrix modeling a mapping criterion), the number of iterations to be performed and α, used to create the first solution at each step of the algorithm. The output is

a mapping vector that associates each application task with one node of the network partition. The creation of this vector obeys the criterion represented in the cost matrix.

3. **INSEE [33]:** This tool simulates the execution of a message-passing application on a multicomputer connected via an interconnection network. INSEE performs a detailed simulation of the interchange of the messages through the network, considering network characteristics (topology, routing algorithm) and application behaviour (causality among messages). The input includes the application's trace file and the mapping vector. The output is a prediction of the time that would be required to process all the application messages, in the right order, including causal relationships and resource contention. However, it only measures the communication costs, assuming infinite-speed CPUs.

5.2 Design of the Experiments

To perform the validation tests in static environments, we created a set of 50 different mappings for each NPB application, using the classic and the TD expressions of the cost matrix. For this environment, we tested two application/network sizes: 16 and 64 tasks/nodes (we only considered one-to-one assignments). Each of the 50 solutions were selected after 50 GRASP iterations with the value of parameter α set to 20%.

For the dynamic environment the set-up was more complex. We used traces of the NPB applications, size 64, class A. The network was four times larger: a torus of 16×16 nodes. Therefore, a running application used a partition containing one fourth of the network nodes. The remaining nodes generated background noise: they interchanged messages using a random, uniform pattern, to emulate the effect of other applications sharing the network. We evaluated the behaviour of the mappings for three different values of the background traffic generation rate: 0.0625 packets per node and per cycle (the theoretical maximum injection rate accepted by the network as determined from the bisection bandwidth [10]), 0.016 packets per node and per cycle (one fourth of the theoretical maximum injection rate), and 0.0003125 packets per node and per cycle (0.5% of the theoretical maximum injection rate). These rates were selected trying to emulate different scenarios of network usage: from a network in saturation state (highest noise injection rate) to an almost empty network (lowest rate).

We tested three forms of network partitions: quadrant (the partition included the 8×8 nodes in the upper-left corner of the network), band (the partition included the 4×16 nodes in the upper network rows) and random (selecting 64 nodes randomly) as shown in Figure 3. For the latter, 20 different random partitions were generated. The mappings, obtained as explained in the previous paragraph, were tested on the assigned partitions.

In both environments, in addition to the classic and the TD mapping strategies, we also evaluated the behaviour of two trivial mappings: consecutive and random.

- Consecutive mapping: Given a network partition (the whole network in the static environment), application tasks are allocated onto the partition's nodes in order of identifiers, starting with the assignment of the first task to the node with the lowest identifier, and ending with the assignment of the last task to the node with the highest identifier.

- Random mapping: In this case, given a network partition, application tasks are assigned to partition nodes randomly. To evaluate this strategy, a set of 50 different random assignments to each partition were performed. In dynamic environments, for the random partitioning schema, due to the use of 20 randomly generated partitions, a total of 20×50 experiments were performed for each application. Note that in dynamic environments (network size larger than application size) with random partitioning (nodes in the partition are selected randomly) the consecutive mapping is just a particular case of random mapping.

Each of the mappings was simulated using the INSEE environment [33] [22]. For dynamic environments and because of the use of random background traffic (noise), each experiment was repeated five times, using each time a different seed.

6 Results for Static Environments

The results of the experiments in static environments are summarized in Table 2 for the smallest size network (16-node 2D torus), and in Table 3 for the largest size network (64-node 2D torus). The tables show the best, the average and the standard deviation of the execution times reported by the simulator for each application-topology-mapping combination. The statistical significance of the

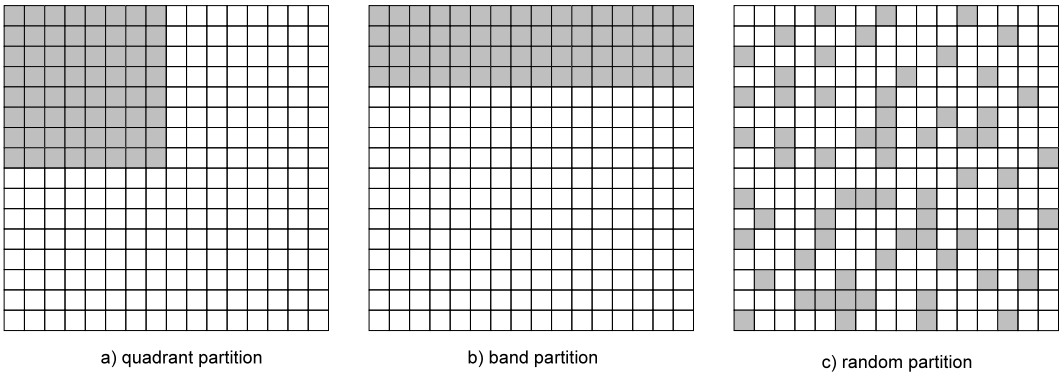

Figure 3: Network partitions used to test different mappings strategies in dynamic environments. a) Quadrant partition b) Band partition c) Example of a random partition

differences between the three non-consecutive mapping strategies (note that the consecutive has not random component) has been assessed running Kruskal-Wallis [16] paired tests on the results obtained for each of the traces. The tests have been performed between the strategy that obtained the best result and each of the others, at one level of significance: 0.05.

In Table 2 we can see that the use of the TD mapping criterion improves, in most cases, the results obtained by the other criteria. There is only one case, the SP trace, in which the TD does not obtain the best result. For this trace the best mapping is the consecutive. However, the use of TD reaches almost the same results. It is remarkable that only in two traces, BT and SP, there exists a significant difference between the use of the classic and the TD criteria, obtaining almost the same results for the other traces. Regarding the random mapping, for this network size it provides the worst results for all traces.

Let us now pay attention to Table 3. Results for the large network differ from those obtained with the 16-node case. The behaviour of the mapping strategies depend strongly on the characteristics of the applications. We can differentiate three application groups. In the first one we include BT, LU and SP. A property of these applications is that the communications pattern (virtual topology) matches the physical one. Therefore the consecutive mapping is clearly the best performer. TD is the second best, with significant differences in two traces with the classic and in three cases with the random

mapping. In a second group we can consider FT and IS. These applications performs basically all-to-all data interchanges, launched by all the tasks almost simultaneously; this generates peaks of substantial network utilization. For this group the best results are given by the random mapping, being the difference of results in relation with TD and classic statistically significant. These two optimization-based mapping strategies tend to allocate as close as possible those tasks that communicate often, and this results in heavy contention for the network links, due to the communications pattern. In this case, the random distribution of the tasks spreads messages evenly throughout the network, helping to reduce contention. Finally, in the third group we find CG and MG. For these two applications there is no match between virtual and physical topologies, and the utilization of the network is not based on intensive all-to-all interchanges. For this group the TD strategy is the best, having significant differences with the classic for CG, and with the random for both applications.

In summary, the novel TD procedure we propose in this paper provides good mappings of applications onto networks in static environments. The trivial mappings are able to produce better results in those cases in which the virtual network matches the actual one (consecutive mapping) and when the communications pattern performs mainly all-to-all communications (random mapping). Note, though, that in actual supercomputers applications run in partitions of a machine, instead of in the whole machine, and in this scenario the perfect matching cannot be expected to happen. We explore results for these scenarios in the forthcoming section.

7 Results for Dynamic Environments

The results of the experiments carried out in dynamic environments are summarized in Tables 4(a), 4(b) and 4(c) for the quadrant partition schema, Tables 5(a), 5(b) and 5(c) for the band partition schema, and Tables 6(a), 6(b) and 6(c) for the randomly generated partitions. The tables show the best, the average and the standard deviation of the execution times reported by the simulator for each application-topology-mapping combination. As done for the static environment, the statistical significance of the differences between the four strategies in dynamic environments has been assessed running Kruskal-Wallis [16] paired tests on the results obtained for each application. Note that in this case the consecutive strategy has a random component derived from the use of background traffic. As

it was done before, we used one significance level: 0.05.

As explained in the previous section, the experiments for dynamic environment were performed in a network that is four times larger than the application size: 256-node 2D torus network, 64-task jobs. Tasks are mapped onto a network partition, using different partitioning schemas: quadrant, band and random (Figure 3). Next we examine the results obtained for each partition schema, and after that we carry out a brief study about the impact that background noise has on the effectiveness of the mapping strategies.

7.1 Analysis of Results

We start analysing the results obtained using the quadrant partition schema. It reserves for a 64-task job a 8×8 square partition of a 16×16 torus. The square is located the upper left corner of the torus; however, due to the symmetry of this topology, the position of the partition in the network does not affect the results.

Again, we will consider three groups of applications: those whose virtual topology is a torus (BT, LU, SP), those that use all-to-all interchanges (FT, IS) and the remaining ones (CG, MG). Obviously, for the first group the best mapping strategy is the consecutive, with significant differences with the other strategies. The second best is TD. For the second group, random mapping is again the best performer, with significant differences with the other methods. For the last group the winner mapping strategy is TD. As we expected, this behaviour is exactly the same observed in static environments. Although most communications are internal to the partition, the average distance traversed by packets is larger than in the case of a 8×8 network. For this reason, execution times are longer than for the case evaluated in the static environment (64-task job on a 64-node network, see Table 3). The presence of background traffic on nodes not belonging to the partition affects internal communications (higher levels of noise results in longer execution times), but does not change the conclusions regarding the best choices of mappings.

Now we analyse the band partitioning schema. In this case the TD strategy performs the best for traces in the first group (BT, LU, SP) and also for CG, with significant differences with the other three strategies. The use of a band partition breaks the perfect topological match, and TD shows its potential. For the second group (FT, IS) the winner is, as in the previous experiments, the random

mapping. For the particular case of MG, the best mapping criterion is TD for low-medium levels of background noise, and consecutive for highly loaded networks. In the latter case, differences between TD and consecutive are statistically significant, but not very large.

Finally, let us analyse the results achieved using randomly generated partitions. In this case, none of the applications benefit from a match between virtual and physical topologies. The group composed by applications FT and IS maintains a differentiated behaviour, and works better with a random mapping. The remaining applications now behave similarly, and in this case TD is the best performer with significant differences with the other mappings.

We have to insist in that, in most current supercomputers, partitioning is carried out without any special consideration. When a job needs a partition, the scheduler selects any available node – which results in something close to the random partitioning schema used in our experiments. This is the most realistic scenario, and corresponds to the one in which the TD mapping strategy performs best for five out of the seven applications included in our experiments.

To summarize this section, we can state that the choice of a mapping criterion to make the task-to-node allocation has a remarkable impact on the performance of parallel applications. The behaviour of a particular mapping, measured in terms of application speed, depends on many aspects: physical topology of the network, partitioning schema, virtual topology of the application, and other characteristics of the application's traffic. Some applications perform the best when tasks are located randomly, because this reduces the presence of network bottlenecks. For most cases it is better to locate communicating tasks in nearby nodes, avoiding bottlenecks, and here is where the TD strategy performs well – although it can be surpassed by consecutive mappings only in the (improbable) case of good virtual-to-physical topology match.

7.2 Effects of the network load on the mapping strategies

As stated in the previous section, the experiments in the dynamic environment were performed injecting synthetic, random background traffic at three different rates in the network. The objective of this traffic is to evaluate the effect that the network status (load) has on the behaviour of the mapping strategies. A direct inspection of the tables shows how the background traffic increments the execution times of the applications, and this increase is larger for higher injection rates. This is expected,

due to the existence of more packets in the network that interfere the inter-tasks communications: there is more contention for the (shared) resources. Our goal now is to compare the stability of the behaviour of the different mapping strategies as we increase network utilization. In table 1 we detail which method obtains the best results at each injection rate for all applications.

Table 1: Best mapping strategies (•) for each application, for different levels of background traffic: Low (0.0003125 packet/node/cycle), Medium (0.016 packet/node/cycle) and High (0.0625 packet/node/cycle).

		Quadrant				**Band**				**Random**			
		Co	R	Cl	TD	Co	R	Cl	TD	Co	R	Cl	TD
	L	•							•	•			•
BT	M	•							•				•
	H	•							•				•
	L				•				•				•
CG	M				•				•				•
	H				•				•				•
	L		•				•				•		
FT	M	•	•				•				•		
	H		•				•				•		
	L	•	•				•				•		
IS	M		•				•				•		
	H		•				•				•		
	L	•							•	•			•
LU	M	•							•				•
	H	•							•				•
	L				•				•				•
MG	M			•	•				•				•
	H			•	•	•							•
	L	•							•				•
SP	M	•							•				•
	H	•							•				•

We can observe that, in most cases, the network load does not affect the choice of best mapping strategy. There are a few exceptions, though. In the case of the quadrant partitioning schema, consecutive and random mappings perform similarly with the medium injection rate for the FT application, and the same happens with he low injection rate for IS. However, when this rate is increased, the best mapping criterion is always the random one. Regarding the MG application, the increase of the injection rate results in better performance for the classic criterion, which is as good as the TD at high injection rates.

If we focus on the band partitioning schema, the MG trace is the only one that deserves comment. The TD criterion is the best at the low and medium injection rates. However, at high rate, the

consecutive mapping becomes the best performer.

Finally, let us pay attention to the random partitioning schema. In this case, the consecutive mapping is as good as the TD for applications BT and LU when used in lowly-loaded networks. However, when the load is higher the best mapping option is TD.

In general, we can conclude that the choice of mapping strategy can be done without taking into consideration the amount of traffic not related to the applications.

8 Conclusions and Future Work

Current supercomputer centres share their resources between different users and applications. A system scheduler plays a crucial role, selecting the jobs to be executed (extracting them from a queue) and assigning some resources to it. Often, the allocation/mapping procedures are carried out using simple mechanisms that ignore the communication patterns of the application and the topology of the network.

In this work we have focused on the mapping problem, describing an optimization-based framework to find good task-to-node assignments that takes into account the communication characteristics of the application, as well as the topology of the network. This framework allows us to represent a mapping criterion as a cost matrix. We have identified a classic form of cost matrix: the node distance matrix. We have proposed a novel criterion that tries to distribute traffic evenly through the network, while still keeping close those tasks that communicate most.

In order to validate our proposal we have used trace-based simulation, using traces obtained from the applications included in the NPB suite. First, we have measured running times on systems built around 2D square tori of sizes 16 and 64, in static environments (a single application uses permanently the whole system). Results show that the proposed TD criterion surpasses the results of the classic one. Random mappings are useful for pathological cases of intense all-to-all communications, and consecutive mappings can perform well only if the application's virtual topology matches the underlying physical topology. In general, the use of an optimization-based search is able to produce a good-quality mapping.

The main goal of our research work, though, is the identification of good mappings for highly dynamic

environments in which the scheduler selects a partition for a job and then maps tasks onto nodes – an assignment that is temporal, because applications are not running forever. To test our proposals in this environments, we have used 64-task applications from the NPB suite running in a 256-node 2D square torus, with three types of partitioning schemas and different levels of background traffic generated by the nodes not allocated to the jobs, trying to simulate different scenarios of network load. In this environment, the TD mapping strategy is the one that allows the applications to run faster, particularly in those cases in which the topologies of partition and application do not match. Note that, as we have stated repeatedly, this is a very realistic assumption.

Although in the experimental part of this paper we have focused on creating good mappings for tori, this work can be extended easily for other topologies, regular or not – for example, to fat-trees, a very common topology in state-of-the-art supercomputers. We just need a criterion to fulfil the cost matrix for the target topology. Also, our experiments have been performed using synthetic traffic. We plan to consider an even more realistic scenario and carry out experiments in which different applications (not synthetic traffic) share a single machine.

We also plan to extend this work in other directions. In particular, we plan to investigate the way "smart", topology-aware partitioning schemas could be incorporated into schedulers. A combination of a good partitioning with an optimization-based mapping strategy could greatly enhance the productivity of large-scale parallel computers. In a previous work we have investigated the great potential of topology-aware partitioning [29], that can be exploited *only* if applications experiment a significant speed-up when tasks are located contiguously. If we incorporate better mappings, this requirement could be relaxed while still improving system's performance.

9 Acknowledgements

This work was supported by the Basque Government [Saiotek, Research Groups 2007-2012 grant number IT-242-07]; the Spanish Ministry of Science and Innovation [grant numbers TIN2007-68023-C02-02, TIN2008-06815-C02-01 and Consolider Ingenio 2010 CSD2007-00018]; and the Carlos III Health Institute [COMBIOMED Network in Computational Biomedicine]. Mr. Pascual is supported by a doctoral grant of the Basque Government.

References

[1] T. Agarwal, A. Sharma, A. Laxmikant, and L. V. Kalé. Topology-aware task mapping for reducing communication contention on large parallel machines. In *IEEE International Parallel and Distributed Processing Symposium*, page 122, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[2] R. Ansaloni. The Cray XT4 Programming Environment. `http://www.csc.fi/english/csc/courses/programming/`, March 2007.

[3] Y. Aridor, T. Domany, O. Goldshmidt, J. E. Moreira, and E. Shmueli. Resource allocation and utilization in the BlueGene/L supercomputer. *IBM Journal of Research and Development*, 49(2–3):425–436, 2005.

[4] A. Bhatele and L. V. Kalé. Application-specific topology-aware mapping for three dimensional topologies. In *Proceedings of Workshop on Large-Scale Parallel Processing (IPDPS)*, pages 1–8, Miami, FL, USA, 2008. IEEE Computer Society.

[5] A. Bhatele and L. V. Kalé. Benefits of topology aware mapping for mesh interconnects. *Parallel Processing Letters*, 18(4):549–566, 2008.

[6] A. Bhatele and L. V. Kalé. An evaluation of the effect of interconnect topologies on message latencies in large supercomputers. In *Proceedings of Workshop on Large-Scale Parallel Processing (IPDPS)*, May 2009.

[7] A. Bhatele and L. V. Kalé. An evaluative study on the effect of contention on message latencies in large supercomputers. In *IEEE International on Parallel and Distributed Processing Symposium*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[8] S. H. Bokhari. On the mapping problem. *IEEE Transaction on Computers*, 30(3):207–214, 1981.

[9] R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB – A quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, 1997.

[10] W. Dally and B. Towles. *Principles and practices of interconnection networks.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[11] T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[12] C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. In *DIMACS Series in Mathematics and Theoretical Computer Science*, pages 173–187, Providence, RI, USA, 1993. American Mathematical Society.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[14] J. Hu and R. Marculescu. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pages 233–239, New York, NY, USA, 2003. ACM.

[15] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[16] W. Kruskal and W. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47:583–621, 1952.

[17] Lawrence Livermore National Laboratory. Simple linux utility for resource management. `https://computing.llnl.gov/linux/slurm/`.

[18] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of the Euromicro Symposium on Digital Systems Design*, page 180, Washington, DC, USA, 2003. IEEE Computer Society.

[19] E. M. Loiola, N. M. M. de Abreu, P. O. B. Netto, P. Hahn, and T. M. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.

[20] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, and F. Hessel. Exploring NoC mapping strategies: An energy and timing aware technique. In *Proceedings of the conference on Design,*

Automation and Test in Europe, pages 502–507, Washington, DC, USA, 2005. IEEE Computer Society.

[21] A. Marzetta and A. Brungger. A dynamic-programming bound for the quadratic assignment problem. In *Computing and Combinatorics*, pages 339–348, London, UK, 1999. Springer-Verlag.

[22] J. Miguel-Alonso, J. Navaridas, and F. J. Ridruejo. Interconnection network simulation using traces of MPI applications. *International Journal of Parallel Programming*, 37(2):153–174, 2009.

[23] A. Misevicius. A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, 30(1):95–111, 2005.

[24] S. Murali and G. D. Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 896–901, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[25] NASA Advanced Supercomputer (NAS) division. NAS parallel benchmarks. `http://www.nas.nasa.gov/Resources/Software/npb.html`, 2002.

[26] J. Navaridas, J. A. Pascual, and J. Miguel-Alonso. Effects of job and task placement on the performance of parallel scientific applications. In *Proceedings 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 55–61, Washington, DC, USA, February 2009. IEEE Computer Society.

[27] J. M. Orduña, F. Silla, and J. Duato. On the development of a communication-aware task mapping technique. *Journal of Systems Architecture*, 50(4):207–220, 2004.

[28] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In *Proceedings of the DIMACS Workshop on Quadratic Assignment Problems*, volume 16, pages 1–42, AMS, Providence, USA, 1994. American Mathematical Society.

[29] J. A. Pascual, J. Navaridas, and J. Miguel-Alonso. Effects of topology-aware allocation policies on scheduling performance. In *Job Scheduling Strategies for Parallel Processing (IPDPS)*, volume 5798, pages 138–156. Springer-Verlag, Berlin, Germany, 2009.

[30] K. G. Ramakrishnan, M. G. C. Resende, and P. Pardalos. A branch and bound algorithm for the quadratic assignment problem using a lower bound based on linear programming. In *State of the Art in Global Optimization: Computational Methods and Applications*, pages 57–73, Dordrecht, The Netherlands, 1995. Kluwer Academic Publishers.

[31] M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[32] M. G. C. Resende and C. Ribeiro. Greedy randomized adaptive search procedures. In *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[33] F. J. Ridruejo and J. Miguel-Alonso. INSEE: An interconnection network simulation and evaluation environment. In *Proceedings of the 11th international Euro-Par conference on Parallel Processing*, pages 1014–1023, Berlin, Germany, 2005. Springer-Verlag.

[34] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[35] R. Tornero, J. M. Orduña, M. Palesi, and J. Duato. A communication-aware topological mapping technique for NoCs. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 910–919, Berlin, Germany, 2008. Springer-Verlag.

Table 2: Execution times (10^4 simulation cycles) for the NPB applications (class W, size 16) running on a 4×4 torus. The experiments were tested using a statistical test with one level of significance: 0.05 (†).

	Consecutive	Random			Classic			TD		
	Value	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	93.78	111.38	121.34†	2.94	93.78	103.69†	6.88	93.77	**93.78**	0.00
CG	348.65	346.13	386.16†	17.97	302.80	307.66	7.86	302.80	**302.86**	0.08
FT	117.94	117.68	124.59†	2.35	113.28	**115.70**	0.98	113.31	116.33	1.53
IS	82.54	80.65	84.42†	1.75	77.70	79.38	0.68	76.61	**79.71**	0.78
LU	74.78	77.92	85.29†	2.71	74.76	**74.78**	0.02	74.77	**74.78**	0.03
MG	50.48	52.95	57.98†	1.66	45.52	45.58	0.07	45.52	**45.56**	0.06
SP	**161.11**	181.30	194.67†	4.28	161.11	175.58†	10.42	161.11	161.46	0.28

Table 3: Execution times (10^4 simulation cycles) for the NPB applications (class A, size 64) running on a 8×8 torus topology. The experiments were tested using a statistical test with one level of significance: 0.05 (†).

	Consecutive	Random			Classic			TD		
	Value	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	**369.35**	728.22	764.38†	16.34	576.15	648.51	29.76	569.67	623.23	28.95
CG	719.35	715.50	789.10†	25.76	572.84	642.17†	35.52	533.42	**614.87**	29.56
FT	1254.25	783.92	**801.16**	6.77	832.72	911.30†	28.33	840.50	910.81†	24.13
IS	404.21	248.89	**253.27**	6.77	273.93	292.19†	28.33	274.88	294.80†	6.93
LU	**136.57**	231.38	258.05†	9.01	187.20	211.53†	10.11	181.01	201.45	9.19
MG	468.47	449.96	477.92†	10.04	378.73	399.74	9.03	377.86	**397.02**	8.61
SP	**598.97**	1095.80	1156.00†	22.61	882.47	953.65†	31.69	823.69	907.23	33.98

Table 4: Execution times (10^4 simulation cycles) for the NPB applications (class A, size 64) running on a **quadrant** partition on a 16×16 torus, with different levels of background traffic. The experiments were tested using a statistical test with one level of significance: 0.05 (†).

(a) Injection rate for background traffic: Low (0.0003125 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	424.88	**427.92**	3.51	1243.34	1338.92†	44.07	821.03	886.64†	37.33	709.94	752.66†	32.00
CG	1015.40	1016.23†	0.56	1244.78	1387.09†	50.81	680.13	813.32†	50.20	648.81	**701.06**	34.04
FT	1521.99	1536.65†	9.78	1495.29	**1514.47**	8.09	1513.17	1622.83†	54.84	1505.33	1579.93†	35.26
IS	505.62	**506.88**	1.49	496.75	507.24	3.59	520.12	542.85†	9.52	517.36	536.74†	12.33
LU	137.03	**137.03**	0.01	386.38	420.56†	18.96	202.79	252.66†	23.76	190.88	217.15†	14.99
MG	635.06	639.62†	3.04	824.44	890.08†	27.47	513.86	535.42†	8.38	489.37	**519.91**	14.35
SP	730.06	**733.22**	3.36	1918.23	2129.24†	71.46	1148.93	1259.65†	37.70	1007.31	1093.41†	34.74

(b) Injection rate for background traffic: Medium (0.016 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	570.70	**579.68**	8.97	1342.04	1452.50†	48.93	884.06	963.14†	46.12	728.85	812.26†	45.42
CG	1056.75	1057.48†	0.73	1337.19	1491.80†	53.20	713.83	864.48†	65.34	682.36	**738.12**	35.16
FT	1644.87	1649.56	4.69	1622.75	**1647.55**	9.15	1632.53	1763.01†	63.78	1638.67	1726.52†	1.53
IS	554.86	555.01†	0.14	536.59	**546.40**	3.90	565.43	578.17†	6.71	549.95	571.67†	11.87
LU	156.61	**156.67**	0.06	414.34	451.31†	20.14	225.99	265.73†	20.72	201.25	227.87†	14.56
MG	689.58	691.32†	1.74	893.85	964.33†	29.92	558.81	573.67	7.13	538.47	**565.80**	19.56
SP	898.83	**899.71**	0.44	2099.81	2297.22†	74.51	1230.93	1352.68†	44.42	1074.66	1168.40†	28.57

(c) Injection rate for background traffic: High (0.0625 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	798.20	**808.80**	5.32	1412.75	1565.46†	48.56	981.13	1050.99†	41.32	811.65	876.95†	47.75
CG	1225.73	1226.47†	0.29	1457.26	1638.02†	58.65	844.28	979.01†	73.71	826.87	**865.89**	24.92
FT	1808.62	1811.30†	9.46	1753.45	**1789.58**	13.51	1775.97	1927.73†	75.54	1866.96	1869.08†	2.12
IS	608.83	612.68†	2.47	583.58	**594.59**	4.53	609.78	636.28†	11.91	613.05	622.84†	4.77
LU	223.66	**224.13**	0.29	447.99	488.82†	20.69	262.70	293.50†	15.34	238.41	261.35†	13.27
MG	768.97	776.14†	4.03	980.98	1048.40†	30.48	633.75	656.10	14.45	619.13	**652.54**	18.52
SP	1229.71	**1244.77**	5.74	2265.83	2473.06†	72.24	1405.28	1526.82†	43.96	1247.65	1304.99†	22.83

Table 5: Execution times (10^4 simulation cycles) for the NPB applications (class A, size 64) running on a **band** partition on a 16×16 torus, with different levels of background traffic. The experiments were tested using a statistical test with one level of significance: 0.05 (†).

(a) Injection rate for background traffic: Low (0.0003125 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	1603.71	1612.01†	6.85	1149.57	1215.63†	36.41	794.87	923.78†	86.57	680.76	**744.20**	44.53
CG	1025.15	1025.34†	0.20	1190.64	1328.34†	44.58	713.85	814.18†	43.49	654.14	**681.16**	22.98
FT	1920.73	1923.84†	2.64	1299.38	**1328.78**	13.59	1393.63	1492.58†	31.04	1425.47	1472.83†	31.30
IS	506.59	513.63†	5.72	414.50	**421.51**	3.27	448.84	483.38†	18.12	453.33	491.62†	25.47
LU	387.25	390.00†	2.65	356.98	395.22†	14.08	221.07	263.38†	22.44	205.16	**230.09**	14.77
MG	534.22	534.98†	0.70	728.29	783.40†	22.77	514.72	555.34	20.50	487.08	**511.89**	12.08
SP	2425.16	2430.58†	6.97	1776.69	1887.00†	59.31	1060.08	1222.78†	76.50	1049.42	**1110.82**	35.50

(b) Injection rate for background traffic: Medium (0.016 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	1649.10	1651.64†	2.54	1181.24	1240.94†	37.46	835.10	945.18†	79.98	719.29	**773.39**	34.56
CG	1045.01	1045.79†	0.78	1232.26	1361.17†	42.19	718.60	819.19†	35.88	683.88	**712.63**	21.06
FT	1986.32	1990.41†	4.09	1342.24	**1375.89**	15.96	1444.27	1549.03†	33.74	1480.44	1501.82†	17.76
IS	548.14	552.14†	4.00	433.35	**440.63**	3.39	469.31	509.64†	20.00	476.39	496.53†	10.21
LU	386.59	387.22†	0.63	361.73	402.23†	13.00	230.52	268.71†	23.16	220.59	**236.01**	11.20
MG	551.82	552.37†	0.56	758.83	813.98†	23.34	541.46	575.69†	17.65	519.84	**543.92**	12.19
SP	2462.18	2467.95†	5.77	1833.71	1942.91†	60.47	1095.88	1258.30†	80.21	1087.29	**1148.72**	32.30

(c) Injection rate for background traffic: High (0.0625 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	1726.23	1736.44†	9.24	1299.75	1370.62†	40.84	886.26	929.50†	32.42	765.86	**825.70**	27.76
CG	1119.93	1122.93†	1.27	1365.53	1480.60†	38.82	768.71	870.04†	35.81	750.61	**777.11**	24.71
FT	2259.05	2261.62†	1.67	1462.12	**1513.49**	18.18	1607.44	1707.12†	36.43	1686.14	1708.58†	22.60
IS	597.14	600.79†	2.57	485.92	**495.91**	4.15	540.47	581.68†	23.55	520.06	556.23†	31.11
LU	597.14	391.98†	1.29	392.63	434.30†	13.01	244.41	281.04†	19.98	227.63	**247.39**	10.03
MG	588.18	**590.37**	1.47	830.41	902.01†	27.11	579.55	614.53†	13.79	586.04	611.84†	14.94
SP	2624.58	2637.76†	8.79	2003.93	2132.24†	64.61	1166.82	1323.25†	80.88	1184.33	**1235.37**	29.27

Table 6: Execution times (10^4 simulation cycles) for the NPB applications (class A, size 64) running on **random** partitions on a 16×16 torus, with different levels of background traffic. The experiments were tested using a statistical test with one level of significance: 0.05 (†).

(a) Injection rate for background traffic: Low (0.0003125 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	537.06	584.32	26.74	641.45	704.24†	26.17	538.75	609.46†	25.99	536.67	**577.38**	16.13
CG	662.46	734.27†	60.47	639.31	710.98†	32.69	525.08	611.12†	39.38	520.11	**585.45**	41.27
FT	880.08	935.05†	33.07	669.75	**708.97**	13.64	679.44	764.48†	28.63	713.80	781.95†	18.91
IS	254.37	272.63†	11.76	199.79	**219.84**	8.18	218.48	237.24†	7.49	216.22	239.53†	9.94
LU	183.31	200.89	12.51	209.74	235.52†	12.48	168.03	198.54†	11.07	168.01	**193.14**	12.69
MG	362.19	390.11†	21.74	377.61	414.16†	17.07	334.89	358.26†	13.00	332.78	**348.60**	8.89
SP	830.56	882.11†	35.45	958.87	1042.50†	31.86	794.01	889.93†	43.03	776.43	**833.07**	28.67

(b) Injection rate for background traffic: Medium (0.016 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	681.54	719.84†	31.11	731.87	840.06†	37.32	639.27	709.78†	30.30	607.19	**658.48**	23.26
CG	827.72	892.92†	64.18	763.39	875.09†	40.54	656.21	735.04†	40.45	617.16	**705.99**	45.94
FT	1229.51	1292.74†	37.35	851.79	**890.01**	14.06	880.16	980.80†	30.81	936.15	996.34†	23.81
IS	399.34	413.39†	10.16	268.96	**289.32**	8.08	301.16	329.11†	11.00	303.92	325.31†	9.97
LU	233.33	245.22†	13.49	253.03	283.06†	14.74	208.37	244.09†	16.21	211.20	**233.55**	12.34
MG	456.44	481.51†	26.40	487.58	534.19†	20.00	421.05	450.93†	14.31	410.90	**439.45**	13.45
SP	1041.56	1100.75†	42.54	1148.71	1287.23†	51.18	964.35	1089.11†	55.43	914.84	**1002.56**	39.47

(c) Injection rate for background traffic: High (0.0625 packets/node/cycle).

	Consecutive			Random			Classic			TD		
	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev	Best	Average	Std Dev
BT	1223.30	1300.21†	34.51	1181.63	1283.79†	31.01	1100.43	1256.54†	50.43	1089.94	**1183.70**	41.52
CG	1451.57	1506.92†	26.53	1348.13	1446.81†	47.07	1240.11	1385.14†	66.04	1164.05	**1256.59**	38.56
FT	1905.98	1964.07†	26.07	1328.94	**1366.29**	12.92	1449.59	1540.50†	29.25	1430.51	1532.25†	37.85
IS	586.99	600.42†	5.61	428.82	**444.90**	4.22	477.72	518.62†	16.31	468.92	512.25†	14.58
LU	460.00	493.66†	16.64	428.43	469.47†	18.56	407.95	453.00†	19.68	377.84	**426.48**	17.40
MG	804.09	851.29†	26.62	796.33	840.75†	13.84	774.89	817.52†	19.99	734.34	**777.76**	18.40
SP	1889.85	1983.33†	51.25	1863.66	1992.44†	48.77	1736.60	1887.61†	53.05	1671.20	**1808.62**	54.28