Konputagailuen Arkitektura eta Teknologia Saila

Departamento de Arquitectura y Tecnología de Computadores

Mechanisms and techniques for scheduling in supercomputers

by

Jose A. Pascual

Supervised by Jose A. Lozano and Jose Miguel-Alonso

Dissertation submitted to the Department of Computer Architecture and Technology of the University of the Basque Country (UPV/EHU) as partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, May 2013

If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?

Seymour Cray.

Acknowledgments

There are so many people I would to thank for helping me and supporting me along these years. I will try to keep this list as brief as possible.

- To both my advisors Prof. Jose Antonio Lozano and Prof. Jose Miguel-Alonso for their guidance and patience.
- To the members of the Intelligent Systems Group for their support. In particular, I would like to thank the Computer architecture section of the ISG and Javier Navaridas for useful discussions and support.
- Author is especially grateful to Prof. Thomas Bäck and Michael Emmerich from the Leiden University and to Prof. Ke Tang and Thomas Weise from the University of Science and Technology of China. Thanks for hosting me in my research visits.
- To Itziar Otaduy for the design of this dissertation cover.
- To my flatmates David, Egoitz and Alex for supporting and putting up with me day and night.
- I want also to dedicate this thesis to Miguel Linares and Alberto Llaguno for his unconditional support.

This work has been supported by the Spanish Ministry of Science and Innovation [projects TIN2007-68023-C02-02, TIN2008-06815-C02-01, TIN2010-14931 and Consolider Ingenio 2010 CSD2007- 00018], the Basque Government [Saiotek, Research Groups 2007-2012, IT-242-07] and the Carlos III Health Institute [COMBIOMED Network]. This work was partially carried out when the author was visiting the University of Science and Technology of China (USTC) under the grant NICaiA (Nature Inspired Computation and its Applications, PIRSES-GA-2009-247619). The author was supported by a doctoral grant of the Basque Government.

Abstract

This thesis analyzes the performance of the scheduling process in space-shared, large-scale supercomputers. These systems are specifically designed to run fine-grained parallel applications in which the communications/computation ratio is high. The way of using the interconnection network has a significant bearing on applications performance and, therefore, on the overall system performance.

The scheduling process can be divided into three stages, driven by a set of policies or strategies. Assuming that users send parallel jobs to a single scheduling queue, (1) a job is selected to run, (2) the resources (set of nodes) required by the job have to be located in the system and reserved for the job, and (3) job task have to be mapped onto the selected nodes.

This dissertation studies ways of improving the performance of the scheduling process focusing on stages 2 (partitioning) and 3 (mapping). In particular we use contiguous partitioning as the strategy to assign partitions to jobs. Contiguous partitioning strategies have a well-known disadvantage: high fragmentation that results in low levels of system utilization. However, they provide jobs with a running environment that, due to the locality of communications and the lack of interference with other running jobs, substantially reduce running times. In order to effectively exploit these advantages an appropriate task-to-node mapping has to be implemented. Through extensive simulation-based experimentation, it is demonstrated that combinations of consecutive partitioning and application-aware mappings achieve excellent job throughput, compared to non-contiguous partitioning alternatives.

Although the main topic of our work is contiguous partitioning, we have also explored some aspects of non-contiguous partitioning strategies, due to its common use in production environments. Regarding system topology, we mainly focus on cube-shaped topologies such as meshes and tori, but we also study alternative partitioning strategies for tree topologies.

Table of Contents

Part II Selected publications

Part I

Scheduling in supercomputers

1

The scheduling process

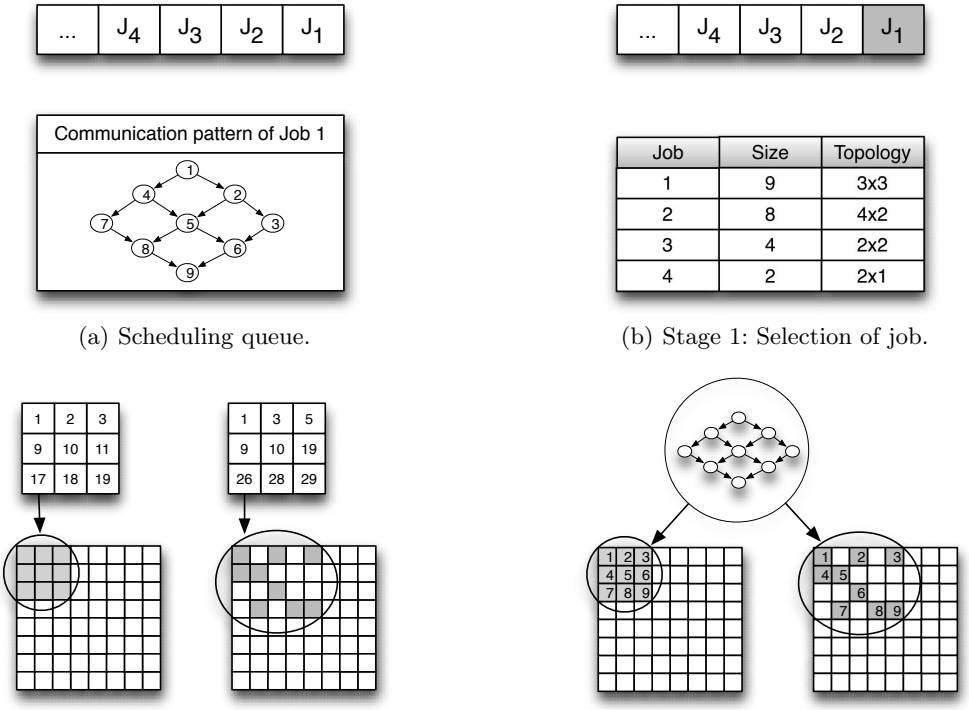
Supercomputer centres are usually designed to provide computational resources to multiple users running a wide variety of applications. In these facilities users send jobs to a scheduling queue, where they wait until the resources required by the job are available. These resources are specified when the jobs are submitted to the system, and may include many requirements such as the number of nodes or a specific partition shape. The most common kind of jobs are parallel applications that need many processors, varying from tens to thousands.

A parallel application is composed of a collection of tasks that exchange information and synchronize among them using different communication patterns. These applications are often designed and developed with a parallel communication 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 logically arranged to perform communications is the *virtual topology* of the application.

State of the art parallel systems are composed of thousands or hundreds of thousands of computing nodes using a network to interconnect them. These networks are built using different topologies such as cubes or trees. Currently, the most powerful supercomputers use networks arranged as 3D tori as shown in the Nov. 2012 edition of the Top500 list [30]. Networks of this kind provide good topological properties such as symmetry, low node degree and low diameter [8]. Examples of these systems are the current Cray supercomputers [14] and the Blue Gene/L [1] and Blue Gene/P [17] systems. More recently, networks with more dimensions are being used. For instance the Blue Gene/Q [10] uses a 5D torus network, and the K computer arranges the nodes using a three dimensional array of 3D cubes, forming a 6D network [2].

The scheduling process in a supercomputer can be split into three stages as represented in Figure 1.1. Jobs arriving to the system are queued until the scheduler selects one of them for running. This decision is made following an established policy. Then, a specific number of available nodes must be found. The way these nodes are arranged is determined by a partitioning strategy. Finally, the tasks of the selected jobs must be assigned to the partition nodes, following a mapping strategy.

The performance of the scheduling process strongly depends on the collection of policies/strategies used to guide each stage. There exist many metrics that measure the scheduling performance, at system-level and at job-level [9]. Inside the first group we can find the *system utilization*, which represents the percent of the computing nodes used along the time, and the *job throughput*, which measures the number of jobs that the system processes in a given period. The metrics inside the second group measure average times such as the *average waiting time* that jobs spend in the queue, the *average turnaround time*, which is composed of the waiting time



(a) Scheduling queue.

(b) Stage 1: Selection of job.

(c) Stage 2: Partitioning step.

(d) stage 3: Mapping step.

Fig. 1.1. Representation of the scheduling process in a 2D cubic network. (a) After a job composed of several tasks arrives in the scheduling queue, (b) the scheduler must select a job to be executed following a scheduling policy. (c) In a second step, the appropriate set of nodes have to be found using a partitioning strategy (contiguous in the left or non-contiguous in the right). (d) Once the partition is found the tasks-to-nodes assignment must be performed following a mapping strategy such as the consecutive one used in the example.

plus the running time, or the *bounded slowdown*, which represents the average ratio between the waiting time and the run time of the jobs.

1.1 Job selection

In the first stage of the scheduling process, one job must be selected from the queue. The selection is performed following a scheduling policy that establishes the order in which the jobs will be executed [16]. A simple policy is First Come First Serve (FCFS) [16], which imposes a strict order in the execution of jobs. These are arranged by their arrival time and order violations are not allowed. The main drawback of this policy is that it severely reduces system utilization. When the job at the head of the queue cannot be put to run because the required resources are not available, all the jobs in the queue must wait due to the sequentially ordered execution of jobs. As a result, many processors could remain idle, even when other waiting jobs could be eligible to use them.

With the goal of increasing the system utilization, several alternatives have been proposed [16], being backfilling (BF) the most widely used due to its easy implementation and proven

benefits. This policy is a variant of FCFS, based on the idea of advancing jobs through the queue. If the job at the head cannot be launched due to resource constraints, a reservation time for it is calculated using the estimated termination time of currently running jobs. This estimation is computed using the expected value of the runtimes (provided by the users when submitting the jobs). While that job is waiting, other queued jobs that can fit in the available resources and are expected to finish before the reservation time can be executed. If one of these advanced jobs run for longer than expected, violating the reservation time, it will be killed and placed again in the queue to be rescheduled. Using this policy, the system utilization is improved because more jobs can be put to run without delaying the expected starting time of other jobs, thus respecting the order imposed by the arrival times.

In this dissertation we use a simple FCFS policy as a baseline to compare the proposed strategies guiding the other stages, and BF because it is the most commonly used in real systems. The scheduling model that we use implements only one queue where the submitted jobs are put to wait. Although in production systems many queues managed by priorities or calendars could be used, we do not consider such scenarios.

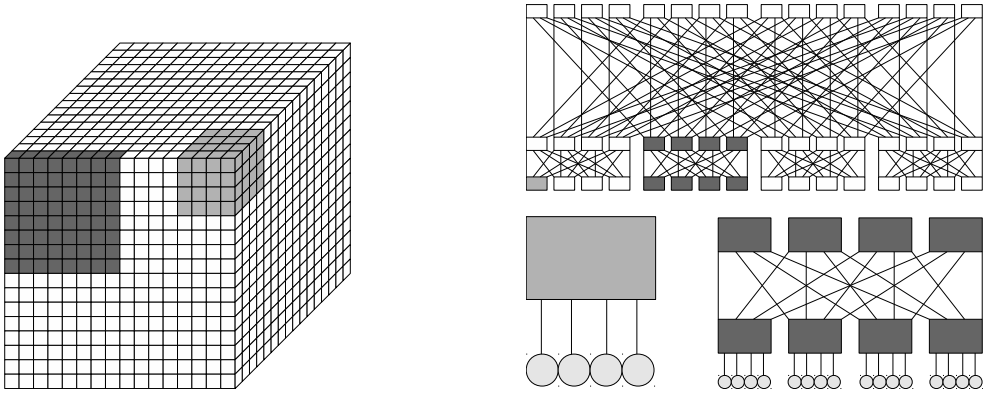
1.2 Partitioning

After the job selection stage is completed, the partitioning stage must be performed. This stage is in charge of searching for the set of nodes requested by the selected job. If found, it will be assigned to the job. Otherwise, the job will have to wait until a suitable set becomes available. The way nodes are physically arranged within the system's interconnection network determines the *physical topology* of the partition.

We can divide partitioning strategies into two broad categories: contiguous and non-contiguous, as illustrated in Figure 1.1(c). Using contiguous partitioning the scheduler must find sets of nodes contiguously arranged. However, the precise definition of contiguity depends on the topology of the network. In cube topologies, as shown in Figure 1.2(a), contiguous partitions form convex shapes. This kind of partitions maintains intra-job communications isolated from other jobs. The topology of the networks restricts the possible topology of the contiguous partitions. For instance, in a 3D network, contiguous partitions can be composed of 3D, 2D or even 1D arranged nodes. When the network topology is a tree, contiguity must be defined considering the switches that form the network. In Figure 1.2(b) we have represented a 64-node fat-tree built with 4-port switches. In this case, a contiguous partition for a job with 4 or less tasks must be composed of nodes attached to the same switch (level 0). If the partition is larger than 4 and smaller than 16 nodes, additional switches that belong to level 1 must be used.

The use of contiguous partitions provides many advantages for running parallel applications. In space-shared systems, contiguity reduces the average distance traveled by messages, and eliminates inter-job communications interference. However, scheduling performance under contiguous strategies suffers seriously due to system fragmentation [31]. *Internal fragmentation* occurs when the assigned partition has more nodes than those requested by the job. *External fragmentation* exists when sufficient nodes are available to satisfy a request, but they are not arranged contiguously and, therefore, cannot be assigned (under this policy). Both kinds of fragmentation drastically reduce system utilization, increasing the waiting time of the jobs.

Non-contiguous partitioning strategies were created to reduce the negative effects of fragmentation [31]. These strategies eliminate the contiguity requirements, allowing the selection of any set of nodes in the network, which results in the increase of system utilization. The main drawback of this kind of strategies is that job communications are not isolated within the assigned partitions. They will compete for the use of the network resources, increasing the contention.



(a) (16x16x16) 3D Cube. (b) 4,3-Fat-tree.

Fig. 1.2. Example contiguous partitions in different network topologies. In (a) two contiguous partition are represented in a 3D cube: a (8x8) 2D mesh and a (4x4x4) 3D mesh. In (b) a 4-node and a 16-node contiguous partitions are represented in a 64-node fat-tree.

Therefore, in general, the jobs will require longer running times. A way to deal with this issue is to oversize the network.

As we have just described, with non-contiguous partitioning, the jobs do not ask for a specific partition shape, just request a node count. But with contiguous partitioning, the request may contain the exact shape of the partition to run the job, or just a node count. In the second case, it is up to the scheduler to make a decision about the contiguous shape to look for. This decision must be performed following a *partition selection strategy*.

After deciding the shape to look for, a partition searching algorithm must be used to check if it is available. In case of contiguous partitioning, it has to determine if a free partition with the chosen shape exists in the system and to locate its position if it does. Many partitioning algorithms have been developed for both contiguous [11] [13] [15] [18] and non-contiguous partitioning [3] [31]. These algorithms implement different searching strategies like First Fit (FF) or Best Fit (BF). As this step can be invoked many times, these algorithms must be implemented very efficiently.

Partitioning strategies are double-edge swords. Systems using non-contiguous partitioning jobs are not affected by external fragmentation, and waiting times are lower. However, different applications share the network resources, resulting in longer running times. Contrarily, applications that run contiguously allocated achieve lower running times due to the lack of interference, but also need to wait longer in the queue due to the external fragmentation caused by this kind of strategies.

1.3 Task mapping

In the last stage, after an appropriate partition has been found, each task that composes the parallel application has to be assigned to a partition node.

This assignment can be performed ignoring which application is being mapped (application agnostic) or taking into account the communication pattern of the application (application-aware). Two examples of application agnostic mappings are the *consecutive*, in which tasks are assigned to nodes in order of identifier, and the *random*, that assigns tasks to nodes randomly. These strategies totally ignore the way the tasks of the application communicate among them

(communication pattern or virtual topology) and the topology of the assigned partition. As a result of this lack of knowledge, the communications will not take advantage of the underlying network topology and applications will generally achieve suboptimal performance [25] [27].

Regarding application aware mappings, they usually rely on optimization techniques to perform the selection of the best task-to-node assignment. They consider the virtual topology of the application, the topology of the partition to be mapped onto and the amount of information exchanged among the tasks. The mapping problem can be seen as an instance of the Quadratic Assignment Problem (QAP) [26], one of the fundamental combinatorial optimization problems in the branch of optimization in mathematics. It considers the problem of how to assign a certain number of facilities to a certain number of locations with the minimum cost.

Mapping was extensively researched in [6] [7] [12] [29]. This research was mainly focused on evaluating the assignment of single-application tasks to partition nodes. The aim was to reduce the contention inside the partition in order to improve application performance. This scenario corresponds to the use of contiguous strategies where different applications do not share network resources. Recently, due to the use of very large applications, this topic is receiving renewed attention from the research community [4] [5]. To the best of our knowledge, mapping has not been studied in the context of space-shared systems using non-contiguous partitioning. Although there is extensive literature showing the benefits of using application-aware mapping strategies to perform the task-to-node assignment, in practice, only consecutive mapping is used in supercomputers.

1.4 Organization of this dissertation

The work collected in this dissertation revolves around the scheduling process in supercomputers, focusing on stage 2 (partitioning) and stage 3 (mapping). It tries to provide answers to this motivating question: knowing the disadvantages of consecutive partitioning (mainly, low system utilization), and also its advantages (mainly, reduction of job run times), does a carefully-chosen combination of consecutive partitioning and application-aware mapping result in higher job throughput, compared to non-consecutive partitioning alternatives?

The dissertation consists of a compilation of research articles that provides partial answers to the motivating question. It is organized into two parts. Part I presents the framework in which the research work has been developed (Sections 1 and 2) and gives an outline of its scientific contributions (Section 3). It also includes a summary of conclusions and proposals for further research in the topic (Section 4) and a brief description of the papers published by the author during the development of this thesis (Section 5). This part ends with a list of references.

Part II is composed of seven Sections (6 to 12), each one being a scientific paper published by the author of this dissertation. In these sections, the contributions described in Section 3 are explained with a higher level of detail. Each section (publication) includes its own list of references, which complements those given in Part I.

2

Experimental framework

In this section, we detail the experimental setup used during the development of this dissertation. As this work is focused on large-scale supercomputers with different network topologies, we rely on simulation to evaluate the proposed techniques. Our framework is composed of two simulation environments. One is devoted to simulate parallel applications execution. The other simulates all the stages of the scheduling process. A general overview of the simulation environment is given in Figure 2.1.

2.1 Simulation of parallel applications execution

All the experiments carried out to compute running times of parallel applications were performed using INSEE [22], a simulation framework to evaluate interconnection networks. INSEE is capable of simulating many direct network topologies such as meshes, tori and twisted tori and indirect topologies such as thin-trees [24] and fat-trees. It also implements a wide variety of flow control mechanisms and routing strategies. Application traffic can be synthetically generated, or loaded using traces in many formats. In order to evaluate the techniques proposed in this dissertation, many modules were developed and integrated into INSEE. Some of these add-ons include the capability of performing automatic mappings of application tasks to nodes and the simulation of space-shared environments, where several parallel applications run concurrently.

The application traces used to feed INSEE were obtained from the NAS Parallel Benchmarks (NPB) [20], a collection of FORTRAN codes derived from computational fluid dynamics (CFD) applications. NPB consist of the following benchmarks: Block Tridiagonal (BT), Conjugate Gradient (CG), Integer Sort (IS), Lower-Up diagonal (LU), Multi-Grid (MG), Scalar Pentadiagonal (SP) and Fourier Transform (FT). We do not use Embarrassingly Parallel (EP) because it does not make intensive use of the network.

Additionally, we also use a set of synthetic applications generated using the methodology presented in [21] [23]. Using this framework, we can generate traces that implement a selected traffic pattern such as 2D and 3D mesh, waterfall or all-to-all. Each application (trace) can be parameterized with a size (number of communicating tasks) and an intensity (number of packets interchanged by the tasks).

In addition to the network definition, the input for INSEE includes the applications trace file, the sets of nodes that form the partitions, and the corresponding instance-to-partition mapping vectors. 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. This time is used as the application execution time, assuming infinite-speed CPUs. In other words,

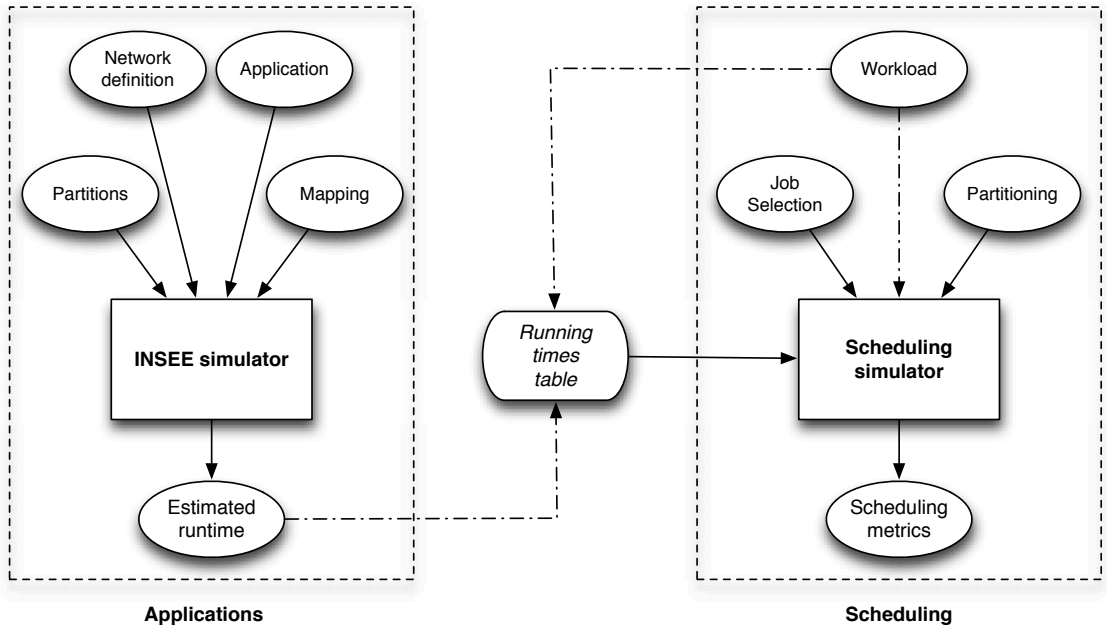


Fig. 2.1. Graphical representation of the simulation environment. The left part represents the simulation of parallel applications and the right part the simulation of the scheduling process. Both simulations can be combined, feeding the scheduling simulator with real applications running times, previously computed using INSEE.

we only consider communication-bounded applications and ignore the effects of execution times at the individual nodes.

2.2 Simulation of the scheduling process

The simulation of the scheduling process has been performed using an in-house developed event-driven simulator. It implements several scheduling policies such as FCFS and BF, and a wide variety of contiguous and non-contiguous partition searching algorithms. The scheduling simulator takes as input a workload that is composed of a sequence of jobs requests, each one with a timestamp, and the application to run (with its parameters, such as size). The simulator also requires as input the run time for that job: it does not simulate the execution of the application.

In this dissertation we have used real and synthetic loads to evaluate the proposed techniques. The real workloads have been extracted from the Parallel Workload Archive (PWA) [28], and contain information about the system on which they were extracted, as described in the Standard Workload Format (SWF) [9]. Two examples of real workloads used in this work are *LLNL Thunder* from a system located in the Lawrence Livermore National Laboratory and *HPC2N* from the High Performance Computing Center North located in Sweden.

For the evaluation of scheduling proposals we have used also synthetic workloads. These kind of workloads provide more flexibility to fine-tune their characteristics. In particular, we use a model developed by Lublin [19] to generate sequences of jobs emulating the actual load received by a system scheduler. Workloads generated with this model are highly customizable by means of a set of parameters that control the characteristics of the jobs using different probability distributions:

1. Interarrival time: Determines the timestamp of jobs arriving to the system. The model calculates this value in two stages: first, the number of jobs arriving at every time interval and then, for each job, its specific arrival time. Both computations are performed using gamma distributions, with different parameters.
2. Job size: Number of parallel tasks of a job. It is modeled by means of a two-stage-uniform distribution.
3. Execution Time: The running time of a job. It is modeled using a hyper-gamma distribution.

The evaluation of some scheduling techniques requires the use of the running times of real applications. In order to perform this kind of simulations, we have to combine both simulation engines. First, INSEE is used to simulate all the application/partition/mapping combinations in the workload, providing as a result a table that contains the running time of each combination. This table is taken as input by the scheduling simulator integrating the running times with the provided workload (real or synthetic). The integration of both simulators is represented graphically in Figure 2.1.

3

Contributions

This dissertation is focused on studying the performance of the scheduling process in space-shared, large-scale supercomputers. These systems are specifically designed to run fine-grained parallel applications in which the communications/computation ratio is high. The way of using the interconnection network has a significant bearing on the performance of the applications and, therefore, on the overall system performance.

After presenting an overview of the scheduling process in the previous sections, and the tools used to analyze it, in this section we summarize the main contributions of this work. Our final goal is to demonstrate that systems managed by schedulers using locality-aware policies (contiguous partitioning together with appropriate mapping strategies) can outperform those using non-contiguous partitioning, even if the achieved levels of utilization are lower. Although this work is mainly focused on cubic topologies such as meshes and tori, we have also explored some aspects of the scheduling process in trees.

3.1 Effects of contiguity on the performance of the applications

Intuitively, applications running inside contiguous partitions should achieve the shortest running times, when compared with non-contiguous partitions. Contiguity reduces the average distance travelled by messages, and eliminates the interference with other applications running concurrently. In order to measure the positive effects of contiguous partitioning on the performance of the applications, we designed a set of experiments, testing different partitioning scenarios.

One of the tested scenarios was a (32×32) 2D torus on which 16 instances of the same application run concurrently. We used four strategies to map application tasks onto network nodes, effectively combining partitioning with consecutive mapping. Row-order and column-order strategies divide the network into non-square partitions (blocks of rows and blocks of columns, respectively). The third strategy divides the network into square partitions (quadrants). The remaining strategy carries out the assignment randomly (therefore, non-contiguously). These arrangements were tested with a collection of applications from the NPB. Here we summarize (see Table 3.1) the results for three applications: LU, MG, and FT.

Results show that the shortest running times are achieved using the quadrant assignment. Applications mapped contiguously in non-square partitions run between two and five times slower. This fact highlights the importance that the partition shape has on application performance. Regarding the random (non-contiguous) assignment, results show that running times are remarkably long. This kind of assignment increases distance traveled by messages, and forces applications to compete for the network resources, creating contention that results in longer running times. See,

	Non-contiguous	Contiguous		
Application	Random	Column-order	Row-order	Quadrant/Row-order
LU	7.9	5.0	5.0	**1.0**
MG	3.6	2.4	2.4	**1.0**
FT	2.5	2.9	2.8	**1.0**

Table 3.1. Running times of three NPB applications on non-contiguous and contiguous partitions. Results are normalized; a value of 1 represents the best assignment.

for example, application LU, which runs 7.9 times faster on a square subnet than on a random collection of nodes.

After gathering evidence about the benefits of contiguous partitions, a second question was investigated: what is the best way of mapping application tasks onto partition nodes? To obtain an answer we evaluated the performance of a single application instance running inside a contiguous partition. The task-to-node assignment (mapping) was performed using three straightforward strategies: random, contiguous in row-order, and contiguous in column-order. Results (summarized in Table 3.2) demonstrated that mapping has an important effect on the running times of the applications, and that there is not a single, universally valid mapping that always achieves the best performance. For example, LU and MG perform the best with consecutive, row-order mappings, while a random mapping is the most appropriate for FT.

	Mapping	
Application	Random	Row-order
LU	2.0	**1.0**
MG	2.1	**1.0**
FT	**1.0**	1.4

Table 3.2. Running times of three NPB applications simulated inside a quadrant (contiguous) partition using two mapping strategies. Results are normalized, in such a way that 1 represents the best assignment.

Results also show that some applications are more sensitive than others to partitioning and mapping. However, in general, we can conclude that the running times of the applications are shorter when running inside contiguous partitions. Additionally, a proper mapping is essential to achieve the best performance. Both conclusions together give support to the fact that schedulers must include topology-aware partition selection and mapping strategies in order to assign resources to applications.

As this scenario is similar to that found in multicore architectures running embedded applications, these conclusions can be applied to such environments. The complete experimental set-up and a comprehensive explanation of the results are presented in Section 6.

3.2 Strategies to map tasks onto contiguous partitions

The results presented in the previous section highlight the importance of choosing a correct strategy to map application tasks to partition nodes. The mappings used in those experiments were simple and did not consider information about application behavior. Using this information, it would be possible to arrange tasks in system partitions in such a way that contention between inter-tasks messages is minimized, reducing application running times. This can be described as an optimization problem, and was the base of two optimization-based mapping strategies.

The first strategy, which we call *classic*, tries to arrange the most communicative tasks onto nodes that are physically close in the network. Apparently, this would be a good choice, because the distance traveled by messages is minimized. However, in practice, this strategy may cause excessive (intra-job) contention when using network resources. In order to reduce this effect, we proposed the *Traffic Distribution* (TD) strategy, which takes into account characteristics of the network (in particular, the routing algorithm) in order to reduce network bottlenecks. Notice that these two strategies are application-aware, because both use information about the communication matrices of the applications.

For the sake of simplicity, we tested the two mapping strategies in single-application scenarios, and compared them with the application-agnostic consecutive and random strategies. Tests were carried out on 2D and 3D partitions. A selection of the results is summarized in Table 3.3. We omit data of the classic strategy because TD always outperforms it.

(a) (8×8) 2D partition.

Application	Consecutive	Random	TD
LU	**1.0**	3.2	1.7
MG	1.2	1.7	**1.0**
FT	1.1	**1.0**	1.2

(b) $(4 \times 4 \times 4)$ 3D partition.

Application	Consecutive	Random	TD
LU	1.1	1.2	**1.0**
MG	**1.0**	1.8	1.4
FT	1.1	**1.0**	1.5

Table 3.3. Running times of three NPB applications using three different mapping strategies in contiguous partitions. They are normalized, in such a way that 1 represents the best strategy.

The analysis of those experiments came up with some interesting results. Application-aware mappings were not able to beat the consecutive and random strategies in some situations. A closer look at the results explains the reason. In the 2D partition (see Table 3.3(a)) LU achieves the best result when used with the consecutive mapping, which is precisely the virtual topology used by LU. The behavior for MG is different, because its virtual topology is 3D, different from the physical one, and in this case TD is the best choice. In summary, when virtual and physical topologies match, the consecutive mapping is unbeatable. However, if there is a mismatch, the TD mapping strategy is the best choice.

Experiments with 3D partitions (physical topologies) confirm this insight, see Table 3.3(b). Consecutive mapping is the best for MG, but TD is the best choice for application LU.

Regarding the FT application, in which inter-tasks communications follow an all-to-all pattern, the best choice is always a random mapping.

Single-application setups are not common in supercomputers. They normally run several applications simultaneously, in a space-shared fashion. Therefore, additional experiments were performed to test multi-application scenarios in which several jobs share the system, running inside contiguous partitions of different shapes. As each application runs internally in its partition, we consider that mapping decisions for single-application environments are still valid.

Results confirm our previous findings. When there is a match of topologies, consecutive mapping is the best performer. Otherwise, TD is the best choice. As regards all-to-all applications, random mapping is the most effective strategy.

Non-contiguous partitioning and consecutive mapping are the most used strategies in production environments. We wanted to test whether or not introducing our TD mapping strategy would be beneficial, even with non-contiguous partitions. Experiments in this set-up, in which virtual and physical topologies never match, showed that TD is an excellent mapping choice for most applications. The exception is once again FT (applications using all-to-all communica-

tions), where random mapping is better, but even in this case TD performs reasonably well. See a summary of results in Table 3.4.

(a) (16×16) 2D torus.

Application	Consecutive	Random	TD
LU	1.40	1.65	**1.00**
MG	1.19	1.48	**1.00**
FT	1.38	**1.00**	1.10

(b) $(8 \times 8 \times 8)$ 3D torus.

Application	Consecutive	Random	TD
LU	1.16	1.14	**1.00**
MG	1.11	1.12	**1.00**
FT	1.68	**1.00**	1.08

Table 3.4. Running times for three NPB applications using different mapping strategies in non-contiguous partitions. They are normalized, in such a way that 1 represents the best strategy.

With the experiments described in this section we have gathered enough evidence about the benefits of using contiguous partitioning with an application-aware mapping in static environments (that is, with a fixed collection of jobs in the system). The complete set of experiments and results are discussed in Sections 7 and 8.

3.3 Topology-aware scheduling

Until now, the gathered evidence shows the benefits of using contiguous partitioning and application-aware mappings. Both techniques reduce the running time required to complete the execution of a job. Therefore, if we include them in the scheduler that manages a supercomputer, then we should expect (1) higher scheduling costs, because it takes longer to find a contiguous partition with a particular shape than to find an arbitrary collection of nodes, and (2) benefits in terms of job running time that should positively affect the job throughput of the systems. However, will this improvement compensate the added scheduling costs?

Before integrating both techniques into the scheduler, a decision has to be made. As we have seen in Section 1.2, jobs can ask for a node count or can request a specific partition shape. In practice, the common situation is the first one; therefore, we will make the scheduler responsible for choosing the appropriate partition shape, guided by a certain *partition selection strategy*. We propose three alternatives.

The Cubic Selection (CS) strategy uses cubic partitions for all requests. Given the good topological characteristics of these partitions, they are expected to be an adequate choice for most applications [8]. The second strategy, Factor Selection (FS), searches for all factorizations of the number of nodes. The aim of this strategy is to reduce the waiting time of the jobs, and it does so by adding flexibility when accepting candidate partitions. Note that CS and FS are application-agnostic partition selection strategies, in contrast with the last one, Best Partition Selection (BPS), which is application-aware: given the communication pattern of an application, it chooses the topology (and the corresponding mapping) that allows it to run the fastest.

After the scheduler decides the partition shape, and locates it in the system, the mapping of the jobs tasks onto the partition nodes has to be performed. We evaluated all the mapping alternatives presented in Section 3.2, in combination with the CS and FS strategies. For BPS, as just stated, the mapping which is used depends on the communication pattern of the application and corresponds to the best performer.

As we were interested in determining which partition selection and mapping strategies provide the best scheduling performance, we used the scheduling simulator to obtain a collection of

performance metrics under a wide variety of workloads. This diversity is important in our analysis because not all applications are equally sensitive to the way resources are assigned to them.

Table 3.5 summarizes some results for systems managed by a scheduler using these partition selection strategies. In all cases, numbers correspond to the mapping providing best results (BM stands for best mapping). If we focus on the average running times of the jobs we can see that BPS achieves the shortest (best) times. As a result of the reduction of running times, resources become available earlier, which is reflected in the waiting time and in the job throughput of the system. This is the final answer to our question: under some conditions (that is, considering application-specific features for partitioning and mapping), contiguous partitioning makes a better utilization of a supercomputer than non-contiguous alternatives.

At this point it is important to pay attention to a metric commonly used to measure the level of efficiency of a supercomputing infrastructure: system utilization. In theory, the higher this value, the more productive the system is. We can see that, using BPS, system utilization is below 80%, while other partitioning-mapping strategies raise this value over 90%. However, we have just described how BPS's job throughput (the real measure of the productivity of the system) is the highest. Therefore, system utilization is not a good metric to measure the performance of the scheduling process.

Part. Selection-Mapping	Avg. Waiting Time	Avg. Runtime	Job Throughput	Utilization
CS-BM	2243.97	314958.30	4472.06	0.90
FS-BM	2286.43	320069.00	4396.60	0.92
BPS	**2087.50**	**260282.20**	**4807.37**	**0.79**

Table 3.5. A selection of performance metrics for a system running a workload of large jobs, using different partition selection strategies. Chosen mapping is always the best one.

Even though our hypothesis has been confirmed, we know that the advantages of contiguous partitioning depend very much on the characteristics of the applications running in the system, which may vary from one supercomputing facility to another. If production systems use non-contiguous partitioning, it is because their advantages in terms of scheduling waiting time are immediate, but the penalization in terms of running times is, in general terms, unknown. We ran some experiments to determine which level of application slowdown would cancel the scheduling benefits of non-contiguous partitioning. Results have been summarized in Table 3.6, for different workload classes (which differ in the average job size).

Workload	CS-BM	FS-BM	BPS
Large	35	37	**25**
Medium	42	40	**27**
Small	45	42	**28**

Table 3.6. Percentage of running time increase (relative to contiguous partitioning) that cancels the benefits of using non-contiguous partitioning.

Results show that when application running times suffer a penalty over 25-30% (compared with the optimum set-up), the popular non-contiguous partitioning is not a good choice. It achieves excellent levels of system utilization (it has been measured: higher than 99%, in contrast with 80% achieved by BPS), but we already know that this metric is not the one to be taken into account. Additionally, if BPS leaves 20% of "idle" nodes, it is possible to make use of them

by means of other job selections strategies (such as backfilling) that would further increase job throughput, or to put them in a low-power state in order to have a greener supercomputer.

The results discussed here, and presented in detail in Section 9, are valid only in the context of systems built around direct interconnection networks with cubic topologies. The effect of contiguous allocation on indirect networks with tree topologies will be explored later in Section 3.6.

3.4 An efficient algorithm to search for contiguous partitions

In the described scenario of contiguous partitioning, the scheduler has to decide the partition shape where each job will run. After this decision is made, a searching algorithm has to be used to locate in the network a collection of idle nodes with that particular shape. In large systems composed of thousands of nodes, this search can be very time consuming, even more taking into consideration that the search has to be performed each and every time a job arrives or leaves the system. In consequence, the use of a very time-efficient partition searching algorithm is required to avoid the scheduler becoming a bottleneck.

In order to efficiently deal with such large networks we have developed Improved First Fit (IFF), an algorithm implementing the first fit searching strategy for multi-dimensional cube networks. IFF maintains information about the network status, allowing the algorithm to rapidly decide whether or not a requested partition shape is available. IFF has been tested against other state-of-the-art algorithms such as Right of Busy Submeshes (RBS) [11], reported in the literature as the most efficient algorithm for this kind of searches.

(a) (1024×1024) 2D mesh.

	Workload Set				
Algorithm	1	2	3	4	5
RBS	12.31	3.93	0.69	**0.08**	**0.05**
IFF	**0.10**	**0.10**	**0.11**	0.15	0.23

(b) $(16 \times 16 \times 16 \times 16 \times 16)$ 5D torus.

	Workload Set				
Algorithm	1	2	3	4	5
NFF	1677.00	1578.00	1815.00	–	–
IFF	**8.00**	**9.00**	**12.00**	**13.30**	**16.80**

Table 3.7. Average allocation times, in milliseconds, achieved by a) RBS and IFF in a 2D mesh and by b) NFF and IFF in a 5D torus. Searching algorithms have been evaluated using five sets of workloads with different average sizes, from small jobs (Set 1) to large jobs (Set 5).

Experimental results show that IFF performs very efficiently in the tested, large size networks, composed of one million nodes. If we compare it with RBS (see Table 3.7(a)), we can see that in general IFF performs better. When dealing with large jobs, RBS is faster, but IFF is still competitive.

A limitation of RBS is that it has been defined only for 2D networks, while supercomputers are evolving towards higher dimensionality. IFF can be extended to cubes with any number of dimensions. We have tested it in a 5D torus (see Table 3.7(b)), comparing its performance against NFF, a straightforward implementation of the first fit searching strategy (to the best of our knowledge, no partition searching algorithm has been published that can be generalized to any number of dimensions). Results show that IFF performs very efficiently in a 5D network with one million nodes, while the searching complexity of NFF makes it impractical in large networks. A detailed description of IFF, including a complexity analysis and a complete set of experiments to assess its performance is presented in Section 10.

3.5 Metrics for partition selection

Some of the results presented in Section 3.2 showed that well-chosen mappings improve the performance of applications running in non-contiguous partitions. In these scenarios, where the virtual and physical topologies do not match and different applications compete for the use of the resources, the use of application-aware mappings, such as TD, help to reduce network contention and therefore reduce the running times of applications.

In these scenarios of non-contiguous allocation, jobs only request a node count and the scheduler has no mechanism guiding it to search for the most appropriate of all the available partitions that could hold the job. In order to help provide such a mechanism, we have developed several metrics to assess the quality of contiguous and non-contiguous partitions when running a specific application. Using them, the scheduler can rank possible alternatives, selecting the most appropriate for the job at hand. Metrics take into account, in addition to the characteristics of the partition and of the underlying network, the traffic pattern of the application and, in some cases, the number of bytes interchanged between tasks.

We define the Task Average Distance (TAD) metric as the average distance between communicating application tasks. TAD is computed taking into account the traffic pattern that the application uses. In addition to TAD we also evaluate its weighted version, WTAD, which also considers the amount of data exchanged between its tasks. From previous results, we know that very communicative tasks allocated physically close increase the risk of network contention, therefore affecting the running time negatively. In order to consider the level of contention created by an application, we define TAD_AL and WTAD_AL. These metrics incorporate knowledge about the asymmetry level (AL) that measures the path diversity between node pairs. Low levels of asymmetry provide higher diversity of paths and are considered better.

We evaluated our proposals against five, application-agnostic, metrics found in the literature. In particular we use the Nodes Affected (NA) and Links Affected (LA) metrics that measure the number of nodes (links) that could be affected by the communications (actually assuming an all-to-all traffic pattern). The remaining three are the Average Distance (AD), Distance From Center (DFC) and Diameter (Diam) metrics that measure different distances between the partition nodes. All the metrics have been tested in 2D and in 3D cube topologies.

	Application-agnostic					Application-aware			
Benchmark	NA	LA	AD	DFC	Diam	TAD	WTAD	TAD_AL	WTAD_AL
LU	-0.23	-0.31	-0.12	-0.13	-0.15	0.42	0.88	0.69	**0.92**
MG	0.44	0.42	0.51	0.52	0.46	0.81	0.83	0.88	**0.89**
FT	0.90	0.88	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	0.74	0.74

Table 3.8. Pearson correlation coefficient between a collection of metrics (both application-agnostic and application-aware) and running times, for three different 64-task applications, on non-contiguous partitions of a $(8 \times 8 \times 8)$ 3D mesh.

In Table 3.8 we have summarized some results obtained in a 3D mesh. They show that previous application-agnostic metrics do not correlate with the running times of the applications: they are not good indicators of how good a partition is to run an application. This is not true for FT, due its particular traffic pattern (notice that some of the metrics actually consider an all-to-all communication pattern). The new metrics we propose present better properties, being excellent indicators of the execution times that we can expect when running an application on the assigned partition. The definition of these metrics, and a complete study of their characteristics, are presented in Section 11.

3.6 Partitioning on trees

In previous sections, we have been focused on mesh and torus topologies. However, tree networks are widely used to interconnect computing nodes in supercomputers. We have also evaluated the possibility of using contiguous allocation on trees.

As discussed in Section 1.2, the definition of contiguity in trees is different from that used in cubes. However, the effects of fragmentation, which translates on low utilization, are the same. In order to reduce the negative effects of contiguous partitioning in trees, the concept of quasi contiguity was introduced. A quasi-contiguous partitioning strategy selects sets of nodes that maintain the maximum possible number of tasks contiguously. The aim is to maintain the intra-jobs communication as private as possible, reducing the contention created in the network links. As finding quasi contiguous partitions is easier than finding totally contiguous sets, the average waiting times of the jobs will be reduced and the system utilization should increase.

However, relaxing the contiguity requirement will negatively affect the running time of the applications. In Table 3.9 we have summarized some results that show to what extent running times are affected by different levels of contiguity. The benefits of contiguous partitioning are clear: applications running non-contiguously are between two and three times slower. Regarding the quasi-contiguous partitioning, the performance is always good, being running times only 30-50% higher to those obtained with a purely contiguous allocation.

	Contiguous	Quasi-contiguous				Non-contiguous
Application	0-task	1-task	2-task	3-task	4-task	16-task (Random)
LU	**1.0**	1.30	1.35	1.45	1.30	3.20
MG	**1.0**	1.50	1.50	1.49	1.48	2.90
FT	**1.0**	1.49	1.49	1.55	1.51	2.30

Table 3.9. Running times for different levels of contiguity of three 16-task applications in a tree. Values are normalized, so that 1 represents the contiguous allocation. x-tasks represents that *16-x* tasks are allocated contiguously, while the remaining x can be anywhere in the network.

After determining that applications perform better when allocated onto contiguous (or quasi-contiguous) partitions, we wanted to measure the level of speed-up that would be required to compensate the penalty suffered by the jobs in terms of waiting times (derived from the use of more restrictive allocation strategies). Table 3.10 shows some results using two real workloads extracted from the Parallel Workload Archive (PWA) [28]. The quasi-contiguous partitioning is evaluated using four Quasi-contiguity levels (QCT), which represents the percentage of the job tasks that can be allocated non-contiguously.

(a) LLNL

		Quasi-contiguity threshold			
Policy	Contiguous	10%	20%	30%	40%
FCFS	31	18	13	11	7
BF	26	14	9	7	5

(b) HPC2N

		Quasi-contiguity threshold			
Policy	Contiguous	10%	20%	30%	40%
FCFS	24	18	15	11	10
BF	23	18	15	10	9

Table 3.10. Percentage of application speed-up (relative to non-contiguous partitioning) necessary to compensate the scheduled overhead caused by contiguous and quasi-contiguous partitioning. Experiments performed using FCFS and backfilling scheduling policies with two workloads from the PWA.

Results showed that small accelerations (smaller when using the quasi contiguity) are required to compensate the cost of finding contiguous (quasi-contiguous) partitions. These levels of speed-up are, actually, realistic as we have shown in Figure 3.9. Also, notice that if the scheduler uses backfilling, system throughput is higher (the workload is processed faster), and the thresholds at which contiguity is advantageous are lower. The study of partitioning strategies in trees is presented in Section 12.

4

Conclusions and future work

This work has been devoted to improve the performance of the scheduling process in large-scale supercomputers. In particular, we were interested in studying contiguous partitioning combined with the use of application-aware mapping strategies as the main mechanism to achieve such an objective.

The use of contiguous partitioning eliminates the interference at the network level between different applications. As a result, applications achieve lower runtimes than in non-contiguous scenarios. However, the use of this kind of partitions causes fragmentation that results in low levels of system utilization. Along this dissertation we have developed and evaluated strategies to improve the performance of applications when running in contiguous partitions. Results show that system throughput can be remarkably improved using these techniques.

Although the main topic of our work is contiguous partitioning, we have also explored some aspects of the non-contiguous partitioning due to its common use in production environments. Additionally, the partitioning of systems built around tree networks has also been explored. Next, a short description of the main contributions of this dissertation is listed.

- We have studied the effects that communications locality has on parallel applications performance in both torus and tree networks. Results show the sensitivity of many applications to the task placement and that compact partitions do not guarantee the best performance. Both locality and appropriate task-to-node assignments have to be combined and incorporated into the scheduler.

- Several application-agnostic and application-aware mapping strategies were evaluated in both single- and multi-application contiguous scenarios. From the results, we identified the best strategy for given an application-partition combination. For the cases in which virtual and physical topologies do not match, our proposal TD, which considers information related to the behavior of applications, is almost always the best performer.

- We have explored the effect that combining contiguous partitioning with the correct mapping strategy has on the overall system performance. We have also proposed strategies to decide the shape of the contiguous partitions when such information is not provided as part of the specification of the jobs. Results show that the BPS strategy, which selects the best known partition-mapping combination, remarkably improves system throughput. We have also compared the selection strategies using contiguous partitioning with a non-contiguous partitioning strategy. Results show that using contiguous partitioning, system throughput can be higher if applications suffer any slowdown as a side-effect of non-contiguity.

- Motivated by the use of large networks in the evaluation of the proposed techniques we developed IFF, an efficient algorithm to search for contiguous partitions in multi dimensional cube networks.
- Mapping strategies were also evaluated using non-contiguous partitioning. It is precisely in these scenarios where application-aware mapping strategies, such as TD, excel.
- The evaluation of the scheduling process in non-contiguous scenarios requires strategies to select partitions to be assigned to the jobs. As a first step towards this direction, a collection of metrics was proposed and evaluated. Using them, the scheduler can decide, given a set of partitions, which one is the best to run a particular application.
- Regarding tree networks, we have presented some work related to the partitioning stage. After showing that exploitation of the locality in these networks has a very positive impact on application performance, we have also explored contiguous partitioning on them. Additionally we have proposed a non-contiguous strategy to improve system utilization, but maintaining the maximum possible degree of locality between intra-job communications.

Undoubtedly, there is a lot of research to be done regarding the areas discussed in this work. We plan to develop strategies to select partitions in non-contiguous scenarios using the metrics introduced in this work. With them, the scheduler will have knowledge about the application in order to select the most appropriate set of non-contiguous nodes, that is, the one in which the application will perform the best. Additionally, we want to study the performance of the scheduling process using other job selection policies such as Backfilling or Shortest Job First using both contiguous and non-contiguous partitioning.

We are particularly interested in evaluating the use of contiguous partitions combined with application-aware mapping strategies in higher dimensional systems such as the Blue Gene/Q. Applications are not designed for such new topologies and if the mapping is not considered, a mismatch between the virtual and physical topologies will appear.

Regarding tree networks, we have only studied the partitioning stage. We plan to extend our mapping framework to these topologies, in which the concept of locality has to be defined specifically.

One of the handicaps of this work is that simulating large parallel applications is very time consuming. Therefore we require other mechanisms to predict the running times of applications. In particular we are interested in developing mathematical models of them running on contiguous and non-contiguous environments. Using these tools, more complex and detailed scheduling scenarios could be explored. Some work has been already carried out in this direction using the non-contiguous metrics as the starting point. These metrics showed high correlation with applications running times, paving a path to find more accurate models.

5

Publications

As a result of this research work several papers have been published in journals and conferences. In this section we introduce the papers selected for this dissertation, presented in Part II. Other published works have been omitted because they consisted of partial results later extended in other publications, or because they are not directly related to the main theme of this dissertation. This section provides a short description of all these publications.

5.1 Publications included in this dissertation

Section 6: Effects of job and task placement on parallel scientific applications performance

- This work explores how locality affects applications performance. We use some simple task placement strategies with a wide variety of applications. Results show that for some applications, the use of specific placement strategies can remarkably improve applications runtimes. These results confirm that including locality-aware policies into the scheduling process could boost the overall performance.

Section 7: Strategies to map parallel applications onto meshes

- Given the benefits of taking advantage of the locality provided by contiguous partitions, in this work we explore strategies to achieve the maximum performance in single-application environments. We focus on the task-to-node assignment, once a partition is provided. As a result, we identify which mapping strategies are the best for parallel applications, based on their communication pattern.

Section 8: Optimization-based mapping framework for parallel applications

- This work explores the use of mapping strategies in single-application and multi-application environments. We use them in both contiguous and non contiguous scenarios where many applications run concurrently. Results show the importance of the partition topology and application communication pattern in order to select the best mapping strategy. They also show that the use of the correct mapping strategy in non contiguous environments can minimize the contention, improving the performance of the applications.

Section 9: Locality-aware policies to improve job scheduling on 3D tori

- In this work we evaluate the effect of combining contiguous partitioning and mapping strategies. We also evaluate the use of three different partition selection strategies. Results show that the use of the correct partition-mapping combination remarkably reduces the average runtime of applications. In addition, we compare the use of the best partition selection strategy with a non contiguous partitioning strategy. Results show that small applications speed-ups can compensate the cost of using contiguous partitioning.

Section 10: A fast implementation of the first-fit contiguous partitioning strategy for cubic topologies

- In this work we describe and evaluate IFF, an algorithm implementing the first fit strategy to search for contiguous partitions. IFF works with cubic topologies of any number of dimensions. Results show that IFF can deal efficiently with very large networks.

Section 11: Application-aware metrics for partition selection in cube-shaped topologies

- In this work we developed metrics to compare the quality of non-contiguous partitions to execute an application. They consider the traffic pattern of the application, the topology of the partition and the routing used by the network. Results showed high correlation between the runtimes and the proposed metrics.

Section 12: Effects of topology-aware allocation policies on scheduling performance

- In this work the cost of contiguous partitioning in tree networks is explored. In addition a quasi-contiguous partitioning strategy is proposed to minimize the negative effects of using contiguous partitions. A simulation based study is carried out using real workloads. The aim of this study is to compare strategies that take advantage of the contiguity with non-contiguous strategies. Results show that small improvements in applications running time compensate the cost of using contiguous strategies.

5.2 Publications not included in this dissertation

Simulating and evaluating interconnection networks with INSEE

- In this work we describe INSEE, a simulation framework initially designed to study interconnection networks, but which has been enhanced to analyze the performance of applications running in parallel computers. In order to evaluate the techniques proposed in this dissertation, many modules were developed and integrated into it. Some of these add-ons include the capability of performing automatic mappings of application tasks to nodes, the simulation of space-shared environments where many parallel applications run concurrently or the use of non-contiguous metrics to evaluate the partitions where applications are simulated.

Effects of job placement on scheduling performance

- This work contains a preliminary study about the effects of contiguous partitioning in tree networks. It was devoted to measure the improvement required, in terms of applications runtime, to compensate the increase in scheduling time caused by contiguous partitioning.

Parallelization of the quadratic assignment problem on the cell

- In this work we developed an enumeration of the permutations space with the property that two consecutive permutations are at distance one swap. Using this enumeration we can partition the search space into non overlapping sets. Inside each set the evaluation of the fitness function is performed very quickly because we can use the value of the previous permutation. In addition the functions to rank and unrank a permutation are developed. Our interest in the Quadratic Assignment Problem is because the mapping problem can be seen as an instance of the QAP.

Optimization-based Mapping Framework for Parallel Applications

- This study was an early stage of the research done in mapping strategies. In this case we used synthetic traffic to represent the communications of parallel applications. This work was later extended using real application traffic.

A study on the complexity of TSP instances under the 2-exchange neighbor system

- This work studied the complexity of the Traveling Salesman Problem (TSP). The study performed an exhaustive search through the permutations space using the enumeration developed for the QAP problem.

Estudio preliminar sobre la complejidad de las instancias del TSP bajo el sistema de vecinos 2-opt

- This is a preliminary study of the complexity of the Traveling Salesman Problem.

5.3 List of publications

In this section we provide the references of all the scientific works published during the development of this dissertation. The papers included in this dissertation are highlighted.

International journals

- **Jose A. Pascual, Jose Miguel-Alonso and Jose A. Lozano. Optimization-based mapping framework for parallel applications. *Journal of Parallel and Distributed Computing* 71 (2011), pp. 1377-1387.**

- Javier Navaridas, Jose Miguel-Alonso, Jose A. Pascual, Francisco Javier Ridruejo. Simulating and evaluating interconnection networks with INSEE. *Simulation Modeling Practice and Theory* 19(1): 494-515 (2011).

- **Jose A. Pascual, Jose A. Lozano and Jose Miguel-Alonso. Application-aware metrics for partition selection in cube-shaped topologies. Submitted to Parallel Computing (PARCO). Preliminary version available as Technical Report EHU-KAT-IK-05-12.**

- **Jose A. Pascual, Jose A. Lozano and Jose Miguel-Alonso. Efficient contiguous partitioning algorithm for cube-like topologies. Submitted to Concurrency and Computation: Practice and Experience. Preliminary version available as Technical Report EHU-KAT-IK-06-12.**

- **Jose A. Pascual, Jose A. Lozano and Jose Miguel-Alonso. Locality-aware policies to improve job scheduling on 3D tori. Submitted to Parallel Computer (PARCO). Preliminary version available as Technical Report EHU-KAT-IK-03-13.**

International conferences

- **Javier Navaridas, Jose A. Pascual, and Jose Miguel-Alonso. Effects of job and task placement on parallel scientific applications performance. *In PDP 09: Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 5561, Washington, DC, USA, 2009. IEEE Computer Society.**

- **Jose A. Pascual, Javier Navaridas, and Jose Miguel-Alonso. Effects of topology-aware allocation policies on scheduling performance. *Proc. 4th Workshop on Job Scheduling Strategies for Parallel Processing In Conjunction with IPDPS 2009.* Rome, Italy May 29, 2009.**

- **Jose A. Pascual, Jose Miguel-Alonso and Jose A. Lozano. Strategies to map parallel applications onto meshes. *International Symposium on Distributed Computing and Artificial Intelligence (DCAI2010)*, Valencia, Spain.**

- Leticia Hernando, Jose. A. Pascual, Alexander Mendiburu and Jose A. Lozano. A study on the complexity of TSP instances under the 2-exchange neighbor system. *SSCI 2011, Symposium on Foundations of Computational Intelligence (FOCI-2011)*, Paris, France, 11-15, April 2011.

National conferences

- Jose A. Pascual and Jose Miguel-Alonso. Effects of job placement on scheduling performance. *Actas de las XIX Jornadas de Paralelismo*. Castellón, 17-19 Septiembre 2008.

- Jose A. Pascual, Jose A. Lozano and Jose Miguel-Alonso. Parallelization of the quadratic assignment problem on the cell. *XX Jornadas de Paralelismo*. 16-18 Septiembre, A Corua.

- Leticia Hernando, Jose A. Pascual, Alexander Mendiburu and Jose A. Lozano. Estudio preliminar sobre la complejidad de las instancias del TSP bajo el sistema de vecinos 2-opt. *VII Congreso Espaol sobre Metaheursticas, Algoritmos Evolutivos y Bioinspirados (MAEB2010)*, Valencia, Spain.

Technical reports

- Jose A. Pascual, Jose Miguel-Alonso and Jose A. Lozano. Optimization-based mapping framework for parallel applications. *Technical Report EHU-KAT-IK-02-10. University of the Basque Country*, April 2010.

References

1. Adiga, NR et all. An overview of the Blue Gene/L supercomputer. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–22, Los Alamitos, CA, USA, 2002.
2. Y. Ajima, T. Inoue, S. Hiramoto, T. Shimizu, and Y. Takagi. The Tofu interconnect. *IEEE Micro*, 32(1):21 –31, 2012.
3. S. Bani-Mohammad, M. Ould-Khaoua, and I. Ababneh. An efficient non-contiguous processor allocation strategy for 2d mesh connected multicomputers. *Information Science*, 177(14):2867–2883, 2007.
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.
5. 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)*, pages 1–8, 2009.
6. S. H. Bokhari. On the mapping problem. *IEEE Transaction on Computers*, 30(3):207–214, 1981.
7. S. W. Bollinger and S. F. Midkiff. Heuristic technique for processor and link assignment in multicomputers. *IEEE Transactions on Computers*, 40(3):325–333, 1991.
8. B. Broeg, B. Bose, Y. Kwon, and Y. Ashir. Lee distance and topological properties of k-ary n-cubes. *IEEE Transactions on Computers*, 44(8):1021–1030, 1995.
9. S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In *Job Scheduling Strategies for Parallel Processing*, pages 67–90. Springer-Verlag, 1999.
10. D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker. The IBM Blue Gene/Q interconnection network and message unit. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 1–10, New York, NY, USA, 2011. ACM.
11. G.-M. Chiu and S.-K. Chen. An efficient submesh allocation scheme for two-dimensional meshes with little overhead. *IEEE Transactions on Parallel and Distributed Systems*, 10:471–486, 1999.
12. T. Chockalingam and S. Arunkumar. Genetic algorithm based heuristics for the mapping problem. *Computers & Operations Research*, 22(1):55 – 64, 1995.
13. P.-J. Chuang and C.-M. Wu. An efficient recognition-complete processor allocation strategy for k-ary n-cube multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 11(5):485–490, 2000.
14. Cray Inc. `http://www.cray.com/Assets/PDF/products/xe/IDC\_948.pdf`.
15. J. Ding and L. N. Bhuyan. An adaptive submesh allocation strategy for two-dimensional mesh connected systems. In *Proceedings of the International Conference on Parallel Processing*, volume 2, pages 193–200, 1993.
16. D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling, – a status report. In *Job Scheduling Strategies for Parallel Processing*, pages 1–16. Springer Verlag, 2005.

17. A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2.3):195 –212, march 2005.
18. K. Li and K.-H. Cheng. A two-dimensional buddy systems for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing*, 12(1):79–83, 1991.
19. U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63:2003, 2001.
20. NASA Advanced Supercomputer (NAS) division. NAS parallel benchmarks. `http://www.nas.nasa.gov/Resources/Software/npb.html`, 2002.
21. J. Navaridas and J. Miguel-Alonso. Realistic evaluation of interconnection networks using synthetic traffic. In *Proceedings of the 8th International Symposium on Parallel and Distributed Computing*, pages 249–252, Lisbon, Portugal, 2009.
22. J. Navaridas, J. Miguel-Alonso, J. A. Pascual, and F. J. Ridruejo. Simulating and evaluating interconnection networks with insee. *Simulation Modelling Practice and Theory*, 19(1):494 – 515, 2011. Modeling and Performance Analysis of Networking and Collaborative Systems.
23. J. Navaridas, J. Miguel-Alonso, and F. Ridruejo. On synthesizing workloads emulating mpi applications. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1 –8, Miami, Florida, 2008.
24. J. Navaridas, J. Miguel-Alonso, F. J. Ridruejo, and W. Denzel. Reducing complexity in tree-like computer interconnection networks. *Parallel Computing*, 36(2-3):71 – 85, 2010.
25. 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.
26. 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.
27. J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. Optimization-based mapping framework for parallel applications. *Journal of Parallel and Distributed Computing*, 71(10):1377 – 1387, 2011.
28. PWA. Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`.
29. K. Taura and A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Proceedings of the 9th Heterogeneous Computing Workshop*, page 102, Washington, DC, USA, 2000.
30. Top500. Top 500 supercomputer sites. `http://www.top500.org`, November 2012.
31. K. Windisch, V. Lo, and B. Bose. Contiguous and non-contiguous processor allocation algorithms for k-ary n-cubes. *IEEE Transactions on Parallel and Distributed Systems*, 8:712–726, 1995.

Part II

Selected publications

6

Effects of job and task placement on parallel scientific applications performance

Effects of Job and Task Placement on Parallel Scientific Applications Performance

Javier Navaridas, Jose A. Pascual, Jose Miguel-Alonso
Department of Computer Architecture and Technology
The University of the Basque Country UPV/EHU
P.O. Box 649, 20080 San Sebastián, SPAIN
{javier.navaridas, ja-pascual, j.miguel}@ehu.es

***Abstract*—this paper studies the influence that task placement may have on the performance of applications, mainly due to the relationship between communication locality and overhead. This impact is studied for torus and fat-tree topologies. A simulation-based performance study is carried out, using traces of applications and application kernels, to measure the time taken to complete one or several concurrent instances of a given workload. As the purpose of the paper is not to offer a miraculous task placement strategy, but to measure the impact that placement have on performance, we selected simple strategies, including random placement. The quantitative results of these experiments show that different workloads present different degrees of responsiveness to placement. Furthermore, both the number of concurrent parallel jobs sharing a machine and the size of its network has a clear impact on the time to complete a given workload. We conclude that the efficient exploitation of a parallel computer requires the utilization of scheduling policies aware of application behavior and network topology.**

Keywords-interconnection networks; parallel job scheduling; performance characterization; resource allocation; trace-driven simulation.

I. INTRODUCTION

Current high-performance computing facilities are composed of thousands of computing nodes executing jobs in parallel. An underlying interconnection network (such as Myrinet [13], Infiniband [6], Quadrics [19], or an *ad-hoc* network) provides a mechanism for tasks to communicate. Most of these facilities belong to national laboratories or supercomputing centers, and are shared by many researchers (see the Top500 list, [5]). However, it is very uncommon to dedicate all the nodes of a site to run a single application. In most cases, nodes are time and/or space shared among users and applications. For example, in [17] authors describe the scheduling mechanisms in use in the supercomputers of the Numerical Aerospace Simulation (NAS) supercomputer facility, located at NASA Ames Research Center.

Supercomputing sites have one or more job queues to which users send their parallel jobs, where they wait until a scheduler allocates some resources to it. A large variety of scheduling policies have been proposed and are in use in order to manage the queues, many of them based on the First-Come First-Served discipline (see [17] again), typically taking into account some restrictions: different levels of priority, quotas (both in terms of CPU time and number of processors per job), maximum waiting time, etc.

It may be shocking to know that many scheduling policies disregard any knowledge about the topological characteristics of the underlying system; they see the system as an unstructured pool of computing resources. We will discuss in the next section how schedulers assign free nodes—*i.e.* resources—to jobs, independently of the location of those nodes in the network, and return them to the free pool when jobs finish or are cancelled. After a certain warm-up time, which depends on the number and variety of executed jobs, physical selection of allocated resources is close to random: nodes assigned to a given job may be located anywhere in the network. In other words, resources are fragmented.

The reader should note that programmers of parallel applications usually arrange tasks in some form of virtual topology. This is a natural way of programming applications in which large datasets (matrices) are partitioned among tasks [2]. Programmers favor communication between neighboring tasks, under the assumption that this strategy should result in improved performance. If the assigned execution nodes are not in close proximity, programmers' efforts are totally useless. Furthermore, if job placement is arbitrary, the messages interchanged by a job may interfere with those interchanged by other, concurrent ones, in such a way that contention for network resources may be exacerbated. Thus, topological information should be taken into account in the scheduler's decision process to effectively exploit locality and to avoid undesired interactions between jobs.

As we stated before, most job placement policies are not locality-aware. In this work, we want to show how the inclusion of topological knowledge in schedulers can improve the performance of parallel computers. We will explore previous work on this issue, as well as the current state of the art about scheduling tools, focusing on the knowledge of the system they manage. Furthermore, we will discuss the impact that job and task placement has on performance when the network of the parallel computer is based on any of the two most commonly used topologies: fat-tree and torus. To do it so, we carried out a simulation-based performance study in which we fed the networks with different application-like workloads: traces, and synthetic traffic patterns that closely emulate the behavior of actual applications. We tested networks of different sizes, and we explored several alternatives of task allocations for a single parallel job, and job and task allocations for concurrent, parallel jobs. Results support what we stated before about the effects of topology-unaware placement: it results in unnecessarily long execution times.

1066-6192/09 $25.00 © 2009 IEEE
DOI 10.1109/PDP.2009.53

To our knowledge, there is no previously published work measuring the interactions between parallel job schedulers, application's sensitivity to placement, and network topology. This is a gap we aim to start filling, with an exploration of the impact that task placement have on the execution time of parallel applications running on supercomputers with different interconnection networks (tori and fat-trees). We will show how some applications are insensitive to placement, but many others run very efficiently under certain topology/placement combinations. The natural continuation of this work will be the inclusion of topology-aware policies in parallel job schedulers.

The rest of this paper is organized as follows. In Section II we discuss some work on job placement and also explore some schedulers and their job placement policies. The experimental environment—studied networks, workloads and placement strategies—is described in Section III. In Section IV we show and analyze the results of the experiments. Section V closes this paper with some conclusions and an outlook of our plans for future work.

II. Related Work

In the literature we can find a variety of strategies for resource allocation and scheduling. These two problems are strongly interconnected. The use of a good allocation algorithm and a good scheduling policy decreases network fragmentation, allowing contiguous allocation of jobs in the parallel system, which can be taken advantage by applications.

In [23] we can see how the contiguous allocation of tasks resulted in improved application performance. Authors run eight sets of 16-node FFTs—benchmark FT, part of the well-known NAS Parallel Benchmarks [14]—concurrently on a 128-node mesh, and compared contiguous vs. random node allocation. They observed a 40% improvement in runtime when using contiguous allocation. The obvious way to go is to introduce contiguous allocation strategies in schedulers for parallel machines. In some other papers addressing this issue [3, 9, 10, 12, 23] allocation algorithms were proposed mainly for *k*-ary *n*-cube topologies. Figures of merit usually did not show how placement strategies affect the runtime of an application instance, but just the completion time of a list of jobs. In [16] we paid attention to tree-based topologies and relied on contiguous allocation of tasks to, by means of an efficient exploitation of communication locality, dilute or even invert the potentially negative effects of reducing the bisection bandwidth of the network. Interestingly, in [10] authors showed how the requirement of contiguous allocation may cause poor utilization of the system due to external or internal fragmentation. To avoid this effect, they evaluated several non-contiguous, but non-random, allocation schemes that improved overall system utilization.

A review of commercial and free schedulers shows that, by default, they are not topology aware—in other words, they do not take care of the actual placement of tasks. This is true for job queuing systems and scheduling managers such as Sun's Grid Engine [24], IBM's LoadLeveler [7] or PBS Pro [18] (the latter used in Cray Supercomputers [1]). Although some of them provide mechanisms for the system administrators to implement their own scheduling/allocation policies, in practice, this is not done. For example, the scheduling strategy used on Cray XT3/XT4 systems (custom-made 3D tori) simply gets the first available compute processors [1]. Maui [4] and Slurm [8], in use in ASC Purple (IBM Federation network) and BSC's MareNostrum (multi-stage Myrinet), have an option to take into account application placement, but they ignore the underlying topology, considering a flat network, i.e. distance between nodes is considered as the difference between node identifiers. The most notable example of current supercomputer that tries to maintain locality when allocating resources is the BlueGene family (3D tori), whose scheduler [3] puts tasks from the same application in one or more midplanes of 8x4x4 nodes.

III. Experimental Set-Up

We used simulation to assess the impact of allocation strategies on application performance. The simulation environment encompasses a network simulator and a workload generator [21]; we describe them in this section. It is important to remark that our simulator measures time in terms of *cycles*; a cycle is the time required by a *phit* (physical transfer unit, fixed to 4 bytes) to traverse one network switch.

A. Workloads

Throughout this work we evaluate networks using realistic workloads, taken from actual or emulated applications. In particular, we used traces taken from the well-know NAS Parallel Benchmarks [14] (NPB), and a set of application kernels described in [15]. In both cases we assumed *infinite-speed* processors, meaning that we only measured the time used by the network to deliver the messages, but not the time used at compute nodes to generate, receive and process them. Note that message causality is preserved, so when the trace states that a node must perform a receive operation, the simulated node stalls until the expected message arrives—we encourage reader to examine [11] for a deeper explanation of our methodology to perform trace-driven simulation. Under these assumptions, reported results only toke into account the communication and synchronization parts of parallel applications; thus, the actual impact on performance of a given scheduling algorithm would depend on the application's computation to communication ratio.

Regarding traces, we used class A of NPB applications Block Tridiagonal (**BT**), Conjugate Gradient (**CG**), Integer Sort (**IS**), Lower-Up diagonal (**LU**), Multi-Grid (**MG**), Scalar Pentadiagonal (**SP**) and Fourier Transform (**FT**). This study did not include Embarrassingly Parallel (**EP**) because it does not make intensive use of the network. In order to reduce required computing resources, and given that they are iterative applications, we drastically reduced the number of iterations in each benchmark, to only 10 to 20 iterations.

Pseudo-synthetic workloads used in this work are binary-tree (**BI**), butterfly (**BU**), distribution in 2D or 3D meshes (**2M**, **3M**) and wave-front in 2D or 3D meshes (**2W**, **3W**), which are detailed and justified in [15]. The message length was 64 Kbytes. Our experimental set-up also included waterfall (**WF**), a pattern observed in the LU NPB application [22]. We modeled this pattern as a burst of 286 wave-fronts

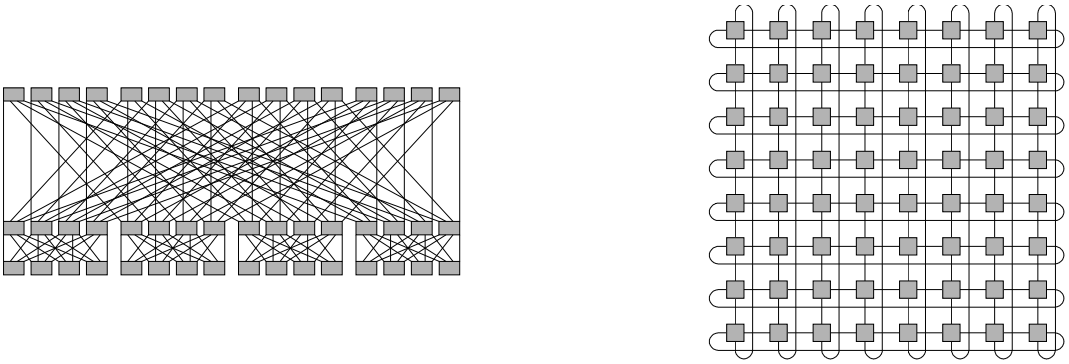


Figure 1. Examples of the topologies used in this study. 4,3-fat-trees (*left*) and 8-ary 2-cube (*right*), both used to interconnect 64 nodes.

(**2W**) starting at once, each of them composed by small messages (256 bytes, or 4 packets).

All of the workloads used in the experiment were captured (or generated) for exactly 64 tasks. In some experiments the network had 64 nodes, so a single application uses the whole computer. In others, the network had 64**N* nodes, so *N* instances of an application shared the computer. Chosen values of *N* were 4 and 16. To simplify the experiments, we never mixed different applications. The figure of merit to measure performance was the time required to consume all the messages in the workload. When using multiple, simultaneous application instances to feed a network, reported time is the one required to complete all the instances (the time taken by the slowest one).

B. Networks and placement

Not all the interconnection networks have the same properties, including the topological ones, and the effect of the placement strategies may vary depending on the network. For this reason, we selected two of the most widely used topologies: fat-trees (typically used to build large-size clusters) and cubes (typically used to build massively parallel computers). The reader can check the Top500 list [5] to see how most computers in the highest positions of the list fit on one of these categories.

In our experiments we used small to medium-size networks, with a number of nodes ranging from 64 to 1024. Given these sizes, we considered only 2D cubes (3D would be recommended for large-scale networks). In order to allow workloads to fit exactly in the network, we used fat-trees built with 8-radix switches. Note how fat-trees raise one level from configuration to configuration. Considering all these restrictions, the networks used in our study are:

- 4-ary, 3-tree and 8-ary, 2-cube (*i.e.* 8x8 torus) for experiments with a single application instance. Both topologies are depicted in Figure 1.
- 4-ary, 4-tree and 16-ary, 2-cube (*i.e.* 16x16 torus) for experiments with 4 instances of the application.
- 4-ary, 5-tree and 32-ary, 2-cube (*i.e.* 32x32 torus) for experiments with 16 instances of the application.

Note that the aim of this paper is *not* to compare the torus against the fat-tree. The evaluation of alternative network topologies goes beyond the scope of this paper. Our focus is on the impact that placement have on the execution time of applications of different sizes, running alone or sharing a parallel computer with other applications.

We assume that parallel jobs are composed of 64 tasks, numbered from 0 to 63. Network nodes are also numbered. In the case of fat-trees, numeration of nodes is: (0,0), (0,1), (0,2), (0,3), (1,0), (1,2), etc., where (*s,p*) should be read as "switch number *s*, port number *p*". Switch numbers correspond, left to right, to the lowest level of the tree, the one to which compute nodes are attached. In the case of 2-cubes, numeration is done using the Cartesian coordinates of the nodes: (0,0), (0,1), (0,2), (0,3), (1,0), etc.

Regarding placement, we consider *task placement* (allocation of the tasks of a single job) and *job placement* (allocation of several jobs that will run concurrently). Actually, we consider task allocation alternatives only for the experiments with a single application instance. In the other cases we evaluate combinations of task and job placement strategies. Now we describe these strategies. In all cases, we assume that assignment is done first in order using the job identifier and that, for a given job, nodes are assigned to task in order of task identifier. In the case of the fat-tree, allocation can be:

- *Consecutive*. Switch/port assignment is done selecting, in order, node (*s,p*), increasing first *p* and then *s*.
- In *shuffle* order. Switch/port assignment is done selecting, in order, node (*s,p*), increasing first *s* and then *p*.

Allocation for the torus can be:

- In *row* order. Assignment is done selecting, in order, node (*x,y*), increasing first *x* and then *y*. This can be seen as partitioning the network in rectangular sub-networks, wider than tall.
- In *column* order. Assignment is done selecting, in order, node (*x,y*), increasing first *y* and then *x*. This can be seen as partitioning the network in rectangular sub-networks, taller than wide.
- When using several application instances, we can partition the network in perfect squares (this is possible because our choice of network and application sizes). We use a *quadrant* scheme in which the network is partitioned this way. Allocation inside each partition is done in row order.

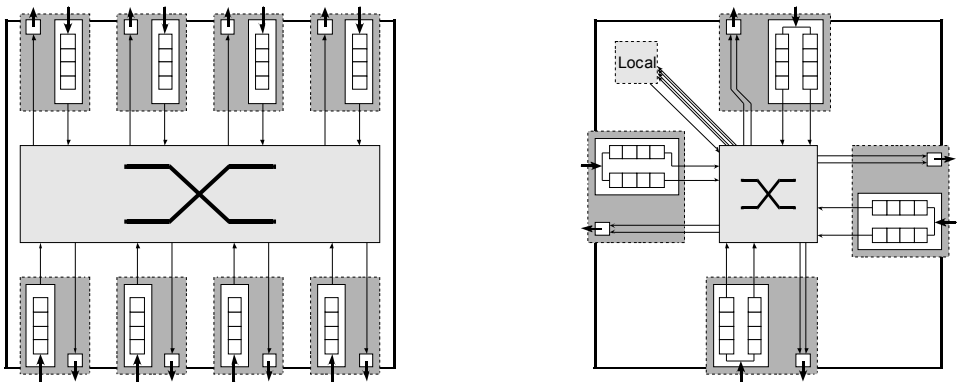


Figure 2. Model of the switches used to build fat-trees (*left*) and 2D tori (*right*). All the ports are depicted showing input queues and output buffers. Note the utilization of two virtual channels sharing a physical link in the torus switch.

Both for torus and fat-tree, allocation of the tasks of *N* 64-task jobs to an *N**64-node machine can be done *randomly*. When running experiments with this placement, we generated five random permutations and plotted the average, maximum and minimum values of the measured execution times.

C. *Models of the components*

Nodes were modeled as reactive traffic sources/sinks with an injection queue able to store up to four packets. In order to model causality, the reception of a message may trigger the release of one or several extra messages as defined by the workloads. When necessary, messages are segmented into fixed-size packets (16 phits). One phit is the smallest transmission unit, fixed to 32 bits. If a message does not fit exactly in an integral number of packets, the last packet contains unused phits.

Simple input-buffered switches were used. Transit queues had room to store up to four packets. The output port arbitration policy was round robin. Switching strategy was virtual cut-through. We depicted the models of switches for the two topologies in Figure 2.

In the case of fat-trees, switches were radix-8. Routing was, when possible, adaptive using shortest paths. A credit-based flow-control mechanism was used, so that when several output ports were viable options to reach the destination, the port with more available credits was selected. Credits were communicated out-of-band, so they did not interfere with normal traffic.

Tori were built using radix-5 switches. Four of the ports were regular transit ports, and the fifth one was an interface with the node. We assumed that the consumption interface was wide enough to allow simultaneous consumption of several packets arriving from different ports. The network relied on bubble flow control [20] to avoid deadlock, making use of two virtual channels: one escape channel in which routing is oblivious DOR (Dimension Order Routing), and an adaptive, minimal routing channel.

IV. Experiments and Analysis of Results

Results of the experiments are depicted in Figure 3. Execution times (actually, communication times) in cycles, as reported by the simulator, were normalized to the best performing task placement, in order to highlight the differences between the different placement strategies. We want to remark that we are not betting for a single, *miraculous* task placement which performs best for all possible applications. In fact, we will see that some applications were not responsive to task placement, or even to the underlying topology. Plots do not allow for a direct comparison of topologies (because values are not absolute) but, as we stated before, this is not the focus of this paper.

For the smallest networks (64-node networks and a single application instance) both in torus and fat-tree, differences between the best and the worst performing placement strategy reached a 250%. This is very significant for such a small network. In general, although there were exceptions, the random placement yielded the worst results, consecutive placement was the best performer for the fat-tree, and both row and column placements performed equally well in the torus topology.

For the medium size configurations (256-node networks and 4 concurrent application instances), the worst-to-best ratio grew up to over 300%, reaching 400% and 450% in the most adverse cases (**LU** in fat-tree and **BT** in torus, respectively). Again, consecutive placement was the best performer in the fat-tree network. In the case of the torus, the best performing strategy was quadrant placement, with the single exception of **MG**, for which row and column placements work equally well.

Finally, for the largest systems in our evaluation (1024-node networks and 16 concurrent application instances) in the fat-tree the ratio for some of the patterns was around 500% and reaches 600% in the most adverse cases. In the case of the torus, these ratios went even higher, being around 700%, and reaching 850% in the worst case. The best placement options were those described for the medium size case. In general, the negative impact of a bad placement depends heavily on the network size. More exactly, on network dis-

tance, that depends on the height (number of levels) of the fat-tree and on the length of the rings of the torus.

If we focus on applications, we can see how **LU**, **BT** and **SP** were very sensitive to task placement, regardless of the topology. This is because their communication patterns cause a significant degree of contention for resources. The interferences between communications from different instances worsen this contention, which in turns increased even more the communication time. On the other hand, **2W** and **3W** were the workloads less responsive to task placement or topology. This is because the high degree of causality intrinsic to their traffic patterns does not saturate the network; thus, in the absence of contention for resources, message delay depends slightly on distance, so the short differences. **IS** also showed not being very sensitive to the placement regardless of the topology.

It is interesting to observe that some workloads were very sensitive to placement when running on one topology, but not that much when the network was different. Extreme examples are **2M** and **BU**. The former adjust perfectly to a mesh topology, and was able to take full advantage of this situation when the placement allowed it. However, **2M** does not map naturally on a fat-tree, so the choice of placement on this network was almost irrelevant. Regarding **BU**, the perfect marriage between this pattern and the fat-tree was exploited only with the consecutive placement, and worked on the torus equally well (or bad) with any placement. For a more detailed explanation of this effect, the interested reader can see [15].

Moreover, if we focus on the plots for single-instance experiments, we can see how consecutive allocation strategies were not always the best performers. Let us pay attention to results of **FT** in the torus. Row and column placements performed worse than random placement. It happens that the allocation strategies we tested were not optimal for this pattern, because its regularity lead to the occurrence of highly congested *hot paths*. Random allocation scatters these contention spots around the network, thus its performance was better. For the multi-instance experiments with **FT**, the quadrant allocation of jobs avoided harmful interferences between instances, an effect that overshadowed the bad task allocation. We would expect better results if we perform the same quadrant job allocation, but with a better task allocation, even random, inside each quadrant.

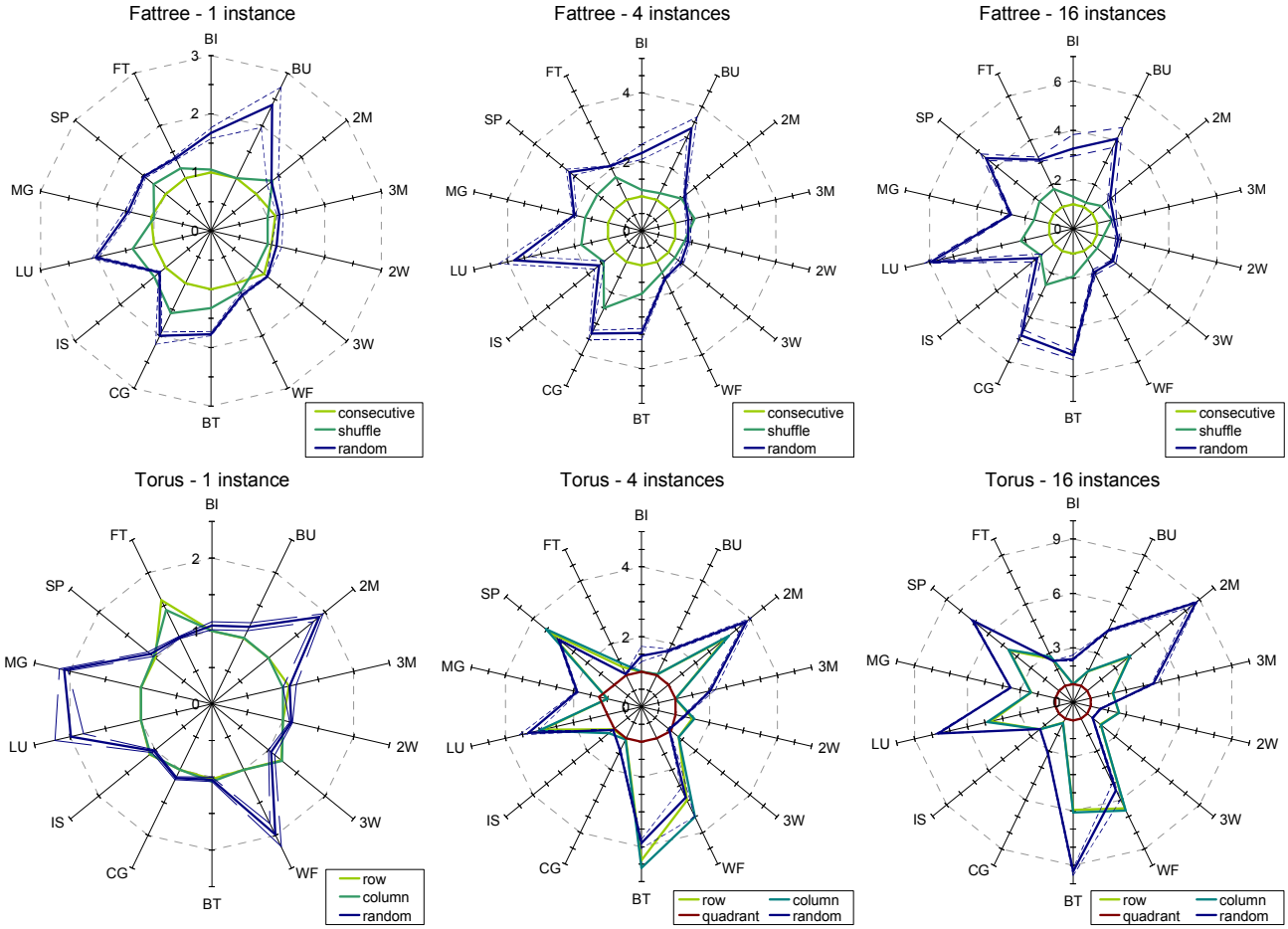


Figure 3. Results of the experiments with different networks, network sizes and workloads. They are normalized, in such a way that 1 represents the execution time for the best placement. Dotted blue lines represent best and worst results for random placement.

To summarize this analysis, we can state that the choice of placement has a very relevant impact on performance on some applications, a fact that should not be taken lightly. Other applications are insensitive to placement, and could be used to fill fragmented gaps of the network, in order to increase system utilization with a minimal impact on the overall performance.

The best performing placements were those that allow for a good matching between the virtual topology (spatial distribution of application's communication) and the physical one, because this way communication locality can be exploited effectively. With very few exceptions, the random placement was the worst performer. The actual benefit of a placement strategy depends heavily on the application and the network topology, but our analysis showed that the flat-network supposition embedded in many schedulers is too simplistic and must be reconsidered in order to accelerate the execution of applications.

We want to remark again that the results presented in this work were obtained under the assumption of infinite-speed processors. Parallel applications pass through computation phases, in addition to communication phases. Our experiments showed how communication can be improved using a good placement; however, computation is not directly affected by placement. Therefore, the actual impact of placement on execution speed would depend on the communication/computation ratio of the application. In other words, the benefits we announce for good placement strategies will be diluted when running actual applications on actual machines. For example, for a 10:1 computation-communication ratio, a 450% increase in communication time will increase total execution time over 30%, which in our opinion still makes worthwhile to continue doing research on this topic

V. Conclusions and Future Work

Most parallel applications rely on different virtual topologies to arrange their tasks, and communication is usually performed between neighboring tasks. When the topology of the physical network matches the virtual one, application performance boosts. Otherwise the interchange of messages is not done in an optimal way, and performance suffers. Even when virtual and actual topologies are similar, a task allocation mechanism that does not allow a good matching between them will result in the impossibility of efficiently exploiting the potential of the network.

In this paper we study the impact of job and task placement strategies on the time parallel applications spend interchanging messages. To do so, we carried out a simulation-based study with two kinds of workloads: traces from applications and application-inspired synthetic traffic. We focused our study on two different network topologies widely used in current supercomputers: tori and fat-trees. We used some very simple placement strategies, as well as random placement.

Results showed that for a small 64-node network in which we run just one application, for almost half of the workloads the difference in speed between the worst and the best placement was around 200%. When increasing the number of concurrent application instances and the size of the network, these differences were more noticeable, reaching increases in excess of 300% for 256-node networks, and close to 1000% for 1024-node networks. The obtained improvements are only applicable for communication phases of the applications, being the computation phases unaffected by placement.

In contrast, and depending on the host topology, some applications or kernels were shown to be only slightly sensitive to task placement. In these cases, the effort of looking for a consecutive region of the network will not pay off. The positive side is that, when mixing different applications, the placement-insensitive ones are good candidates to be used to fill gaps that would otherwise remain unused while placement-sensitive applications are waiting for a consecutive portion of the system.

We conclude that the inclusion of locality-aware placement policies within scheduling tools could boost parallel application performance. The way to carry out this inclusion is still a line of research. We plan to apply different optimization techniques in order to decide the degree of responsiveness to task placement of an application. If this degree is low, we can use the application to fill fragmented regions of the network. Alternatively, if an application is sensitive to placement, we should find appropriate placement for it, even when sharing a parallel computer with other jobs.

Initial results showed that dividing a network in sub-networks with the same topology result in excellent performance, especially when these networks match the virtual topologies used within applications. Still, both the pros and cons of this approach have to be considered, because the effort required to allocate an optimal sub-network may surpass the possible performance drop derived from a simple, random allocation.

The work described in this paper is focused on parallel applications running on high performing computing systems, and on the kind of interconnection networks used in them. However, the effects of efficiently exploiting locality could be even more noticeable when using a hierarchy of networks. Let us consider a cluster of multiprocessors. In this machine, the communication time within an on-chip network is smaller than that of the external node-to-node network, so if communicating tasks are located in the same node, the execution time should be improved. Furthermore, if the computing resource is a grid of clusters, the cluster-to-cluster communication links are orders of magnitude slower than the other networks, so the allocation of processors for the tasks of a job must avoid the utilization of these links.

Acknowledgments

This work has been supported by the Spanish Ministry of Education and Science, grant TIN2007-68023-C02-02, and by Basque Government grant IT-242-07. Mr. Javier Navaridas is supported by a doctoral grant of the UPV/EHU. Mr. Jose A. Pascual is supported by a doctoral grant of the Basque Government.

REFERENCES

[1] R Ansaloni, “The Cray XT4 Programming Environment”. Slides available (November 2008) at: http://www.csc.fi/english/csc/courses/programming_environment

[2] Y. Aoyama and J. Nakano. "RS/6000 SP: Practical MPI Programming". IBM Red Books SG24-5380-00, ISBN 0738413658. August, 1999.

[3] Y Aridor, T Domany, O Goldshmidt, JE Moreira and E Shmueli “Resource allocation and utilization in the Blue Gene/L supercomputer”. IBM J. Res. & Dev. Vol. 49 No. 2/3 March/May 2005. Available (November 2008) at: http://www.research.ibm.com/journal/rd/492/aridor.pdf

[4] Cluster Resources. “Maui Admin Manual”. Available (November 2008) at: http://www.clusterresources.com/products/mwm/moabdocs/MoabAdminGuide52.pdf

[5] JJ Dongarra, HW Meuer and E Strohmaier. “Top500 Supercomputer sites”. Available (November 2008) at: http://www.top500.org/

[6] Infiniband Trade Association. “Infiniband® Trade Asociation”. Available (November 2008) at: http://www.infinibandta.org

[7] S Kannan, M Roberts, P Mayes, D Brelsford and JF Skovira “Workload Management with LoadLeveler”. IBM Red Books SG24-6038-00. ISBN 0738422096. November 2001.

[8] Lawrence Livermore National Laboratory. “Simple Linux Utility for Resource Management”. Available (November 2008) at: https://computing.llnl.gov/linux/slurm/

[9] Y Liu, X Zhang, H Li and D Qian. “Allocating Tasks in Multi-core Processor based Parallel System”. 2007 IFIP International Conference on Network and Parallel Computing Work-shops (NPC 2007), September 2007 pp. 748-753.

[10] V Lo, KJ Windisch, W Liu and B Nitzberg “Noncontiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers”, IEEE Transactions, on Parallel and Distributed Systems, July 1997 (Vol. 8, No. 7) pp. 712-726. DOI: 10.1109/71.598346

[11] J. Miguel-Alonso, J. Navaridas and F.J. Ridruejo. “Interconnection network simulation using traces of MPI applications”. International Journal of Parallel Programming, in press. DOI: 10.1007/s10766-008-0089-y

[12] DH Miriam, T Srinivasan and R Deepa. “An Efficient SRA Based Isomorphic Task Allocation Scheme for k-ary n-cube Massively Parallel Processors”. International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), September 2006 pp. 37-42.

[13] Myricom. “Myrinet home page”. Available (November 2008) at: http://www.myri.com/

[14] NASA Advanced Supercomputing (NAS) division. “NAS Parallel Benchmarks” Available (November 2008) at: http://www.nas.nasa.gov/Resources/Software/npb.html

[15] J Navaridas, J Miguel-Alonso and FJ Ridruejo. “On synthesizing workloads emulating MPI applications”. The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-08). April 14-18, 2008, Miami, Florida, USA.

[16] J Navaridas, J Miguel-Alonso, FJ Ridruejo and W Denzel “Reducing Complexity in Tree-like Computer Interconnection Networks”. Technical report EHU-KAT-IK-06-07. Department of Computer Architecture and Technology, The University of the Basque Country. Submitted to Elsevier’s Journal of Parallel Computing

[17] J Patton-Jones and B Nitzberg. “Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization.” In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1659, pages 1-16. Springer-Verlag, 1999.

[18] PBS GridWorks. “PBS Pro”. Available (November 2008) at: http://www.pbsgridworks.com/

[19] F Petrini, W Feng, A Hoisie, S Coll and E Frachtenberg. “The Quadrics Network: High-Performance Clustering Technology”. IEEE Micro 22, 1 (Jan. 2002), 46-57. DOI: 10.1109/40.988689

[20] V Puente, C Izu, R Beivide, JA Gregorio, F Vallejo and J M. Prellezo “The Adaptive Bubble router”, Journal on Parallel and Distributed Computing, vol 61, Sept. 2001. DOI: 10.1006/jpdc.2001.1746

[21] FJ Ridruejo and J Miguel-Alonso. “INSEE: an Interconnection Network Simulation and Evaluation Environment”. Lecture Notes in Computer Science, Volume 3648 / 2005 (Proc. Euro-Par 2005).

[22] FJ Ridruejo, J Navaridas, J Miguel-Alonso and C Izu “Realistic Evaluation of Interconnection Network Performance at High Loads”. The International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Adelaide, December 3-6, 2007.

[23] V Subramani, R Kettimuthu, S Srinivasan, J Johnson and, P Sadayappan “Selective Buddy Allocation for Scheduling Parallel Jobs on Clusters”. Fourth IEEE International Conference on Cluster Computing, (CLUSTER'02), September 2002 pp. 107.

[24] Sun Microsystems, Inc. “N1 Grid Engine 6 User’s Guide”. Available (November 2008) at: http://docs.sun.com/app/docs/coll/1017.3.

7

Strategies to map parallel applications onto meshes

Strategies to Map Parallel Applications onto Meshes

Jose A. Pascual, Jose Miguel-Alonso, and Jose A. Lozano

Abstract. The optimal mapping of tasks of a parallel program onto nodes of a parallel computing system has a remarkable impact on application performance. We propose a new criterion to solve the mapping problem in 2D and 3D meshes that uses the communication matrix of the application and a cost matrix that depends on the system topology. We test via simulation the performance of optimization-based mappings, and compare it with consecutive and random trivial mappings using the NAS Parallel Benchmarks. We also compare application runtimes on both topologies. The final objective is to determine the best partitioning schema for large-scale systems, assigning to each application a partition with the best possible shape.

Keywords: mapping, parallel applications, QAP, k-ary n-mesh, scheduling.

1 Introduction

Message-passing 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 scheme that tries to use efficiently the underlying network. The way tasks are arranged to perform communications is called the virtual topology. However, these applications, once programmed, can be executed in a wide variety of parallel architectures, with different interconnection networks. These architectures vary from clusters with simple LAN-based networks to supercomputers with custom-made networks organized as meshes, tori, trees, etc. Given this variety of interconnects, it is not uncommon to find a mismatch between the physical arrangement of the compute elements (that depends on the system's topology) and the virtual topology

Jose A. Pascual, Jose Miguel-Alonso, and Jose A. Lozano
Intelligent Systems Group, The University of the Basque Country UPV/EHU
e-mail: {joseantonio.pascual,j.miguel,ja.lozano}@ehu.es

A.P. de Leon F. de Carvalho et al. (Eds.): Distrib. Computing & Artif. Intell., AISC 79, pp. 197–204.
springerlink.com

(application dependent). This may result in an inefficient utilization of the network, that materializes in large delays and bandwidth bottlenecks that, in turn, results in a general performance loss. The way of mapping tasks onto network nodes has a remarkable impact on the overall system performance [10] [12] [1].

The most used network topology [5] in current supercomputers is the 3D torus. This topology provides desirable topological characteristics such as symmetry, low node degree and low diameter. However, sharing a 3D torus between multiple applications results in partitions with the form of 2D or 3D sub-meshes, maybe with some wrap-around links. Most of the research made in partitioning 3D topologies [6] focused on the search for 3D sub-topologies, assuming that all applications will benefit from them. However, this assumption does not take into account the virtual topologies of applications, that may not match with a 3D (sub-)mesh. Moreover, the search for just one shape for the partitions can affect the scheduling performance, due to an increase of external fragmentation [7].

In this paper, we adapt an optimization-based mapping strategy previously developed for 2D topologies [12] to work with 3D meshes. The procedure is expressed as an optimization problem, in which a target functions has to be minimized. Within this strategy, we can use a classic optimization criterion that tries to minimize the average distance traversed by messages. We also proposed an alternative criterion, called TD (Traffic Distribution) that tries to reduce contention for the use of network resources. Within this framework, a given mapping is better than another one if the value of the target function is smaller; still, we must check if a theoretically better mapping actually makes the application run faster. In order to carry out this validation we use simulation. The simulation workbench is INSEE [14] which, given a trace of an application, a target network, and a mapping, can provide an estimations of the execution time. As applications we use (traces of) a subset of the well-known NAS Parallel Benchmarks (NPB) [9]. For simplicity, we consider only configurations of 2D and 3D meshes in which the number of tasks equals the number of network nodes. Regarding mappings, in addition to those obtained using optimization, we have tested some trivial ones: consecutive and random.

All this work is necessary to answer the following question. If we have a very large-scale system, arranged in a 3D structure, in which we want to allocate a much smaller application, which option would be better, a 3D sub-structure (a sub-cube or sub-mesh) or a 2D structure (a plane, or a portion of it)? The way we deal with this issue will have a great impact on terms of application performance, and also on terms of scheduling costs. As we will see, 3D sub-structures perform better for many applications, but not for all. Meanwhile, it may be easier (less expensive, in terms of scheduling costs) to partition a cube into planes instead of sub-cubes.

2 Optimization-Based Mapping Framework

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 $\pi : T \longrightarrow P$ that assigns

a task t to a node p 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 (Quadratic Assignment Problem) [11] 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:

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

The formulation of the problem as an instance of the QAP allows computing mappings using techniques developed for the generic QAP. In particular, we used a GRASP (Greedy Randomized and Adaptive Search Procedure) [13] solver, which is fast and provides the best results known when solving the generic QAP. This constructive, multi-step algorithm iterates over two steps: first, an initial solution is created by means of an adaptive greedy randomized algorithm; then a local search improves that solution. A detailed explanation of the algorithm can be found in [12].

3 Mapping Criteria on Meshes

The mapping problem has been stated often as a location problem [3], 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 running time, because it may create contention hot-spots inside the network [1] [2] [12]. For this reason, we propose a different cost matrix to be used instead of the distance matrix. The criterion of minimizing average distance is relaxed, in order to favour communication paths that distribute traffic through different network axes, reducing the risk of contention. We have called this the *Traffic Distribution* (TD) criterion.

The definition of both criteria (cost matrices) depends on the topological characteristics of the target network in which applications will run. In this work we only consider meshes with two and three dimensions. 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) [4], in which messages traverse first all the necessary hops in each dimension, but we do not make any assumption regarding the routing algorithm, except that it must use minimal paths.

We will describe the different criteria for 3D meshes with $N = n_x \times n_y \times n_z$ nodes. The expressions for 2D meshes can be easily inferred, assuming that the number of nodes in the Z axis is 1. Given two nodes with identifiers i and j, the distance between them is given by Equation 2.

$$dist(i,j) = dist_x(i,j) + dist_y(i,j) + dist_z(i,j) \tag{2}$$

where each term of the sum is the number of hops through each axis a message must traverse when going from i towards j. Equation 3 shows terms $dist_x$, $dist_y$ and $dist_z$.

$$\begin{aligned} dist_x(i,j) &= |(j \bmod (n_x \times n_y)) \bmod n_x - (i \bmod (n_x \times n_y)) \bmod n_x| \\ dist_y(i,j) &= |(j \bmod (n_x \times n_y))/n_y - (i \bmod (n_x \times n_y))/n_y| \\ dist_z(i,j) &= |j/(n_x \times n_y) - i/(n_x \times n_y)| \end{aligned} \tag{3}$$

The classic criterion identifies C as the distance matrix, whose components can be filled using Equation 2 for each pair of source-destination nodes.

$$C = [c_{i,j}]_{i,j=1\ldots N} \text{ where } c_{i,j} = dist(i,j) \tag{4}$$

In this paper, we adapt the TD mapping criterion developed in [12] to 3D meshes. The new criterion is expressed as an alternative cost matrix that allows us to find mappings that, while not optimal in terms of distance, avoid network bottlenecks. In particular, we pay attention to the number of hops per dimension that messages have to travel. In a 3D mesh, a 6-hop route in which all the hops are in the X axis is worse than another 6-hop route with 2 hops per dimension, because the latter imposes less pressure in the X axis, spreading the utilization of network resources evenly among the three axes. Formalizing this, 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 5.

$$td(i,j) = dist + |dist_x - dist_y| + |dist_x - dist_z| + |dist_y - dist_z| \tag{5}$$

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

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

4 Experimental Set-up

In this section we detail the collection of experiments carried out to evaluate the effectiveness of the two approaches to the mapping problem described before. We use networks with 64 nodes: a 2D, 8x8 mesh and a 3D, 4x4x4 mesh. This way we can assess the influence of network topology on that effectiveness.

We use traces extracted from the NPB suite [9], a suite of small kernels and pseudo-applications which are derived from computational fluid dynamics (CFD)

applications. For each application we need to generate the traffic matrix W. We have done so for the class A, size 64 instances of the benchmarks. These applications were run in a real parallel machine, in which traces of all the messages interchanged by tasks were captured to fill W.

The QAP Solver 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) and the number of iterations to be performed. The output is a mapping vector that obeys the criterion represented in the cost matrix.

Simulations have been carried out using INSEE [14] [8]. This tool simulates the execution of a message-passing application on a multicomputer connected via an interconnection network. It 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. INSEE only measures the communication costs, assuming infinite-speed CPUs.

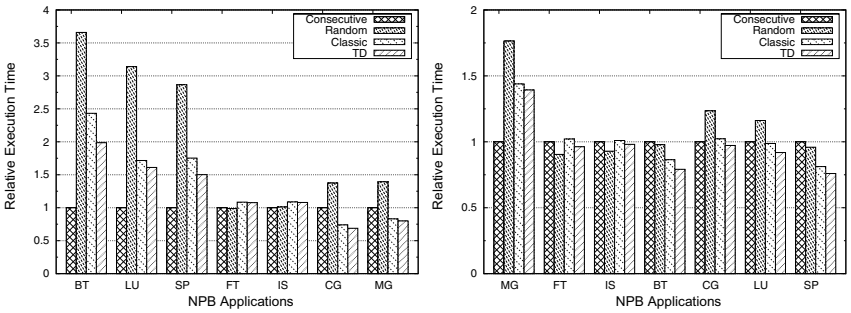
To perform the validation of mappings, we generated a set of 50 different vectors for each NPB application, using the classic and the TD expressions of the cost matrix. Each of the 50 tested vectors were selected after 50 GRASP iterations. In addition to the classic and the TD mapping strategies, we also evaluated the behaviour of the consecutive and random trivial mappings. Given a network, the consecutive mapping criterion allocates the application tasks onto the network nodes in order of identifiers, starting with task / node 0. Regarding the random criterion, application tasks are assigned to network nodes randomly; we used 50 different random assignments to evaluate this strategy.

5 Analysis of Results

We present the results of the experiments in three graphs. The first and the second allow us to assesses the quality of trivial vs. optimization-based mappings for the 64-node, 2D and 3D meshes. The third graph is designed to identify the topology that best matches each of the applications used in this study.

Figure 1 summarizes the results of experiments with 2D and 3D meshes. Due to limitations in space, we do not show average and standard deviation of the execution times as reported by the simulator for each application-topology-mapping combination. Instead, we discuss only average results, normalized to the consecutive case.

For the 2D mesh, it is clear that we can divide the applications into three groups. The first one contains applications BT, LU and SP. A property of these three applications is that the virtual topology matches the physical one; therefore, the consecutive mapping is the best performer. In a second group we include applications FT and IS; for these, optimization-based mappings are not better than the trivial ones – although the TD criterion provides almost the same results. These applications are implemented using non-optimized all-to-all communications, for which locality



(a) 2D Mesh (b) 3D Mesh

Fig. 1 Execution times (simulation cycles) for the NPB applications (class A, size 64) running on a 2D mesh (a) and on a 3D mesh (b) for each mapping criterion. Values are normalized to the consecutive case.

cannot be efficiently exploited. In the third we put CG and SP. Is for these applications when the optimization-based strategies in general, and the TD in particular, shows their potential, providing results clearly better than those of the trivial criteria.

Regarding the 3D mesh, there is a case of perfect virtual-to-physical match of topology: the MG application – therefore the consecutive mapping is the best one. For FT and IS we observe again that most mappings provide similar results, being the random one slightly better. In the remaining cases TD is the best performer.

In summary, the TD procedure provides good mappings of applications onto mesh topologies. 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). We have to remark that the TD criterion performs better than the classic for all the applications under test.

In the literature, it is generally assumed that the topological characteristics of 3D structures (that are better than those of same-size, 2D networks) should result in better execution times for applications. For this reason, partitions of 3D networks are generally done in terms of 3D sub-networks. The results of the experiments clearly state that applications perform best when there is a good virtual-to-physical matching of topologies. As not all applications have a 3D virtual topology, a mapping onto a 3D structure can be inadequate. In Figure 2 we have summarized the execution times of the NPB applications for 64-node, 2D and 3D meshes. As we can expect, for applications BT, LU and SP (2D-mesh virtual topology) a trivial, consecutive mapping onto a 2D network is faster than any mapping onto a 3D mesh. In contrast, for MG (3D-mesh virtual topology) the consecutive mapping onto the 3D network is clearly a better option. For the remaining applications, for which the structure of the communications does not match these topologies (or does not have any structure), the TD mapping onto a 3D partition provides excellent results.

We have to consider this in the context of a massively parallel computer managed by a topology-aware scheduler. Given an application, for which a system

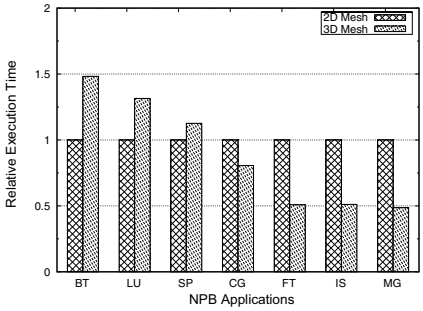


Fig. 2 Comparison of the execution time (simulation cycles) of the NPB applications in 2D and 3D meshes. Values are normalized to the 2D mesh case.

partition has to be assigned, it is desirable to locate a partition whose shape matches the most adequate one for that application; this can result in a search of a 2D submesh (a plane, or a sub-plane) inside a 3D torus. If this match is impossible (or the application's virtual topology is unknown), then a 3D partition along with an optimization-based mapping is a good option. The search of a partition for an application should be done taking into account not only the effects on the performance of the application, but also the scheduling costs: the procedure should minimize system fragmentation.

6 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 and assigning some resources to it (nodes grouped as partitions). 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 a new criterion for mesh topologies 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. We have focused on meshes because, even though large-scale supercomputers are built around 3D torus, partitions do not always enjoy wrap-around links.

We have shown the importance of searching for the correct topology for a particular application; this will result in an improvement of applications run times. And we have also stated that the partitioning mechanism should be included in the scheduling process, selecting the best sub-network for a given job, but trying to keep fragmentation under control. A good mapping algorithm, such as the optimization-based TD strategy, is useful to keep good performance levels. Our main line of future work is precisely the integration of partitioning and mapping mechanisms into schedulers for large-scale systems.

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

References

1. Agarwal, T., et al.: Topology-aware task mapping for reducing communication contention on large parallel machines. In: IEEE Intl. Parallel and Distributed Processing Symposium, Los Alamitos, CA, USA (2006)
2. Bhatele, A., Kalé, L.V.: An evaluation of the effect of interconnect topologies on message latencies in large supercomputers. In: Workshop on Large-Scale Parallel Processing (2009)
3. Bokhari, S.H.: On the mapping problem. IEEE Trans. on Computers 30(3), 207–214 (1981)
4. Dally, W., Towles, B.: Principles and practices of interconnection networks. Morgan Kaufmann Publishers Inc., San Francisco (2003)
5. Dongarra, J., Meuer, H., Strohmaier, E.: Top500 supercomputer sites, `http://www.top500.org`
6. Kang, M., et al.: Isomorphic strategy for processor allocation in k-ary n-cube systems. IEEE Trans. on Computers 52, 645–657 (2003)
7. Lo, V., et al.: Noncontiguous processor allocation algorithms for mesh-connected multicomputers. IEEE Trans. on Parallel and Distributed Systems 8, 712–726 (1997)
8. Miguel-Alonso, J., Navaridas, J., Ridruejo, F.J.: Interconnection network simulation using traces of MPI applications. Intl. Journal of Parallel Programming 37(2), 153–174 (2009)
9. NASA Advanced Supercomputer (NAS) division: NAS parallel benchmarks (2002), `http://www.nas.nasa.gov/Resources/Software/npb.html`
10. Navaridas, J., Pascual, J.A., Miguel-Alonso, J.: Effects of job and task placement on the performance of parallel scientific applications. In: 17th Euromicro Intl. Conf. on Parallel, Distributed, and Network-Based Processing, pp. 55–61 (2009)
11. Pardalos, P.M., Rendl, F., Wolkowicz, H.: The quadratic assignment problem: A survey and recent developments. In: DIMACS Workshop on Quadratic Assignment Problems, vol. 16, pp. 1–42 (1994)
12. Pascual, J.A., Miguel-Alonso, J., Lozano, J.A.: Optimization-based application framework for parallel applications. Tech. rep., The University of the Basque Country (2010)
13. Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133 (1995)
14. Ridruejo, F.J., Miguel-Alonso, J.: INSEE: An interconnection network simulation and evaluation environment. In: 11th Intl. Euro-Par conf. on Parallel Processing, pp. 1014–1023. Springer, Berlin (2005)

8

Optimization-based mapping framework for parallel applications

J. Parallel Distrib. Comput. 71 (2011) 1377–1387

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Optimization-based mapping framework for parallel applications

Jose A. Pascual *, Jose Miguel-Alonso, Jose A. Lozano

Intelligent Systems Group, School of Computer Science, The University of the Basque Country UPV/EHU, P. Manuel Lardizabal 1, San Sebastian, 20018, Spain

ARTICLE INFO

Article history:
Received 8 July 2010
Received in revised form
24 November 2010
Accepted 16 June 2011
Available online 5 July 2011

Keywords:
Optimization-based mapping of parallel applications
Mapping as an instance of the Quadratic Assignment Problem
Scheduling for supercomputers
Mappings on 2D and 3D cubes

ABSTRACT

The 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 matrix is usually a distance matrix (the classic approach), but we propose a novel definition of the cost criterion, applicable to torus networks, that tries to distribute traffic evenly over the different axes; we call this the Traffic Distribution criterion. As the mapping problem can be seen as a particular instance of the Quadratic Assignment Problem (QAP), we can apply any QAP solver to this problem. In particular, we use a greedy randomized algorithm. Using simulation, we test the performance levels of the optimization-based mappings, and compare them with those of trivial mappings (consecutive, random), in two different environments: single application (one application uses all system resources all the time) and space sharing (several applications run simultaneously, on different system partitions), using systems with 2D and 3D topologies and real application traffic. Experimental results show that some applications do not benefit from optimization-based mappings: those in which there is a match between virtual and physical topologies, and those that carry out massive all-to-all communications. In other cases, optimization-based mappings with the TD criterion provide excellent performance levels.

1. Introduction

Parallel applications are composed of a collection of tasks that exchange information and synchronize among them using different communication patterns. These applications are often designed and developed with a parallel communication 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. 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's *physical topology* (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 [2].

In most cases, a supercomputing facility is shared by many users that run different applications. This means that the system can be running several parallel jobs simultaneously, normally using a space-sharing approach: each job uses a different system partition. A system scheduler is in charge of allocating resources to jobs, applying different policies to select the partition (collection of nodes) on which a given job 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 application performance due to the efficient exploitation of communication locality [38,42]. 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

* Corresponding author.
E-mail addresses: joseantonio.pascual@ehu.es (J.A. Pascual), j.miguel@ehu.es (J. Miguel-Alonso), ja.lozano@ehu.es (J.A. Lozano).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.06.005

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.

In this paper we propose an optimization-based framework to deal with the mapping problem. Although the use of optimization-based mapping approaches is not new, our framework is more general and allows us to easily select different criteria when searching for good mappings. These criteria need to be captured into a cost matrix that finally translates into a function to optimize. In addition to the commonly used, distance (*Classic*) criterion, that tries to minimize the average distance traversed by messages [40,49], 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 can create network contention due to the locality of communications, resulting in a degradation of application 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 [12] can be seen as a specific instance of the Quadratic Assignment Problem (QAP, see Section 4.1). Consequently, we can tackle it using any method developed to solve the QAP. This problem has been proved to be NP-hard [21] and, therefore, we use heuristic approaches to find good solutions for it. In particular, we have selected the Greedy Randomized Adaptive Search Procedure (GRASP) [19,45]. This algorithm, once the criterion has been selected, 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 behaviour at run-time. Therefore we have to *validate* the mappings, proving that in most cases 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 [36] 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) [35]. For the interconnection networks, we have restricted our study to 2D and 3D tori. We have carried out two different sets of experiments, to evaluate mappings for single-application and space-sharing environments. For simplicity, in the *single-application environment* we only consider configurations in which the number of application tasks 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 obtained with different mappings depend on the applications' communication pattern, but the most notable result is that a consecutive mapping excels when virtual and physical topologies match; otherwise, optimization-based mappings are good options. For applications based on all-to-all collective communications we found that random mappings are adequate. We will discuss these results later.

To test our techniques on *space-sharing environments*, we have measured the performance of several instances of the same application sharing a system. Regarding the allocation of system partitions to jobs, we compared random partitions with simple forms of consecutive partitions, using bands or squares (for 2D) and planes and sub-cubes (for 3D). For each partitioning schema, we have tested the same allocation strategies used in the single-application scenario: trivial mappings (consecutive, random) and mappings obtained using optimization (classic and TD). Again, we verify that a mismatch between virtual and physical topology results in a bad behaviour of the consecutive mapping, revealing the great potential of mappings based on the TD criterion.

The rest of the paper is organized as follows. Section 2 makes a review of work related to the mapping problem and its incorporation into schedulers. Section 3 is devoted to the definition of the Quadratic Assignment Problem, presenting some approaches to solve it. In Section 4 we define formally the mapping problem and its expression as a particular instance of the QAP, and present the particular algorithm chosen to obtain solutions. In Section 5 we detail the criteria used to obtain mappings with some specific characteristics: the classic one and the proposed TD. Section 6 presents the experimental workbench, following in Sections 7 and 8 with the results of the experiments carried out to validate different mapping criteria on both single-application and space-sharing environments. Finally, the paper ends in Section 9 with some conclusions and future lines of work.

2. Related work

In this section we carry out a brief review of the literature about the mapping problem, with special focus on those works in the field of high-performance computing.

Due to the complexity of the problem, heuristic algorithms are often the choice to find solutions. Different optimization-based approaches have been developed, in which some system characteristics and performance metrics are modelled. Some authors use Tasks Interaction Graphs and Task Precedence Graphs as ways to represent parallel programs. While capturing the main relationships between tasks, they fail to capture the temporal behaviour of programs (different phases with different communication needs) — a limitation that also applies to the framework proposed in this paper. Due to this, in [43], the authors developed a variation of Lo's Temporal Communication Graphs [26] that also models temporal behaviour. Their goal is to minimize, using a greedy algorithm, an evaluation function that represents the resource contention on a k-ary n-cube network.

In [30], the authors used genetic algorithms targeting the minimization of message latencies. This kind of algorithms have been combined with local search strategies in [16], creating a hybrid algorithm. Simulated annealing approaches have been also used [13], expressing the problem as a minimization of both the communication delays and the contention for network resources. Bokhari stated the problem as a minimization of the number of task graphs that fall onto the processor graph [12]. He used a local search technique combined with non-deterministic jumps into intermediate solutions to avoid local optima.

In the context of massively parallel computers, in [38,42], 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 [7,8], 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 [25,34] or on the reduction of power consumption [22,28].

Ideally, in large-scale systems managed by a scheduler, topology and application-aware allocation and mapping policies should be taken into consideration, in order to increase the overall system 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, communication is inefficient due to inter- and intra-application interferences (as studied in [42]).

To our knowledge, only the Blue Gene family of supercomputers provides a way to assign to a job a file containing a mapping for its tasks [5]. To illustrate the importance of this issue, authors of [8] perform a study with three applications in two supercomputers arranged as 3D tori: a Blue Gene/L [5] and a Cray XT3 [4]. 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 [9] the authors evaluate the effect that contention has on packets' latencies, concluding that developers should take into account the network topology when mapping tasks to nodes.

3. The quadratic assignment problem

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 Quadratic Assignment Problem (QAP) [41]. This is one of the fundamental combinatorial optimization problems in the branch of optimization in mathematics. 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 weights.

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:

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

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 [29], and branch and bound techniques [44]. 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 [27].
- Heuristic methods: These include improvement methods such as local search and tabu search [33], simulated annealing [24] and genetic algorithms [20]. An heuristic widely used to solve the QAP is the Greedy Randomized Adaptive Search Procedures (GRASP) [46]. 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 4.2 for a description of this method.

[1] As a notable exception, SLURM [48] includes topology-aware partitioning for Blue Gene, Cray XT, Sun Constellation and tree-based hierarchical networks.

4. A framework to solve the mapping problem

In this section we explain in more detail our proposal of an optimization-based framework to find application-to-partition mappings. First we formalize the definition of the mapping problem, and then we describe the use of GRASP to generate the mappings. The framework we define is valid for both single-application and space-sharing environments. In the first case, it is assumed that the application runs on a target machine 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 the space-sharing environment several applications compete for the resources of a machine, each of then running on the partition assigned by a scheduler. The solution still has to be good, but there are time restrictions: it has to be obtained rapidly, to keep scheduling overheads low. For the sake of homogeneity in the process of analysing the performance of our proposals, in the experiments we have set the same time restrictions (solver iterations) for both environments.

4.1. The mapping problem

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

$$\pi : T \longrightarrow P$$
$$t \longrightarrow \pi(t) = p$$

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:

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

This formulation 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 volume of information interchanged by the tasks, and the cost matrix C which expresses the criterion selected to distribute the traffic within 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. Given the communication traces of a particular parallel application, the number of messages interchanged between task pairs, and the lengths (in bytes) of those messages are used to compute W. The procedure to compute C depends on the criterion selected to create mappings, represented as a function to be optimized.

4.2. Mapping algorithm

In our experiments we have used the GRASP solver, because we found in the literature that this is the algorithm that provides the best known results when solving the generic QAP [14]. 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 α% of those with the 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 first, 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 improvement 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.

5. 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 Eq. (1). Often the mapping problem has been seen as a location problem [12], 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. Intuitively, 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 [2,10]. 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. In both cases we are using the cost unit described before: the interchange of one data byte through a single node-to-node link. Therefore, the cost of sending a message depends on its length and on the distance from source to destination.

The definition of both criteria, that materialize in cost matrices, depends on topological characteristics of the target network in which applications will run. In this work we consider two network topologies: 2D and 3D tori of $N = n_x \times n_y (\times n_z)$ nodes, as represented in Fig. 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 destination. The simplest form of routing is Dimension Order Routing (DOR) [18], in which messages traverse first all the necessary hops in the X axis, then switches to the Y axis (and then to the Z axis in the 3D case); however, adaptive routing through minimal paths improves system performance, and it is the one used in the experiments. We are assuming that routing decisions are made at each router through which a message passes. A router receiving a message selects the next node to forward it, making its decision considering the status of the links, trying to avoid resource contention. This kind of routing is used in state-of-the-art supercomputers like IBM Blue Gene [1,3] and will be implemented in the next generation of routers for Cray's supercomputers [17,50].

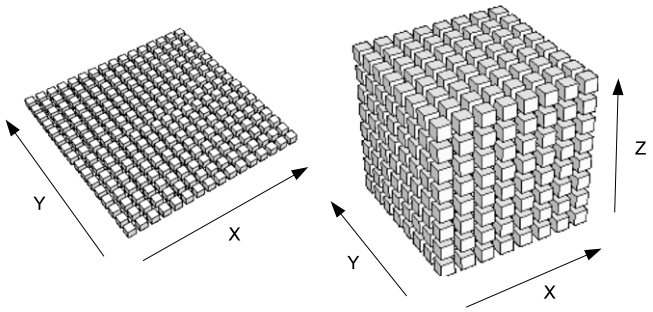


Fig. 1. Examples of the topologies used in the experiments: 2D torus of (8×8) nodes and 3D torus of $(4 \times 4 \times 4)$ nodes.

5.1. Classic distance criterion

In a 3D torus, given two nodes with identifiers i and j, the distance between them is given by Eq. (3).

$$d(i,j) = d_x(i,j) + d_y(i,j) + d_z(i,j) \tag{3}$$

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, the second term is the number of hops through the Y axis, and the third term is for the Z axis. Eq. (4) shows terms d_x and d_y for 2D tori and Eq. (5) shows terms d_x, d_y and d_z for 3D tori.

$$\begin{aligned} d'_x(i,j) &= |\lfloor j/n_x \rfloor - \lfloor i/n_x \rfloor|; \\ d_x(i,j) &= min(d'_x, n_x - d'_x) \\ d'_y(i,j) &= |j \bmod n_y - i \bmod n_y|; \\ d_y(i,j) &= min(d'_y, n_y - d'_y) \end{aligned} \tag{4}$$

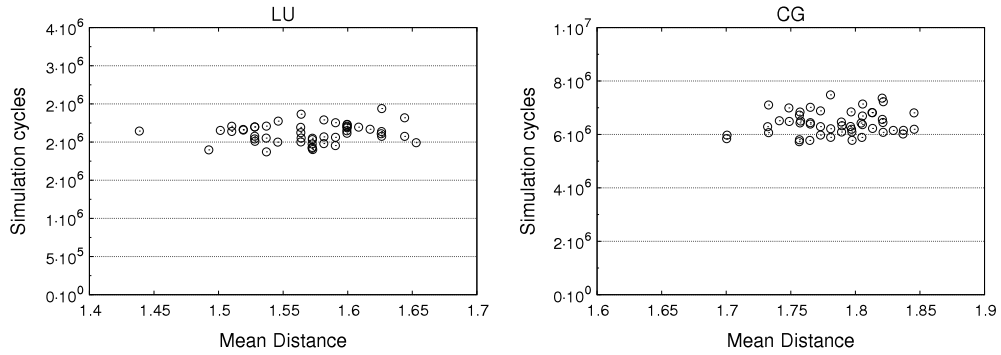
$$\begin{aligned} d'_x(i,j) &= |(j \bmod (n_x \times n_y)) \bmod n_x \\ &\quad -(i \bmod (n_x \times n_y)) \bmod n_x|; \\ d_x(i,j) &= min(d'_x, n_x - d'_x) \\ d'_y(i,j) &= |(j \bmod (n_x \times n_y))/n_y - (i \bmod (n_x \times n_y))/n_y|; \\ d_y(i,j) &= min(d'_y, n_y - d'_y) \\ d'_z(i,j) &= |j/(n_x \times n_y) - i/(n_x \times n_y)|; \\ d_z(i,j) &= min(d'_z, n_z - d'_z). \end{aligned} \tag{5}$$

Given these expressions, the classic criterion to tackle the mapping problem identifies C as the distance matrix, whose components can be filled using Eq. (3) for each pair of source–destination nodes.

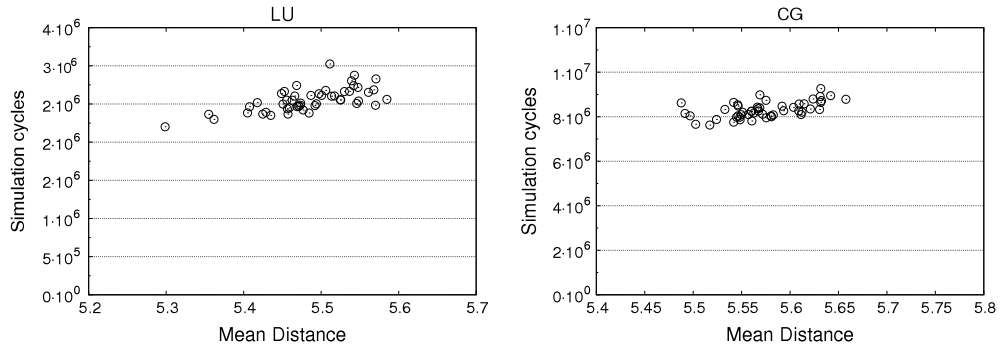
$$C = [c_{i,j}]_{i,j=1\ldots N} \quad \text{where } c_{i,j} = d(i,j). \tag{6}$$

5.2. Traffic distribution criterion

The rationale behind the 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 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 (Fig. 2(a)) and on a $4 \times 4 \times 4$ torus (Fig. 2(b)), and measured the mean distance traversed by packets. We have also measured the execution time of



(a) 2D Torus (8 × 8).



(b) 3D Torus (4 × 4 × 4).

Fig. 2. Relationship between the simulation time (in cycles) and the communication mean distance in (a) 8 × 8 torus for NPB applications LU and CG, class A, size 64. (b) 4 × 4 × 4 torus for NPB applications LU and CG, class A, size 64.

the applications as reported by INSEE. All this data has been plotted in Fig. 2, using as ordering criterion the average distance of message interchanges for the different mappings. We would expect a clear relationship between distance and execution time, but the figure does not show it — at least, for the small range of distances considered in these experiments. Although there is evidence that longer routes result in longer execution times [11], distance is a coarse-grain indicator of expected performance, but not a fine-grain one. Mapping criteria based only on distance ignore a fundamental characteristic of interconnection networks: resources (routers, links) have to be shared between many simultaneous communications, from the same or from different applications. 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 causes latency increases.

We propose in this paper a new criterion, that allows us to find mappings that, while not optimal in terms of distance, helps avoiding network bottlenecks by taking advantage of adaptive routing (through minimal paths). In particular, we pay attention to the number of hops per dimension that messages have to travel. A 8-hop route in which all the hops are in the X axis is worse than another 8-hop route with 4 hops per dimension, because the first fixes a single path using a single axis (dynamic adaptation through minimal paths is impossible) while the latter imposes less pressure in the X axis, spreading the utilization of network resources evenly among all axes and providing more opportunities for adaptation in case of detecting resource contention. 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 Eqs. (7) (2D) and (8) (3D).

$$\cos t(i,j)_{2D} = d(i,j) + |d_x(i,j) - d_y(i,j)| \tag{7}$$

$$\begin{aligned}\cos t(i,j)_{3D} \;=\; & d(i,j) + |d(i,j)_x - d(i,j)_y| \\ & + |d(i,j)_x - d(i,j)_z| + |d(i,j)_y - d(i,j)_z|.\end{aligned} \tag{8}$$

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

$$C = [c_{i,j}]_{i,j=1\ldots N} \quad \text{where } c_{i,j} = \cos t(i,j). \tag{9}$$

It should be clear that, while the classic criterion favours optimal placements in terms of distance, the TD criterion partially sacrifices this optimality to take advantage of adaptive routing, with the hope that spreading evenly the traffic through the different network axes will result in a reduction of the time required to deliver them. The cost to pay is that routes followed by messages *could* be longer. The dynamic behaviour of the application will decide whether or not the choice is worthwhile. Experiments discussed in the next sections show that, in fact, it is.

6. Experimental set-up

In this section we detail the collection of experiments carried out to evaluate the effectiveness of the TD approach to the mapping problem described in the previous section. We will use networks of three different sizes for the 2D torus topology: 16 (4 × 4), 64 (8 × 8) and 256 (16 × 16) nodes, and networks of two different sizes for the 3D torus topology: 64 (4 × 4 × 4) and 512 (8 × 8 × 8) nodes, in order to test the impact of network size on that effectiveness. Now we describe the workbench used for the experimentation.

6.1. Experimental workbench

The workbench comprises three key pieces:

1. **NPB traces:** 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 [35], 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 (including the point-to-point messages used to implement collective communications). 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 modelling 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** [36]: 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) [36]. 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. This tool has been widely used in the evaluation of massive parallel systems [15,37,39,31].

Regarding the applications, and to better understand the results of the experiments, we will classify them in three groups considering the virtual topology used to carry out inter-task communications.

- Type I (BT, LU, SP). In these three applications tasks communicate with neighbours using a 2D virtual topology [6].
- Type II (CG, MG). In these applications tasks also communicate with neighbours, but using a 3D virtual topology [6,35].
- Type III (FT, IS). These applications use all-to-all communications, without an specific virtual topology in mind. Traces reflect a non-optimized implementation of all-to-all primitives, using point-to-point messages [47].

For the first two application types we should expect excellent results for those partitioning/mapping combinations that provide perfect matches between physical and logical topologies. It is for the remaining scenarios, *where this match does not exist*, for which we expect good performance from the optimization-based mapping strategies.

6.2. Design of the experiments

To perform the validation tests in single-application environments, we created a set of 50 different mappings for each NPB application, using the classic and the TD expressions of the cost matrices. For this environment, we tested two application/network sizes in 2D: 16 and 64 tasks/nodes, and one application/network size in 3D: 64 tasks/nodes. Note that we only consider one-to-one assignments. Each of the tested mappings were selected after 50 GRASP iterations with the value of parameter α set to 20%. In this environment, INSEE [36,32] is used to estimate the time required to complete the application — actually, to deliver all the messages as required by the application.

For the space-sharing environment the set-up was more complex because, in addition to focus on mappings, we had to include some partitioning strategy. We used traces of the NPB applications, size 64, class A. For the 2D experiments, the network was a torus of 16×16 nodes, housing four application instances. We tested three simple partitioning schemas, as shown in Fig. 3(a)–(c): quadrant, band and random. For 3D, the network (a torus of $8 \times 8 \times 8$ nodes) was capable of housing eight application instances. Regarding partitioning, we used extensions of the schemas tested for 2D: sub-cubes, planes and random, as shown in Fig. 3(d)–(f). When using random partitions, experiments were repeated 20 times. The different mapping alternatives were tested for each partitioning schema. In this environment, INSEE is used to estimate the time required by multiple instances of the same application to interchange all the messages. These instances have to compete for shared resources, although for some partitioning schemas the level of inter-application interference is higher than for others. A good mapping should minimize intra-application contention, and help alleviating inter-application interference. As we used adaptive routing, that has a random component (to choose between different, valid routes), each experiment was repeated five times, using different seeds.

Note that, for 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 partition (which could be whole system), application tasks are allocated onto the partition's nodes in the 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: Given a network partition of a larger system, application tasks are assigned to partition nodes randomly. To evaluate this strategy, a set of 50 different random assignments to each partition were performed. Note that the consecutive mapping on a random partition is just a particular case of random mapping. In space-sharing environments, we tested 50 different mapping instances of each non-consecutive criterion (classic, TD and random) over 20 randomly generated partitions, performing a total of 20×50 experiments per mapping criterion.

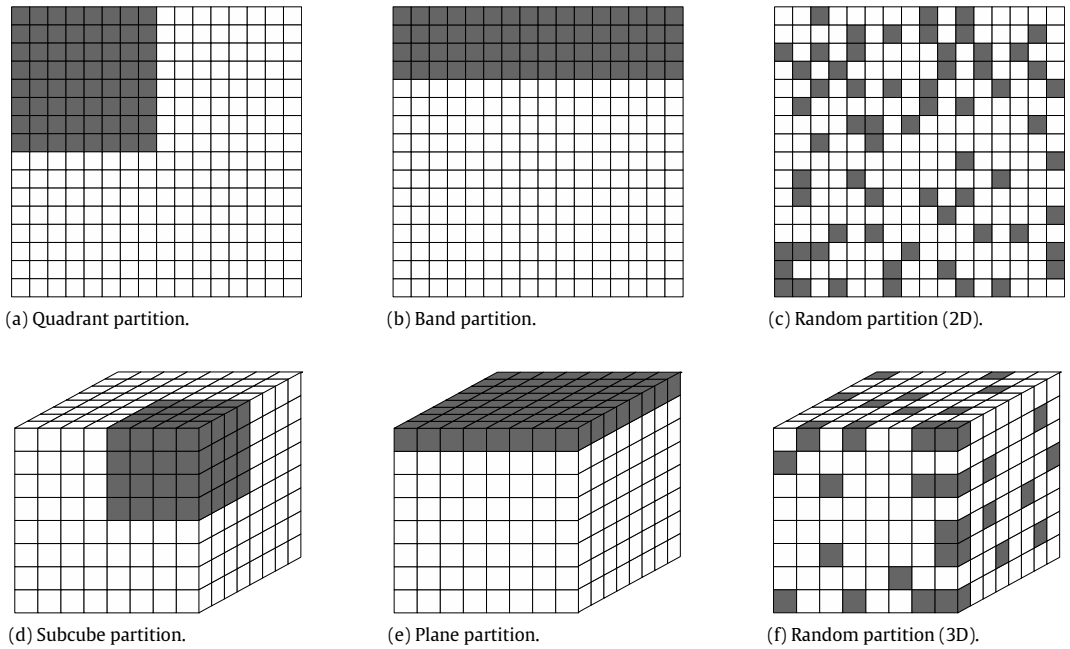
7. Results for single-application environments

The results of the experiments in single-application environments are summarized in Table 1(a) for the small network (16-node 2D torus), and in Tables 1(b) and 2 for the large networks (64-node 2D torus and 64-node 3D torus). Tables show best, average and standard deviation of the execution times reported by the simulator for each application-topology-mapping combination. The statistical significance of the differences between the mapping strategies has been assessed running statistical tests with one level of significance: 0.05. When the winner is the consecutive mapping we perform a t-test. In the remaining cases, we perform a Kruskal–Wallis [23] paired test. Each mapping strategy is tested against the best.

7.1. Analysis of results for 2D topologies

Results for the small network (Table 1(a)) only tell us that random mappings are not a good choice for any application. TD mappings are the best performers in most cases and, when surpassed by other criteria, the differences are not significant.

Results for the large network (Table 1(b)) deserve more comments. Now we see how the benefits of the different mapping alternatives depend strongly on the application type. For Type I applications (BT, LU, SP) the consecutive mapping yields the perfect match between virtual and physical topologies, therefore achieving the best results; TD is the second best performer, but



(a) Quadrant partition. (b) Band partition. (c) Random partition (2D).

(d) Subcube partition. (e) Plane partition. (f) Random partition (3D).

Fig. 3. Network partitions used to test different mappings strategies in space-sharing environments: (a, b, c) 16 × 16 torus. (d, e, f) 8 × 8 × 8 torus.

Table 1
Execution times (10^4 simulation cycles) on 2D torus topologies. Statistical tests were performed against the best strategy.

	Consecutive	Random			Classic			TD		
	Value	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev
(a) NPB applications (class W, size 16) running on a 4 × 4 torus										
BT	**93.78**	111.38	121.34*	2.94	93.78	103.69*	6.88	93.77	93.78	0.00
LU	**74.78**	77.92	85.29*	2.71	74.76	74.78	0.02	74.77	74.78	0.03
SP	**161.11**	181.30	194.67*	4.28	161.11	175.58*	10.42	161.11	161.46	0.28
CG	348.65	346.13	386.16*	17.97	302.80	307.66	7.86	302.80	**302.86**	0.08
MG	50.48	52.95	57.98*	1.66	45.52	45.58	0.07	45.52	**45.56**	0.06
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
(b) NPB applications (class A, size 64) running on a 8 × 8 torus										
BT	**369.35**	728.22	764.38*	16.34	576.15	648.51*	29.76	569.67	623.23*	28.95
LU	**136.57**	231.38	258.05*	9.01	187.20	211.53*	10.11	181.01	201.45*	9.19
SP	**598.97**	1095.80	1156.00*	22.61	882.47	953.65*	31.69	823.69	907.23*	33.98
CG	719.35	715.50	789.10*	25.76	572.84	642.17*	35.52	533.42	**614.87**	29.56
MG	468.47	449.96	477.92*	10.04	378.73	399.74	9.03	377.86	**397.02**	8.61
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

* Represents that significant differences do exist between the marked strategy and the best one.

provide significantly worse results. For Type II applications (CG, MG) we do not have this good match, and TD shows its potential, being significantly better than the remaining alternatives. Finally, for Type III applications (FT, IS) the clear winner is the random mapping. In these cases all tasks interchange the same amount of information with the remaining tasks, therefore any effort of finding optimal routes is useless because all of them are equally good — or bad. The dynamic behaviour of these interchanges (carried out in order of identifier) gives advantage to random mappings, that helps using diverse routes, therefore avoiding contention.

7.2. Analysis of results for 3D topologies

Results for the 3D torus are summarized in Table 2. We could state that the consecutive mapping is the best performer for Type I and Type II applications, while random is the best for the Type III group. However, given the small size of the network, and its topological characteristics (very small average distance) the differences between the winner and the second best are not very large in most cases. In general, we can conclude that at this network size, the diameter and the average hops messages travel is short and, hence, we do not expect a significant improvement from any particular mapping.

7.3. Conclusions for single-application environments

The novel TD criterion for optimization-based mappings offer good results for scenarios with relatively large average distance, because it favours reduced-distance paths while avoiding contention and providing opportunities for in-route adaptation. However, for small networks, and scenarios with good matching between virtual and physical topologies, trivial mappings are good performers and optimization is not worthwhile. Note, though, that in massively parallel supercomputers applications run in system partitions, instead of in the whole machine, and a perfect

Table 2
Execution times (10^4 simulation cycles) for the NPB applications (class A, size 64) running on a 4 × 4 × 4 torus. Statistical tests were performed against the best strategy.

	Consecutive	Random			Classic			TD		
	Value	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev
BT	**432.14**	465.76	482.61*	10.45	452.61	485.08*	13.33	457.95	482.69*	12.26
LU	**160.85**	162.41	174.05*	7.36	150.85	162.46*	5.36	156.35	168.57	6.75
SP	**692.51**	718.72	747.78*	16.56	710.93	738.03*	16.28	703.33	734.75*	14.48
CG	**462.82**	497.76	525.27*	16.89	453.61	497.84*	25.62	468.03	515.28*	25.11
MG	**249.83**	301.64	313.12*	7.98	286.89	299.91*	7.99	284.37	303.68*	8.87
FT	516.23	506.61	**516.69**	5.09	523.50	540.30*	8.81	508.56	522.50*	7.66
IS	183.23	165.84	**168.95**	1.48	171.73	176.55*	2.04	167.25	170.97	1.75

* Represents that significant differences do exist between the marked strategy and the best one.

Table 3
Execution times (10^4 simulation cycles) for the NPB applications (class A, size 64) running on a 16 × 16 torus, with different partitioning schemas. Statistical tests were performed against the best strategy.

	Consecutive	Random			Classic			TD		
	Value	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev
(a) Quadrant partitioning schema										
BT	**370.49**	1360.97	1407.31*	30.32	899.25	944.70*	28.26	743.27	783.83*	22.62
LU	**136.57**	425.84	452.63*	22.56	239.58	260.24*	20.77	209.89	238.53*	23.48
SP	**727.02**	2119.00	2172.27*	44.08	1269.63	1312.62*	33.72	1095.93	1125.03*	24.77
CG	1005.04	1379.65	1462.25*	59.99	733.99	800.39*	51.93	699.76	**725.55**	22.92
MG	642.38	906.77	938.32*	26.69	533.67	556.39*	12.68	531.48	**545.71**	15.68
FT	1530.88	1519.11	**1528.00**	8.23	1679.78	1718.45*	24.15	1566.52	1611.30*	25.81
IS	**499.70**	506.09	510.66*	3.37	528.59	549.47*	12.78	533.35	545.71*	9.97

	Consecutive			Random			Classic			TD		
	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev
(b) Band partitioning schema												
BT	1620.36	1626.31*	5.44	1237.91	1289.43*	25.45	807.13	966.90*	93.72	770.88	**813.00**	22.42
LU	392.29	394.00*	1.49	397.46	415.19*	13.44	245.25	279.35*	18.28	223.93	**253.40**	18.47
SP	2434.14	2445.02*	7.10	1919.18	1969.22*	39.84	1307.53	1347.37*	43.01	1088.57	**1168.33**	68.45
CG	1029.19	1030.61*	1.60	1335.35	1416.12*	64.24	769.11	888.75*	73.96	707.82	**740.69**	41.68
MG	515.75	**518.06**	2.22	778.58	809.19*	22.98	550.74	565.93*	17.38	511.43	528.35*	12.85
FT	1922.13	1928.43*	3.96	1322.88	**1342.65**	12.08	1482.46	1537.83*	39.50	1509.00	1564.06*	40.39
IS	510.77	514.33*	2.35	422.81	**425.98**	1.52	481.69	520.89*	19.96	492.10	528.00*	23.77
(c) Random partitioning schema												
BT	1125.12	1177.79*	26.90	1397.60	1451.59*	27.52	879.73	948.53*	27.71	835.38	**875.09**	18.70
LU	383.72	394.75*	8.20	429.36	465.37*	12.07	294.68	310.64*	8.52	262.91	**281.51**	7.88
SP	1726.38	1779.35*	32.68	2094.88	2216.95*	40.74	1312.90	1397.11*	40.24	1251.14	**1306.42**	26.64
CG	1313.09	1360.12*	47.35	1377.19	1433.48*	32.12	883.08	926.47*	20.47	817.95	**866.90**	21.05
MG	778.88	789.98*	9.74	940.14	983.85*	17.43	632.69	668.53*	11.43	640.53	**664.21**	8.82
FT	2352.60	2499.23*	91.30	1568.50	**1590.88**	9.00	1670.15	1734.77*	25.93	1680.27	1728.22*	22.09
IS	763.35	800.55*	24.40	517.47	**524.08**	3.09	563.10	587.56*	10.12	560.05	578.87*	10.12

* Represents that significant differences do exist between the marked strategy and the best one.

match cannot be expected to happen. We explore results for these scenarios in the forthcoming section.

8. Results for space-sharing environments

The results of the experiments carried out in 2D space-sharing environments are summarized in Table 3(a) for quadrant partitions, in Table 3(b) for band partitions and in Table 3(c) for the randomly generated partitions. For the 3D topology the results are shown in Table 4(a) for the plane partitions, in Table 4(b) for the sub-cubes and in Table 4(c) for the randomly generated partitions. Again, tables show the best, the average and the standard deviation of the execution times reported by the simulator for each application-topology-mapping combination. In this case the statistical significance of the differences between the mapping strategies has been assessed running Kruskal–Wallis [23] paired test when the best performer has a random component. There are two deterministic cases: the consecutive mapping strategy over quadrant and subcube partitioning schemas. When these strategies are the best performers, we use a t-test to assess the statistical significance of the difference with the others. As in the single-application environment, both tests have been performed with one level of significance: 0.05.

8.1. Analysis of results for 2D topologies

As explained above, experiments were performed running four 64-task instances of the same application on partitions of a 256-node 2D torus, see Fig. 3 to understand the partitioning schemas.

We start analysing the results obtained arranging applications onto quadrants: a 8 × 8 square partition of a 16 × 16 torus is assigned to each application instance; the four instance compete for resources. Intra-application interference does not appear, because all inter-task communications are confined within the partition. For Type I applications, the consecutive mapping is the best one, with significant differences with the remaining methods. This was expected, because of the perfect match between topologies. The second best mapping is TD. For Type II applications, the best mapping is TD: the perfect match does not occur, and TD does a good job placing tasks. For Type III applications we observe

Table 4
Execution times (10^4 simulation cycles) for the NPB applications (class A, size 64) running on a 8 × 8 × 8 torus with different partitioning schemas. Statistical tests were performed against the best strategy.

	Consecutive	Random			Classic			TD		
	Value	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev
(a) Subcube partitioning schema										
BT	694.21	699.70	720.17*	17.14	616.86	639.85*	14.80	556.31	**585.23**	17.92
LU	**195.51**	234.07	249.00*	10.82	197.41	211.14*	12.77	184.73	196.83	7.60
SP	1077.76	1068.37	1104.96*	38.88	904.44	940.05*	27.77	834.75	**870.72**	26.49
CG	**573.67**	739.57	761.60*	15.23	598.22	645.54*	38.92	562.32	618.52*	29.45
MG	**249.97**	448.83	463.39*	9.45	364.86	376.59*	7.54	352.00	361.01*	9.22
FT	798.23	723.59	**730.99**	7.32	841.32	857.50*	16.81	768.25	791.20*	13.32
IS	274.89	257.13	**261.81**	2.62	274.50	286.43*	9.91	270.02	282.06*	6.51

	Consecutive			Random			Classic			TD		
	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev	Best	Average	Std dev
(b) Plane partitioning schema												
BT	369.34	**369.35**	0.00	791.45	804.52*	8.57	689.17	718.28*	27.84	637.36	672.85*	22.94
LU	136.57	**136.57**	0.00	258.16	274.78*	10.12	223.87	235.04*	6.85	201.27	216.02*	6.87
SP	598.97	**598.97**	0.01	1177.89	1208.26*	20.71	992.81	1031.35*	32.49	939.91	967.91*	16.25
CG	719.58	719.68*	0.08	813.00	840.06*	21.74	677.77	708.00*	26.05	641.53	**670.66**	33.20
MG	468.11	469.53*	1.73	480.71	492.89*	8.42	409.56	417.12*	6.52	400.73	**408.92**	8.69
FT	1254.53	1258.30*	4.51	808.72	**818.23**	8.44	916.34	943.58*	13.06	919.59	936.82*	12.66
IS	404.79	406.56*	2.38	256.67	**260.15**	2.83	290.52	303.91*	10.42	302.36	308.24*	6.35
(c) Random partitioning schema												
BT	636.03	648.32*	10.87	645.37	664.53*	9.24	599.88	632.88*	14.05	570.14	**595.68**	12.96
LU	233.40	240.66*	5.00	219.59	235.63*	6.37	210.78	222.26*	5.42	197.10	**206.83**	5.98
SP	958.79	975.79*	15.41	987.79	1015.46*	12.42	897.71	939.01*	16.03	867.60	**897.51**	21.93
CG	719.11	745.01*	12.56	668.10	699.30*	14.78	624.78	645.75*	16.87	587.10	**615.49**	14.25
MG	430.16	441.65*	5.08	431.92	447.29*	7.37	388.98	405.65*	5.90	383.58	**397.85**	6.65
FT	1059.12	1115.00*	25.45	681.93	**687.59**	2.85	712.39	728.55*	6.65	703.50	717.51*	5.88
IS	353.52	382.57*	12.65	226.08	**229.06**	1.26	245.32	250.92*	2.96	241.11	246.82*	2.12

* Represents that significant differences do exist between the marked strategy and the best one.

the same behaviour seen for the single-application case: random mappings are the best ones, for the same reasons explained before.

Regarding the band partitioning schema, we break the perfect topology match for Type I applications, and this results in TD showing its good behaviour: now, it is the best performer. It is also good (although not always the best) for Type II applications. For Type III there is nothing new to say: random mappings win.

Finally, let us analyse results achieved using randomly generated partitions. In this case, none of the applications benefit from a match between virtual and physical topologies. As happened with the band partitions, TD performs the best for applications of Type I and II. Random partitioning is still the best for Type III, although results provided by TD are much better than those provided by the consecutive mapping.

8.2. Analysis of results for 3D topologies

In this case, eight 64-task jobs share a 512-node 3D torus. Partition schemes are extensions of those used in the 2D case: sub-cubes, planes and random (Fig. 3).

We start analysing the plane partitioning schema. Note that for the selected network size, 512-node (8 × 8 × 8), the partition of the network in 64-node planes (8 × 8) results in the partition of the network in eight 2D torus, each one dedicated to an application instance, and without interferences between them. Therefore, results are not surprising, corresponding to those of the single-application scenario for the large 2D network: consecutive mapping is the winner for Type I applications, TD for Type II, and random for Type III.

In the sub-cubes case, we find now that consecutive mapping is the best one for Type II applications — those with a 3D grid virtual topology. This is something we could expect, because again we find a perfect virtual-to-physical topology match. This does not happen

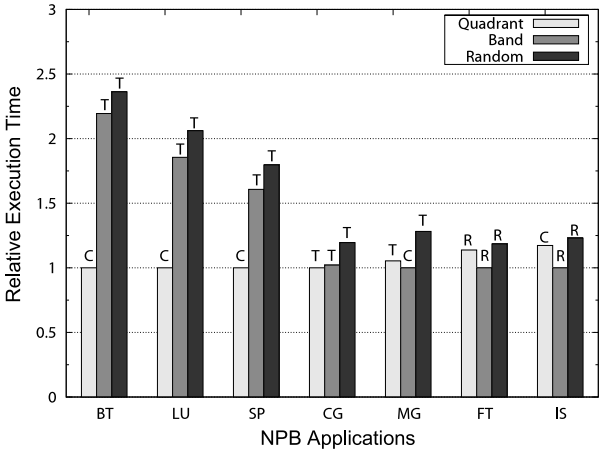


Fig. 4. Execution times of the best mapping criterion for each NPB application on each partition type in a 2D torus (16 × 16), relative to the best partitioning/mapping combination. The label on top of each bar indicates the best mapping: (C)onsecutive, (T)raffic Distribution and (R)andom.

for Type I, and here TD is the best performer. For FT and IS results are as expected: the random mapping is still the best performer.

Finally, we analyse the results for the random partitioning schema. In this case the best mapping strategy is TD: it provides the best results for application Types I and II. For Type III the random mapping is still the best, but results of TD are not very far.

8.3. Conclusions for space-sharing environments

In Figs. 4 and 5 we summarize the main results for the space-sharing experiments. From these figures we can conclude that application performance is greatly affected by network

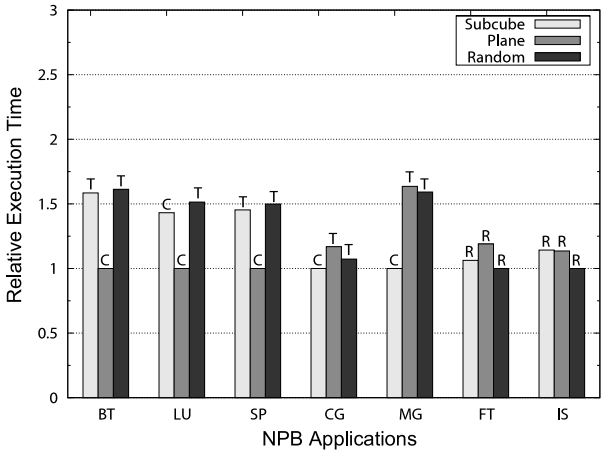


Fig. 5. Execution times of the best mapping criterion for each NPB application on each partition type in a 3D torus ($8 \times 8 \times 8$), relative to the best partitioning/mapping combination. The label on top of each bar indicates the best mapping: (C)onsecutive, (T)raffic Distribution and (R)andom.

topology, partitioning schema, and mapping schema. When using a supercomputer we cannot choose the network topology, because it is part of the system. We could, though, incorporate onto the scheduler policies to choose partitioning-mapping combinations, that should be chosen taking into account the characteristics of the application. Finding perfectly regular partitions that match with the virtual topology of the application is desirable, but difficult to achieve. Random mappings, which work reasonably well for small networks, become unusable for larger networks, for which the distance and the inter-application interference increase correspondingly. In general, given reasonably compact partitions (non-random), TD mappings provide excellent results.

9. 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, called TD, 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 and 3D square tori of size 64, in single-application environments. Results show that the proposed TD criterion surpasses the results of the classic one in all cases. 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 good quality mappings.

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 these space-sharing environments, we have used 64-task applications from the NPB suite running in a 256-node 2D square torus and in a 512-node 3D square torus, with three types of partitioning schemas in each topology running four (2D topology) and eight (3D topology) instances of each NAS application. 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.

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. We plan to consider an even more realistic scenario and carry out experiments in which different applications share a single machine dynamically by means of a scheduler.

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 already investigated the effects of contiguous partitioning schemas on scheduling performance [42]. The results show improvements *only* if applications experiment a significant speed-up when tasks are located contiguously. Therefore, if we incorporate better mappings, this requirement could be relaxed while still improving system's performance.

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, TIN2010-14931 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. Mr. Miguel-Alonso is supported by the HiPEAC European Network of Excellence.

References

[1] N.R. Adiga, et al., An overview of the Blue Gene/L supercomputer, in: Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, IEEE Computer Society Press, Los Alamitos, CA, USA, 2002, pp. 1–22.
[2] T. Agarwal, A. Sharma, A. Laxmikant, L.V. Kalé, Topology-aware task mapping for reducing communication contention on large parallel machines, in: IEEE International Parallel and Distributed Processing Symposium, IEEE Computer Society, Los Alamitos, CA, USA, 2006, p. 122.
[3] Ahmad Faraj, Sameer Kumar, Brian Smith, Amith Mamidala, John Gunnels, MPI collective communications on the Blue Gene/P supercomputer: Algorithms and optimizations, in: Proceedings of 17th IEEE Symposium on High Performance Interconnects.
[4] R. Ansaloni, The Cray XT4 Programming Environment, March 2007. http://www.csc.fi/english/csc/courses/programming/.
[5] Y. Aridor, T. Domany, O. Goldshmidt, J.E. Moreira, E. Shmueli, Resource allocation and utilization in the BlueGene/L supercomputer, IBM Journal of Research and Development 49 (2–3) (2005) 425–436.
[6] D. Bailey, T. Harris, W. Saphir, R.V.D. Wijngaart, A. Woo, M. Yarrow, The NAS Parallel Benchmarks 2.0. Technical report, NASA Advanced Supercomputer (NAS) division, 1995.
[7] A. Bhatele, L.V. Kalé, Application-specific topology-aware mapping for three dimensional topologies, in: Proceedings of Workshop on Large-Scale Parallel Processing, IPDPS, IEEE Computer Society, Miami, FL, USA, 2008, pp. 1–8.

[8] A. Bhatele, L.V. Kalé, Benefits of topology aware mapping for mesh interconnects, Parallel Processing Letters 18 (4) (2008) 549–566.
[9] A. Bhatele, 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, IEEE Computer Society, Washington, DC, USA, 2009, pp. 1–8.
[10] A. Bhatele, 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.
[11] A. Bhatele, L.V. Kalé, Quantifying network contention on large parallel machines, Parallel Processing Letters 19 (4) (2009) 553–572.
[12] S.H. Bokhari, On the mapping problem, IEEE Transaction on Computers 30 (3) (1981) 207–214.
[13] S.W. Bollinger, S.F. Midkiff, Heuristic technique for processor and link assignment in multicomputers, IEEE Transactions on Computers 40 (3) (1991) 325–333.
[14] R.E. Burkard, S.E. Karisch, F. Rendl, QAPLIB — a quadratic assignment problem library, Journal of Global Optimization 10 (4) (1997) 391–403.
[15] J.M. Camara, M. Moreto, E. Vallejo, R. Beivide, J. Miguel-Alonso, C. Martinez, J. Navaridas, Twisted torus topologies for enhanced interconnection networks, IEEE Transactions on Parallel and Distributed Systems 99 (PrePrints) (2010).
[16] T. Chockalingam, S. Arunkumar, Genetic algorithm based heuristics for the mapping problem, Computers & Operations Research 22 (1) (1995) 55–64.
[17] Cray Inc. http://www.cray.com/assets/pdf/products/xe/idc_948.pdf.
[18] W. Dally, B. Towles, Principles and practices of interconnection networks, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
[19] T. Feo, M. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (1989) 67–71.
[20] C. Fleurent, J.A. Ferland, Genetic hybrids for the quadratic assignment problem, in: DIMACS Series in Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, USA, 1993, pp. 173–187.
[21] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.
[22] J. Hu, 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, ACM, New York, NY, USA, 2003, pp. 233–239.
[23] W. Kruskal, W. Wallis, Use of ranks in one-criterion variance analysis, Journal of the American Statistical Association 47 (1952) 583–621.
[24] P.S. Laursen, Simulated annealing for the qap — optimal tradeoff between simulation time and solution quality, European Journal of Operational Research 69 (2) (1993) 238–243.
[25] T. Lei, 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, IEEE Computer Society, Washington, DC, USA, 2003, p. 180.
[26] V.M. Lo, Temporal communication graphs: Lamport's process-time graphs augmented for the purpose of mapping and scheduling, Journal of Parallel and Distributed Computing 16 (4) (1992) 378–384.
[27] E.M. Loiola, N.M.M. de Abreu, P.O.B. Netto, P. Hahn, T.M. Querido, A survey for the quadratic assignment problem, European Journal of Operational Research 176 (2) (2007) 657–690.
[28] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, F. Hessel, Exploring NoC mapping strategies: An energy and timing aware technique, in: Proceedings of the conference on Design, Automation and Test in Europe, IEEE Computer Society, Washington, DC, USA, 2005, pp. 502–507.
[29] A. Marzetta, A. Brungger, A dynamic-programming bound for the quadratic assignment problem, in: Computing and Combinatorics, Springer-Verlag, London, UK, 1999, pp. 339–348.
[30] H. Mühlenbein, M. Gorges-Schleuter, O. Krämer, New solutions to the mapping problem of parallel systems: the evolution approach, Parallel Computing 4 (3) (1987) 269–279.
[31] J. Miguel-Alonso, C. Izu, J. Gregorio, Improving the performance of large interconnection networks using congestion-control mechanisms, Performance Evaluation 65 (3-4) (2008) 203–211.
[32] J. Miguel-Alonso, J. Navaridas, F.J. Ridruejo, Interconnection network simulation using traces of MPI applications, International Journal of Parallel Programming 37 (2) (2009) 153–174.
[33] A. Misevicius, A tabu search algorithm for the quadratic assignment problem, Computational Optimization and Applications 30 (1) (2005) 95–111.
[34] S. Murali, G.D. Micheli, Bandwidth-constrained mapping of cores onto NoC architectures, in: Design, Automation and Test in Europe Conference and Exhibition, vol. 2, IEEE Computer Society, Los Alamitos, CA, USA, 2004, pp. 896–901.
[35] NASA Advanced Supercomputer (NAS) division. NAS parallel benchmarks, 2002, http://www.nas.nasa.gov/Resources/Software/npb.html.
[36] J. Navaridas, J. Miguel-Alonso, J.A. Pascual, F.J. Ridruejo, Simulating and evaluating interconnection networks with insee, Simulation Modelling Practice and Theory 19 (1) (2011) 494–515. Modeling and Performance Analysis of Networking and Collaborative Systems.
[37] J. Navaridas, J. Miguel-Alonso, F.J. Ridruejo, W. Denzel, Reducing complexity in tree-like computer interconnection networks, Parallel Computing 36 (2–3) (2010) 71–85.
[38] J. Navaridas, J.A. Pascual, 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, Washington, DC, USA, February 2009, IEEE Computer Society, 2009, pp. 55–61.
[39] J. Navaridas, L.A. Plana, J. Miguel-Alonso, M. Luján, S.B. Furber, Spinnaker: impact of traffic locality, causality and burstiness on the performance of the interconnection network, in: CF '10: Proceedings of the 7th ACM International Conference on Computing Frontiers, ACM, New York, NY, USA, 2010, pp. 11–20.
[40] J.M. Orduña, F. Silla, J. Duato, On the development of a communication-aware task mapping technique, Journal of Systems Architecture 50 (4) (2004) 207–220.
[41] P.M. Pardalos, F. Rendl, H. Wolkowicz, The quadratic assignment problem: a survey and recent developments, in: Proceedings of the DIMACS Workshop on Quadratic Assignment Problems, vol. 16, AMS, Providence, USA, 1994, pp. 1–42. American Mathematical Society.
[42] J.A. Pascual, J. Navaridas, J. Miguel-Alonso, Effects of topology-aware allocation policies on scheduling performance, in: Job Scheduling Strategies for Parallel Processing, IPDPS, Springer-Verlag, Berlin, Germany, 2009, pp. 138–156.
[43] R. Perego, G. De Petris, Minimizing network contention for mapping tasks onto massively parallel computers, in: PDP '95: Proceedings of the 3rd Euromicro Workshop on Parallel and Distributed Processing, IEEE Computer Society, Washington, DC, USA, 1995, p. 210.
[44] K.G. Ramakrishnan, M.G.C. Resende, 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, Dordrecht, The Netherlands, 1995, pp. 57–73. Kluwer Academic Publishers.
[45] M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.
[46] M.G.C. Resende, C. Ribeiro, Greedy randomized adaptive search procedures, in: Handbook of Metaheuristics, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003, pp. 219–249.
[47] W. Saphir, R.V.D. Wijngaart, A. Woo, M. Yarrow, New implementations and results for the NAS parallel benchmarks 2, in: 8th SIAM Conference on Parallel Processing for Scientific Computing, 1997.
[48] Lawrence Livermore National Laboratory. Simple linux utility for resource management, https://computing.llnl.gov/linux/slurm/.
[49] R. Tornero, J.M. Orduña, M. Palesi, J. Duato, A communication-aware topological mapping technique for NoCs, in: Proceedings of the 14th International Euro-Par Conference on Parallel Processing, Springer-Verlag, Berlin, Germany, 2008, pp. 910–919.
[50] A.J. van der Steen, Overview of recent supercomputers, September 2010. http://www.euroben.nl/reports/overview10.pdf.

Jose A. Pascual obtained his M.Eng. in computer science from the University of the Basque Country, Gipuzkoa, Spain, in 2005. He is currently pursuing his Ph.D. at the Department of Computer Architecture and Technology of the same university. His research interests include high-performance computing, scheduling for parallel processing, and performance evaluation of parallel systems.

Jose Miguel-Alonso received his Ph.D. in computer science from the University of the Basque Country UPV/EHU (Spain) in 1996. He is currently a Full Professor at the Department of Computer Architecture and Technology of the UPV/EHU. His research interests include interconnection networks for parallel systems, network (cluster, grid) computing, performance evaluation of parallel and distributed systems, and scheduling for parallel processing.

Jose A. Lozano received a B.S. degree in mathematics and a B.S. degree in computer science from the University of the Basque Country, Spain, in 1991 and 1992 respectively, and the Ph.D. degree in computer science from the University of the Basque Country, Spain, in 1998. Since 2008 he is full professor of the University of the Basque Country, Spain, where he leads the Intelligent System Group. He is the coauthor of more than 50 ISI journal publications. His major research interests include machine learning, pattern analysis, evolutionary computation, data mining, metaheuristic algorithms, computer architecture and real-world applications. Prof. Lozano is associate editor of IEEE trans. on Evolutionary Computation and member of the editorial board of Evolutionary Computation journal, Soft Computing and other three journals.

9

Locality-aware policies to improve job scheduling on 3D tori

Locality-aware Policies to Improve Job Scheduling on 3D Tori

Jose A. Pascual, Jose Miguel-Alonso, Jose A. Lozano

Intelligent Systems Group, School of Computer Science.
University of the Basque Country UPV/EHU.
P. Manuel Lardizabal 1, San Sebastian, Spain 20018.

Abstract

This paper studies the influence that contiguous job placement has on the performance of schedulers for large-scale computing systems. In contrast with non-contiguous strategies, contiguous partitioning enables the exploitation of communication locality in applications, and also reduces inter-application interference. However, contiguous partitioning increases scheduling times and system fragmentation, reducing the overall system utilization. We propose and evaluate several strategies to select contiguous partitions to allocate incoming jobs. These strategies also take advantage of different mapping mechanisms to perform the task-to-node assignment in order to further reduce application run times. A simulation-based study has been carried out, using a collection of synthetic applications performing common communication patterns. Results show that the exploitation of communication locality by means of the correct topology-mapping results in an effective reduction of application run times, and the achieved gains more than compensate the scheduling inefficiency, therefore resulting in better overall system performance.

Keywords: Scheduling, Contiguous partitioning, Tasks mapping, Locality-aware policies.

1. Introduction

Supercomputer centers are usually designed to provide computational resources to multiple users running a wide variety of applications. Users send jobs to a scheduling queue, where they wait until the resources required by the job are available. The scheduler is in charge of selecting the order in which jobs will be executed, find the partition to allocate it, and perform the assignment of the application tasks to the selected nodes. Note that we use the terms job and application interchangeably.

In a parallel system, application tasks (running on network nodes) communicate interchanging messages. Depending on the set of nodes assigned to them, severe delays may appear due to network contention; delays that result in longer execution times. An effective exploitation of locality using contiguous partitioning results in smaller communication overheads, which reflects in lower running times [14]. However, searching for this locality is expensive in terms of scheduling time, because jobs cannot leave the queue until contiguous resources are available. The result is an increase in system fragmentation: free nodes that cannot be used because they are not contiguous.

Once a partition is selected for a job, the tasks-to-nodes assignment must be performed. The effectiveness of this mapping strongly depends on the topology of the partition and the communication pattern of the application [14]. Therefore the selection of the correct mapping for each application becomes crucial to exploit efficiently system resources and, in consequence, achieve the best run times.

The gains attainable via exploitation of locality, combined with the use of appropriate mapping strategies, may compensate the negative effects of increasing fragmentation. This is precisely the focus of this paper. Our final goal is to demonstrate that systems using locality-aware policies (contiguous partitioning +

Email addresses: joseantonio.pascual@ehu.es (Jose A. Pascual), j.miguel@ehu.es (Jose Miguel-Alonso), ja.lozano@ehu.es (Jose A. Lozano)

appropriate mapping) can perform better than systems using non-contiguous partitioning, even with lower levels of overall system utilization.

In order to evaluate the partition selection strategies (and the associated mappings) a simulation-based study has been carried out using synthetic applications. Due to the complexity of simulating the whole scheduling process with actual applications, we have split the evaluation in two stages. First we have selected a collection of synthetic applications (communication patterns) and used INSEE [12], a very detailed interconnection networks simulator, to estimate their run times in a wide diversity of combinations of partitions and mappings. With contiguous partitioning, applications can run concurrently without interference between them; therefore the run time of an application will be the same when executed using the same partition topology and the same mapping strategy in a larger system. The result of this stage is a table of run times that is used in a second stage, in which we evaluate topology-aware scheduling strategies on a large-scale supercomputing with a 3D torus topology.

Results show that locality-aware policies have a huge impact on system performance. The use of these policies is able to remarkably reduce applications average run times. As consequence other scheduling metrics, such as the average waiting time, are also improved. Actually, the whole scheduling process improves, using metrics such as waiting times and job throughput. The best results are achieved by an strategy that we call BPS (Best Partition Selection), despite the fact that it exhibits the lowest levels of system utilization: even "wasting" 20% of the system nodes, it is the best performer of all contiguous strategies. Additionally, we also show that locality-aware policies can outperform non-contiguous strategies.

The rest of the paper is organized as follows. In Section 2 we describe the scheduling process, presenting in Sections 3 and 4 different strategies to select partitions and perform the tasks-to-nodes assignment (mapping). Section 5 presents in detail the experimental set-up, while Section 6 presents and discusses experiments showing the effect of partitioning strategies on application performance. In Section 7 we explain the design of the experiments involving topology-aware scheduling, and analyze the results. Section 8 is focused on the search of a trade-off between application slowdown and scheduling acceleration using non-contiguous partitioning. Finally, Section 9 closes the paper with some conclusions and future lines of work.

2. Overview of the scheduling process

In this paper we consider the scheduling process for supercomputers as depicted in Figure 1. The process involves three steps since the arrival of a job to the queue until it leaves the system. First the scheduler must select a job from the queue. Then, it has to select a system partition based on the number of nodes requested. Finally, tasks must be assigned to nodes (mapping) to put the job to run. Each step of this process is guided by a policy or strategy.

2.1. Job selection

Almost all supercomputers are managed by a scheduler that is in charge of assigning resources to the jobs submitted by users. These jobs are temporarily stored in a queue (or several queues) that is managed using a given job selection policy. Many job selection policies have been proposed in the literature, being a widely used one First Come First Served (FCFS) [7]. Using this policy, jobs are processed strictly in arrival order, and executed as soon as the scheduler locates the available (free) resources required by the job. While resources are not available for the job at the head of the queue, that job and all submitted after it must wait, despite the possibility of allocating jobs not at the head. The main drawback of this policy is that it produces system fragmentation, that can be severe for large jobs, because some system nodes will remain idle during a long period of time until the necessary collection is ready.

Other job selection policies try to reduce this drawback of FCFS. Backfilling (BF) [7] allows the advance of jobs even when they are not at the head of the queue. Shortest Jobs First (SJF) [7]) selects the jobs to be executed using their expected run time instead of their arrival time. In this work we *do not* evaluate scheduling policies. For simplicity, we have used in all the experiments the FCFS job selection policy, even knowing that alternatives such as BF and SJF could yield better performance.

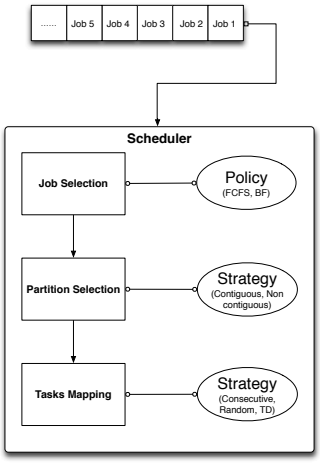


Figure 1: Representation of the different steps that compose the scheduling process.

2.2. Partition selection

After a job is selected from the queue (in the scenarios we consider it will always be the oldest one), the scheduler has to locate a system partition in which it will run. A partitioning strategy guides the search. We can differentiate two wide sets of partitioning strategies:

- Contiguous strategies: They search for partitions that form convex shapes. These partitions reduce the average distance traveled by the messages interchanged between tasks, and also the interference caused between different jobs sharing the system because most, if not all, communications are confined inside the partition. The negative side of contiguous partitioning is that it exacerbates fragmentation: the selected job has to wait until enough resources *with the expected shape* are available.
- Non-contiguous strategies: This class of partitioning was proposed to reduce fragmentation [9] [3], increasing system utilization at the cost of sacrificing communication locality. The contiguity requirement is relaxed or totally eliminated, looking for free nodes anywhere in the network. The expected effect is an increase in application run times due to non-optimal communications and inter-application interference.

In this work we introduce and evaluate a collection of contiguous partitioning strategies. We know the cost to pay in terms of scheduling time: jobs must wait until contiguous partitions with the required shapes become available. However, once allocated, jobs will run efficiently and without meddling with other jobs sharing the system, therefore releasing assigned resources earlier. This improvement can compensate the previous inefficiency.

In order to search for contiguous partitions a basic strategy, first-fit, is used: the system is traversed in a fixed order, until the expected available shape is located. We use an optimized algorithm to implement this strategy, called IFF and described in [15]. In some experiments we also use non-contiguous partitioning. In this case we use a simple algorithm that searches free nodes in sequential order ignoring locality. This is the most used technique in commercial systems, like the Cray XT3/XT4, that simply gets the first available compute processors [1]. Note that this strategy provides a flat vision of the network, ignoring its topological characteristics and the virtual topologies of the scheduled application. Also, note that in the long run it behaves as a random allocation of resources.

2.3. Tasks mapping

Once the required partition has been assigned to the job to be run, the scheduler has to map the tasks onto the physical nodes. The mapping can have a significant impact on application performance [14] [4]

[19] [21]. Therefore, in this paper we will evaluate different combinations of mapping and partition selection strategies: a bad mapping cannot take advantage of having reserved a contiguous partition.

The mapping problem can be formally expressed as follows [14]: 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 bijection π that assigns a task t to a node p

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

minimizing the following objective function:

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

where T and P are two equal size matrices representing parallel application tasks and processors respectively, matrix $W = [w_{i,j}]_{i,j \in T}$ represents the number of bytes interchanged between each pair of tasks, and matrix $C = [c_{i,j}]_{i,j \in P}$ represents a certain cost criterion involving pairs of network nodes.

Most schedulers do not perform task-to-node assignment taking into consideration application's characteristics. Instead, they carry out a simple, consecutive assignment. In some cases, given the right shapes of partition and application's virtual topology, this assignment can be good. However, in many cases it will not help applications to achieve its maximum performance; in fact, it can severely increase run time. In this paper we will consider different strategies to perform the best possible mappings.

3. Partition selection strategies

Once the overall scheduling process has been discussed, in this section we focus on partition selection strategies. We consider a scenario in which a characteristic of a waiting job is the number of nodes needed to run in. The scheduler will then select the most appropriate partition shape to run it. Note that in the literature some works assume that job requests include the expected partition shape [20] [8] [2]. We delegate this decision to the scheduler.

We propose and evaluate three contiguous partition selection strategies: Cubic Selection (CS), Factors Selection (FS) and Best Partition (BS), together with a basic non-contiguous strategy. Note that all of them are discussed in the context of a supercomputer built around a 3D torus network.

3.1. Cubic selection strategy

This strategy looks for the *most cubic* partition for each job. This kind of partitions provides interesting topologic properties including symmetry, low node degree and low diameter to execute applications [5].

Schedulers implementing the Cubic Selection (CS) strategy, using the partition size requested by the job to run, determine the most cubic partition. In Table 1 we have summarized all the possible partition topologies for different requests sizes. The first column (P1) contains the most cubic partitions, those CS will look for. Once a valid partition has been found and reserved, job tasks have to be mapped to the partition nodes, but this depends on the chosen mapping strategy, discussed in the next section.

The aim the CS strategy is not only to take advantage of the good topological properties of the cubic partitions. It also takes advantage of the use of compact, symmetric partitions that can help reducing network fragmentation and, therefore, reducing the job waiting times.

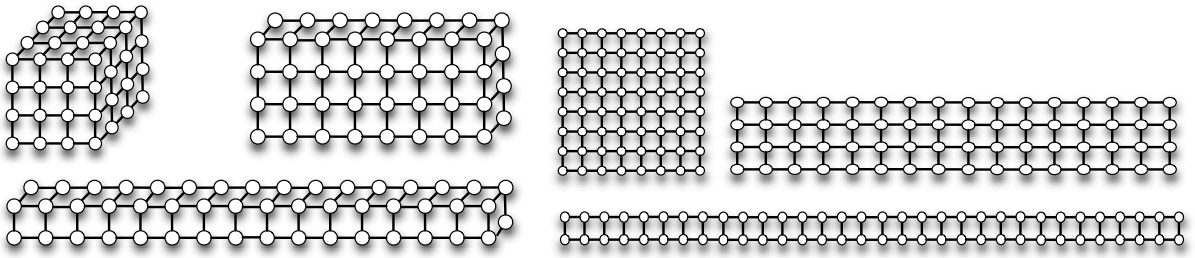
3.2. Factors selection strategy

The Factors Selection (FS) strategy searches for contiguous partitions in which to allocate the requested number of nodes. We call factor to each one of the possible partitions, that can have a 2D or a 3D rectangular shape (always in the context of a 3D torus, and omitting 1D partitions for performance reasons). In Table 1 we have summarized all the possible factors for different request sizes.

Size	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
8	$(2 \times 2 \times 2)$	(4×2)	–	–	–	–	–	–	–	–	–
16	$(4 \times 2 \times 2)$	(4×4)	(8×2)	–	–	–	–	–	–	–	–
32	$(4 \times 4 \times 2)$	$(8 \times 2 \times 2)$	(4×4)	(16×2)	–	–	–	–	–	–	–
64	$(4 \times 4 \times 4)$	$(8 \times 4 \times 2)$	$(16 \times 2 \times 2)$	(8×8)	(16×4)	(32×2)	–	–	–	–	–
128	$(8 \times 4 \times 4)$	$(16 \times 4 \times 2)$	$(8 \times 8 \times 2)$	$(32 \times 2 \times 2)$	(16×8)	(32×4)	(64×2)	–	–	–	–
256	$(8 \times 8 \times 4)$	$(16 \times 4 \times 4)$	$(16 \times 8 \times 2)$	$(32 \times 4 \times 2)$	$(64 \times 2 \times 2)$	(16×16)	(32×8)	(64×4)	(128×2)	–	–
512	$(8 \times 8 \times 8)$	$(16 \times 8 \times 4)$	$(16 \times 16 \times 2)$	$(32 \times 4 \times 4)$	$(32 \times 8 \times 2)$	$(64 \times 4 \times 2)$	$(128 \times 2 \times 2)$	(32×16)	(64×8)	(128×4)	(256×2)

Table 1: List of possible 2D and 3D partitions (factors) for different size requests.

Note that, given a partition size, there are multiple valid factors. For example, for a 64-node request we can choose between six options as shown in Figure 2. Having many different options increases the probability of finding a valid partition, therefore the waiting time of the jobs is expected to be reduced. However, the topology of the partition affects applications performance. Not all the factors are equally appropriate for the application to run at maximum performance. We have to determine if the improvement provided by FS in terms of scheduling time compensates the possible negative effect in application run times.



(a) 3D partitions (b) 2D partitions

Figure 2: Set of possible partitions to accommodate a 64 nodes request.

A scheduler implementing FS will search for any valid factor, but a searching order has to be established to try to locate first the most promising ones. A well accepted criterion of performance in contiguous partitioning is the Average Pairwise Distance (APD) between the nodes forming the partition. Therefore the scheduler will search the factors in increasing order of APD.

3.3. Best partition selection strategy

The third partition selection strategy we propose is Best Partition Selection (BPS). It requires application-specific knowledge, about the communication patterns of the message interchange. It then searches for the best partition to execute the application. The main disadvantage of this strategy is that it must wait for an specific shape, potentially increasing the waiting time (until that partition becomes available). Again, it is to be seen if the potential gains in run times compensates this loss.

3.4. Non-contiguous partitions

The selection of non-contiguous partitions clearly benefits the scheduling performance in terms of waiting times. Finding this kind of partitions is easy because the contiguity requirement is eliminated: as soon as enough resources are available, anywhere in the network, the job can be allocated. The risk is a severe damage to application performance, because of the lack of locality in communications, exacerbated by inter-application interference.

4. Mapping strategies

Parallel applications performance is strongly related to the way the tasks are assigned to the network nodes of the assigned partition. Mapping can be done using different strategies and their effectiveness

depends on many factors, such as the shape of the partition, the communication pattern of the application and the amount of data interchanged between application tasks.

In this work we use three different mapping strategies to be combined with the partition selection strategies introduced in the previous section. The first two have a common characteristic: they are application-agnostic, not requiring information about the application's communication patterns. In contrast, the third one is based on optimization, and require application specific information (matrices W and C) to find a good mapping. The mapping strategies we consider in this study are:

- Consecutive Mapping (CM): Given a partition, application tasks are assigned to the nodes in order of identifier, 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 (RM): Given a network partition, application tasks are assigned to nodes chosen uniformly at random, one task per node.
- Optimization based (TDM): Tasks are assigned to nodes optimizing a given criterion, as defined in Subsection 2.3. In this work we use a criterion called Traffic Distribution (TD) [14], which allows finding mappings that, while not optimal in terms of communications average distance, helps avoiding network bottlenecks by taking advantage of adaptive routing (through minimal paths) in 3D tori. The criterion favors those routes in which the number of hops per dimension is equalized or, in other words, it penalizes those routes in which the number of hops per dimension is not equal. The level of penalization depends on the level of asymmetry between each pair of communicating tasks.
- Best Mapping (BM): Given a detailed knowledge of the application and the partition in which it will run, it is possible to determine the most appropriate mapping between the three previously defined ones. Best Mapping Strategy allows the scheduler to make this decision.

A topology-aware scheduler may use different combinations of selection and mapping strategies. The aim in all cases is to find a trade-off between communication locality (efficiently running jobs) and scheduling performance (reducing waiting times in the job queue). In table 2 we have summarized all the combinations. Note that for the Best Partition Selection strategy we only consider the Best Mapping strategy because both concepts of optimality are tightly linked.

Partition Selection	Maping	Cobination
Cubic Selection	Consecutive	CS-CM
Cubic Selection	Random	CS-RM
Cubic Selection	TD	CS-TDM
Cubic Selection	Best Mapping	CS-BM
Factors Selection	Consecutive	FS-CM
Factors Selection	Random	FS-RM
Factors Selection	TD	FS-TDM
Factors Selection	Best Mapping	FS-BM
Best partition	Best Mapping	BPS-BM

Table 2: Description of the different combinations of partition selection and mapping strategies

5. Experimental environment

In this section we detail the experimental set-up used to evaluate the partition selection strategies proposed in this paper. We have carried out several experiment sets:

- The first set evaluates the influence of the different combination of contiguous partitioning and mapping strategies on single-application performance. They will serve to show how a good choice of partition topology, together with an appropriate mapping for the application-partition combination, can boost application performance.
- In contrast, the second sets measures the (negative) effect of non-contiguous partitioning on job run times.
- The third set of experiments, the core of this paper, evaluates our proposals of topology-aware scheduling, presenting the impact of consecutive mapping and partitioning strategies in the context of a large 3D supercomputer. The benefits of these strategies will be shown in terms of per-application and per-system metrics.
- Finally, we trade-off the potential benefits of non-contiguous partitioning with the expected negative impact on application run times.

It has been an extensive collection of simulation-based experiments, carried out with the set of tools we present in the forthcoming sections.

5.1. Workloads

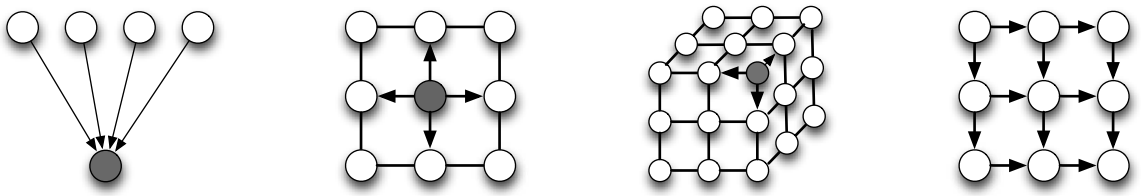
We need to "feed" our simulated supercomputer with a real, or realistic, workload. In this paper we have used a collection of 17 synthetic applications that communicate using a variety of common patterns. Actually we use traces of these applications, generated using the framework proposed in [13] [11]. Each application (trace) can be parameterized with a size (number of communicating tasks) and an intensity (number of packets interchanged by the tasks). The applications can be grouped in four classes, summarized in Table 3, depending on the communication pattern it uses, that implies a certain virtual topology.

Class I	Class II	Class III	Class IV
All-to-All	2D Mesh	3D Mesh	BinaryTree
All-to-One	2D Torus	3D Torus	Butterfly
One-to-All	2D Sweep	3D T-Mesh	GUPS
–	2D T-Mesh	3D T-Torus	Waterfall
–	2D T-Torus	–	–
–	2D T-Sweep	–	–

Table 3: Synthetic applications arranged in classes

- Class I: Applications based on collective communications. For example, Figure 3(a) depicts the communication pattern of *All-to-One*. Bullets represent tasks and arrows represent message flows.
- Class II: Applications that perform communications using a 2D pattern. Figure 3(b) corresponds to *2D Mesh*.
- Class III: Applications that perform communications using a 3D pattern. Figure 3(c) corresponds to *3D Mesh*.
- Class IV (Miscellanea): Four different applications (*Binary Tree*, *Butterfly*, *GUPS* and *Waterfall*), each one with its particular pattern. As an example, in Figure 3(d) we have depicted the *Waterfall* application.

Some applications can use any number of tasks (size), but others have restrictions. In Table 4 we have detailed the actual sizes used for each class. See for example how Class II applications must have a quadratic number of tasks, while Class III applications are cubic. For simplicity reasons, in all cases sizes are powers of 2, but this is not a limitation of our proposed scheduling strategies.



(a) All-to-One (b) 2D Mesh (c) 3D Mesh (d) Waterfall

Figure 3: Communication patterns of some applications.

	8	16	32	64	128	256	512
Class I	•	•	•	•	•	•	•
Class II	–	•	–	•	–	•	–
Class III	•	–	–	•	–	–	•
Class IV	•	•	•	•	•	•	•

Table 4: Application sizes

5.2. Computing applications run times

Once we have an application (actually, a trace), we need to compute its running time in a given system partition, after applying a given task-to-node mapping. We make these computations using the INSEE [12] tool set, that can predict the time required to run a parallel application on a parallel computing system. Using simulation, INSEE measures the time required to process all the messages in a trace, in the right order, considering the effects of application-induced causal relationships and also those derived from network-induced resource contention. INSEE is very flexible in terms of the interconnection networks it models; it accepts different topologies, switching mechanisms, routing strategies, etc. In particular, we have used cut-through switching, two virtual channels per physical link, adaptive routing through minimal paths, and bubble flow control to avoid deadlock [17].

5.3. Simulating scheduling strategies

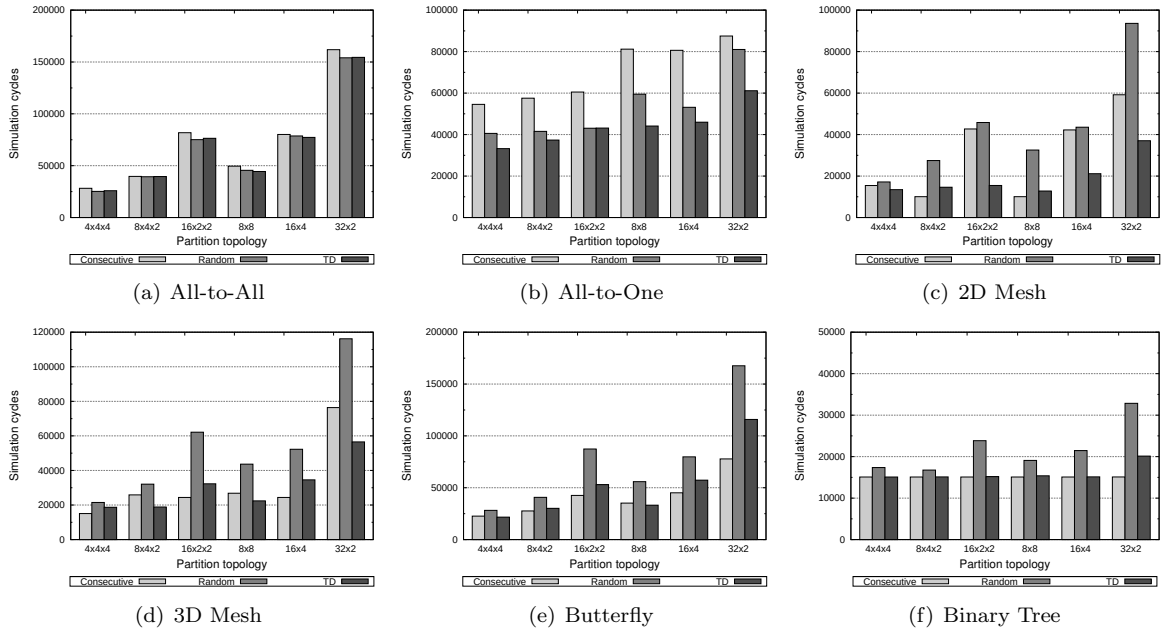
An scheduling simulator has been developed in-house [16] as a tool to evaluate scheduling strategies. Among other features, it implements the FCFS job selection policy, the IFF algorithm to search for contiguous partitions, and all the partition selection and mapping strategies proposed in this paper, including the trivial non-contiguous strategy defined in Section 2.1.

The scheduler simulator takes as input a sequence of jobs requests, each one with a timestamp, and the application to run (with its parameters, such as size). The simulator *also requires* as input the run time for that job (after selecting partition and mapping): it does not simulate the execution of the application. To that extent, the run times of all possible combinations of application, sizes, partition and mapping were pre-computed with INSEE, and tabulated to be easily used by the scheduling simulator.

In all of the experiments we simulate a supercomputer built around an interconnection network with a $(32 \times 32 \times 32)$ 3D torus topology.

6. Effects of partitioning and mapping on application performance

In this section we evaluate the effect of partitioning and mapping combinations on the performance of parallel applications. Note that in these experiments no scheduler is involved. First we evaluate the use of contiguous partitions with different mapping strategies, with the objective of showing how the topology of the partitions influence run times. Applications are mapped into a selection of partitions using different mapping strategies: both must be considered in combination.



(a) All-to-All (b) All-to-One (c) 2D Mesh

(d) 3D Mesh (e) Butterfly (f) Binary Tree

Figure 4: Run times of six different applications executed in six different contiguous partitions topologies. Three mapping strategies have been used in each application-partition combination.

We also include an evaluation of application performance in space-shared supercomputers using non-contiguous partitioning, in order to assess to what extent messages interchanged by applications suffer delays (due to the use of larger routes, and to inter-application interference) that increase run times.

6.1. Contiguous partitioning strategies

Given a job and a partition in which to run it, additional information is required in order to select the most appropriate mapping, that is, the one allowing the application to run faster. In [14] some hints are given about which mapping could be the best, based on knowledge of the application's communication pattern:

- Consecutive: when there is a perfect match between application's virtual topology and the physical topology of the partition.
- Random: when applications create high levels of contention in the whole partition, for example using All-to-All collectives.
- TD: a general-purpose technique based on optimization that works well when virtual and physical topologies do not match. Even in the previous case (high levels of contention) its performance is similar to that of Random mapping. Notice that TD has been defined only for cubic topologies.

In Figure 4 we have depicted the run time of six different applications (of different classes) formed by 64 tasks, each one running in six different 64-node topologies using the three different mappings. For each application separately, it is easy to visually asses how for some applications achieved performance differs widely depending on the resource assignment, while others are able to work similarly well in different environments. Let us now analyze the figure case by case.

Results for *All-to-All* and *All-to-One* (Figures 4(a) and 4(b)) show that these applications work best in the most cubic (compact) partitions. This arrangement minimize communication overheads. Regarding the mapping, TDM is in general the best option, although Random works slightly better with *All-to-All*.

The cubic arrangement is also the best partition choice for *3D mesh* and *Butterfly*. In the first case (Figure 4(d)) because there is a perfect topology match when the mapping is consecutive (other mappings perform worse). In the second (Figure 4(e)) this match is not perfect, but the topological characteristics of the partition, together with a good mapping (either consecutive or TD) allows message interchange to be carried out with minimum interference. Notice how Random mapping is never the best option and how, when the chosen partition is not cubic, TD is a very competitive mapping strategy.

For the *2D Mesh* application, obviously the best partition choice is a flat one providing the perfect topology match, together with consecutive mapping (Figure 4(c)). Again, when using other partition performance suffers, but TD can do a good work optimizing communications.

Regarding the *Binary Tree* application, its behavior is interesting because apparently it is not very affected by the underlying topology and mapping. This is because in this application communications are bounded by dependency chains, thus not imposing a high pressure on network resources. This being said, Consecutive and TD mappings work well, being Random a bad choice.

Summarizing this experiment set, we can confirm our expectations: the best partition-mapping option is application dependent, and the obtained run times for the same application differ widely in different resource assignment scenarios. A bad choice may multiply run times by a large factor. In this restricted set of experiments, over $\times 6$ for the *3D Mesh* application.

6.2. Non-contiguous partitioning

The previous experiments were run with applications running in isolation, without interferences from other, concurrently running jobs. This does not correspond with the scenario we are interested in: a space-shared supercomputer. However, assuming that a contiguous partition strategy is being used, results should be equally valid, because each partition will work in isolation, without interfering, pass-through traffic.

When using non-contiguous partitioning, the situation changes drastically, and interference (in terms of contention for the use of shared network resources) will appear, negatively affecting run times. In this section we carry out a set of experiments to evaluate this effect. We have simulated eight instances of each synthetic application, with size 64 tasks, running simultaneously in a $(16 \times 16 \times 16)$ 3D mesh. Each instance was mapped to a 64-node, non-contiguous partition, with nodes chosen at random. Then we have measured the run times. For the sake of clarity, we present results for just three applications, of different classes: *3D Torus*, *Binary Tree* and *All-to-All*.

Due to the random selection of nodes for each job, each experiment will yield a different run time, depending on the characteristics of the resource assignment. For this reason, we have performed a large collection of repetitions, generating different partitions each time. The results summarized in Figure 5 represents all the run times returned by INSEE. In order to better understand the results, we have ordered them taking into consideration the *Average Pairwise Distance* of the generated partitions: those with lower APD values are more "compact" than those with larger values. Also, we have normalized the results in such a way that 100% represents the run time in a contiguous $(4 \times 4 \times 4)$ 3D mesh with the best mapping. In *all* cases we observe how achieved run times are longer in non-contiguous settings, sometimes up to $\times 3.5$ worse than in a contiguous partition. However, the degree of slowdown strongly depend on the class of application: for example, *Binary Tree* is slightly affected (something to be expected because the communication bottleneck for this application is in the chain of dependencies, not in the network itself). The relationship between APD and performance is clear only for the *All-to-All* application, which imposes high levels of pressure onto the interconnection network.

To summarize this section, we can state that non-contiguous partitioning always provide worse results, in terms of run times, than contiguous partitioning, but the degree of slowdown depends strongly on application characteristics. For communication-intensive workloads, the increase of the distance traversed by messages, together with the inter-application interference, severely damage run times.

7. Scheduling with contiguous partitioning

After assessing the positive effects of contiguous partitioning and mapping strategies, we go a step further and analyze in this section the effect of introducing these techniques in the resource manager, the

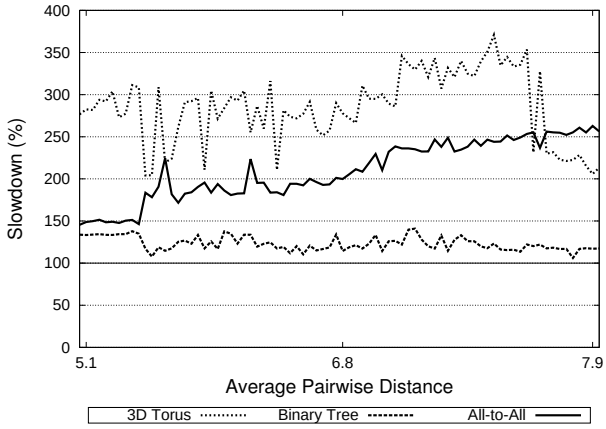
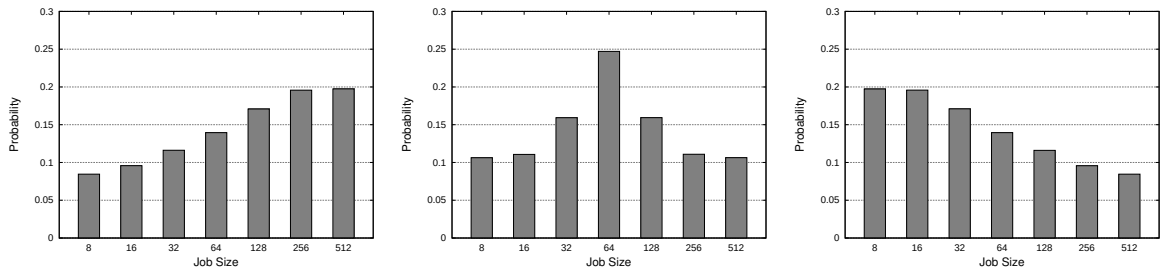


Figure 5: Slowdown of three 64-node applications running in non-contiguous environments. Values normalized to 100%, the run time in a contiguous $(4 \times 4 \times 4)$ mesh.



(a) Large size jobs (b) Medium size jobs (c) Small size jobs

Figure 6: Probabilities assigned to each size to generate large, medium and small average size workloads.

supercomputer scheduler. To that extent we use the scheduler simulator, feed it with a variety of workloads, and obtain a collection of performance metrics that will help us to determine which partition selection and mapping strategies provide the best scheduling performance.

7.1. Design of the experiments

The workload used in the experiments play an important role in this evaluation because, as we have seen in the previous sections, not all applications are equally sensitive to the way resources are assigned to them. In this section we define the way workloads have been generated, mixing different applications, and the way the running times have been pre-computed in order to feed the simulator. We also present the design of the experimental study carried out.

7.1.1. Definition of the workloads

We have created five different sets of workloads to asses in which way the performance of the proposed selection strategies depend on the application class. The first set tries to reflect a general scenario in which we mix all the applications defined previously. The remaining four sets are class-specific. Given a set, a workload is generated generating at uniformly at random an application of the selected class(es), and a communication intensity. Regarding task sizes, each set includes three workloads, with different average job size (number of tasks). These sizes are sampled using the probability distributions depicted (in histogram form) in Figure 6(a). We use the terms Large, Medium and Small to refer to workloads in which *most* of the jobs are large (128 tasks or more), medium (around 64 nodes) and small (less than 32 nodes) respectively. Figure 7 summarizes all this workload generation process.

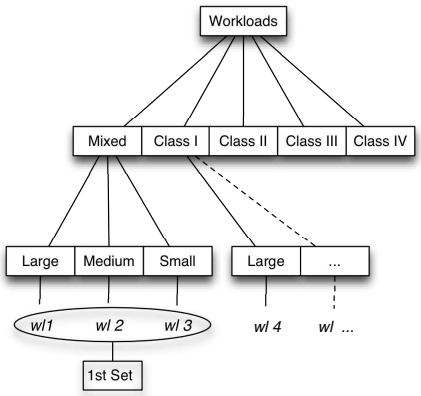


Figure 7: Sets of generated workloads, combining application classes and average job sizes.

7.1.2. Tabulating running times

As we have stated before, in contiguous partitioning there is no inter-application interference. Therefore, the run time of one application will be the same in any partition of a space-shared system provided that we use the same partition topology and mapping function. Based on this idea, and due to the complexity of evaluating complete scheduling scenarios, in a first stage we computed the run time of each application/intensity/size/partition topology/mapping combination using INSEE. This stage involved thousands of simulations, and its result is a comprehensive table to be used in the scheduling simulator stage.

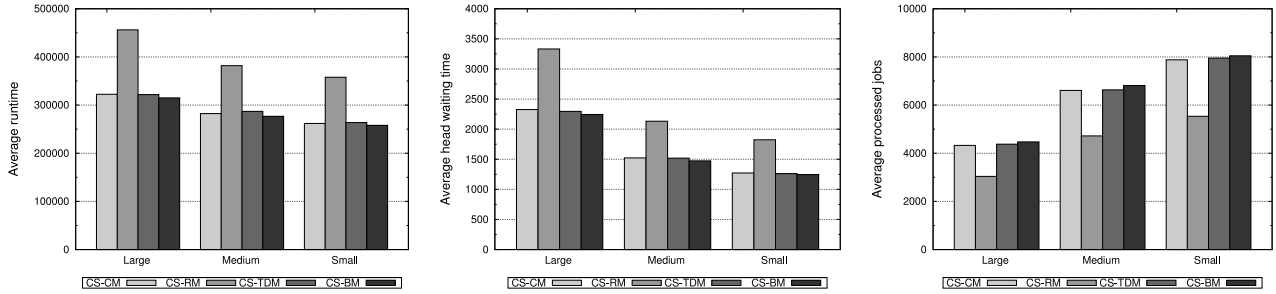
7.1.3. Experimental methodology

The evaluation of the system has been performed in two scenarios:

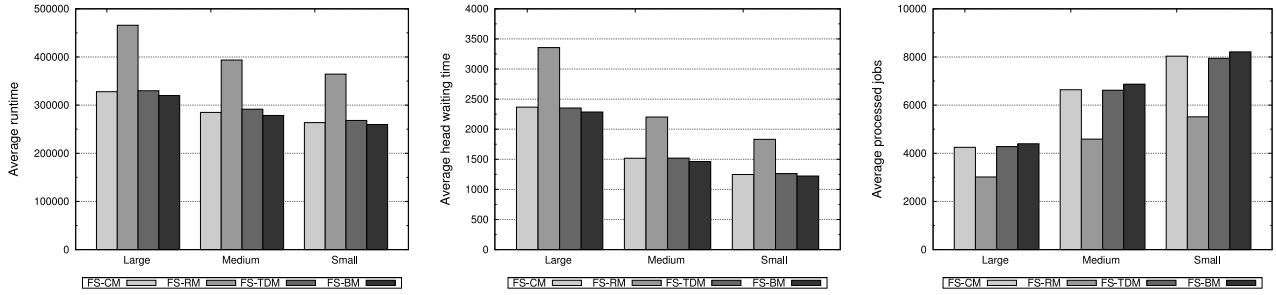
1. Simultaneous arrival of jobs (maximum input load): An ordered array of jobs arrive simultaneously to the scheduler, that must process it in order. During the duration of the experiment, the scheduler always has a next job to process. The objective of this scenario is to measure the processing capacity of the system at full blast, looking for eventual performance bottlenecks.
2. Time-spaced arrival of jobs: the scheduler receives a collection of jobs with different inter-arrival times between them. Depending on these times, the queue can be sometimes empty. This scenario better reflects a realistic use of a supercomputer, a working batch system. Inter-arrival times are modeled using an exponential distribution with parameter λ, in such a way that we can study system behavior under different levels of system demand.

Regarding collected statistics, this is a non-comprehensive list:

- Run time of each individual job.
- Number of processed jobs.
- Head waiting time: the time that a job spends since it reaches the head of the scheduling queue until it is actually put to run.
- Waiting time: the time that a job spends in the queue since its arrival until it is put to run.
- Total time: the waiting time of a job plus its run time. In other words, the time lapse since a job arrives to the system until it finishes.
- System utilization: a measurement of the number of system nodes that are active (that is, run application tasks) during an experiment.



(a) Results of the CS partition selection strategy using different mappings.



(b) Results of the FS partition selection strategy using different mappings.

Figure 8: Results of the CS and FS partition selection strategies.

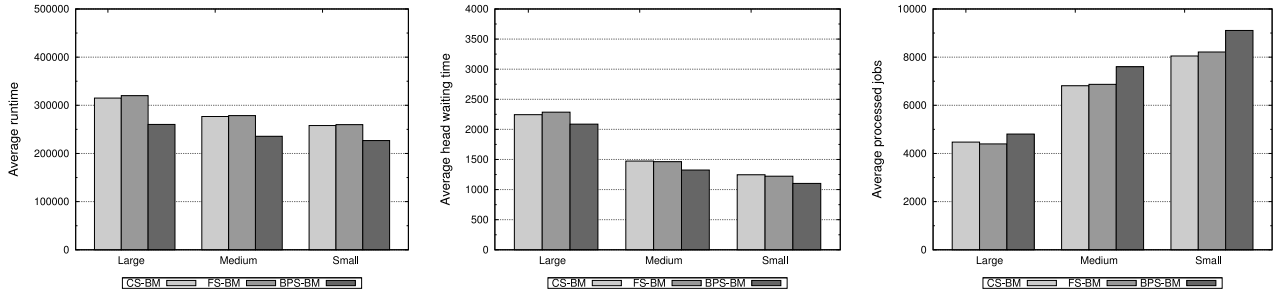


Figure 9: Performance of CS-BM, FS-BM and BPS-BM strategies.

In the first scenario, we submit the complete workload to the simulator and obtain global metrics, focusing mainly on system-related measurements. For the second scenario with non-simultaneous arrivals, we have used a method called batch mean analysis [6]. In this method a long simulation run is divided in fixed intervals (batches), after a warm-up period to allow the system to reach a steady-state regime. In each batch we compute separate means of metric samples. After several batches, we compute total means. We have used 30 batches, each one of 10000000 INSEE simulation cycles. For the sake of clarity, in the figures we do not show the standard deviation of the results, only global means.

The statistical significance of the differences between the different combinations of selection and mapping strategies have been assessed running Kruskal-Wallis [18] paired test with 0.05 level of significance. We do not show the results of these tests in the following figures and tables because, in all cases, differences between the best strategy and the remaining ones are statistically significant.

7.2. Results for contiguous partitioning with simultaneous arrival

In this subsection we analyze the behavior of the scheduling system when submitted to the simultaneous arrival of a large number of jobs, always focusing on the performance of different topology-aware partitioning and mapping strategies.

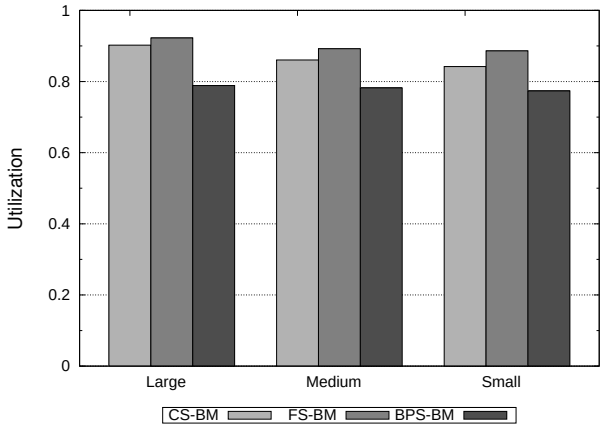


Figure 10: System utilization achieved by CS-BM, FS-BM and BPS-BM partition selection strategies, for mixed workloads and simultaneous arrivals.

7.2.1. CS and FS strategies

In Figures 8(a) and 8(b) we plot the results for mapping strategies CS and FS respectively, just for the mixed application workload in its three different sizes. Strategy CS-BM shows the best results, for any size: average run times and head waiting times are the lowest, and the system takes the shortest time to process all the applied workload. Note that, in general, the behavior of FS instead of CS is slightly, but not remarkably, worse.

Focusing on mapping, using only RM is a bad option for these mixed workloads. Choosing always CM or TDM shows good results, that can be improved selecting, for each particular application, the best one (BM strategy).

7.2.2. BPS-BM strategy

Let us now choose the best mapping strategy, BM, and compare how the different partition selection strategies behave. Results are summarized in Figure 9.

If we focus on the average run time metric, we can see that CS-BM and FS-BM strategies achieve almost the same results, being CS-BM slightly better. BPS-BM is clearly the winner, specially for large-size jobs: this partition selection strategy is able to remarkably reduce the average application run times. As a result the jobs stay less time using system resources, which also improves the average head waiting time metric for other jobs. Note that BPS is the most expensive strategy in terms of the time required to wait for the expected partition (each job requires waiting for an specific partition shape), but the acceleration experienced by applications more than compensates this overhead.

We can see a different view of this advantage when looking at the number of jobs processed by the system: the job throughput using BPS-BM is clearly the best. This is particularly true for small jobs, but also applies for medium and large-size jobs.

Generally, supercomputing centers summarize their performance metrics in terms of system utilization: the more the better. Although it is clearly not desirable to have expensive resources idle, for us it is clearer that job throughput is a much better figure describing actual system utilization, and quality of users' experience. Contiguous partitioning strategies provide excellent results in these terms, with BPS-BM remarkably increasing job throughput, even though system utilization is always below 80% of system resources (see Figure 10). As our experiments have been done in a $(32 \times 32 \times 32)$ torus (that is, 32768 nodes) this means that, on average, more than 6500 nodes are idle. We will see later in Section 8 how a non-contiguous set-up can reach a 99% utilization, but not necessarily the same levels of job throughput.

(a) Average run time.

Class	CS-CM	CS-RM	CS-TDM	CS-BM	FS-CM	FS-RM	FS-TDM	FS-BM	BPS-BM
I	202546	195318	190943	190739	212208	206168	201331	200604	**190504**
II	20902	27128	22192	20413	21045	27694	22572	20559	**17648**
III	**15600**	29721	21128	**15600**	16304	30231	21427	16008	**15600**
IV	884508	1262367	906168	877364	875849	1286402	920205	869785	**712574**

(b) Average number of processed jobs.

Class	CS-CM	CS-RM	CS-TDM	CS-BM	FS-CM	FS-RM	FS-TDM	FS-BM	BPS-BM
I	5493	5715	5809	**5820**	5183	5336	5428	5453	5438
II	96171	70046	90622	98151	95666	68494	88991	97645	**107182**
III	**66540**	29575	45705	**66540**	63304	29109	45286	65167	**66540**
IV	2549	1585	2530	2614	2702	1583	2576	2740	3136

(c) Average head waiting time.

Class	CS-CM	CS-RM	CS-TDM	CS-BM	FS-CM	FS-RM	FS-TDM	FS-BM	BPS-BM
I	1827	1758	1727	**1724**	1939	1885	1852	1844	1847
II	103	142	110	101	104	145	112	102	**93**
III	**150**	337	218	**150**	157	343	220	153	**150**
IV	3948	6401	3978	3854	3718	6379	3904	3673	**3207**

Table 5: Performance of partition selection strategies for class-specific workloads. Medium size.

7.2.3. Influence of the application class in the selection strategies

The previous analysis only used mixed workloads. In this section we study to what extent the obtained results are valid for any class of application. For the sake of clarity, we focus the study on workloads of medium size. Results of the experiments are summarized in Table 5 (Best results are highlighted).

For Class I applications (based on collective communications) the best results in terms of job throughput are achieved by the CS-BM strategy. The compact, cubic partitions provided by CS provide an excellent foundation for collective operations. Interestingly, even though BPS-BM result in a better execution speed, this acceleration does not compensate the losses in head waiting time.

Class II applications correspond to 2D virtual topologies. BPS-BM achieves the best throughput, even considering the high cost of waiting until a matching 2D partition becomes available. In most cases the best mapping is CS (note how performance of CS and BM is very similar), although in some cases TDM is chosen.

For applications with 3D virtual topologies (Class III) the best results are achieved using CS-CM, CS-BM and BPS-BM. This result was expected because these applications communicate using a 3D pattern that matches perfectly the partition shapes chosen by CS, efficiently exploited with a consecutive mapping. The results confirm that the best selection is always CS and the best possible mapping is CM, therefore the three strategies mentioned are identical.

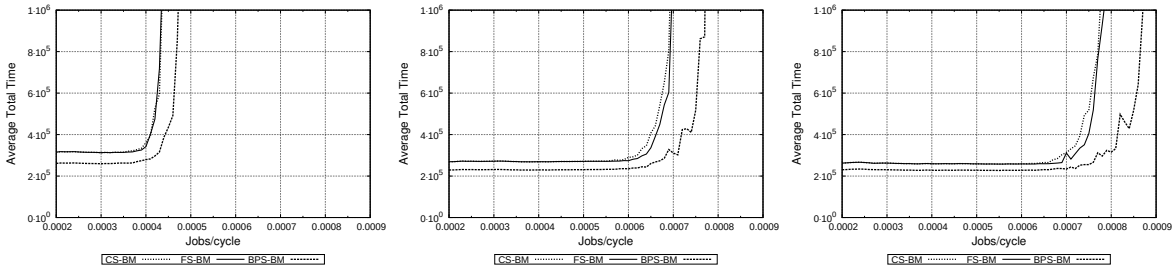
Regarding Class IV (miscellaneous) applications, results are similar to those obtained with the mixed workload: the best performing strategy is BPS-BM. It is interesting to see how, in this case, the FS partition selection strategy is a good choice in terms of reducing head waiting time, but the selected partitions are not optimal for the jobs.

In summary, BPS-BM is an excellent choice of partition selection and mapping strategy for a great diversity of applications. Only for workloads dominated by collective operations CS-CM (or CS-BM) provide slightly better metrics.

7.3. Results for contiguous partitioning with time-spaced arrivals

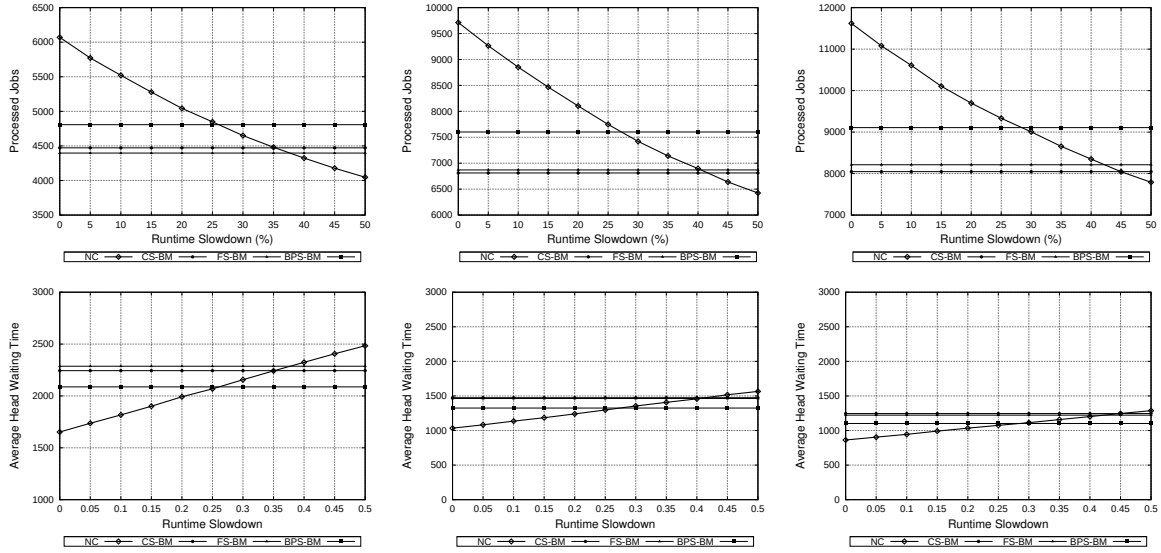
These experiments were carried out using the mixed workload set composed by applications of all classes. For the sake of clarity we will only discuss the results for strategies using best mappings (BM).

We will focus our attention to the total time a job remains in the system, which adds to run time the time it waits in the queue, first waiting for previous jobs to be processed, then waiting until the required



(a) Large size jobs. (b) Medium size jobs. (c) Small size jobs.

Figure 11: Average total time using the CS-BM, FS-BM and BP-BM selection strategies with different levels of load.



(a) Large size jobs. (b) Medium size jobs. (c) Small size jobs.

Figure 12: Number of processed jobs and head waiting time for contiguous and non-contiguous partition selection strategies, for different slowdown factors.

resources are found. Notice that, in this simulation scenario, jobs arrive at a certain time (its timestamp) and, therefore, the total time depends on its characteristics and on those of all of the previously arriving jobs. It is even possible, in lightly-loaded situations, to put a job to run immediately.

In Figure 11 we have depicted the average total time at different levels of applied load (inter-arrival times) for workloads of different sizes. Note how the load accepted by the system before reaching a saturated state is highly dependent on job size: larger workloads saturate the system faster, because finding large partitions is harder and creates higher degrees of fragmentation. Conclusions are not very different from those of the previous section: the best performer strategy is BPS-BM, for all load levels and workload sizes. The benefits in terms of run times provided by this strategy compensate the wait for the "perfect" partition. As the jobs will spend less time running, nodes will become earlier ready for other waiting jobs.

8. Trading off costs and benefits of non-contiguous partitioning

The previous section has compared different contiguous partitioning strategies that can be implemented in a scheduler. However, we did not include in the comparison non-contiguous strategies, something we will perform in this section.

We have demonstrated in Section 6 that allocating jobs non-contiguously slows down their execution, but we also know that it will reduce the time spent at the queue waiting for resources. It would be desirable to carry out experiment sets such as those done in the previous section, but we have a limitation here: there are a myriad of possible non-contiguous partitions for a given job size, and it would be impossible to run INSEE off-line to compute the expected run time for all the corresponding configurations. An alternative would be to include INSEE as an on-line tool in the scheduling simulator, but this would make experimentation impossible due to exceedingly high simulation times.

We have tackled the problem using a different approach. Given an application, we already know (it is tabulated) its best running time in a contiguous partition: that provided by BPS-BM. We also know that running the application non-contiguously will introduce an *unknown* degree of slowdown that would depend on many factors. We can run the scheduling simulator providing this *run time slowdown* as a parameter, in order to have total flexibility to find trade-offs between scheduling gains and run time losses. Using this method, we compare the non-contiguous strategy with CS-BM, FS-BM and BPS-BM contiguous strategies.

In Figure 12 we have plotted the average number of processed jobs in the simultaneous-arrival setup; the corresponding head waiting times are plotted below. Notice the horizontal lines for contiguous strategies, because in these cases we have not applied the slowdown factor (tabulated data is available). Regarding the non-contiguous strategy, we see some obvious results: when applications do not suffer slowdown, the advantages of this strategy are clear: scheduling gains due to the ease of finding collections of available, but scattered, nodes immediately reflects in better system utilization and, therefore, a higher system throughput. However, when applications suffer slowdown due to non-optimal resource assignment, the picture changes. For large jobs, a 25% increase in run time makes BPS-BM a better choice. Even CS-BM and FS-BM are better if slowdown is over 35-38%. For other job sizes (medium and small) the disadvantages of non-contiguous partitioning disappear at slightly higher, but similar, slowdown factors. It is important to remember now how, in Subsection 6.2 we showed that there is no a single, universally valid slowdown factor for non-contiguous partitioning, but a certain degree of slowdown is unavoidable. System utilization has not been plotted in the figures: it reaches an excellent 99% with the non-contiguous strategy, while BPS-BM almost reaches an 80%.

It is important to remark that in this work we have used the simplest non-contiguous strategy: the one that just searches for free nodes anywhere in the system. Other, more complex strategies try to maintain some degree of locality between the partitions assigned to jobs, with the hope of providing applications with a better running environment. These applications should perform somewhere in the middle of the two extremes considered in this work, but the actual mileage of each different proposal should be evaluated on a case-by-case basis.

9. Conclusions and future work

Most current supercomputing sites are built around parallel systems shared between different users and applications. The optimal use of resources is a complex task, due to the heterogeneity in user and application demands.

Supercomputers are expensive to build and maintain, so the general thought is to maintain their utilization as high as possible. However, the efficient use of a parallel computer cannot be measured only by the amount of unused nodes. Other utilization characteristics, such as the job throughput, provide a better view of system's usefulness and user experience.

In this paper we have studied the impact on performance of different locality-aware resource assignment strategies, using contiguous partitioning in 3D tori (although they are equally valid for mesh topologies). Partitioning strategies that search for contiguous resources are expensive in terms of system fragmentation and waiting times, but also are able to accelerate the execution of applications. We have evaluated several alternatives combining partition selection and mapping strategies, measuring their impact not only in application run times but also on the overall scheduling process.

Results show that providing the scheduler with additional information about application characteristics may allow it to select optimally-shaped partitions and to apply the best-known mapping to run the application on it – this is the BPS-BM strategy, able to drastically reduce application run times in such a way that

it more than compensates the additional waiting times. BPS-BM, as such, has a negative effect in terms of system utilization, but this not noticeable to users: in fact it provides excellent results in terms of job throughput.

A non-contiguous strategy offers the opposite picture: system utilization is maximized, reaching values up to 99%. However, when applications suffer from non-optimal resource assignments (something that happens for most applications), the benefits of this utilization, derived from the ease to find free (but scattered) resources, disappear. Locality-aware contiguous policies, such as BPS-BM, are clearly better options if the average slowdown suffered by application is over 25-30%.

The best of the locality-aware strategies discussed in this paper, BPS-BM, makes at best use of only a 80% of system resources, remaining on average a 20% of unused nodes. We can see this as an advantage in terms of power consumption: using strategies such those discussed in [10], BPS-BM can provide a greener, cheaper supercomputer while providing excellent user-visible performance. Alternatively, it would be possible to put those nodes to work by means of job selection policies different from FCFS, such as Backfilling and Shortest Job First, not discussed in this paper. This is a future line of work.

A more detailed study of non-contiguous partitioning would require the combination of the scheduling simulator and the interconnection network simulator. However, as we have explained before, the computational cost of performing all the experimental work necessary for a comprehensive study would be unfeasible. We are working on the development of models of the behavior of applications in non-contiguous environments, in such a way that application run times would be predicted without performing the whole simulation.

Acknowledgements

This work has been supported by programs Saiotek and Research Groups 2007-2012 (IT-242-07) and project IT609-13 from the Basque Government, project TIN2010-14931 from the Spanish Ministry of Science and Innovation, and by the COMBIOMED network in computational biomedicine (Carlos III Health Institute). Mr. Pascual is supported by a doctoral grant of the Basque Government. Prof. Miguel-Alonso is a member of the HiPEAC European Network of Excellence. This work was partially done when Mr. Pascual was visiting the University of Science and Technology of China under grant NICaiA (Nature Inspired Computation and its Applications, PIRSES-GA-2009-247619).

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

[2] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. M. Mackenzie. Comparative evaluation of contiguous allocation strategies on 3d mesh multicomputers. *Journal of Systems and Softtware*, 82(2):307–318, Feb. 2009.

[3] M. A. Bender, D. P. Bunde, E. D. Demaine, S. P. Fekete, V. J. Leung, H. Meijer, and C. A. Phillips. Communication-aware processor allocation for supercomputers: Finding point sets of small average distance. *Algorithmica*, 50(2):279–298, Jan. 2008.

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

[5] B. Broeg, B. Bose, Y. Kwon, and Y. Ashir. Lee distance and topological properties of k-ary n-cubes. *IEEE Transactions on Computers*, 44(8):1021–1030, 1995.

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

[7] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling – a status report. In *Job Scheduling Strategies for Parallel Processing*, pages 1–16, Berlin, Heidelberg, 2005. Springer-Verlag.

[8] M. Kang, C. Yu, H. Y. Youn, B. Lee, and M. Kim. Isomorphic strategy for processor allocation in k-ary n-cube systems. *IEEE Transactions on Computers*, 52(5):645 – 657, may 2003.

[9] V. Lo, K. Windisch, W. Liu, and B. Nitzberg. Noncontiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 8(7):712–726, 1997.

[10] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS '09, pages 205–216, New York, NY, USA, 2009. ACM.

[11] J. Navaridas and J. Miguel-Alonso. Realistic evaluation of interconnection networks using synthetic traffic. In *Proceedings of the 2009 Eighth International Symposium on Parallel and Distributed Computing*, pages 249–252, Lisbon, Portugal, 2009. IEEE Computer Society.

[12] J. Navaridas, J. Miguel-Alonso, J. A. Pascual, and F. J. Ridruejo. Simulating and evaluating interconnection networks with insee. *Simulation Modelling Practice and Theory*, 19(1):494 – 515, 2011.

[13] J. Navaridas, J. Miguel-Alonso, and F. Ridruejo. On synthesizing workloads emulating mpi applications. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –8, Miami, Florida, april 2008.

[14] J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. Optimization-based mapping framework for parallel applications. *Journal of Parallel and Distributed Computing*, 71(10):1377 – 1387, 2011.

[15] J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. A fast implementation of the first-fit contiguous partitioning strategy for cubic topologies. Technical report, University of the Basque Country, 2012.

[16] 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)*, pages 138–156. Springer-Verlag, Rome, Italy, 2009.

[17] V. Puente, C. Izu, R. Beivide, J. Gregorio, F. Vallejo, and J. Prellezo. The adaptive bubble router. *Journal of Parallel and Distributed Computing*, 61(9):1180 – 1208, 2001.

[18] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 edition, 2007.

[19] B. E. Smith and B. Bode. Performance effects of node mappings on the ibm blue gene/l machine. In *Proceedings of the 11th international Euro-Par conference on Parallel Processing*, pages 1005–1013, Berlin, Heidelberg, 2005. Springer-Verlag.

[20] K. Windisch, V. Lo, and B. Bose. Contiguous and non-contiguous processor allocation algorithms for k-ary n-cubes. *IEEE Transactions on Parallel and Distributed Systems*, 8:712–726, 1995.

[21] H. Yu, I.-H. Chung, and J. Moreira. Topology mapping for blue gene/l supercomputer. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006. ACM.

10

A fast implementation of the first-fit contiguous partitioning strategy for cubic topologies

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **00**:1–18
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

A fast implementation of the first-fit contiguous partitioning strategy for cubic topologies.

Jose A. Pascual*, Jose Miguel-Alonso, Jose A. Lozano

Intelligent Systems Group, School of Computer Science. University of the Basque Country UPV/EHU. P. Manuel Lardizabal 1, San Sebastian, Spain 20018.

SUMMARY

In this paper we propose and evaluate IFF, a fast implementation of the first-fit partitioning strategy. It has been devised to accelerate the process of finding contiguous partitions in space-shared parallel computers in which the nodes are arranged forming multi-dimensional cube networks. IFF uses system status information to drastically reduce the cost of finding partitions of the requested shape. The use of this information, combined with the advance detection of zones where requests can not be allocated, remarkably improves the search speed in large networks. An exhaustive set of simulation-based experiments have been carried out to test IFF against other algorithms implementing the same partitioning strategy. Results, using synthetic and real workloads, show that IFF can be several orders of magnitude faster than competitor algorithms.

Received ...

KEY WORDS: Parallel workloads; contiguous partitioning; first fit strategy; scheduling; d-dimensional networks.

1. INTRODUCTION

Scheduling in a supercomputer system refers to the way resources (compute nodes) are selected and assigned to run the parallel jobs submitted to it. These resources must be shared among many jobs belonging to multiple users. Many different policies and strategies can be implemented in order to assign them, trading off aspects such as decision speed and system utilization. We could separate the scheduling process into two steps: first the scheduler decides which one of the awaiting jobs will be executed (FCFS is not the only possible policy) and then a set of resources has to be allocated to run that job on it [1]. The second step only succeeds if the required resources are available and found.

A parallel job is composed of a set of tasks that run concurrently, while synchronizing and interchanging messages if necessary. An interconnection network will make these operations possible: the parallel computer is actually a networked set of compute nodes. We will assume that the system is space-shared among different jobs, meaning that several jobs may be running simultaneously, but a given node only executes a single task of a particular job. A scheduler orchestrates the assignment of compute nodes to job tasks.

The strategy chosen to select the particular partition (set of resources) on which to run a job has a remarkable impact on the execution time of that job. There is significant evidence [2] [3] [4] showing that most applications run faster when they are allocated onto contiguous partitions. This kind of partitions reduces the average distance traveled by the messages interchanged between tasks,

*Correspondence to: Intelligent Systems Group, School of Computer Science. University of the Basque Country UPV/EHU. P. Manuel Lardizabal 1, San Sebastian, Spain 20018. E-mail: joseantonio.pascual@ehu.es

and also the interference caused between the different jobs sharing the system. The negative side of contiguous partitioning is that it may cause severe *external fragmentation*: nodes that are free but cannot be allocated to the selected job because they are not contiguous.

In systems that use contiguous partitioning (this work focuses on d-dimensional meshes and tori), we assume that an incoming job requests, when enqueued, a node set with a particular shape, not just a node count. The scheduler has to locate a partition with the requested shape trying to guarantee a low response time and avoiding *internal fragmentation*. This means assigning exactly the required number of nodes. Some allocation strategies may assign nodes in excess of those requested, that will be unused [5] [6] [7]. The combined effect of both kinds of fragmentation can drastically reduce system utilization.

Many strategies have been developed to search for contiguous partitions in supercomputers [5] [6] [7] [8] [9], most of them focusing on 2D meshes. However, state of the art supercomputers are starting to use higher dimensional networks; for instance, the IBM Blue Gene/Q implements a 5D torus network interconnecting thousands of nodes [10]. In this paper we use a very simple search strategy, *first fit*, and provide an efficient algorithm to implement it. Our proposal, IFF (*Improved First Fit*), is able to find contiguous partitions in cubic networks with any number of dimensions, scaling efficiently to systems formed by millions of nodes. We take for granted that systems of this large size will be used to run simultaneously hundreds, even thousands of jobs, under the control of a scheduling system. In these scenarios the procedure to search within the network for a requested partition could be invoked many times until it is available. Partitioning algorithms must be efficient not only when locating requested partitions, but also determining whether or not they exist in the system.

The first fit search strategy consists of visiting, in a predefined order, the system nodes, looking for candidate partitions to map the request onto. The algorithm stops as soon as a suitable partition is identified. A naive implementation of first fit can be achieved using minimal node status information (is a node already allocated to a running job?) but extensive search. In contrast, IFF uses additional status information to drastically reduce the number of visited candidate places, therefore reducing search time. In the context of this paper, we often use terms such as "visiting a node" or "traversing the system", meaning that we are accessing data structures holding system node status. Also, as we focus on supercomputers built as networked compute nodes, we will use the words "system" and "network" interchangeably.

In IFF, each node (as stated before, the data structure representing the status of that node) stores information not only about its busy/free status, but also about the number of consecutive free nodes in each dimension. This information, combined with the detection of regions on which the allocation of the current request is not possible (*forbidden regions*) is the key of IFF's speed. The time complexity of IFF depends on many factors such as the request size and current network status, that evolves with time. We perform a formal analysis of the cost of the worst case, and also carry out an empirical evaluation of the average cost using synthetically generated workloads.

The actual performance of IFF is compared against three implementations of the first fit strategy: NFF (*Naive First Fit*), ZFF (*Zhus's First Fit* [8]) and RBS (*Right of Busy Submeshes* [9]). Results using synthetic workloads in 2D (mesh) and 5D (torus) networks show that IFF outperforms the other algorithms in most scenarios, being up to two or three orders of magnitude faster. In particular, these good results are obtained in large and heavily used systems, the most demanding ones. Evaluations with actual workloads from the Parallel Workloads Archive (PWA) [11] confirm these results.

The remaining of this paper is organized as follows. Section 2 makes a review of related work describing contiguous allocation strategies and algorithms. Sections 3 and 4 describe the proposed IFF algorithm and the data structures required to implement it. We continue in Sections 5 and 6 with the description of the simulation-based experiments carried out to evaluate IFF, starting with a definition of the workloads used. Section 7 performs an analysis of IFF from the complexity viewpoint. The paper ends in Section 8 with some conclusions and future lines of work.

DOI: 10.1002/cpe

2. RELATED WORK

In this section we discuss several strategies proposed in the literature to solve the contiguous partitioning problem. Most of these proposals have been developed for 2D meshes or tori and, to the best of our knowledge, have not been generalized to topologies with a higher number of dimensions. However, supercomputer manufacturers are building systems with networks with five and even six dimensions. For this reason we also discuss how system partitioning is done in a representative state-of-the-art supercomputer, IBM's Blue Gene/Q (the youngest and most powerful of the Blue Gene family) and the role IFF may play in d-dimensional cubic systems.

2.1. Contiguous allocation strategies for meshes

The Two-Dimensional Buddy System (2DBS), proposed in [5], is a generalization of the one-dimensional binary buddy system, useful only in square meshes whose sizes are powers of 2. Assuming a mesh with size $n = 2^r$ (for some $r \neq 0$, $r \in \mathbb{N}$), 2DBS stores a list of available square submeshes of size $2^{r'}$ for each $0 < r' < r$. An incoming task requesting an (R_x, R_y) partition will be allocated a submesh of size $2^k \times 2^k$, where $k = \lceil log(max(R_x, R_y)) \rceil$. Due to this kind of allocation, 2DBS causes severe internal fragmentation and also external fragmentation, resulting in poor scheduling performance. The allocation and deallocation complexities of this strategy are both $O(log\ n)$.

In order to solve the square size limitation of 2DBS, the Frame Sliding (FS) and Adaptive Scan (AS) strategies were developed. The FS strategy was proposed in [6]: it allows the allocation of a submesh with the exact size and shape of the request. Therefore the use of this strategy will result in the elimination of internal fragmentation. For an incoming task, FS searches for a free submesh examining a sequence of *frames*, sets of nodes with size equal to the request, starting with the lowest leftmost available node. If not all the nodes in the current frame are available, the frame is slid over the mesh to the next candidate frame, with a stride equal to the width of the requested submesh. When a set of rows is visited without success, a vertical stride equal to the partition height is performed. The main drawback of this strategy is that it does not guarantee finding the requested submesh even when it does exist, because of the way the sliding frame advances. In [7] an Adaptive Scan strategy was proposed as an improvement over FS. It uses an adaptive horizontal stride and a vertical stride of one, and allocates a submesh of the exact size and shape of the request. In this case the complexities of the allocation and deallocation phases are both $O(n)$, being n the number of nodes.

In [8] Zhu describes the First Fit and Best Fit (FF and BF) strategies. Both of them are applicable to meshes of arbitrary sizes and shapes, allocating submeshes of the exact size of the request and avoiding internal fragmentation. For its implementation two arrays of booleans are used: the *busy array*, to record the status of the nodes, and the *coverage array*, that is computed for each incoming request and records the set of nodes that cannot be used as the base for allocating the request. FF selects the first submesh large enough for the request, while BF scans all nodes, selecting the minimum-area submesh capable of satisfying the request. The complexity of the allocation and deallocation phases is $O(n)$. Throughout the rest of the paper we will use the acronym ZFF to refer to Zhus's implementation of the first fit search strategy.

More recently the Right of Busy Submeshes (RBS) algorithm was proposed in [9]. RBS is based on the use of a list of busy submeshes. The allocation process in this case can be split into two phases: detection and assignment. In the detection phase, the set of forbidden submeshes that can not hold the request is constructed, based on the current busy list. Any node that is left out of this set is a candidate to serve as base of the requested submesh. In the assignment phase, one node is selected from the set of candidate nodes. The actual assignment can be done following first fist, best fit or even worst fit policies. However, in order to reduce allocation overheads, in [9] the first fit policy was used. RBS does not generate internal fragmentation. The complexity of the allocation and deallocation phases is stated to be $O(m^2)$ and $O(m)$ respectively, being m the number of tasks allocated. However m is not independent from n, the number of nodes. The reason is that n limits the maximum number of jobs that could be allocated simultaneously in the system. Therefore the worst

Prepared using cpeauth.cls
DOI: 10.1002/cpe

case complexity is $O(n^2)$ for the allocation phase and $O(n)$ for the deallocation phase. Recently, RBS has been used as the base to develop a non-contiguous allocation strategy [12].

Notice that, when we compare IFF against competitor algorithms, as we do in Section 6.1, we are using in all cases the same search strategy, which is first fit. Therefore, all the evaluated algorithms detect and return the same partitions when invoked, but their performance differ, being IFF significantly faster. We also want to remark that the implementations of ZFF and RBS used in this study are those provided by their authors.

2.2. Contiguous allocation in Blue Gene systems

IBM's Blue Gene family of supercomputers is specially interesting in the context of this study because its latest implementation, the Blue Gene/Q (BG/Q), uses a 5D torus interconnection network [10]. As described before, IFF can be used to search for contiguous partitions in networks with this number of dimensions, while competitor algorithms do not.

Previous members of the Blue Gene family, series L and P, were built around 3D tori [13] [14]. The use of an interconnection network with higher radix improves bisection bandwidth and communication locality: in the BG/Q each node has ten neighbors at distance one (two in each dimension) instead of just six. The BG/Q is not the only current machine implementing a high dimensional network: the K computer [15] uses a 6D network but, as nodes are arranged using a three dimensional array of 3D cubes, we can not consider it as a "pure" 6D cubic system.

In Blue Gene systems the building block is known as a mid-plane, composed of 512 compute nodes (c-nodes). The c-nodes inside each mid-plane are arranged forming a (8x8x8) 3D torus in BG/L and BG/P systems and a (4x4x4x4x2) 5D torus in the BG/Q. Mid-planes are the minimum unit used by the allocation algorithms of Blue Gene schedulers. This is convenient in terms of reducing partition search time and guaranteeing locality (to some extent), but causes internal fragmentation when the number of requested nodes is not a multiple of a mid-plane [16]. It is important to remark a feature of Blue Gene systems: connections between mid-planes can be reconfigured, in such a way that a given mid-plane may be connected to others that are not physically adjacent.

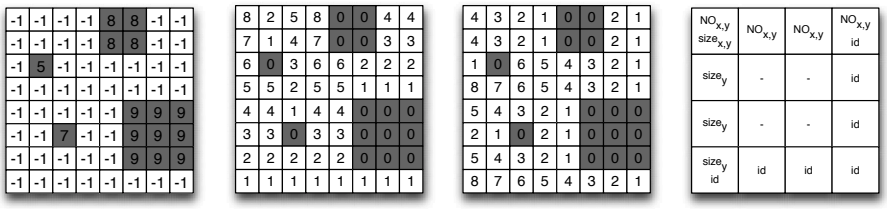
Job allocation in these systems is divided into two successive phases: first the system is scanned for all rectangular and spatially contiguous sets of free mid-planes that match the shape and size of the request. This phase uses a simple first fit strategy that checks for all possible free locations that can accommodate the request [16]. In a second step the algorithm must find the required links to connect the selected mid-planes, forming a mesh or a torus as requested in the job specifications. If those links do not exist, the job must continue waiting. In [17] authors propose imposing less restrictions to the second phase when requests are torus-shaped, arranging the links to form a mesh instead. This allocation is evaluated in terms of its impact on application run times.

In Section 6.2 we will evaluate the performance of IFF in three 5D tori with 4K, 64K and 1M nodes, comparing it against a naive version of first fit (NFF) similar to the one used in Blue Gene systems to find mid-planes. The only status information used by NFF is the busy/free status of nodes. Note that the allocation granularity will be the node instead of the 512-node mid-plane. IFF can be thought as an efficient algorithm to search for contiguous partitions in very large, d-dimensional networks, but also as an efficient algorithm to search for contiguous sets of mid-planes in Blue Gene systems.

3. DATA STRUCTURES USED BY IFF

In this section we introduce the data structures required by IFF. The main ones are Neighborhood Occupancy (NO), that reflects the occupancy of consecutive nodes (free vs. busy nodes) and Forbidden Regions (FR), that mark portions of the network that are not suitable to hold a given request. For the sake of clarity, the structures will be described for the 2D mesh case. Later these structures will be generalized for d-dimensional tori.

DOI: 10.1002/cpe



(a) Status (b) NO_x (c) NO_y (d) Fields to update

Figure 1. Representation of some of the structures used by IFF.

The search for a requested partition involves the selection of nodes fulfilling some conditions. As a network can be seen as a collection of nodes, we can represent the whole system as a 2-dimensional matrix, $network[M_x][M_y]$ that contains information about each node. We use matrix notation where x values make reference to rows – vertical axis and y values to columns – horizontal axis. When required, the dot operator will be used to access the fields (data structures) associated to a given node. The per node status information contains four fields:

- *Status*: It represents the current status of the node. A -1 status means that the node is free, while a positive value means that the node is busy (used by a running job). In the later case, the status value is the identifier (id) of the job as assigned by the scheduler (see Figure 1(a)).
- *Partition Size*: When a node is busy, the per-dimension size of the partition is stored here. This information will be used in order to reduce the number of nodes to visit during the search, stepping over assigned partitions without checking them.
- *Neighborhood Occupancy (NO)*: This field contains the number of nodes in each dimension until the first busy node. For the 2-dimensional case it is composed by two subfields NO_x and NO_y (Figures 1(b) and 1(c) respectively).
- *Forbidden Region (FR)*: This field represents the membership of a node to a forbidden region. It can take positive values or zero (when the node does not belong to any of these regions). This value represents the number of nodes that composes the region in one dimension, and it will be used to jump over them without checking for suitable nodes. In practice, only some nodes are marked with the size of the forbidden region because the remaining nodes will be overpassed.

Effectively, only some nodes of each partition, those in the periphery, are required to hold status information as shown in Figure 1(d), reducing the number of updates required to assign a partition to a job.

3.1. Neighborhood Occupancy (NO)

Each node has associated two NO values, one per dimension. These values represent the number of nodes until the first busy one as shown in Figures 1(b) and 1(c). Using these values we can state the conditions that a node must fulfill to serve as base node for a request. We call *base node* to the upper-left node of a partition.

Definition 1
Given a request (R_x, R_y), we say that node $N_{i,j}$ is a *candidate node* for this request when the following two conditions are fulfilled:

$$N_{i,j}.NO_y \geq R_y \tag{1}$$

$$N_{i,j}.NO_x \geq R_x \tag{2}$$

These two conditions are necessary but not sufficient for $N_{i,j}$ to be a base node for the request.

DOI: 10.1002/cpe

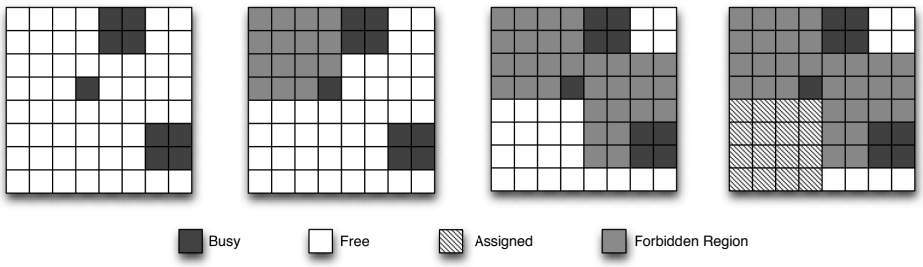


Figure 2. Example of the allocation of a (4x4) request and the forbidden regions detected in the process.

Definition 2
Given a request (R_x, R_y), we say that a node $N_{i,j}$ is a *base node* for the request if it is a candidate node (as stated in Definition 1) and also fulfills the following condition:

$$N_{k,j}.NO_y \geq R_y \text{ for } k = i+1, \ldots, i+R_x-1 \tag{3}$$

IFF differentiates between candidate and non-candidate nodes, skipping the latter.

3.2. Forbidden Regions (FR)

We can define the forbidden regions as subsets of nodes that cannot accommodate the current request. In other words, they cannot serve as base nodes. These subsets form submeshes inside the network and are dynamically created during the allocation process of a request. These regions are created using the NO values stored in free nodes and the *Partition Size* values stored in busy nodes. Once a node is identified as candidate, but discarded as base node, a forbidden region will be created. Initially, the forbidden region is determined by the rectangle that goes from the candidate node to the last node that fulfills Condition 3. This last node is in the same row of the last node that fulfills Condition 2. The column is given by the NO_y value of the node that does not fulfill Condition 3. This rectangle is later combined with the information about the partition size. In Figure 2 we have represented an example of the forbidden regions detected for a request of size (4×4).

4. THE IMPROVED FIRST FIT ALGORITHM

In this section we provide a detailed description of the IFF partitioning algorithm. We divide it into two phases: (1) the *allocation* phase that checks for the availability of the requested partition and, in case it exists, allocates the jobs' tasks onto the assigned nodes, and (2) the *deallocation* phase that releases the nodes of the assigned partition when job execution finishes. The proposed algorithm works as NFF and ZFF do: it sets an order on the nodes, and then they are checked in this previously established order. Once the algorithm finds a base node to allocate the requested partition, it stops the search. Otherwise, the algorithm finishes when all the nodes have been checked. The key point of IFF is the use of the previously described data structures to avoid exhaustive availability test throughout all system nodes.

4.1. Allocation Phase

This phase starts determining if the requested partition exists (detection stage). In such case, the base node to allocate the partition is returned; otherwise the allocation fails. A second *assignment* stage takes as input the base node and the per-dimension partition size, and assigns the job's tasks to the allocated nodes, updating the status structures as required.

4.1.1. Detection stage. Given the current request (R_x, R_y), IFF starts searching for a suitable node $N_{i,j}$ to be the base node for the allocation. The search is performed following a row-major order.

DOI: 10.1002/cpe

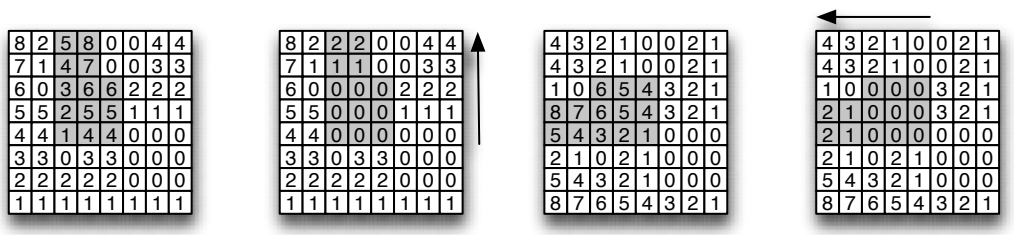
As stated in Section 3.1, a node $N_{i,j}$ must fulfill the conditions of the definitions 1 and 2 to serve as base node. Once a node of this type is found, as the allocation strategy is first fit, the search is stopped. In case the current node is candidate but it cannot be a base, the corresponding forbidden region is marked and the search continues. Otherwise, if the current node cannot be candidate, the algorithm will use the stored information to jump over some nodes. These jumps can be done in two situations:

1. If the current node belongs to an already assigned partition, the algorithm will skip over it, excluding from checking those nodes assigned to the partition in the current row. The required information to perform the jump is stored in the *Partition Size* field of the node.
2. When the current node is free but it does not fulfill Condition 1 there are two possible scenarios: (a) The condition is not fulfilled because of the presence of an assigned partition; in this case the algorithm will perform a jump with value NO_y plus the y-size of the assigned partition, (b) the remaining nodes in the current row are not enough to accommodate the request; in this case, the algorithm will proceed to the next row.

If the current node does not fulfill Condition 3, a forbidden region will be created. Regarding these regions, they are created dynamically while checking the nodes. When IFF decides that a set of nodes belong to a forbidden region, these nodes are marked correspondingly. When testing the suitability of a node as base node, IFF checks if, in a previous step, it has been included in a forbidden region. If this is the case, the algorithm will jump over that node and other marked nodes, without further checks. Once the detection stage ends, the forbidden regions have to be deleted because they depend on the dimensions of the requested partition and therefore of no use in subsequent allocations.

4.1.2. Assignment stage. If the detection stage is successful, we have to allocate the job to the assigned partition and update the status information of all the nodes affected by the allocation. This includes the *Status* and *Partition Size* of the allocated nodes, and also the NO values of the affected nodes. As stated in Section 3, it is not necessary to update all the nodes inside the assigned partition (see Figure 1(d)). Performing the update in this way will result in an important reduction of the number of update operations, specially for large partitions.

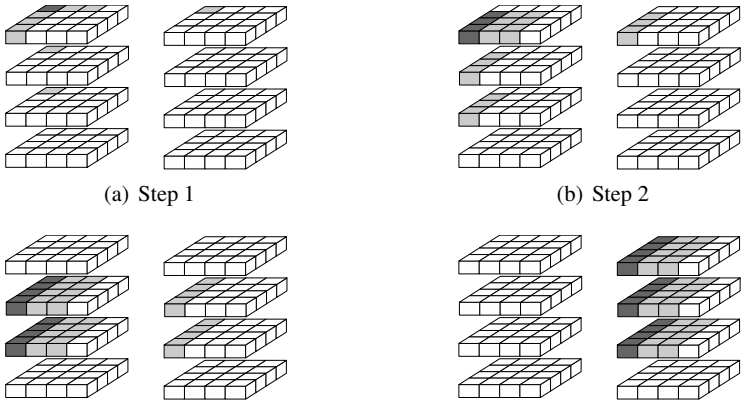
In Figure 3 we can see an example of updating NO values. For the sake of clarity we update all the nodes inside the assigned partition. We have to set both NO_x and NO_y to zero in the allocated nodes, update of the NO_x values of the affected nodes to the left (starting from value 1 increasing in 1 unit until a busy node is reached) and those of NO_y upwards (again, starting from value 1 increasing in 1 unit until a busy node is reached). In the example we have allocated a partition with base node $N_{3,3}$ and size (3×3). The nodes affected by this allocation will be $N_{3,2}$, $N_{3,1}$, $N_{4,2}$, $N_{4,1}$, $N_{5,2}$, $N_{5,1}$ in the Y dimension and $N_{2,3}$, $N_{2,4}$, $N_{2,5}$, $N_{1,3}$, $N_{1,4}$, $N_{1,5}$ in the X dimension. In general, possible nodes affected by the allocation of $N_{i,j}$ are $N_{i',j'}$ such that $i' = 1, \ldots, i-1$ and $j' = 1, \ldots, j-1$.



(a) NO_x values before (left) and after (right) allocation. (b) NO_y values before (left) and after (right) allocation.

Figure 3. Example of nodes that will be updated when allocating a (3×3) partition from node $N_{3,3}$. Arrows indicate the order in which the nodes will be updated.

DOI: 10.1002/cpe



(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4

Figure 4. Steps required to detect a (4x3x3x2) request in a (4x4x4x2) 4D torus using $N_{0,0,0,0}$ as candidate node. IFF checks the black nodes at each step and detects the gray nodes as available.

4.2. Deallocation Phase

The deallocation phase involves the release of the partition assigned to a job when it finishes running. As in the assignment stage, the algorithm has to update the status of the nodes that belonged to the released partition, as well as the NO values of the nodes affected by the deallocation. The procedure is almost the same as in the assignment stage. The only difference is that the update of the NO values will be performed starting inside the partition just released, considering the NO value of the previous node in each dimension.

4.3. Extension of IFF to d-dimensional cube topologies

In the previous sections we have described IFF for 2D meshes, but it can be generalized to d-dimensional torus topologies. Small changes when dealing with the NO structures are required in the initialization process, in the allocation phase, and in the deallocation phase.

When extending IFF to tori changes are required to take into account the wrap-around links. As NO values represent the number of free nodes until the first busy node in each dimension, the initial NO values will be, in all nodes, equal to the number of nodes per dimension. In the assignment stage and the deallocation phase, updates do not stop when reaching the extreme nodes in each dimension, but must continue through the wrap-around links.

To extend IFF to d-dimensional topologies we had to add the required fields to each node in the structure representing the network: *Partition Size* and NO for each dimension. Small changes are required to explore the additional dimensions but the main mechanisms of the algorithm remain the same.

In Figure 4 we have represented an example of the detection of a (4x3x3x2) request in a (4x4x4x2) 4D torus using $N_{0,0,0,0}$ as candidate node. In a first step (Figure 4(a)), IFF uses the NO information of the candidate node to detect if some of the required nodes are available (gray nodes). As they are free ($NO_1 \geq 4$ and $NO_2 \geq 3$ and $NO_3 \geq 3$ and $NO_4 \geq 2$), IFF will continue checking the availability of the remaining nodes using their NO information. In the following step (Figure 4(b)), IFF checks if the NO values of the black nodes fulfill the conditions of the request ($NO_2 \geq 3$ and $NO_3 \geq 3$ and $NO_4 \geq 2$). Note that at each step, less network dimensions are considered and the useful information provided by the NO values is reduced. The process continues the same, checking in steps 3 and 4 (Figures 4(c) and 4(d)) if the nodes fulfill the requirements ($NO_3 \geq 3$ and $NO_4 \geq 2$) and ($NO_4 \geq 2$) respectively. If during this search a busy node is found, the process stops and the corresponding forbidden region will be created. The assignment stage and the deallocation phase remain the same, updating the affected nodes in each dimension.

DOI: 10.1002/cpe

5. EXPERIMENTAL SETUP

In this and the next section we describe a collection of experiments designed to compare IFF against other algorithms implementing first fit, namely NFF, ZFF [8] and RBS [9]. Each of them is evaluated simulating a supercomputing system fed with different workloads. In some cases, these workloads are real (taken from the Parallel Workload Archive (PWA) [11]), and in some other cases they are synthetic, generated using the Lublin model [18]. Synthetic workloads provide more flexibility to fine-tune their characteristics. In both cases, we are interested in measuring the average time that each algorithm expends allocating each job. All times have been measured running single-threaded implementations of the evaluated algorithms on an Intel Xeon X5650 2.67GHz processor.

5.1. Synthetic workloads

We use a model developed by Lublin [18] to generate synthetic workloads: sequences of jobs emulating the actual load received by a system scheduler. Workloads generated with this model are highly customizable by means of a set of parameters that control the jobs' characteristics using different probability distributions:

1. Interarrival time: Determines the timestamp of jobs arriving to the system. The model calculates this value in two stages: first, the number of jobs arriving at every time interval and then, for each job, its specific arrival time. Both computations are performed using gamma distributions, with different parameters.
2. Job size: Number of parallel tasks of a job. It is modeled by means of a two-stage-uniform distribution.
3. Execution Time: The running time of a job. It is modeled using a hyper-gamma distribution.

Given a workload, a metric is suggested in [19] to measure the *load* managed by the scheduler. Selecting workloads with different values of this metric allows us to check our proposals in different scenarios of system utilization. The *load* is computed as follows:

$$load = \left(\frac{\sum_{j \in J} size_j \times runtime_j}{P \times (T_{end} - T_{start})} \right) \tag{4}$$

where P is the number of processors, J is the set of jobs between T_{start} and T_{end}, T_{end} is the time of the last arrival and T_{start} is the time of the last termination within the first 1% of the jobs. This 1% of first arriving jobs and the jobs that terminate after the last arrival are removed, in order to reduce warm up and cool down effects.

We have worked with three networks sizes: 4K, 64K and 1M nodes. For each of these sizes, we have defined five different average job sizes, which are relative to system size; see the summary of combinations of system and job sizes in Table I, in which Set 1 represents workloads formed by small jobs and Set 5 represents workloads with large-size jobs. In all cases the *maximum* partition size is equal to one quarter of the network size. Finally, the job interarrival times for these combinations have been adjusted to generate target loads of 0.1, 0.4 and 0.7.

System Size	Max Job Size	Set 1	Set 2	Set 3	Set 4	Set 5
4K nodes	1K nodes	5.79	8.71	20.93	67.63	125.58
64K nodes	16K nodes	45.26	78.66	187.80	650.70	1251.29
1M nodes	256K nodes	516.03	901.10	2163.40	7510.96	14704.51

Table I. Network and average job size combinations of the synthetic workloads used in the experiments.

5.2. Real workloads

As stated before, in this work we also consider actual workloads extracted from the PWA (Parallel Workload Archive, [11]). These logs contain information about the system on which they were

Prepared using cpeauth.cls
DOI: 10.1002/cpe

extracted, in the SWF format (Standard Workload Format) [20]. From all the workloads available in the PWA we have selected six, obtained from different large-scale parallel computers. In Table II we summarize some characteristics of each of them. For the first four workloads we have simulated a (64×64) 2D mesh system and a $(8 \times 8 \times 4 \times 4 \times 4)$ 5D torus. For the SHARCNET and LLNL Atlas logs we have used a larger (128×128) 2D mesh system and a $(8 \times 8 \times 8 \times 8 \times 4)$ 5D torus system.

Workload	Average Job Size	Load	Number of Jobs	Simulated Systems
LLNL-Thunder	45.07	0.76	128,662	$(64 \times 64) - (8 \times 8 \times 4 \times 4 \times 4)$
LLNL-uBGL	736.18	0.28	112,611	$(64 \times 64) - (8 \times 8 \times 4 \times 4 \times 4)$
SDSC-DS	63.21	0.27	96,089	$(64 \times 64) - (8 \times 8 \times 4 \times 4 \times 4)$
SDSC-Blue	43.00	0.86	243,314	$(64 \times 64) - (8 \times 8 \times 4 \times 4 \times 4)$
SHARCNET	3.15	0.89	1,195,242	$(128 \times 128) - (8 \times 8 \times 8 \times 8 \times 4)$
LLNL-Atlas	429.22	0.41	60,332	$(128 \times 128) - (8 \times 8 \times 8 \times 8 \times 4)$

Table II. Characteristics of the workloads extracted from the Parallel Workload Archive

6. EVALUATION OF IFF

In this section we present the results obtained when comparing the proposed IFF algorithm against ZFF and RBS in 2D meshes and against NFF in 5D tori. The reason for not using ZFF and RBS in 5D is that they were not defined for high dimensional networks and their adaptation to such topologies, if possible, is beyond the scope of this work. We focus on allocation times because, as all of the algorithms implement the same strategy (first fit) the remaining figures of merit (for example, average waiting times and system utilization) are exactly the same. All experiments have been carried out for different sizes of 2D meshes and 5D tori, and using FCFS [1] as scheduling strategy.

Note that an allocation would require a single invocation to the partitioning algorithm if the scheduler finds the requested partition the first time it looks for it. Otherwise, it will require several invocations (indicating that the request cannot be satisfied) before the final, successful one. In this section, we define the allocation time for a request as the combined time of all invocations to the partitioning algorithm required to fulfill the request.

6.1. Synthetic workloads in 2D meshes

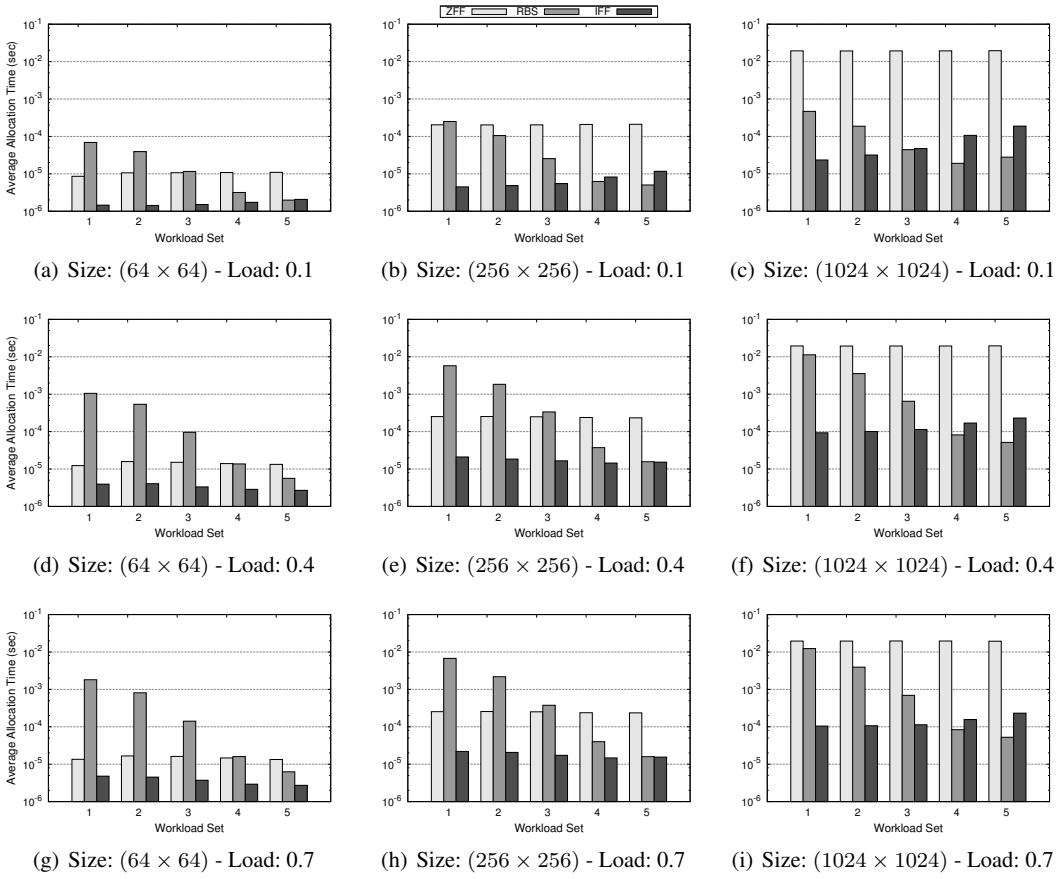
The first set of experiments measure average time required to perform an allocation, for the different synthetic workloads at different levels of input load, on a system arranged as a 2D mesh. All the collected results are summarized in Figure 5.

As we can see, ZFF is always slower than IFF, in all scenarios. Regarding RBS, it outperforms IFF in only a few cases: large systems running large jobs (independently of the input load, Figures 5(c) 5(f) and 5(i)), and lightly loaded medium-size systems running large jobs (Figure 5(b)).

In general, RBS performs well when the number of jobs running in the system simultaneously is low, because in these situations managed busy lists are short. Lightly loaded systems fall in this case, and the same happens when average job size is large because, evidently, only a few large jobs can be running in the system. Note how RBS is very sensitive to average job size (the longer this size the better the performance), because increasing this size means a reduction in the number of allocated jobs. It is important to remark that those scenarios in which RBS performs well are not desirable from the point of view of overall system efficiency: these are situations of low system utilization, either because the input load is low, or because of the high levels of external fragmentation that results from the allocation of large jobs onto contiguous partitions.

Under any other situation, IFF is the best performer. It is able to efficiently deal with large collections of running jobs. IFF run times grow with system size, but are not very sensitive to input load and average job size. This is a very desirable property, because allocation times are under

DOI: 10.1002/cpe



(a) Size: (64×64) - Load: 0.1 (b) Size: (256×256) - Load: 0.1 (c) Size: (1024×1024) - Load: 0.1

(d) Size: (64×64) - Load: 0.4 (e) Size: (256×256) - Load: 0.4 (f) Size: (1024×1024) - Load: 0.4

(g) Size: (64×64) - Load: 0.7 (h) Size: (256×256) - Load: 0.7 (i) Size: (1024×1024) - Load: 0.7

Figure 5. Average allocation times in seconds achieved by ZFF, RBS and IFF for 2D meshes of different sizes. Note the logarithmic scale in Y axis.

control even for very large systems (more than one million nodes). The management of a large number of jobs is not a problem for IFF, just the opposite. In a system with many jobs already allocated IFF will be fast because it will skip checking many nodes during the detection stage, and will require a small number of updates during the assignment stage.

We can summarize this section stating that, except for workloads with very large jobs, IFF performs better (up to two orders of magnitude faster) than RBS. Regarding ZFF, in all situations, for any system size and workload, IFF is between one and three orders of magnitude faster.

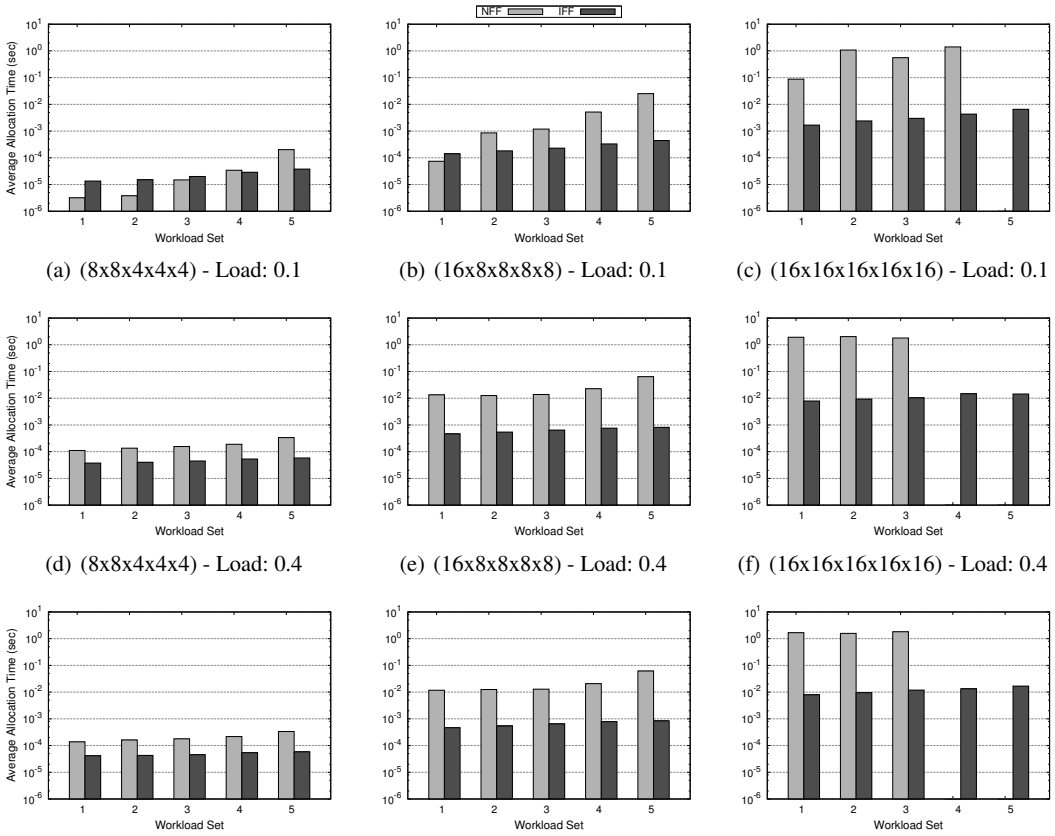
6.2. Synthetic workloads in 5D tori

We extend now the evaluation of IFF to tori of higher dimensionality. We have chosen 5D because this is the number of dimensions of the BG/Q system. However, we have not used IFF with the mid-plane as allocation unit: this will be the node, as in the previous section.

For this study we cannot compare IFF against RBS and ZFF because, as discussed before, they are defined for 2D networks only. Instead, we will compare our proposal against NFF, a naive implementation of the first fit strategy. Results of the experiments are summarized in Figure 6. Note that some results for NFF are not represented in the figures because measured allocation times are in excess of 10 seconds.

The most important remark is that, even in the worst situation, allocation time of IFF is less than 0.015 seconds in systems with more than a million nodes. For these large systems, NFF is two (or more) orders of magnitude slower. Only in scenarios of very low utilization running small jobs the

Prepared using cpeauth.cls
DOI: 10.1002/cpe



(a) (8x8x4x4x4) - Load: 0.1 (b) (16x8x8x8x8) - Load: 0.1 (c) (16x16x16x16x16) - Load: 0.1

(d) (8x8x4x4x4) - Load: 0.4 (e) (16x8x8x8x8) - Load: 0.4 (f) (16x16x16x16x16) - Load: 0.4

(g) (8x8x4x4x4) - Load: 0.7 (h) (16x8x8x8x8) - Load: 0.7 (i) (16x16x16x16x16) - Load: 0.7

Figure 6. Average allocation times in seconds achieved by NFF and IFF for 5D tori of different sizes. Note the logarithmic scale in Y axis.

simplicity of NFF results in smaller allocation times – but these scenarios should not be common when making a good use of an expensive supercomputing infrastructure.

When working with 5D tori, IFF shows again that it is slightly affected by load and job size. Comparing results with those of a 2D mesh with the same number of nodes, allocation times are one order of magnitude worse. This is due to the complexity added by dealing with wrap-around links and a larger number of dimensions. This will be further discussed in Section 7.

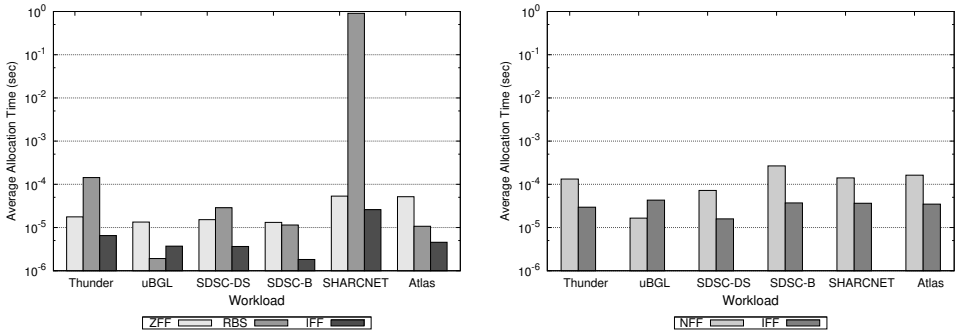
6.3. Performance with real workloads

Experiments with synthetic workloads in 2D meshes gave us information about the situations in which IFF performs the best, and those in which RBS outperforms our proposal. In this section we confirm these insights using real workloads taken from the PWA. We also evaluate IFF against NFF in a 5D torus using these workloads.

We summarize in Figure 7(a) allocation times in 2D meshes (each of them corresponds to the average time of 20 simulation runs) for the six workloads and system sizes described in Table II. Note that system sizes are similar in size to those considered "small" in the previous experiments with synthetic workloads.

Results show that IFF outperforms RBS for all cases but the LLNL-uBGL workload. This workload is composed of very large-size jobs in a system with low input load: the ideal situation for RBS. This scenario is similar to that represented in Figures 5(c), 5(f) and 5(i). In all other scenarios, with higher loads and not-that-large jobs, IFF excels, being on average almost one order of magnitude faster than RBS. Special remark has to be done to the results achieved using the

DOI: 10.1002/cpe



(a) 2D mesh (b) 5D torus

Figure 7. Average allocation time in seconds with real workloads from PWA. Note the logarithmic scale in the Y axis.

SHARCNET workload. In this case IFF outperforms RBS being more than four orders of magnitude faster: in this workload the average job size is very small, thus the number of jobs allocated simultaneously is high.

Regarding the ZFF algorithm, results confirm that IFF achieves better allocation times in all situations.

When using 5D tori networks the analysis is basically the same. We have summarized the results in Figure 7(b). IFF performs better than NFF for all the workloads but LLNL-uBGL. In this case, the system has 4096 nodes, thus it is only possible to run, on average, 8 jobs concurrently. In all of the other situations, with more jobs sharing the system, IFF outperforms NFF, in agreement with the experiments performed with synthetic workloads.

7. COMPLEXITY ANALYSIS OF IFF

In this section we assess IFF from the time complexity point of view. Note that the space complexity of IFF analysis is trivial from the explanations of the data structures in Section 3. As all of them are static, (no dynamic allocation is involved) the complexity is $O(N)$, being N the number of system nodes.

First we analyze the detection stage, the one that identifies the existence of the requested partitions. For this stage, we derive a worst-case analysis and, in addition to that, we present an empirical study aiming to determine the average behavior of this stage. This average-behavior study is necessary because, even though dealing with worst-case situations is expensive ($O(n)$), they are pathological cases very unlikely to appear. Then we analyze the second stage of the allocation phase, which is equal to that of the deallocation phase. For these analyses we consider requests for partitions with size $(R_1 \times R_2 \ldots \times R_d)$ on a d-dimensional torus with dimensions $(M_1 \times M_2 \times \ldots \times M_d)$. Although the complexity analysis is carried out for tori, it is also valid for d-dimensional meshes.

Throughout this section we will compute complexity in terms of metrics NVN (*Number of Visited Nodes*), which measures the number of nodes checked during the detection stage, and NUN (*Number of Updated Nodes*), which measures the number of nodes updated in the assignment stage or the deallocation phase.

7.1. Detection Stage

7.1.1. Worst case. The analysis of the worst case must be performed taking into account the size of the requested partition and the status of the system (busy and free nodes). Figure 8 represents a (8×8) 2D torus and a $(4 \times 4 \times 4)$ 3D torus in which half the nodes have been allocated to 32 different jobs of size one. In the 2D torus case, a request shaped $(k \times 1)$ represents the worst possible

DOI: 10.1002/cpe

situation, that is, the one that requires the largest number of nodes being visited. For a 3D torus, this situation arises with requests shaped $(k \times 1 \times 1)$. We now compute the cost for these pathological cases.

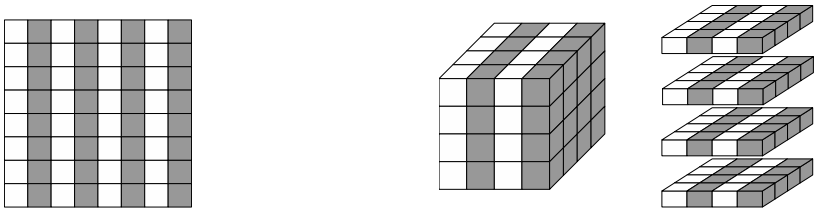
As we can see in Figure 8(a) representing a 2D torus, in each row we check all the nodes ($N_{i,j}$ with j odd) and skip the busy nodes ($N_{i,j}$ with j even). These checks must be performed over all of the rows. In the case of the 3D torus, represented in Figure 8(b), the situation is similar: we must perform the same number of checks on each plane along the third dimension. Generalizing to a d-dimensional torus with dimensions $(M_1 \times M_2 \times \ldots \times M_d)$ and half the nodes busy with this kind of alternating pattern, the NVN to detect a request shaped $(k \times 1 \times \ldots \times 1)$ is:

$$(\frac{M_1}{2} \times M_2 \times \ldots \times M_d) = \frac{1}{2} \prod_{i=1}^{d} M_i \tag{5}$$

Therefore, the worst case complexity of IFF in a network of shape $(M_1 \times M_2 \times \ldots \times M_d)$ is:

$$O(M_1 \times M_2 \times \cdots \times M_d) \tag{6}$$

In other words, the worst case complexity of IFF is in the order of the number of system nodes.



(a) 2D torus (b) 3D torus

Figure 8. Network status in which the strategy achieves the worst performance. For the sake of clarity the wrap-around links are omitted.

7.1.2. Average behaviour. As we stated before, the system configuration determining the worst case for IFF is very unlikely to appear, and cannot be considered as representative of the scenarios in which IFF would actually operate. We are interested in estimating the behavior of IFF in more representative cases, that will provide a meaningful average behavior. We have done this using simulation, measuring the number of nodes visited by IFF (NVN) for different request sizes and input loads. Regarding system configurations, we focus on large-size networks (1M nodes) arranged as tori with different numbers of dimensions (from 2D to 5D).

The methodology used to perform the measurements is as follows. We use the workloads defined in the last row of Table I (the one for systems with 1M nodes). As input loads, we use the same used in Section 6 and a new one, that we call *MAX*, that represents a scenario in which all the jobs arrive to the system simultaneously (inter-arrival times are 0). This covers the whole spectrum of load, from very light to extremely high. The obtained values of NVN are summarized in Table III. Notice that there are two values of NVN for each (topology, load, workload set) tuple. The first ones measures the cost of the detection stage when allocation is successful. The second one is for those cases in which the whole network is traversed while looking for a suitable partition, but it does not exist and, therefore, the detection is unsuccessful. In networks with high degrees of external fragmentation, as happens in some of the configurations (in particular, those with large-size jobs) the NVN for unsuccessful detections becomes critical, because this stage of IFF is invoked many times before a request is actually granted.

When analyzing the impact of input load on NVN, we can observe in the table how, for successful detections, cost increases with the load. This is because, in lightly loaded systems, a requested partition would be found easily. We observe the opposite behavior for unsuccessful detections. These

DOI: 10.1002/cpe

Topology	Load	Set 1		Set 2		Set 3		Set 4		Set 5	
		Successful	Failed	Successful	Failed	Successful	Failed	Successful	Failed	Successful	Failed
1024x1024	0.1	137.66	0.00	106.66	9.02	92.71	12.55	104.98	18.98	109.26	23.88
	0.4	387.70	36.66	265.87	28.03	167.07	23.57	129.10	28.19	122.26	25.51
	0.7	439.99	33.91	295.03	26.96	175.44	25.14	130.38	26.14	124.33	27.35
	MAX	468.51	26.07	297.37	23.99	176.05	23.81	130.58	27.80	126.26	28.20
128x128x64	0.1	158.28	0.00	142.36	0.00	154.55	0.00	207.70	157.47	296.69	395.68
	0.4	575.53	130.58	484.39	109.05	330.28	102.04	272.74	171.68	353.96	160.61
	0.7	751.01	87.34	541.94	81.04	379.04	81.54	335.24	113.78	356.03	150.38
	MAX	796.17	67.30	569.38	77.52	383.14	80.34	337.51	112.59	359.80	158.04
32x32x32x32	0.1	280.35	381.98	286.49	0.00	376.42	1539.45	615.55	1808.51	848.20	1342.92
	0.4	823.68	422.96	719.33	391.54	675.08	435.08	821.54	614.09	1002.30	649.03
	0.7	852.13	427.30	749.40	348.66	686.23	460.54	828.98	591.66	1001.16	639.37
	MAX	903.65	430.23	747.43	367.12	677.96	439.74	824.05	587.64	1004.41	660.36
16x16x16x16x16	0.1	393.14	566.14	446.66	4174.07	628.21	5878.54	1082.44	3860.13	1527.48	3685.07
	0.4	995.74	1359.06	1019.63	1177.73	1067.67	1091.72	1507.93	1711.89	1839.89	1755.58
	0.7	1047.11	1106.71	989.38	1151.23	1084.62	1132.15	1488.63	1688.27	1846.59	1859.07
	MAX	1068.99	1008.59	992.97	1056.60	1075.01	1233.24	1523.50	1673.62	1897.37	1803.74

Table III. Results of the experiments for measuring the Number of Visited Nodes during the detection stage using four large networks with 1M of nodes at different input loads.

happen in systems already populated with running jobs and IFF is able to rapidly check the network status taking advantage of the data structures it manages, including the forbidden regions.

It is worth noticing how increasing network dimensionality (for the same number of nodes) adds difficulty to the detection of requested partitions. However, even in the worst of the measured cases, NVN is below 6000 nodes (0.6% of the total number of nodes), and this corresponds to very lightly loaded systems. This figure lowers to less than 2000 (0.2%) for medium-high input loads. The reason behind this behavior is the way IFF takes advantage from forbidden regions. For example, if we had an allocated job of 2K nodes in a partition of shape (64×64) in a 2D system, the number of nodes IFF could jump over would be 64. In a 3D system with a $(16 \times 16 \times 16)$ job (notice that job size is the same), this number would be reduced to 16. We could continue this reasoning to find values of 8 and 4 skipped nodes for 4D and 5D networks with partitions shaped $(8 \times 8 \times 8 \times 8)$ and $(8 \times 8 \times 4 \times 4 \times 4)$, respectively. Therefore, the higher the dimensionality, the lower the benefit provided by forbidden regions.

7.2. Assignment stage and deallocation phase

Finding the requested partition is only the first stage of the allocation phase. It has to be followed by an assignment stage in which IFF's structures are updated to reflect the new status of the system. The deallocation phase also require updating the same data structures, but when a job finishes. The worst case complexity in both cases is the same, although worst case scenarios differ. Cost is measured in terms of the number of nodes in which NO values are updated (NUN).

The complexity of the assignment stage for a partition shaped $(R_1 \times R_2 \ldots \times R_d)$ is:

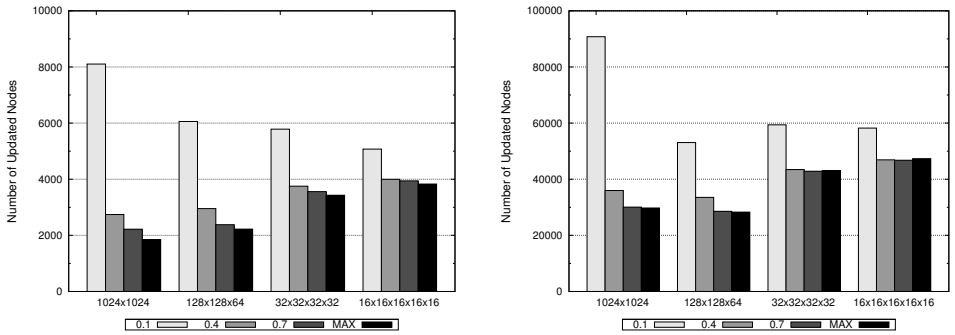
$$O(max\{(M_1 - R_1 + 2) \times \cdots \times R_d, \cdots, R_1 \times \cdots \times (M_d - R_d + 2)\}) = O(\prod_{i=1}^{d} M_i) \qquad (7)$$

In this stage we do not have to update all the nodes inside the detected partition: it is sufficient to update the nodes located in the boundaries of the partition. Due to this, the worst possible scenario arises when the system is empty and the partition being allocated has this shape:

$$(R_1 \times R_2 \ldots \times R_d) \text{ and } \forall j \neq i \; R_i \leq 2 \text{ and } R_j = M_j \qquad (8)$$

The deallocation phase cannot make use of this optimization, requiring the update of all the nodes within the released partition. The worst case situation for this phase involves the update of all network nodes. Therefore, the complexity for a partition shaped $(R_1 \times R_2 \ldots \times R_d)$ is:

DOI: 10.1002/cpe



(a) NUN using the smallest jobs (Set 1) (b) NUN using the largest jobs (Set 5)

Figure 9. Representation of the Network Utilization and the Number of Updated Nodes achieved by 1M nodes networks using the smallest and the largest jobs (Sets 1 and 5).

$$O(max\{M_1 \times R_2 \times \cdots \times R_d, \cdots, R_1 \times R_2 \times \cdots \times M_d\}) = O(\prod_{i=1}^{d} M_i) \tag{9}$$

The worst possible scenario for deallocation arises when releasing a job which is the only one running in the system in a partition with shape:

$$(R_1 \times R_2 \ldots \times R_d) \text{ and } \forall j \neq i \; R_i \leq M_i \text{ and } R_j = M_j \tag{10}$$

Notice how these expressions reflect pathological scenarios with almost empty systems, which are unlikely to happen in production supercomputers. IFF updates nodes advancing, in each dimension, until it finds a neighboring job, something that happens more easily when the network is more occupied. Therefore NUN should be smaller for higher input loads. This behavior has been observed experimentally, and results are represented in Figure 9. We only include data for workload sets 1 and 5 because results with the remaining sets show the same behavior. As expected, for lightly loaded systems, independently of job size, the NUN required to perform an allocation is higher than for medium and heavily loaded systems.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed, discussed and evaluated a novel algorithm that implements the first fit partitioning strategy, with the purpose of finding contiguous partitions for parallel jobs running in a space-shared supercomputer. IFF uses status information to greatly accelerate allocation operations, by skipping over network portions on which allocation is impossible.

The computational cost of IFF depends on many factors such as current request size, network occupancy status and average size of the allocated jobs. We have presented a worst-case complexity analysis of IFF, as well as a detailed simulation-based study of the average behavior. Results show that, in order to locate a requested partition, IFF checks, in average, a very small percent of the total number of network nodes (less than 1%). This result has been tested for different partition sizes in large systems with different number of dimensions.

We have compared IFF against ZFF and RBS, two alternative implementations of first fit in 2D meshes. ZFF is a simple but allegedly efficient implementation of the first fit strategy in which all system nodes are traversed sequentially. RBS is based on the use of a busy list that contains currently allocated tasks. We have evaluated the three algorithms using simulations with synthetically generated workloads, and also with real workloads extracted from the Parallel Workload Archive. Results show the excellent performance of IFF for small and large average-size jobs. Only in scenarios with light load and very large jobs RBS beats IFF. Experiments with actual

DOI: 10.1002/cpe

workloads confirm these findings. In all other cases, RBS performs worse, or even much worse, than IFF. Our proposal has the additional advantage of not being as influenced by system occupancy as RBS is. Results for the ZFF algorithm show that it is not competitive in any scenario.

IFF has been designed to be used in cubic networks of any number of dimensions. As most recent supercomputers implements 3D and 5D networks, we have also carried out a collection of experiments testing IFF in tori with different number of nodes an dimensions. As baseline to comparatively analyze the performance of IFF, we have used a naive implementation of the first fit strategy (NFF). Results show that IFF can be over three orders of magnitude faster than NFF.

For those schedulers that use non-contiguous partitioning, different algorithms must be devised. GABL (*Greedy Available Busy List*) is a proposal described in [12], in which jobs are sequentially divided into smaller sub-jobs that are allocated using RBS. Lists managed by GABL are, therefore, longer than those managed by the original version of RBS, and this has a bearing in terms of cost. We are developing a new version of IFF with the objective of providing efficient and scalable partitioning algorithm when contiguity is not a requirement.

ACKNOWLEDGEMENTS

This work has been supported by programs Saiotek from the Basque Government, projects TIN2010-14931 from the Spanish Ministry of Science and Innovation, and by the COMBIOMED network in computational biomedicine (Carlos III Health Institute). Mr. Pascual is supported by a doctoral grant of the Basque Government. Prof. Miguel-Alonso is a member of the HiPEAC European Network of Excellence.

REFERENCES

1. Feitelson DG, Rudolph L, Schwiegelshohn U. Parallel job scheduling, – a status report. *Job Scheduling Strategies for Parallel Processing*, Springer Verlag, 2005; 1–16.
2. Navaridas J, Pascual JA, Miguel-Alonso J. Effects of job and task placement on the performance of parallel scientific applications. *Proceedings 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, IEEE Computer Society: Washington, DC, USA, 2009; 55–61.
3. Pascual JA, Miguel-Alonso J, Lozano JA. Optimization-based mapping framework for parallel applications. *Journal of Parallel and Distributed Computing* 2011; **71**(10):1377 – 1387.
4. Weisser D, Nystrom N, Brown S, Gardner J, D O, Urbanic J, Lim J, Reddy R, Raymond R, Wang Y, *et al.*. Optimizing job placement on the cray xt3. *48th Cray User Group Proceedings*, 2006.
5. Li K, Cheng KH. A two-dimensional buddy systems for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing* 1991; **12**(1):79–83.
6. Chuang PJ, Tzeng NF. Allocating precise submeshes in mesh connected systems. *IEEE Transactions on Parallel and Distributed Systems* February 1994; **5**:211–217.
7. Ding J, Bhuyan LN. An adaptive submesh allocation strategy for two-dimensional mesh connected systems. *International Conference on Parallel Processing*, vol. 2, 1993; 193–200.
8. Zhu Y. Efficient processor allocation strategies for mesh-connected parallel computers. *Journal of Parallel and Distributed Computing* 1992; **16**(4):328 – 337.
9. Chiu GM, Chen SK. An efficient submesh allocation scheme for two-dimensional meshes with little overhead. *IEEE Transactions on Parallel and Distributed Systems* 1999; **10**:471–486.
10. Chen D, Eisley NA, Heidelberger P, Senger RM, Sugawara Y, Kumar S, Salapura V, Satterfield DL, Steinmacher-Burow B, Parker JJ. The IBM Blue Gene/Q interconnection network and message unit. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, ACM: New York, NY, USA, 2011; 1–10.
11. PWA. Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`.
12. Bani-Mohammad S, Ould-Khaoua M, Ababneh I. An efficient non-contiguous processor allocation strategy for 2d mesh connected multicomputers. *Information Science* 2007; **177**(14):2867–2883.
13. Gara A, Blumrich MA, Chen D, Chiu GLT, Coteus P, Giampapa ME, Haring RA, Heidelberger P, Hoenicke D, Kopcsay GV, *et al.*. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development* march 2005; **49**(2.3):195 –212.
14. IBM Journal of Research and Development staff. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development* Jan 2008; **52**(1/2):199–220.
15. Ajima Y, Inoue T, Hiramoto S, Shimizu T, Takagi Y. The Tofu interconnect. *Micro, IEEE* 2012; **32**(1):21 –31.
16. Aridor Y, Domany T, Goldshmidt O, Moreira JE, Shmueli E. Resource allocation and utilization in the Blue Gene/L supercomputer. *IBM Journal of Research and Development* 2005; **49**(2–3):425–436.
17. Desai N, Buntinas D, Buettner D, Balaji P, Chan A. Improving resource availability by relaxing network allocation constraints on Blue Gene/P. *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, IEEE Computer Society: Washington, DC, USA, 2009; 333–339.
18. Lublin U, Feitelson DG. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing* 2001; **63**:2003.

DOI: 10.1002/cpe

19. Tsafrir D. Modeling, evaluating, and improving the performance of supercomputer scheduling. PhD Thesis, School of Computer Science and Engineering, the Hebrew University, Jerusalem, Israel September 2006.
20. Chapin SJ, Cirne W, Feitelson DG, Jones JP, Leutenegger ST, Schwiegelshohn U, Smith W, Talby D. Benchmarks and standards for the evaluation of parallel job schedulers. *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, 1999; 67–90.

DOI: 10.1002/cpe

11

Application-aware metrics for partition selection in cube-shaped topologies

Application-aware metrics for partition selection in cube-shaped topologies

Jose A. Pascual, Jose Miguel-Alonso, Jose A. Lozano

Intelligent Systems Group, School of Computer Science.
University of the Basque Country UPV/EHU.
P. Manuel Lardizabal 1, San Sebastian, Spain 20018.

Abstract

Non-contiguous partitioning strategies are often used to select and assign a set of nodes of a parallel computer to a particular job. The main advantage of these strategies, compared to contiguous ones, is the reduction of system fragmentation. However, locality in communications cannot be easily exploited, resulting in longer execution times. Several metrics have been proposed in the literature to assess how fit a partition is to run an application on it. These metrics are computed considering the dispersion of the partition. In this paper we demonstrate that metrics based solely on dispersion are not always valid. Using simulation, we show how, for some applications, metrics of a partition do not correlate with the execution times of applications running on it. We define new metrics that do not only consider partition-related properties, but also application's communication patterns and path diversity for communicating tasks. We evaluate these metrics in 2D and 3D meshes, using the NAS Parallel Benchmarks suite of applications as workload. A simulation based study was carried out with a large set of partitions. Results show how metrics that include information about applications' traffic patterns have consistent strong (and positive) correlations with applications' run times.

Keywords: Partition selection strategy; performance metric; application run time; simulation-based evaluation

1. Introduction

Current parallel computing systems are composed of thousands of interconnected compute nodes. It is uncommon to have all these nodes devoted to run a single program. Instead, many simultaneous jobs (applications) of different users time-share resources. The resource manager locates a system partition (collection of free nodes) and assign application's tasks onto partition's nodes. The way this partitioning (and mapping) is done has a remarkable impact on application's performance and on the performance of the whole parallel system, as discussed in [2] [5] [13].

Multiple partitioning strategies have been proposed in the literature. Some of them look for contiguous sets of nodes forming convex shapes (these are contiguous strategies). In [11] [12] [13] it is shown that most applications run faster in this kind of partitions, compared to non-contiguous alternatives, because they optimize communications in terms of locality and non-interference. There is a cost to pay, though: they can cause severe system fragmentation (nodes that cannot be assigned to any job), both internal and external [5]. The class of non-contiguous partitioning strategies was proposed to reduce fragmentation, increasing system utilization at the cost of sacrificing communication locality. It is to be demonstrated whether the expected benefit outperforms the performance penalties experienced by applications.

With the intention of assessing how good a given (non-contiguous) partition would be when used to run a parallel application on it, authors of [6] [7] propose computing some *metrics*. All these metrics focus on

Email addresses: joseantonio.pascual@ehu.es (Jose A. Pascual), j.miguel@ehu.es (Jose Miguel-Alonso), ja.lozano@ehu.es (Jose A. Lozano)

topological characteristics of a partition, measuring its "dispersion degree", trying to reflect, when a parallel application runs on it, the contention that application's messages will experience. Note that contention is understood as waiting times when resources (links) have to be shared by packets trying to cross them simultaneously. In the cited work, authors claim that there is a high and positive correlation between some dispersal metrics and the contention reported by a simulator (they used Procsimity [16]). Even though dispersion can be related to contention and, therefore, with application run times, the proposed metrics for partitions do not consider the characteristics of the applications running on it. The collection of experiments consider only a few applications, running on small partitions of small computing systems. In this paper we show that, in different (but more realistic) scenarios the expected correlation does not exist.

Authors of [4] measure the goodness of partitions using solely the average inter-node distance, showing a correlation between this metric and run times. However, when tested in a real system, the expected correlation does not show up [2]. Authors claim that the previous results were caused by the use of communication patterns in which all-to-all message interchanges were dominant. In this paper we will avoid this pitfall evaluating a diversity of applications with different communication patterns.

A first aim of this paper is to make a thorough assessment of the correlation between dispersal metrics and run times using larger applications with more diverse communication patterns. We will see that correlation is *very weak* for some applications. We claim that this is because these metrics are *application agnostic*: they do not consider how applications communicate, or how many bytes application tasks interchange. For this reason, strong and positive correlations are achieved only for *a few, specific* patterns.

As the metrics proposed in the literature do not have the expected properties, we propose alternative ones. In [13] authors show that, given a partition and an application communication pattern, the shortest run times are achieved when there is a perfect match between application's *virtual topology* and partition's *physical topology*. Most virtual topologies correspond to very regular communication patterns (for example, near neighbor communication on a tree or a mesh); if a job is assigned to a non-contiguous partition, the desirable matching cannot occur. We try to capture in the new metrics the characteristics of *both* partition and application that would result in shorter running times. We focus on (1) the average distance between communicating application tasks, (2) the amount of interchanged traffic and (3) the asymmetry level (or diversity of paths) between communicating tasks. We propose and assess four metrics with different combinations of these factors: TAD (Task Average Distance), WTAD (Weighted Task Average Distance), TAD_AL (TAD + Asymmetry Level) and WTAD_AL (WTAD + Asymmetry Level). Simulation-based studies verify that these metrics consistently show strong and positive correlation with applications' run times.

These metrics have been designed with a particular purpose in mind: to be used by a scheduler as a tool to rank possible (free) partitions on which a particular job can run, choosing the partition that is expected to allow the job to run faster (that should be the one with the smallest metric). They have been developed mainly to evaluate non-contiguous partitions, but they are applicable to contiguous partitions too – in fact, in these cases metrics provide consistently lower values. Therefore, if a contiguous partition were available, the scheduler would choose it.

The experimentation workbench is based on INSEE [10], a simulator that, given a trace of an application, a target network, a partition and a mapping of tasks to nodes, provides an estimation of the execution time of the application; the unit of time is an abstract "cycle". We feed INSEE with (traces of) a subset of the well-known NAS Parallel Benchmarks (NPB) [9]. We use seven of the NPB benchmarks to cover a wide range of communication patterns, and two classes of each one of them to cover different communication intensities: Workstation and A. Regarding system topology, for simplicity we consider only 2D and 3D meshes (see Section 4.1).

The rest of this paper is organized as follows. Section 2 reviews the published (dispersion-based) metrics. In Section 3 we describe the propose metrics TAD, WTAD, TAD_AL and WTAD_AL. Section 4 describes the experimental workbench and the simulation-based experiments carried out. The analysis of results is carried out in Section 5. The paper ends in Section 6 with some conclusions and a sketch of our future lines of work.

2. Dispersion-based metrics for non-contiguous partitions

To our knowledge, there are only two works completely focused on developing and testing metrics for non-contiguous partitions [6] [7]; authors call them *dispersal metrics*. They were defined and evaluated for systems with $k_1, k_2, \ldots, k_n$-ary n-mesh network topologies, where k_i is the number of nodes in dimension i, and n is the number of dimensions. In these networks each node $\mathbf{a}=(a_1, a_2, \ldots, a_n)$ is identified by a coordinates vector with n components, with $a_i \in \{0, 1, \ldots, k_i - 1\}$. Node coordinates can be easily converted into a unique identifier, and vice-versa.

The definition of the first two metrics implicitly assume that nodes communicate using an all-to-all pattern. In this scenario, the *nodes affected* metric (**NA**) measures the number of nodes that could be involved in that communication: those belonging to the partition and others, external to it. Given a (possibly non-contiguous) partition P, this metric measures the area covered by the partition and is defined as follows:

$$NA = \prod_{i=1}^{n} \left(\max_{a \in P}\{a_i\} - \min_{a \in P}\{a_i\} + 1 \right) \tag{1}$$

where $\max_{a \in P}\{a_i\}$ and $\min_{a \in P}\{a_i\}$ are the maximal and minimal coordinates in the i-th dimension respectively for every node a belonging to partition P.

Similarly, the *links affected* metric (**LA**) measures the number of links that could be affected by the all-to-all communication. This metric is computed by projecting the partition nodes onto each dimension, measuring the maximal and minimal coordinates as well as the number of distinct coordinates affected. The formal definition is as follows:

$$LA = \sum_{i=1}^{n} \left(\left(\max_{a \in P}\{a_i\} - \min_{a \in P}\{a_i\} \right) \times \prod_{j=1 \wedge j \neq i}^{n} count_j \right) \tag{2}$$

where in addition to the maximal and minimal coordinates, LA uses the number of different coordinates that have, at least, one allocated node, for each dimension j ($count_j$).

An additional set of metrics based on topological distance is also proposed: *average distance* (**AD**), *distance from center* (**DFC**) and *diameter* (**Diam**). If we use $d(a, b)$ to represent the distance between nodes a and b, and $|P|$ is the number of nodes in partition P, the definition of these three metrics is as follows:

- Average Distance: Represents the average distance between all pairs of nodes in P.

$$AD = \frac{\sum_{a \in P} \sum_{b \in P} d(a, b)}{|P| \times (|P| - 1)} \tag{3}$$

- Distance from Center: Represents the sum of the distances from each node in P to the node that is most central.

$$DFC = \min_{a \in P}\{\sum_{b \in P} d(a, b)\} \tag{4}$$

- Diameter: Represents the maximum distance between two nodes in P.

$$Diam = \max_{a,b \in P}\{d(a, b)\} \tag{5}$$

The distance used in all these metrics is the Manhattan distance, as defined in Equation 6.

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i| \tag{6}$$

In Section 4 we describe the experiments carried out to assess these five metrics, focusing on how good they are to provide clues about the fitness of a given partition to run parallel applications. Results are presented and analyzed in Section 5. As advanced in the Introduction, metrics and run times do not correlate as expected; this is the reason we are proposing new metrics in the next section.

3. Metrics that incorporate knowledge of communication patterns

We propose four new metrics for partitions, always with the aim of providing hints about how fast a given application would run on that partition. The five metrics defined in the previous section are application agnostic. Actually, they assume an all-to-all broadcast between all partition nodes – or, what is the same, between all application tasks if one-to-one mapping is used. The metrics we propose need information about the communication pattern of the application, so they actually measure the fitness of partition-application combinations. The task-to-node mapping procedure also influences these metrics (actually, it strongly influences application performance [13]. In this paper we use a straightforward consecutive mapping: tasks are allocated onto the partition's nodes in the 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.

The communication pattern used by an application can be represented in a communication matrix. We define two of these: a weighted version, and a boolean alternative. Given the set of application's tasks T, matrix $W = [w_{i,j}]_{i,j \in T}$ represents the number of bytes exchanged between each task pair. Matrix B (Equation 7) is the boolean version of W, and only captures whether or not a pair of tasks carry out any message interchange. B represents the (unweighted) communication graph of the application.

$$[b_{i,j}]_{i,j \in T} = \begin{cases} 1 & \text{if } w_{i,j} > 0 \\ 0 & \text{if } w_{i,j} = 0 \end{cases} \tag{7}$$

In [13] we characterized the impact of the shape of a partition on the performance of an application running on it, stating that the shortest run time is achieved when the (virtual) topology of the application matches perfectly the (physical) topology of the partition. Application programmers usually arrange tasks in a way that communications are optimized for a regular, compact network topology. This effort is useless if the scheduler assigns to that application a non-contiguous partition, far different from the one expected by the programmer.

Metrics defined in the previous section were designed to assess how fit a partition is for a generic application. Now we refine these metrics for *specific* applications, using the communication matrices. As explained before, this cannot be done without previously defining a mapping of tasks to partition nodes. Given a set of tasks T that form a parallel application, and a set of nodes P of a partition, the mapping π of T on P is defined as follows:

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

Given a task t, $\pi(t)$ returns the identifier of node a in which t has been allocated. Inversely, given a node a, $\pi^{-1}(a)$ returns the identifier of the task t running on it.

We define the **TAD** (Task Average Distance) metric as the *average distance between communicating application tasks*. More formally, TAD is computed as:

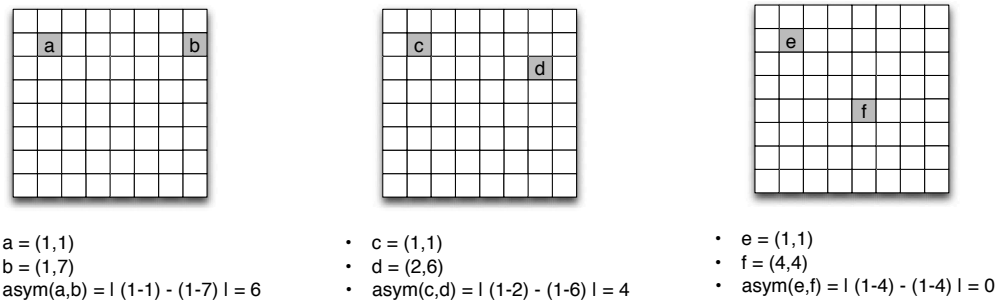


Figure 1: Examples of computation of the asymmetry level in a 2D network. Left: a very asymmetric arrangement of a pair of nodes, with low path diversity. Right: a totally symmetric arrangement, maximizing path diversity.

$$TAD = \frac{1}{|T| \cdot |T| - 1} \sum_{a \in P} \sum_{b \in P} d(a,b) \cdot B_{\pi^{-1}(a),\pi^{-1}(b)} \tag{8}$$

Note that computing TAD requires matrix B, reflecting all pairs of communicating tasks. However, in many applications the intensity of traffic between task pairs is not homogeneous. Frequent communication increase contention for resources. A talkative pair of tasks should be allocated to nearby nodes in order to reduce resource utilization. Similarly, the relative positions of tasks that only interchange a few messages is not very relevant. For this reason, we define **WTAD** as a weighted variant of TAD, in which matrix W is used instead of B. More formally:

$$WTAD = \frac{1}{|T| \cdot |T| - 1} \sum_{a \in P} \sum_{b \in P} d(a,b) \cdot W_{\pi^{-1}(a),\pi^{-1}(b)} \tag{9}$$

Note that, as with TAD, only communicating tasks contribute to this metric, in a measure that depends on the intensity of the communication.

To define the remaining two metrics, we add additional knowledge. In a communication network, it is desirable to have a high degree of path diversity between node pairs, to avoid congestion hot-spots and to take advantage of adaptive routing [13]. We can capture this using a measure of the "asymmetry level" of an arrangement of two nodes, defined in Equation 10. In the equation, $d_i(a,b) = |a_i - b_i|$. Note that $asym(a,b)$ measures the sum of the differences between per-dimension distances for nodes a and b, yielding higher values for more asymmetric arrangements. Those pairs with lower asymmetry levels provide a higher diversity of paths and are considered better.

$$asym(a,b) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |d_i(a,b) - d_j(a,b)|; \tag{10}$$

In Figure 1 we have depicted three node-pair arrangements, in all cases at distance 6, and the corresponding computations of *asym*: 6 (left), 4 (middle) and 0 (right). The most asymmetric arrangement allows for a single path communicating the two nodes, while the extreme case of 0 asymmetry allows up to 35 different paths (in the middle case there are 21 possible paths). Path diversity can be exploited by a network implementing adaptive routing to reduce contention.

Once *asym* has been defined, we can introduce metrics TAD_AL and WTAD_AL. The first one adds the asymmetry level to the average inter-task distance TAD. Formally, **TAD_AL** is computed as follows:

$$TAD\_AL = \frac{1}{|T| \cdot |T| - 1} \sum_{a \in P} \sum_{b \in P} (d(a,b) + asym(a,b)) \cdot B_{\pi^{-1}(a),\pi^{-1}(b)} \tag{11}$$

Similarly, WTAD can be extended to take asymmetry into account. **WTAD_AL** is therefore defined as follows:

$$WTAD\_AL = \frac{1}{|T| \cdot |T| - 1} \sum_{a \in P} \sum_{b \in P} (d(a,b) + asym(a,b)) \cdot W_{\pi^{-1}(a),\pi^{-1}(b)} \quad (12)$$

These new metrics will be assessed in the following sections, and compared against those that do not use application-related information.

4. Experimental set-up

In this section we detail the collection of experiments carried out to evaluate the effectiveness of the different metrics. Given a target network, in this case a 2D or a 3D mesh, we generate a set of partitions, all of them with 64 compute nodes. We run, on those partitions, some parallel applications selected from the well-known NAS Parallel Benchmarks (NPB) [9] suite, classes Workstation and A, for 64 compute nodes. We compute, for each partition-application pair, the metrics defined in sections 2 and 3. Also, using a simulator, we compute a prediction of the run times of the applications on the partitions, and evaluate the correlation between the metrics and the run times.

We are interested in large parallel systems space-shared by several applications. For this reason, in the experiments we will use a system with 1024 (32×32) nodes arranged as a 2D mesh, and also a 3D mesh-arranged system with 512 ($8 \times 8 \times 8$) nodes. This would give us some insights into the influence of the topology on the effectiveness of the metrics.

In the following subsections we define in more detail the characteristics of the applications, the simulation environment, and the design of the experiments.

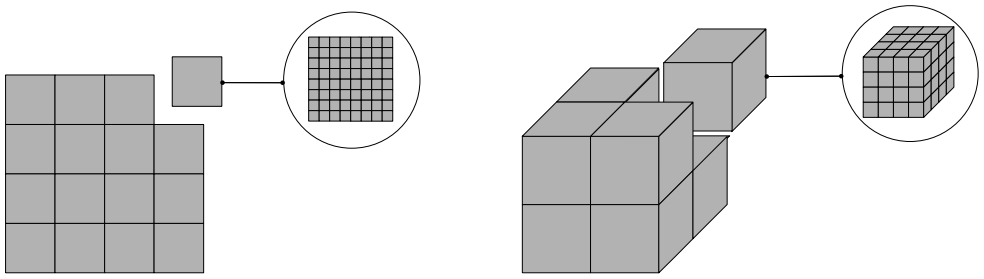
4.1. Experimental workbench

The workbench comprises two key pieces:

- NPB traces: For each application and size we need to generate the traffic matrices W and B. We have done so for particular instances of the benchmarks included in the NPB [9], classes Workstation and A with size 64. The difference between these classes is in the problem size. This means that the communication pattern is the same, but in class A tasks interchange more data. These applications were run in a real parallel machine, in which traces of all the messages interchanged by tasks were captured (including the point-to-point messages used to implement collective communications). Traces contain the necessary information to fill matrix W, from which B can be easily derived. Traces are also required by the INSEE simulation environment.

- INSEE [10]: This tool simulates the execution of a message-passing application on a multicomputer. It performs a very detailed simulation of the interchange of the messages through the interconnection network, considering network characteristics (topology, routing algorithm) and application behavior (causality among messages). INSEE can simulate the simultaneous execution of several instances of the same application, running on different system partitions. The input for INSEE includes the application's trace file, the sets of nodes that form the partitions, and the corresponding instance-to-partition mapping vectors. 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. This tool has been widely used in the evaluation of massive parallel systems [3] [8]. In order to highlight the influence of partitioning on run times, and to avoid the effects that the characteristics of the compute nodes would have in those times, we have chosen to use infinite-speed CPUs. This means that, in fact, we are only measuring the communication costs of the applications.

Regarding the applications, we use seven of the NPB benchmarks. We classify them into four types considering the virtual topology used to carry out inter-task communications:



(a) Blocks for the 32×32 mesh. (b) Blocks for the $8 \times 8 \times 8$ mesh.

Figure 2: Subdivision in blocks used in the procedure to generate configurations. Each network configuration is created sleeting a k number of block and assigning k applications to those preselected k blocks.

- LU, BT and SP: In these three applications tasks communicate with neighbors using a 2D virtual topology [1]. LU uses a perfect 2D mesh communication pattern. BT and SP use a 6 point stencil pattern on a 2D mesh (each node communicates with its NE and SW neighbors in addition to N, E, W, and S) [17].
- MG: In this application tasks also communicate with neighbors, but using a hypercube virtual topology [17].
- CG: This application has its own distinctive communication pattern, that corresponds with a 3-point stencil [17].
- FT and IS: These applications use all-to-all communications, without a specific virtual topology in mind. Traces reflect a non-optimized implementation of all-to-all primitives, using point-to-point messages [14].

4.2. Design of the experiments

In this section we detail the procedure used to generate sets of partitions. The objective is to split the complete system in several, 64-node partitions, with *similar* characteristics. These partitions will be used by simultaneously-running applications (actually, instances of the same application). This means that we will evaluate the system in conditions of maximum occupancy. In the case of the 2D system, the network is a mesh of 32×32 nodes, and therefore we generate 16 partitions. For the 3D case (with $8 \times 8 \times 8$ nodes) 8 partitions are generated. The resulting partitions are provided as inputs for INSEE. Regarding mapping vectors, we use a simple, consecutive mapping strategy.

We define a *network configuration* as a collection of partitions in which a network is split. We want to test different configurations, with partitions with a variety of shapes, inter-node distances, diameters, etc. that will result in a diversity of scenarios. Now we explain the mechanism used to generate non-contiguous and contiguous configurations.

4.2.1. Non-contiguous network configurations

We focus first on the 2D case. The network is divided into blocks of 64 contiguous nodes: 16 square blocks of 8×8 nodes (see Figure 2(a)). An initial, trivial configuration is considering each block a contiguous partition. To generate non-contiguous partitions, we use a parameter k that can take values 2, 4, 8 and 16. Given a value of k, we select (randomly) a *block set* with k blocks. Collectively, this block set can accommodate k applications. Now we generate k partitions within the block set selecting randomly 64 nodes. The procedure is repeated until generating the 16 partitions, guaranteeing that each block is used only once. Note that $k = 1$ would provide a contiguous configuration (see next subsection), and $k = 16$

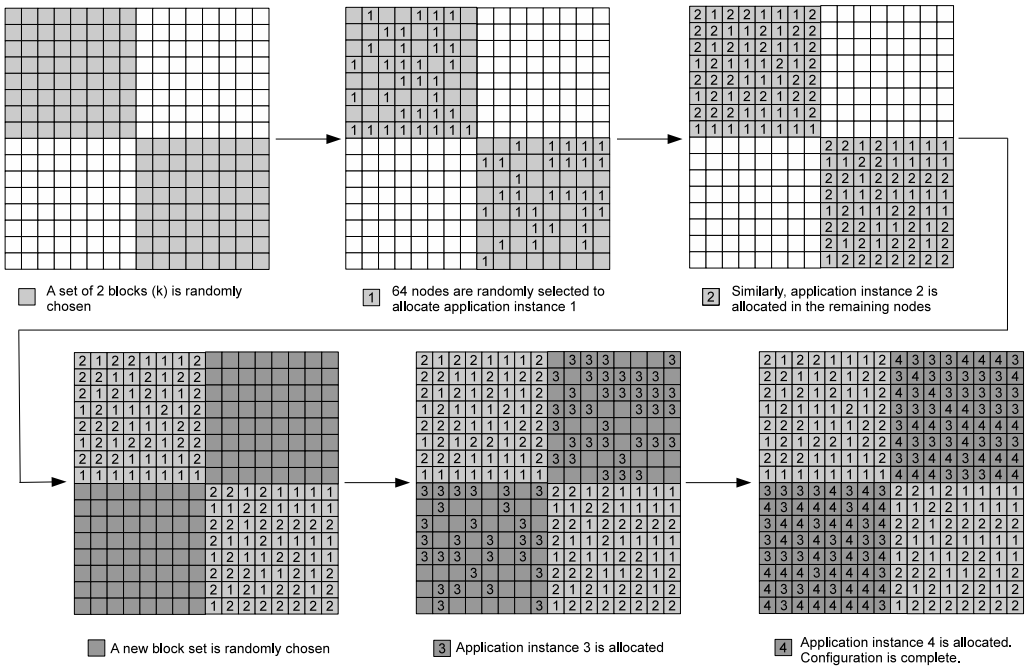


Figure 3: Example of the generation of a network configuration with $k = 2$. For the sake of clarity we represent a 16×16 mesh. First we choose two blocks randomly. Then, we select 64 nodes randomly inside those blocks and map first application tasks into them using a consecutive strategy. The second application is mapped into the remaining nodes of the selected blocks. The same procedure is followed to assign the remaining blocks to the remaining applications.

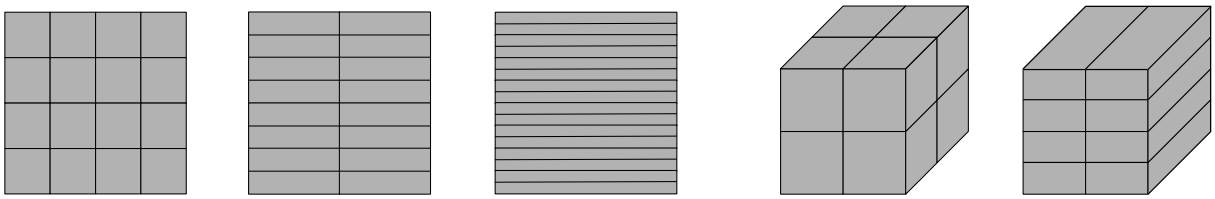
generates completely random partitions. A simplified, graphical explanation of this process is depicted in Figure 3. In the 3D case, there are only 8 blocks of $4 \times 4 \times 4$ nodes (see Figure 2(b)) and k can take values 2, 4 and 8. A total of 80 different non-contiguous configurations per topology and application have been included in the tests.

4.2.2. Contiguous network configurations

The procedure described before, using $k = 1$, generates square (cubic) contiguous partitions of 8×8 (for 2D) and $4 \times 4 \times 4$ (for 3D). These have been included in the experiments. Other contiguous, prismatic (but not cubic) partitions have been included too. In the case of the 32×32 2D network we have defined partitions of 16×4 and 32×2 nodes (see Figure 4(a)). For the 3D case, we have included in the experiments partitions with shape $8 \times 4 \times 2$ as shown in Figure 4(b). Note that, in general, the asymmetry levels of node pairs are smaller for the most regular partitions. This fact will be reflected in the corresponding _AL metrics.

4.3. Evaluation of the metrics

Given the partition set, the dispersion metrics described in Section 2 can be easily computed, because they only need topology-related information. The new metrics proposed in this paper can also be computed using this information together with communication matrices. Notice that all metrics were defined for a partition (or partition + application), but we follow the methodology proposed in [6] [7] to extend them to measure network configurations. In a given configuration there are several application instances running concurrently, each one on its partition (and, therefore, with its own metric). The metric of the configuration is the average of the metrics of its partitions.



(a) 8×8, 16×4 and 32×2 (b) $4 \times 4 \times 4$ and $8 \times 4 \times 2$

Figure 4: Contiguous 64-node partitions used in the experiments. a) for a (32×32) 2D mesh. b) for a $(8 \times 8 \times 8)$ 3D tori.

Note that configurations are also fed into INSEE, together with application traces, to measure the total time to complete all simultaneously-running applications.

Once all this information has been collected, we have calculated the Pearson correlation coefficient between configuration metrics and run times. This coefficient is a value in the range $[-1..1]$. Values close to the extremes indicate strong (positive or negative) correlation between the two tested set of values. Any relationship between two variables must also be assessed for its significance because strong correlations do not imply significance. We perform the test of significance for a Pearson product-momentum correlation coefficient as explained in [15]. This way we can guarantee the significance of the relationship between the variables. We perform the tests against the null hypothesis of no relationship between each metric and each runtime with a confidence level of $\alpha = 0.01$.

5. Analysis of results

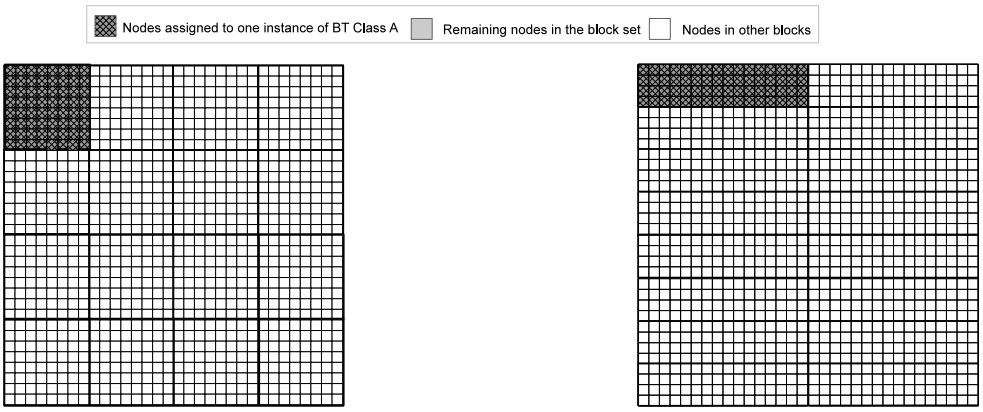
In this section we analyze the results of the experiments described in the previous section for each topology (the 32×32 mesh and the $8 \times 8 \times 8$ mesh). In the context of this work, we are concerned about the *performance* of a metric, measured as the correlation with application run times. We do not provide exhaustive tables of metrics and run times, because those would be extensive and tedious to analyze. However, to have a glimpse about the managed values for metrics and run times, we provide in Figure 5 four different configurations for application BT (class A, 64 tasks), including contiguous and non-contiguous set-ups, together with the corresponding values of the metrics. A quick look to these results show that BT runs remarkably faster in contiguous partitions, compared to the non-contiguous alternatives. Additionally, the run time for this application is shorter in the square partition than in the rectangular one. These facts are adequately reflected by metric TAD_AL (see Table 1).

In the remaining of this section we will pay attention solely to the Pearson correlation values between metrics and run-times, for a given application and topology, considering the generated configurations: 83 for the 2D network and 82 for the 3D network. These results are summarized in Tables 2 and 3.

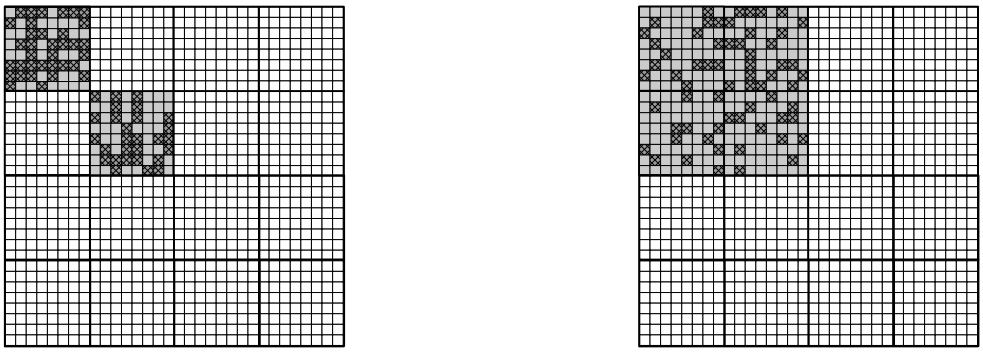
Each table is arranged in three sections: one for metrics considering affected nodes / links (NA, LA), one for the distance-related metrics (AD, DFC, Diam) and the last one for the newly proposed metrics (TAD,

Configuration	NA	LA	AD	DF	Diam	TAD	WTAD	TAD_AL	WTAD_AL	**Run time**
NC1	64	3136	5.33	256	14.00	0.41	545812	0.75	921106	3704882
NC2	64	2880	6.66	320	18.00	0.56	799189	1.03	1359653	8318829
NC3	256	57150	10.69	507.25	28.81	0.98	999325	1.35	1403923	13091018
NC4	256	56700	10.69	510.12	28.25	1.17	1220024	1.80	1900848	14619862

Table 1: Values of metrics and run times for application BT (class A, 64 tasks) for the four configurations depicted in Figure 5. Values in the "Run time" column are INSEE cycles measuring communication cost.



(a) Examples of contiguous configurations.



(b) Examples of non-contiguous configurations. Left: $k = 2$. Right: $k = 4$.

Figure 5: Examples of four network configurations. The figures depict a single application instance. The remaining nodes are assigned to other, concurrently running instances.

WTAD, TAD_AL, WTAD_AL). All results in Table 2 have a significance level of 0.01. As this is not the case for the 3D mesh, in Table 3 we have used a (†) to identify values with this significance level.

5.1. Results for the 2D mesh

Let start analyzing the results for the 2D mesh topology using the NPB applications, as summarized in Table 2. The top part correspond to class (problem size) Workstation of these benchmarks, while the bottom part is for class A. The difference between these classes, as explained before, is not in the communication pattern, but in the amount of traffic interchanged by tasks (larger for A).

If we focus on class Workstation, we can see that for all benchmarks except FT and IS the newly proposed metrics perform better than those proposed in [6] [7]. Even for those applications, only DFC is marginally better. When dealing with class A, the new metrics' performance is similar to the distance-related ones, except for CG, for which introducing the knowledge of the communication pattern clearly increases the performance of the derived metrics. Results for FT and IS are not surprising: these applications use all-to-all communications, meaning that the communication matrices are complete and homogeneous. In other words: these matrices do not contribute with any relevant information. In this context of dense (and intense) communication, searching for path diversity does not provide any additional help, because all resources are equally used.

In general, metrics NA and LA show poor performance levels, compared to those that consider the distance between nodes (tasks), which are all the remaining ones. Performances of AD, DFC and Diam are very similar, being DFC slightly better.

(a) NPB size 64 class *Workstation*.

Benchmark	NA	LA	AD	DFC	Diam	TAD	WTAD	TAD_AL	WTAD_AL
BT	0.80	0.76	0.88	0.89	0.85	0.91	0.88	**0.93**	0.90
LU	0.84	0.83	0.89	0.90	0.87	0.96	0.93	**0.97**	0.94
SP	0.81	0.77	0.88	0.89	0.85	0.92	0.89	**0.94**	0.91
CG	0.59	0.58	0.68	0.71	0.64	0.88	0.88	**0.92**	**0.92**
MG	0.66	0.64	0.77	0.80	0.73	0.85	0.87	0.91	**0.92**
FT	0.89	0.83	0.96	**0.98**	0.94	0.96	0.96	0.96	0.96
IS	0.87	0.78	0.94	**0.96**	0.91	0.94	0.94	0.95	0.94

(b) NPB size 64 class *A*.

Benchmark	NA	LA	AD	DFC	Diam	TAD	WTAD	TAD_AL	WTAD_AL
BT	0.88	0.81	**0.94**	**0.94**	0.91	0.93	0.91	0.92	0.91
LU	0.86	0.84	0.91	0.91	0.89	0.96	0.93	**0.97**	0.94
SP	0.88	0.81	0.94	**0.95**	0.91	0.93	0.91	0.92	0.91
CG	0.66	0.65	0.76	0.78	0.72	0.91	0.91	0.92	**0.93**
MG	0.82	0.77	0.90	0.92	0.87	0.92	0.92	**0.93**	0.92
FT	0.93	0.86	0.97	**0.98**	0.95	0.97	0.97	0.95	0.95
IS	0.90	0.81	0.95	**0.96**	0.93	0.95	0.95	0.94	0.94

Table 2: Pearson correlation coefficient between NPB applications run times and the different metrics in a 32×32 mesh. All relationships have 0.01 level of significance.

Regarding the results for the last set of metrics, if we make TAD our reference, we can see that using matrix W instead of B does not improve the metric. That is, the performance of WTAD is not better than that of TAD. Adding the asymmetry level to the metrics improves performance for all applications except FT and IS. We have explained how these applications work, and how searching for path diversity for them is worthless.

It is important to note that the performance of the new metrics is good independently of the application (and class). TAD_AL and WTAD_AL show correlation coefficients that are always over 0.9. And, when performing worse than distance-related metrics, the difference is not large.

5.2. Results for the 3D mesh

Now we analyze the results using the 3D network topology, summarized in Table 3. Again the top part correspond to application class Workstation and the bottom part to class A. Note that there are many negative values in the table but, as they are not statistically significant, we will not pay attention to them.

Again, NA and LA perform poorly (even worse than in the 2D case), so we will not discuss them. We also confirm that distance-related metrics are appropriate only for applications performing all-to-all communications (FT, IS). This fact was already discussed in [2]. For the remaining applications, the influence of distance in application performance is not that direct. Other factors, such as the capability of applications to exploit the large diversity of paths provided by this cubic topology have to be taken into account. This explains the good performance achieved by TAD_AL and WTAD_AL.

In general, the new metrics are the best performers. Using (as in the previous section) TAD as the reference point, including in the metric information about the intensity of the inter-task communication does significantly improve performance (WTAD show better results than TAD). Also, except for FT and IS, adding the asymmetry level further increases performance, for the reasons explained before. Globally, the best performing metric is WTAD_AL except for all-to-all based applications, for which TAD would be a better choice.

5.3. General discussion

Experiments shows that metrics based solely on distance are valid only in low-degree topologies, when running applications performing all-to-all and intense data interchanges. In these scenarios, path diversity is

(a) NPB size 64 class *Workstation*

Benchmark	NA	LA	AD	DFC	Diam	TAD	WTAD	TAD_AL	WTAD_AL
BT	-0.33	-0.43	-0.18	-0.18	-0.18	0.31†	0.64†	0.49†	**0.68†**
LU	-0.32	-0.37	-0.25	-0.26	-0.28	0.34†	0.79†	0.65†	**0.87**†
SP	-0.36	-0.45	-0.21	-0.21	-0.22	0.34†	0.71†	0.53†	**0.76**†
CG	-0.04	-0.04	-0.04	-0.04	-0.1	0.62†	0.62†	**0.80**†	**0.80**†
MG	0.26†	0.25†	0.30†	0.30†	0.24†	0.68†	0.70†	**0.85**†	0.84†
FT	0.86†	0.82†	**0.94**†	0.93†	**0.94**†	**0.94**†	**0.94**†	0.72†	0.72†
IS	0.89†	0.87†	**0.93**†	**0.93**†	**0.93**†	**0.93**†	**0.93**†	0.76†	0.74†

(b) NPB size 64 class *A*

Benchmark	NA	LA	AD	DFC	Diam	TAD	WTAD	TAD_AL	WTAD_AL
BT	-0.21	-0.32	-0.05	-0.05	-0.05	0.45†	0.79†	0.59†	**0.80**†
LU	-0.23	-0.31	-0.12	-0.13	-0.15	0.42†	0.88†	0.69†	**0.92**†
SP	-0.19	-0.29	-0.02	-0.01	-0.02	0.47†	**0.80**†	0.60†	**0.80**†
CG	0.11†	0.09†	0.14†	0.14†	0.08†	0.76†	0.76†	0.83†	**0.87**†
MG	0.44†	0.42†	0.51†	0.52†	0.46†	0.81†	0.83†	0.88†	**0.89**†
FT	0.90†	0.88†	**0.95**†	**0.95**†	**0.95**†	**0.95**†	**0.95**†	0.74†	0.74†
IS	0.91†	0.89†	**0.95**†	**0.95**†	0.94†	**0.95**†	**0.95**†	0.77†	0.76†

Table 3: Pearson correlation coefficient between NPB applications run times and the different metrics in a $8 \times 8 \times 8$ mesh. Values with a (†) have a 0.01 level of significance.

poor and, furthermore, all paths are equally and heavily used. Best partitions are those with small average distance that help removing packets from network as soon as possible. Metrics TAD and WTAD provide, in this case, basically the same information, which is not very different from that of AD.

For higher-degree networks (in this case, 3D meshes), the situation changes. These networks are richer in terms of path diversity. Taking advantage of this diversity can reduce possible contention hot-spots. It may be a better choice to have a partition with larger diameter if, as a result, congestion can be avoided. This is the reason why purely distance-related metrics do not correlate with application run times. Experiments show that, if path diversity (asymmetry level) is considered to compute the metric, correlation is much better.

In summary WTAD_AL is an excellent indicator of the fitness of a given partition to run a given parallel application. If the W matrix is not available (or is expensive to obtain), but the communication pattern (given by B) is known, TAD_AL would be a good alternative. For FT and IS, with complete and homogeneous communication matrices, any distance-related metric would be a good choice.

6. Conclusions and Future Work

Current parallel systems share their resources between multiple, simultaneously-running applications. In space-shared environments, they run in independent partitions, selected by means of a partitioning strategy. These strategies can be broadly classified into contiguous and non-contiguous. In this paper, we have studied distance-related metrics proposed in the literature to evaluate the appropriateness of a given non-contiguous partition to execute a parallel application. In addition we have also proposed new metrics that take into account application and topology related characteristics. These metrics can be applied to contiguous as well as non-contiguous partitions, and both cases have been included in the experiments carried out to test their performance.

From the metrics proposed in [6] [7], NA and LA are not good performance indicators. Distance-related metrics (AD, DFC and Diam) are better but only in some cases: low-degree networks, intense communications, all-to-all patterns. The metrics proposed in this paper that include application-related information (TAD, WTAD) as well as available path-diversity (TAD_AL, WTAD_AL) are of more general applicability.

After analyzing the results of an extensive set of experiments we can conclude that application agnostic metrics should not be used by a scheduler to make partition selection decisions. This kind of metrics are indicators of the fitness of a partition to run an application only in very limited scenarios. The additional knowledge provided by the application and the path-diversity improves remarkably the capacity of a metric to determine the suitability of a non-contiguous partition.

As future work we plan to further investigate the relationship between the proposed metrics and the runtime of the applications. These metrics provide correlation coefficients close to 1, indicating a strong linear relationship between application run times and partition shapes. Therefore, the value of these metrics could be used to estimate the run time of an application on a new partition when the run time on a different partition is known.

This knowledge is planned to be included in a topology-aware scheduling framework in which the scheduler searches for a system partition to run an job (application) considering how well the job would run on it, assuming that jobs request a node count and the scheduler has to choose the best available set of free nodes. The scheduler should make this decision taking into consideration not only application and available partition shapes, but also the task-to-node mapping, an issue that has not been explored in this paper.

Acknowledgements

This work has been supported by programs Saiotek and Research Groups 2007-2012 (IT-242-07) from the Basque Government, project TIN2010-14931 from the Spanish Ministry of Science and Innovation, and by the COMBIOMED network in computational biomedicine (Carlos III Health Institute). Mr. Pascual is supported by a doctoral grant of the Basque Government. Prof. Miguel-Alonso is a member of the HiPEAC European Network of Excellence.

[1] D. Bailey, T. Harris, W. Saphir, R. V. D. Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical report, NASA Advanced Supercomputer (NAS) division, 1995.

[2] D. P. Bunde, V. J. Leung, and J. Mache. Communication patterns and allocation strategies. *18th International International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[3] J. M. Camara, M. Moreto, E. Vallejo, R. Beivide, J. Miguel-Alonso, C. Martinez, and J. Navaridas. Twisted torus topologies for enhanced interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(12):1765–1778, 2010.

[4] V. J. Leung, E. M. Arkin, M. A. Bender, D. Bunde, J. Johnston, A. Lal, J. S. B. Mitchell, C. Phillips, and S. S. Seiden. Processor allocation on cplant: Achieving general processor locality using one-dimensional allocation strategies. In *Proceedings of the IEEE International Conference on Cluster Computing*, CLUSTER '02, pages 296–304, Washington, DC, USA, 2002. IEEE Computer Society.

[5] V. Lo, K. J. Windisch, W. Liu, and B. Nitzberg. Noncontiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 8:712–726, 1997.

[6] J. Mache and V. Lo. Dispersal metrics for non-contiguous processor allocation. Technical report, University of Oregon, 1996.

[7] J. Mache and V. Lo. The effects of dispersal on message-passing contention in processor allocation strategies. In *Proceedings of the Third Joint Conference on Information Sciences, Sessions on Parallel and Distributed Processing*, pages 223–226, 1997.

[8] J. Miguel-Alonso, C. Izu, and J. Gregorio. Improving the performance of large interconnection networks using congestion-control mechanisms. *Performance Evaluation*, 65(3-4):203 – 211, 2008.

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

[10] J. Navaridas, J. Miguel-Alonso, J. A. Pascual, and F. J. Ridruejo. Simulating and evaluating interconnection networks with INSEE. *Simulation Modelling Practice and Theory*, 19(1):494 – 515, 2011.

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

[12] J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. Strategies to map parallel applications onto meshes. In *Distributed Computing and Artificial Intelligence*, volume 79 of *Advances in Intelligent and Soft Computing*, pages 197–204. Springer Berlin / Heidelberg, 2010.

[13] J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. Optimization-based mapping framework for parallel applications. *Journal of Parallel and Distributed Computing*, 71(10):1377 – 1387, 2011.

[14] W. Saphir, R. V. D. Wijngaart, A. Woo, and M. Yarrow. New implementations and results for the NAS parallel benchmarks 2. In *8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[15] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 edition, 2007.

[16] K. Windisch, J. V. Miller, and V. Lo. Procsimity: an experimental tool for processor allocation and scheduling in highly parallel systems. In *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95)*, pages 414–421, Washington, DC, USA, 1995. IEEE Computer Society.

[17] Q. Xu, J. Subhlok, R. Zheng, and S. Voss. Logicalization of communication traces from parallel execution. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 34–43, Washington, DC, USA, 2009. IEEE Computer Society.

12

Effects of topology-aware allocation policies on scheduling performance

Effects of Topology-Aware Allocation Policies on Scheduling Performance

Jose Antonio Pascual, Javier Navaridas, and Jose Miguel-Alonso

The University of the Basque Country, San Sebastian 20018, Spain
{joseantonio.pascual,javier.navaridas,j.miguel}@ehu.es

Abstract. This paper studies the influence that job placement may have on scheduling performance, in the context of massively parallel computing systems. A simulation-based performance study is carried out, using workloads extracted from real systems logs. The starting point is a parallel system built around a k-ary n-tree network and using well-known scheduling algorithms (FCFS and backfilling). We incorporate an allocation policy that tries to assign to each job a contiguous network partition, in order to improve communication performance. This policy results in severe scheduling inefficiency due to increased system fragmentation. A relaxed version of it, which we call quasi-contiguous allocation, reduces this adverse effect. Experiments show that, in those cases where the exploitation of communication locality results in an effective reduction of application execution time, the achieved gains more than compensate the scheduling inefficiency, therefore resulting in better overall performance.

1 Introduction

Supercomputer centres are usually designed to provide computational resources to multiple users running a wide variety of applications. Users send jobs to a scheduling queue, where they wait until the resources required by the job are available. These jobs may vary from large parallel programs that need many processors, to small sequential programs. The scheduler manages system resources, taking into consideration different policies that may restrict the use in terms of maximum number of processors or maximum execution time. Other restrictions may be implemented such as user or group priorities, quotas, etc.

Generally, site performance is measured in terms of the utilization of the system and the slowdown suffered by jobs while waiting in the queue until the required resources become available. Consequently, a variety of scheduling policies [1] and allocation algorithms [2] [3] [4] have been developed aiming to minimize both the number of nodes that remain idle and the job waiting times. Scheduling policies are in charge to decide the order in which jobs are launched. Scheduling decisions may be based on different variables, such as job size, user priority or system status. Allocation algorithms map jobs onto available resources (typically, processors). Locality-aware policies select resources taking into account network characteristics, such as its topology or the distance between processors.

E. Frachtenberg and U. Schwiegelshohn (Eds.): JSSPP 2009, LNCS 5798, pp. 138–156, 2009.

The most commonly used scheduling policies are FCFS (First-Come First-Serve) and FCFS + backfilling, sometimes with variations. The FCFS discipline imposes a strict order in the execution of jobs. These are arranged by their arrival time and order violations are not permitted, even when resources to execute the first job are not available but there are enough free resources to execute some other (or others) jobs in the queue. The main drawback of this policy is that it produces severe system fragmentation because some processors can remain idle during a long period of time due to the sequentially ordered execution of jobs. Idle processors could be used more efficiently running less-demanding jobs, thus achieving a performance improvement.

With the goal of minimizing the effect of this strictly sequential execution order, several strategies have been developed [1], backfilling being the most widely used due to its easy implementation and proven benefits. This policy is a variant of FCFS, based on the idea of advancing jobs through the queue. If some queued jobs require a smaller amount of processors than the one at the head, we can execute them until the resources required by the job at the head become available. This way, utilization of resources is improved because both network fragmentation and job waiting times decrease. The reader should note that, throughout this paper, we will often use the word *network* to refer to the complete parallel system.

Network fragmentation caused by scheduling algorithms is known as external fragmentation [5]. But a different kind of fragmentation appears in topologies like meshes or tori when the partitions reserved to jobs are organized as sub-meshes or sub-tori; for example, to allocate a job composed by 4x3 processes, some algorithms search for square sub-meshes, 4x4 being the smallest size that can be used to run the job. In this case, four processors reserved for the job will never be used. This effect is named internal fragmentation [5]. Some job allocation algorithms try to minimize this effect. However, this work *does not* consider this effect, because each parallel job will be assigned to the exact number of required nodes.

Neither FCFS nor backfilling are allocation algorithms, as they do not take into account the placement of job processes onto network nodes. In a parallel system, application processes (running on network nodes) communicate interchanging messages. Depending on the communication pattern of the application, and the way processes are mapped onto the network, severe delays may appear due to network contention; delays that result in longer execution times. If we have several parallel jobs running in the same network, each of them randomly placed along the network, communication locality inside each job will not be exploited; and what is more, messages from different applications will compete for network resources, greatly increasing network contention. An effective exploitation of locality results in smaller communication overheads, which reflects in lower running times. Note that searching for this locality is expensive in terms of scheduling time, because jobs cannot be scheduled until contiguous resources are available (and allocated), so that network fragmentation increases. In order to avoid this effect, we propose the utilization of quasi-contiguous allocation schemes in which some restrictions of the purely-contiguous policy are relaxed, allowing the non-contiguous allocation of part of the required network nodes.

This way network occupancy can be increased, at the cost of some penalty in terms of application run times.

A trade-off has to be found between the gains attainable via exploitation of locality and the negative effects of increasing fragmentation. This is precisely the focus of this paper. We study only the placement in k-ary n-tree topologies [6], but the tools and methodology presented here will be extended to other topologies such as meshes or tori. Our final goal is to demonstrate that the introduction of locality-aware policies in the schedulers may provide important performance improvements in systems with multiple users and different applications.

The rest of the paper is organized as follows. In Section 2 we discuss some previous work on scheduling and allocation policies, describing in Section 3 those used in this paper. The simulation environment and the workloads used for the experiments are described in Section 4. Section 5 analyze a few preliminary experiments that provide evidence of the pros and cons of consecutive allocation schemes. These experiments are further elaborated in Section 6, that focuses on the search of a trade-off between application speedup and scheduling slowdown. Section 7 closes the paper with some conclusions and future lines of research.

2 Related Work

Extensive research has been conducted in the area of parallel job scheduling. Most works were focused on the search of new scheduling policies that minimize job waiting times, and on allocation algorithms that minimize network fragmentation. In [1] authors analyzed a large variety of scheduling strategies; however, none of them took into account virtual topologies of applications (the logical way of arranging processes to exploit communication locality) or network topology.

To our knowledge, only [5] described a performance study of parallel applications taking into account locality-aware allocation schemes. The starting point of this job was the fact that, in schedulers optimized for certain network topologies (they focused on meshes and tori), allocation was always done in terms of sub-meshes (or sub-tori). This policy optimized communication in terms of locality and non-interference, but caused severe fragmentation, both internal and external. The authors did not use scheduling with backfilling, a technique that would partly reduce this undesirable effect. However, they tested a collection of allocation strategies that sacrifice contiguity in order to increase occupancy. They claimed that the effect on application performance attributable to the partial loss of contiguity was low, and more than compensated by the overall improvement in system utilization.

A more recent paper [7] evaluated the positive impact that locality-aware allocations have on applications performance, but focused on three particular applications, running on supercomputers connected by 3-D interconnection networks.

Part of our experiments corroborates the conclusions of the cited papers. However, our work differs from them in several important aspects. Previous research work shows that, depending on the communication pattern of the application,

contiguous allocation provides remarkable performance improvements [8]. Therefore, we do not make extensive use of non-contiguity to increase system utilization; instead, we incorporate backfilling scheduling policy into the scheduler. Additionally, we focus on k-ary n-trees, instead of meshes or tori.

A review of schedulers in use in current supercomputers, such as Maui, Sun Grid Engine, and PBS Pro, shows that they do not implement contiguous allocation strategies. Some of them provide methods for the system administrator to develop their own strategies but, in practice, this is rarely done. To our knowledge, the only two current schedulers that maintain the locality are the one used by the BlueGene family supercomputers [9] and SLURM. The BlueGene scheduler puts tasks from the same application in one or more midplanes of 8x8x8 nodes which decreases network contention and allows locality exploitation. SLURM performs always a best-fit algorithm building first a Hilbert curve through the nodes on the Sun Constellation and Cray XT systems in order to keep locality as higher as possible. In contrast, the scheduling strategy used by the default scheduler (PBS Pro) on Cray XT3/XT4 systems (also a custom-made 3D tori) simply gets the first available compute processors [10].

3 Scheduling and Placement Policies

We used simulation to carry out an analysis of the impact that contiguous and quasi-contiguous allocation strategies have on scheduling performance. Our simulator implements two different scheduling policies (FCFS with and without backfilling), as well as three allocation algorithms (non-contiguous, contiguous, and quasi-contiguous) implemented for k-ary n-trees. The workloads used to feed the simulations have been obtained from actual supercomputers and are publicly available at the Parallel Workload Archive [11].

The details of the scheduling algorithms used in the experiments are as follows:

1. **First Come First Serve (FCFS):** In this policy, jobs are strictly processed in arrival order and executed as soon as there are enough available resources. The scheduling process is stopped until this condition is reached, even if there are enough free resources that could be allocated to other waiting jobs.
2. **Backfilling (BF):** This strategy permits the advance of jobs, even when they are not at the head of the queue, in such a way that system utilization increases, but without delaying the execution of the jobs that arrived first. The mechanism works as follows. A reservation for the first job in the queue is done, if enough resources are not currently available; the reservation time is computed taking into account the estimated termination time of currently running jobs. Other waiting jobs demanding fewer resources may be allowed to run while the first one is waiting. When the time of the reservation is reached, the waiting job has to run; if at that point resources are not available, some running, advanced jobs must be killed, because otherwise the reservation would be violated. This way, the starvation of the first job is avoided. Reservations are computed using a parameter called User

Estimated Runtime, which represents a user-provided estimation of the job execution time [12]. In some cases the scheduling system itself may provide this value, based on estimations made over the historical system logs [13].

Other scheduling methods have been proposed in the literature, such as SJF (Shortest Jobs First [1]) which selects the jobs to be executed by their size instead of their arrival time, and several variations of backfilling (see [1]). However, the most commonly used algorithm in production systems is the EASY backfilling [1], also known as aggressive backfilling. EASY performs reservations only over the first job in the queue. This is the policy used in this study.

Regarding the allocation algorithms, the following are included in the study:

1. **Non-contiguous:** This policy performs a search of free nodes making a sequential search over them, ignoring the locality. This is the most used technique in commercial systems, like the Cray XT3/XT4 systems, that simply gets the first available compute processors [10]. This scheme provides a flat vision of the network, ignoring its topological characteristics and the virtual topologies of scheduled applications [4]. Note that in the long run it behaves as a random allocation of resources.
2. **Contiguous:** In this scheme job processes are allocated to nodes maintaining them as close as possible. To minimize the distance between processes (nodes) in a k-ary n-tree, we have defined the concept of level of a job. This level is related to the number of stages in the tree (n), and the number of ports per switch (k up and k down) [6]. Stage 1 corresponds to switches at the bottom of the tree, *i.e.*, those directly connected to compute nodes. Small jobs of less than k nodes can be allocated to a collection of nodes attached to the same stage-1 switch, without requiring communication involving switches in upper stages of the tree. These are level-1 jobs. However, jobs larger than k will require the utilization of switches at stages 2, 3, etc. In general, up to k^i nodes can be allocated using stage-i switches.
3. **Quasi-contiguous:** This algorithm is a relaxed version of the previous one. It searches nodes that are contiguously allocated but, if the required number of free nodes is not found at the job level, it searches for the remaining nodes using switches *one* level above; contiguity is partly kept. The threshold of required-but-not-found free nodes that triggers the search on a higher level is a parameter provided to the algorithm, and the value providing best results is highly dependent on the size and type of the jobs that are executed in the systems. This parameter, which we call *qct* (quasi-contiguity threshold) is actually a percentage of the job size representing the number of tasks of that job allowed to be allocated using one extra level of the tree. Using this equation

$$max_{j \in J} = \left\lceil \frac{qct}{100} \times size_j \right\rceil \quad . \tag{1}$$

the algorithm computes $max_{j \in J}$, the maximum number of tasks of the job j allowed to be allocated using switches at the next level.

The utilization of additional stages of the tree may increase network contention, so we try to keep it under control by reducing the number of messages traversing high-level switches. To do so, we maintain the maximum possible number of nodes under switches belonging to the same level; actually, in favorable conditions this algorithm behaves exactly like the purely contiguous one. However, as some tasks can be assigned to non-contiguous portions of the network, external fragmentation is reduced. The *qct* threshold will maintain the number of quasi-contiguously allocated tasks limited, in order to reduce the interference created by the messages of different applications.

The contiguous algorithm starts computing the level to which the job belongs, and the size of this level (*level_ size*, the number of compute nodes below a single switch located at that level, which is the maximum size of a job that can be contiguously allocated below that level). After this preliminary step, the search of free nodes is performed, in groups of *level_ size* nodes following a first fit allocation scheme, because this way all the allocated nodes would be contiguous, that is, connected by the same switch or switches at the required level. If the complete tree is traversed but the necessary number of nodes has not been found, the job cannot be allocated. For example, in a 4-ary 3-tree topology, if we need

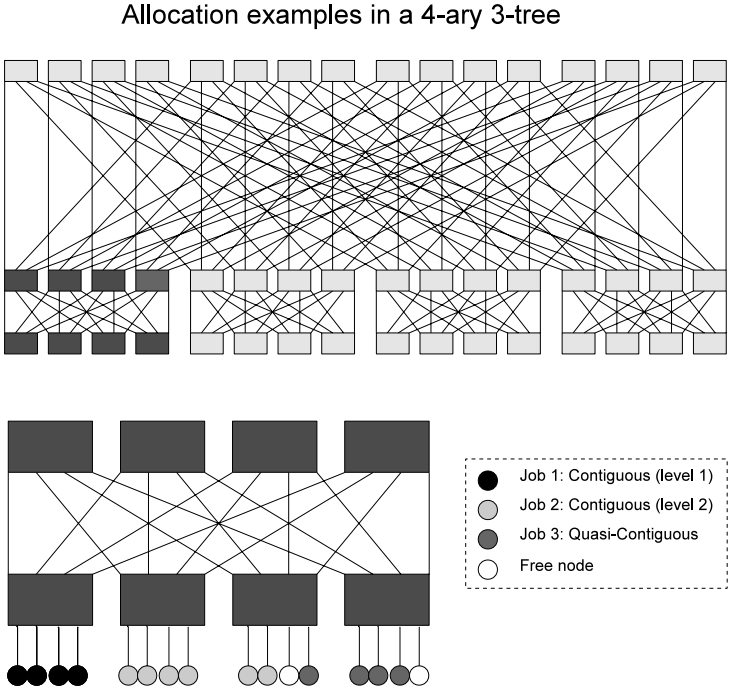


Fig. 1. Top: a 4-ary 3-tree; compute nodes are not represented for the sake of clarity. Bottom: a section of the network, with some examples of allocated jobs.

to allocate a 4-node job, we have to find a completely empty stage-1 switch. For a 6-node job (level-2) we need to find 6 free nodes that are connected using only stage-1 and stage-2 switches.

The quasi-contiguous algorithm requires two steps. Firstly, it performs a search for contiguous partitions as we stated before. If not found, because there are not enough free nodes at the job level, and the percentage of non-allocated tasks is below the *qct* threshold, the search continues in the level above. For example, in a 4-ary 3-tree topology, if we need to allocate a 4-node job, we start searching for completely empty stage-1 switches but, if none is available, another search is performed using stage-2 switches.

In Figure 1 we represent some simple allocation examples in a 4-ary 3-tree topology. We can observe how Job 1, of size 4, can be allocated into a single stage-1 switch; this is a contiguous allocation. The level of Job 2, of size 6, is 2; this means that it is allocated to two stage-1 switches that directly connected via switches at stage 2. Therefore, allocation of Job 2 is also contiguous. Job 3 is quasi-contiguously allocated because it should be a level-1 job (size is 4) but it requires the utilization of stage-2 switches.

4 Description of the Workloads

As we stated before, in this work we evaluate the performance of schedulers using logs of workloads extracted from real systems that are available from the PWA (Parallel Workload Archive, [11]). These logs have information about the system as described in the SWF format (Standard Workload Format) [14]. In this study we used the following fields:

1. **Arrival Time:** The timestamp at which a job arrives to the system queue. Logs are sorted by this field.
2. **Execution Time:** The interval of time that the job was running in the system. In order to simulate the improvement of performance due to the exploitation of communication locality, we scale this field by applying a speed-up factor.
3. **Processors:** Number of processors required by the job.
4. **User Estimated Runtime:** This information is used only by the backfilling scheduling policy and represents a user estimation of the job execution time.
5. **Status:** This field represents the status of a job. Jobs can fail, or be cancelled by the user or by the system, before or after they started the execution. Some studies do not include in the simulations those jobs that were not successfully completed (due to failure or cancellation), but we consider important all the jobs because they stayed in the queues, delaying the execution of other jobs.

In our experiments, all times were measured in minutes. We only used workloads that provide User Estimated Runtime information, because of the need of this parameter to perform a backfilling scheduling policy.

In [15], the authors suggested a metric to measure the *load* managed by the scheduler. Selecting workloads with different values of this metric allows us to check our proposals on different scenarios. The *load* is computed as follows:

$$load = \left(\frac{\sum_{j \in J} size_j \times runtime_j}{P \times (T_{end} - T_{start})} \right) \quad . \tag{2}$$

where P is the number of processors, J is the set of jobs between T_{start} and T_{end}, T_{end} is the last termination time and T_{start} is the last arrival time of the first 1% of the jobs. This 1% of firstly arrived jobs and the jobs that terminate after the last arrival are removed, in order to reduce warm up and cool down effects.

From the workloads available at the PWA, we have selected these three:

1. **HPC2N (High Performance Computing Center North).** This is a system located in Sweden, composed by 240 compute nodes and using the Maui scheduler. The workload log contains information of 527,371 jobs. Load: 0.62.
2. **LLNL Thunder (Lawrence Livermore National Laboratory).** This is a Linux cluster composed by 4008 processors in which the nodes are connected by a Quadrics network. The scheduler used in this system is Slurm. The log is composed by 128,662 job records. Load: 0.76.
3. **SDSC BLUE(San Diego Supercomputer Center).** This system is an IBM SP located in San Diego, with 1152 processors. The scheduler in use is Catalina, developed at SDSC, and performs backfilling. The log contains information of 243,314 jobs. Load: 0.86.

We simulated these workloads in k-ary n-trees adapted to each system sizes. For the first workload we have simulated a 4-ary 4-tree with 256 nodes. For the other two we have used a 4-ary 6-tree with 4096 nodes. The number of nodes of the topologies does not match with the nodes of the workloads, so we have considered that the extra processors are not installed and they are ignored in the simulation.

5 Costs and Benefits of Contiguous Allocation Policies

Parallel applications performance depends on many factors, such as the communication pattern, distance between the application tasks, network contention, etc. The first one is an application-dependent characteristic, but the others are affected by the way the application is allocated.

A contiguous allocation strategy reduces the distance between the application tasks, to accelerate the interchange of messages and to reduce network utilization. An important, additional effect is that interference with other running applications is also reduced. This interference, that causes contention for network resources, may result in severe performance drops. Therefore, the contiguous allocation of a job improves the overall performance of the system, not only of that job.

In [8], the authors evaluate the possible benefits of contiguity for a collection of parallel applications. These benefits are highly dependent on the communication patterns of the applications. However, as we will show, the search of contiguity can be very expensive in terms of scheduling time. The execution of jobs may be

delayed for a long time, until the required resources are available, the external fragmentation increases and the overall system utilization suffers. To minimize these negative effects we have introduced the concept of quasi-contiguity, a relaxed version of the contiguous allocation scheme which is expected to be less harmful in terms of scheduling time, while providing the same (or nearly the same) benefits in terms of application acceleration.

In order to validate the benefits of a contiguous and quasi-contiguous allocation policy, we have carried out several simulations using the INSEE simulator [16]. This tool does not simulate a scheduling algorithm, just the execution of a message-passing application on a multicomputer connected via an interconnection network. To feed this simulator we need traces of the messages interchanged by the communicating tasks. We have obtained these traces using a selection of the well-known NAS Parallel Benchmarks (NPB [17]). INSEE performs a detailed simulation of the interchange of the messages through the network, considering network characteristics (topology, routing algorithm) and application behavior (causality among messages). The output is a prediction of the time that would be required to process all the messages in the application, in the right order, and including causal relationships. Therefore, it only measures the communication costs, assuming infinite-speed CPUs. When using actual machines, a good portion of the time (ideally, most of the time) would be devoted

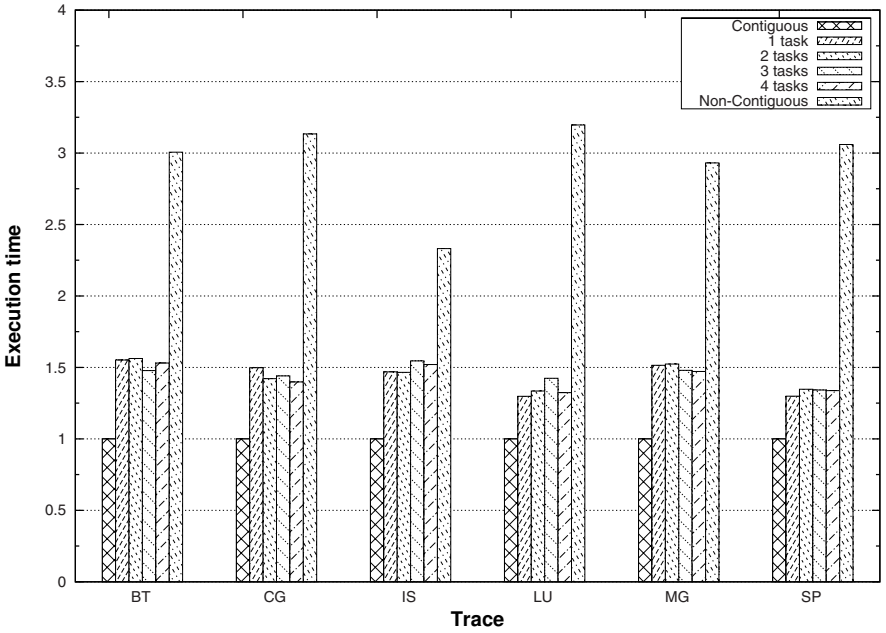


Fig. 2. Execution time for different allocation policies simulating the traces of some NAS Parallel Benchmarks in a 4-ary 4-tree topology. Values are normalized, so that 1 represents the contiguous allocation.

to CPU processing, and the impact of accelerated communications in overall execution time would be smaller.

The simulated topology is a 4-ary 4-tree, with 256 nodes. Instead of one application, we simulate the simultaneous execution of sixteen instances (jobs) of the same application (actually, trace), each one using sixteen nodes. The sixteen jobs have been allocated onto the network using three strategies:

1. **Contiguous:** Each job is allocated onto four level-2 switches, so the communications between tasks of the same job never need links or switches at level 3.
2. **Quasi-Contiguous:** In this strategy, we allow a partial non-contiguous allocation of the job tasks. The four experiments performed allow the non contiguous allocation of 1, 2, 3 or 4 tasks of each job, respectively.
3. **Non-Contiguous:** Tasks of each job are distributed along all the switches at level 4 (the maximum level of this tree). This means that intra-job communications do use level-4 switches, and also that messages of different jobs compete for network resources.

Figure 2 shows the execution time of each application using each strategy normalized to the time required by the contiguous placement. The benefits of contiguous allocation strategies are clear: non-contiguously allocated applications

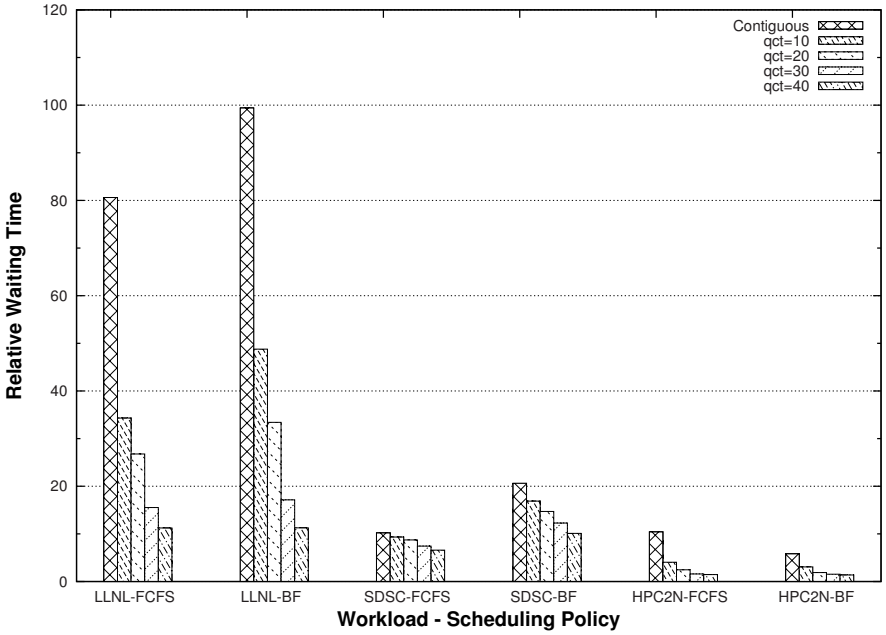


Fig. 3. Cost of contiguous and quasi-contiguous allocation, in terms of waiting time. A value of 1 would represent the average job waiting time for the non-contiguous allocation with the same scheduling policy.

run between 2 and 3 times slower. Regarding the quasi-contiguous allocation, we can appreciate that performance is always good, being only 30-50% higher to that obtained with purely contiguous allocation. These results confirm our expectations: a good allocation strategy can substantially reduce the execution time of a set of applications sharing a parallel computer as stated in [8].

Now we will asses the real cost of contiguity on scheduling. Using the scheduling simulator with the selected workloads (those from the PWA), we measure application waiting time for FCFS and backfilling scheduling algorithms, for purely contiguous allocation and quasi-contiguous allocation for four values of *qct*: 10, 20, 30 and 40%. Results are plotted in Figure 3. Note that values are relative to those obtained with the same workload and scheduling using non-contiguous allocation. Results are devastating: waiting times can be up to 100 times worse if contiguity is a requirement. Values are better for quasi-contiguity, but still bad. However, note that we *did not* take into consideration the acceleration that jobs experience due to better allocation. We will explore this issue in the next section. It is remarkable the difference between the LLNL workload waiting times and the other workloads waiting times, due to the presence of big size jobs (some of them of 1024 nodes). Finding contiguous partitions of this size is quite difficult, which results in longer waiting times for them and for the jobs that follow.

6 Tradding Off Costs and Benefits of Contiguous Allocation

In this section we carry out a collection of experiments to thoroughly evaluate the effect that contiguous allocation may have on scheduling performance. In these experiments we consider that contiguous allocation is able to accelerate the execution of parallel jobs. However, the actual values of attainable speed-ups are not available to us – they strongly depend on the communication characteristics of the applications, something that requires an exhaustive knowledge of each and all the applications included in the workload logs. We do not have that knowledge. For this reason, we introduce speed-up as a *parameter* of the simulation. With this setup we are able to know to what extent a certain level of application speed-up compensates the performance drop introduced by a restrictive allocation policy. This parameter is applied only to the parallel applications of the workload remaining the sequential jobs with the same runtime.

We have studied several combinations of scheduling and allocation policies. We evaluate them in terms of these two measurements:

1. **Job waiting time.** The time jobs spent in the queue.
2. **Job total time.** All the time spent in the system, which includes the time waiting at the queue and the execution time.

As stated before, when using contiguous and quasi-contiguous allocation, a speed-up factor has been applied to reduce the execution time. Note again that

applying a speed-up factor to a running time improves not only the application finish time, but also reduces the time spent by the jobs using system resources; and therefore, the scheduling performance is increased too. In the simulations we used the workloads from the PWA described in Section 4.

The quasi-contiguous strategy has been evaluated with four values of *qct*. Results are depicted in Figures 4, 5, 6 and 7. Note that, as the range of values is very wide, we used a logarithmic scale in the Y axis of all figures. We represent the averages of total time (waiting plus running) and, in some cases, waiting time alone. In each graph we can see six lines, one per allocation policy. Tested speed-up factors range from 0% to 50%. When this factor is 0% it means that, although the scheduler seeks contiguity, using it does not accelerate program execution. In all other cases we accelerate the execution times reported in the logs using the indicated speed-up factors (a value of 10% means that the execution requires 10% less time to be executed with that allocation scheme). Obviously, we cannot assume any acceleration with non-contiguous allocation, and for this reason the corresponding line is flat.

Let us now pay attention to Figure 4, where the LLNL workload is studied in detail. In all scheduling-allocation combinations, results with speed-up=0 are as appalling as described in the previous section. However, when this value increases (that is, when applications really run faster when allocated contiguous resources) the picture changes. At speed-up values between 5% - 30% the contiguous and quasi-contiguous approaches show their potential. It is clear that the quasi-contiguous strategies prove beneficial at lower speed-ups than the purely contiguous. Also, note that if the scheduler uses backfilling, global system efficiency is higher (the workload is processed faster), and the thresholds at which contiguity is advantageous are lower.

Figure 5 shows the results of the same experiments, but from a different perspective. Only waiting times are shown. A direct comparison with the previous figure help us to determine which part of the total time is spent in the queue, and which part is running time. For the cases with small speed-ups, most of the time is waiting time. When applying a speed-up factor, running time is accordingly reduced, but waiting time is also reduced.

In Figures 6 and 7 we have summarized results for workloads HPC2N and SDSC. To be succinct, and given that the qualitative analysis performed with LLNL is still valid, we only show results of total times for the FCFS and backfilling. For the SDSC workload, the threshold at which contiguous and non-contiguous allocation starts being beneficial falls between 15% and 25% (higher than that of LLNL). Similar, although slightly lower, values required by HPC2N are between 10% and 25%.

In all figures, we can see the benefits of using the quasi-contiguous policy. The scheduler performs better and, as described in the previous section, the expected speed-ups would be only slightly lower that those attainable with contiguous allocation. We have to remark that the implementation of this strategy tries always to find first a contiguous allocation, and only uses non-contiguous nodes as the last alternative. Therefore, if we estimate that we can obtain a certain

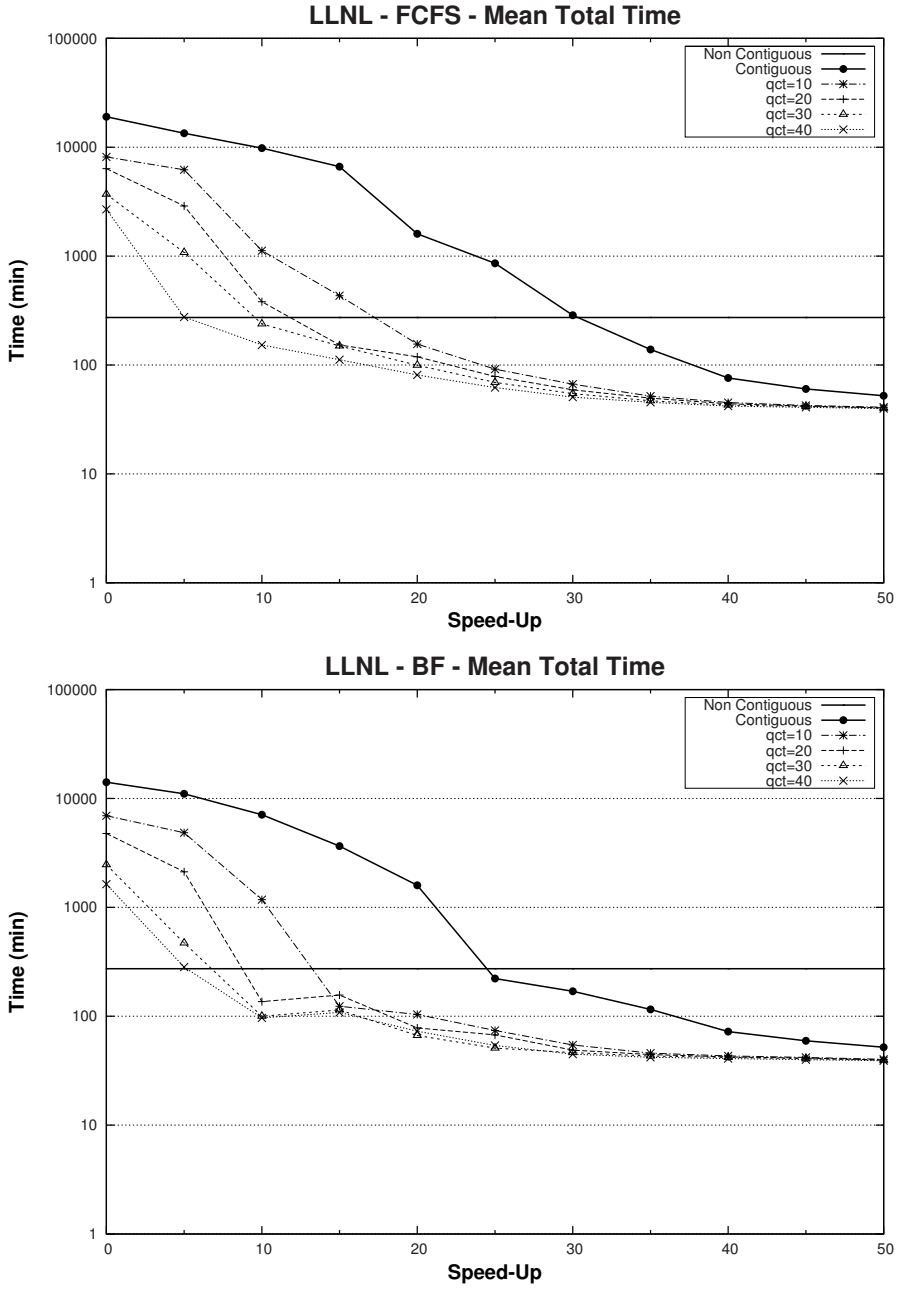


Fig. 4. Results of the experiments with the LLNL workload for FCFS and backfilling scheduling policies for various allocation strategies. Mean Total Time (Wait Time + Execution Time) at different speed-ups. The scale of the Y axis is logarithmic.

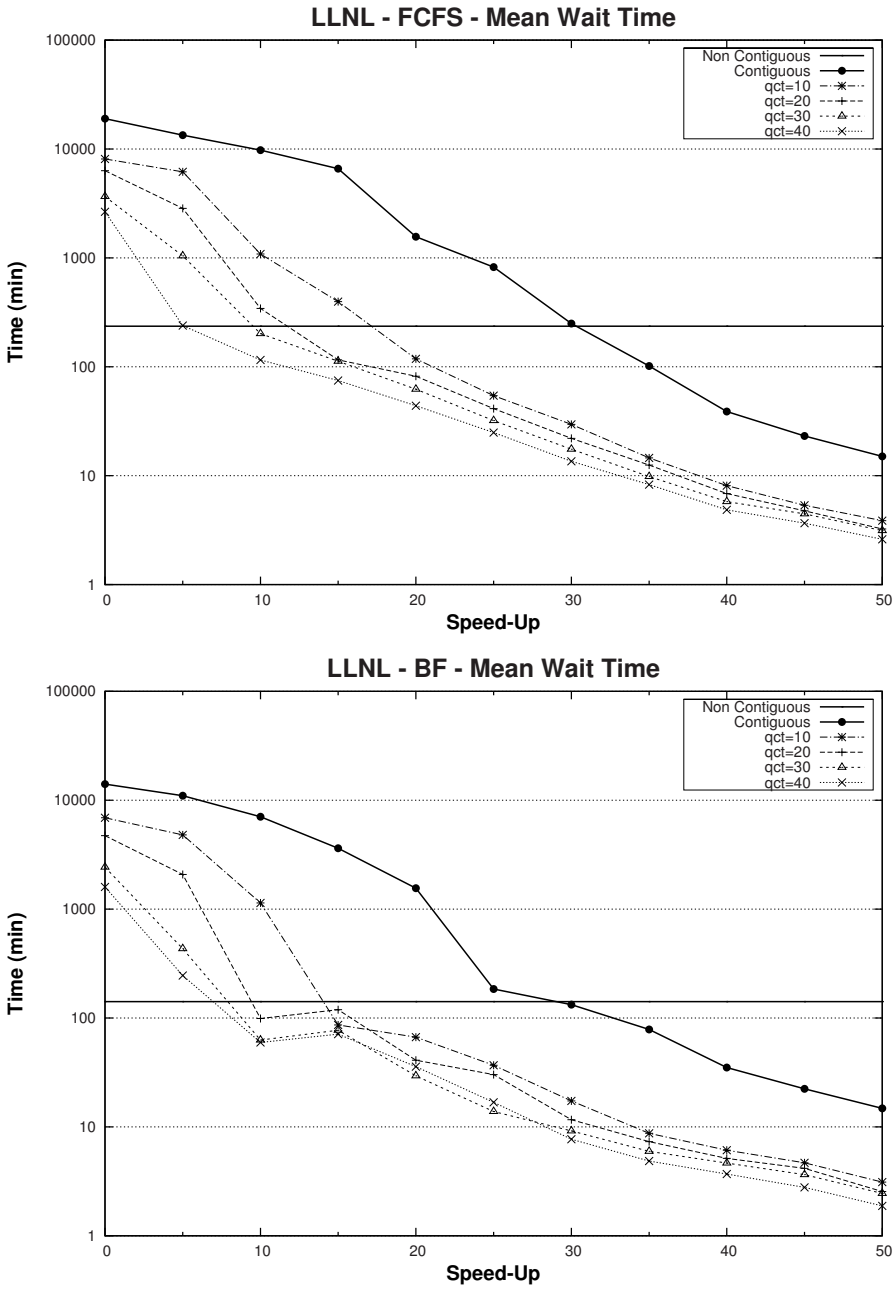


Fig. 5. Results of the experiments with the LLNL workload for FCFS and backfilling scheduling policies for various allocation strategies. Mean Wait Time at different speed-ups. The scale of the Y axis is logarithmic.

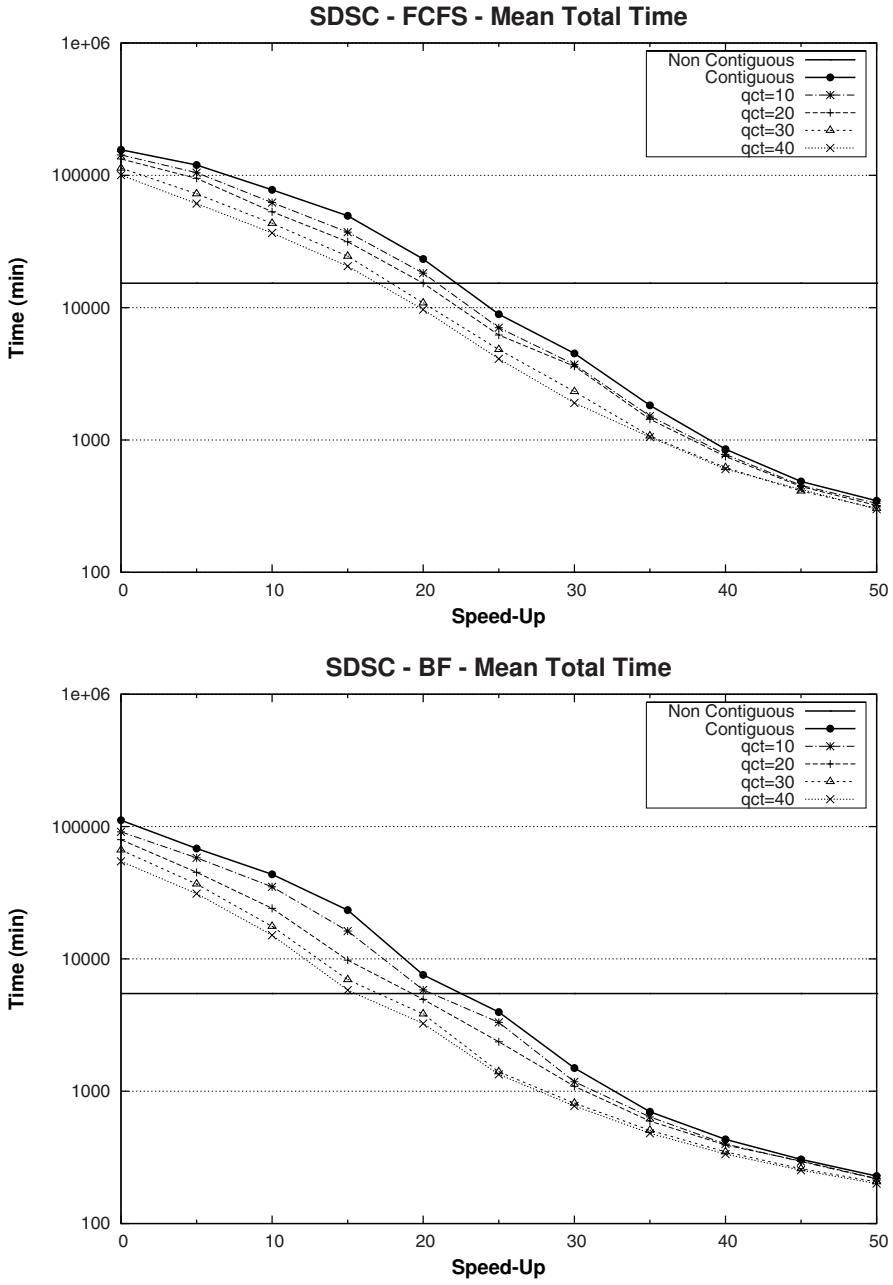


Fig. 6. Results of the experiments with the SDSC workloads for FCFS and backfilling scheduling policies for various allocation strategies. Mean Total Time (Wait Time + Execution Time) at different speed-ups. The scale of the Y axis is logarithmic.

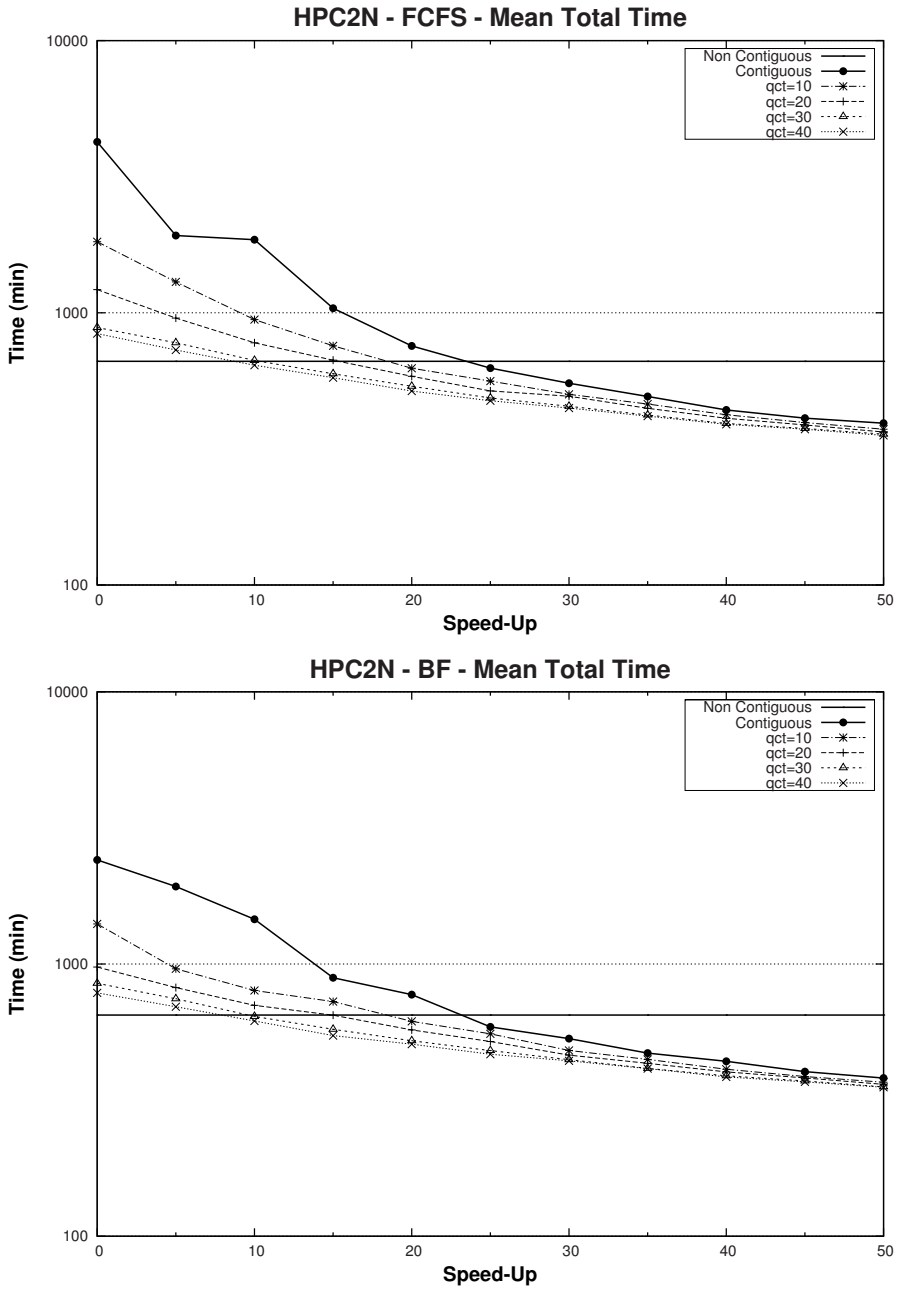


Fig. 7. Results of the experiments with the HPC2N workload for FCFS and backfilling scheduling policies for various allocation strategies. Mean Total Time (Wait Time + Execution Time) at different speed-ups. The scale of the Y axis is logarithmic.

speed-up when using a given value of *qct*, we will actually obtain better speed-ups, because in some cases the scheduler will obtain a contiguous allocation for the jobs.

Note that the increase of the *qct* parameter results in an equalization of the FCFS and backfilling performance reducing the difference between them. The reason is that the quasi-contiguous allocation strategy has a similar effect to the backfilling policy allowing the schedule of more jobs and thus, reducing the waiting time in the queue.

7 Conclusions and Future Work

Most current supercomputing sites are built around parallel systems shared between different users and applications. The optimal use of resources is a complex task, due to the heterogeneity in user and application demands: some users run short sequential applications, while others launch applications that use many nodes and need weeks to be completed.

Supercomputers are expensive to build and maintain, so that conscious administrators try to keep utilization as high as possible. However, the efficient use of a parallel computer cannot be measured only by the lack of unused nodes. Other utilization characteristics, although not that evident, may improve the general system performance.

In this paper we have studied the impact on performance of allocation and scheduling policies. We compared two scheduling techniques combined with three allocation algorithms in a k-ary n-tree network topology. Allocation algorithms that search for contiguous resources have an elevated cost in terms of system fragmentation, but also are able to accelerate the execution of applications. With the quasi-contiguous allocation, this acceleration is slightly penalized but the scheduling performance is significantly improved.

Experiments with actual workloads demonstrate that the cost of contiguous allocation is very high, but when the improvement of run time experienced by jobs is around 20-30%, this cost is compensated. Using relaxed versions of the contiguous allocation strategy (which we have called quasi-contiguous) this threshold lowers significantly, in such a way that in some cases speed-ups around 10% are enough to provide improvements in terms of scheduling efficiency.

This study has focused only in tree-based networks; the next step will be a performance study for other topologies (in particular, for k-ary n-cubes and k-ary n-tori). Because of the highly dependency of the allocation algorithms on the underlying topology, new quasi-contiguous allocation strategies should be developed for each new studied topology. We have provided application acceleration as a simulation parameter, although we know that the real acceleration depends heavily on the communication pattern of the applications, and on the way processes are mapped onto system nodes. For this reason, we plan to perform more complex simulations, in which the actual interchanges of messages are considered; to that end, we plan to integrate INSEE [16] into the scheduling simulator.

Finally, we plan to implement our allocation techniques into a real (commercial or free) scheduler in order to make real measurements in production environments with real applications.

References

1. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U.: Parallel job scheduling, – a status report. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2004. LNCS, vol. 3277, pp. 1–16. Springer, Heidelberg (2005)
2. Gupta, E.K.S., Srimani, P.K.: Subtori Allocation Strategies for Torus Connected Networks. In: Proc. IEEE 3rd Int'l Conf. on Algorithms and Architectures for Parallel Processing, pp. 287–294 (1997)
3. Choo, H., Yoo, S.M., Youn, H.Y.: Processor Scheduling and Allocation for 3D Torus Multicomputer Systems. IEEE Transactions on Parallel and Distributed Systems 11(5), 475–484 (2000)
4. Mao, W., Chen, J., Watson, W.I.: Efficient Subtorus Processor Allocation in a Multi-Dimensional Torus. In: HPCASIA 2005: Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region, Washington, DC, USA, p. 53. IEEE Computer Society, Los Alamitos (2005)
5. Lo, V., Windisch, K., Liu, W., Nitzberg, B.: Noncontiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers. IEEE Transactions on Parallel and Distributed Systems 8, 712–726 (1997)
6. Petrini, F., Vanneschi, M.: Performance Analysis of Minimal Adaptive Wormhole Routing with Time-Dependent Deadlock Recovery. In: IPPS 1997: Proceedings of the 11th International Symposium on Parallel Processing, Washington, DC, USA, p. 589. IEEE Computer Society, Los Alamitos (1997)
7. Bhatele, A., Kale, L.V.: Application-specific Topology-aware Mapping for Three Dimensional Topologies. In: Proceedings of Workshop on Large-Scale Parallel Processing (held as part of IPDPS 2008) (2008)
8. Navaridas, J., Pascual, J.A., Miguel-Alonso, J.: Effects of Job and Task Placement on the Performance of Parallel Scientific Applications. In: Proc 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Weimar, Germany (February 2009)
9. Aridor, Y., Domany, T., Goldshmidt, O., Moreira, J.E., Shmueli, E.: Resource Allocation and Utilization in the Blue Gene/L Supercomputer. IBM Journal of Research and Development 49(2–3), 425–436 (2005)
10. Ansaloni, R.: The Cray XT4 Programming Environment, `http://www.csc.fi/english/csc/courses/programming/`
11. PWA: Parallel workloads archive, `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`
12. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Modeling User Runtime Estimates. In: Feitelson, D.G., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2005. LNCS, vol. 3834, pp. 1–35. Springer, Heidelberg (2005)
13. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling Using System-Generated Predictions Rather than User Runtime Estimates. IEEE Trans. Parallel Distrib. Syst. 18(6), 789–803 (2007)

14. Chapin, S.J., Cirne, W., Feitelson, D.G., Jones, J.P., Leutenegger, S.T., Schwiegelshohn, U., Smith, W., Talby, D.: Benchmarks and standards for the evaluation of parallel job schedulers. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1999, IPPS-WS 1999, and SPDP-WS 1999. LNCS, vol. 1659, pp. 67–90. Springer, Heidelberg (1999)
15. Tsafrir, D.: Modeling, Evaluating, and Improving the Performance of Supercomputer Scheduling. PhD thesis, School of Computer Science and Engineering, the Hebrew University, Jerusalem, Israel (September 2006) Technical Report 2006–78
16. Ridruejo, F.J., Miguel-Alonso, J.: INSEE: An Interconnection Network Simulation and Evaluation Environment. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1014–1023. Springer, Heidelberg (2005)
17. NASA Advanced Supercomputer (NAS) division: Nas parallel benchmarks, `http://www.nas.nasa.gov/Resources/Software/npb.html`