

# Analyzing the Performance of Allocation Strategies based on Space-Filling Curves

Jose A. Pascual<sup>1\*\*</sup>, Jose A. Lozano<sup>1,2</sup>, and Jose Miguel-Alonso<sup>1</sup>

<sup>1</sup> Intelligent Systems Group (ISG)

University of the Basque Country UPV/EHU

<sup>2</sup> Basque Center for Applied Mathematics (BCAM)

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

**Abstract.** Future exascale supercomputers will be composed of thousands of nodes. In those massive systems, the search for physically close nodes will become essential to deliver an optimal environment to execute parallel applications. Schedulers manage those resources, shared by many users and jobs, searching for partitions in which jobs will run. Significant effort has been devoted to develop allocation strategies that maximize system utilization, while providing partitions that are adequate for the communication demands of applications. In this paper we evaluate a class of strategies based on space-filling curves (SFCs) that search for partitions in which nodes are physically close, compared to other alternatives that relax this requirement (e.g. non-contiguous), or make it even more strict (e.g. contiguous). Several metrics are used to assess the quality of an allocation strategy, some based on system utilization, some others measuring the quality of the resulting partitions. Contiguous allocators suffer from severe degradation in terms of system utilization, while non-contiguous allocators provide inadequate partitions. Somewhere in the middle, SFC allocators offer good system utilization while using quite compact partitions. The final metric to decide which allocator is the best depend on the severity of the slowdown suffered by applications when running in non-optimal partitions.

**Keywords:** Space-filling curves, scheduling, allocation, partitioning, contiguity, non-contiguity.

## 1 Introduction

In the coming years, supercomputer vendors will deliver massive exascale systems with many thousands of nodes (millions of computing cores) to execute parallel jobs (applications). These jobs are composed of tasks that communicate among them using an underlying fabric: an interconnection network (IN) which determines the way compute nodes are connected.

Most supercomputers are shared by many users, who request the execution of jobs through a submission queue. The *scheduler* is in charge of selecting the

---

\*\* This author is currently with the APT group in The University of Manchester.

job or jobs to run, following a given policy. The most common scheduling policies are First Come First Serve (FCFS) [6] and Backfilling [6]. Often, several jobs can fit in the system simultaneously, as the size of a job is normally smaller than the size of the complete system (in terms of compute cores).

Once a job is selected, an *allocator* must find a set of free nodes (a partition) and perform the mapping of job tasks onto system nodes. An allocation strategy is used to carry out the search. We can differentiate two broad classes of strategies. *Contiguous* strategies look for convex sets of free nodes, normally with hyper-rectangular shapes. *Non-contiguous* strategies remove this shape restriction. Contiguous strategies try to reduce the execution time of jobs, as they allocate partitions with very low inter-node distance, and free of interference from other partitions in which other jobs run. However, they can cause *internal fragmentation*, as they normally reserve for a job a set of nodes larger than the number of job’s tasks. *External fragmentation* is also common, when enough nodes are available to run a job, but they are not arranged with the required shape. Thus, the price of contiguity is a low level of system utilization. For this reason, non-contiguous strategies were developed [14] [25] [15]: jobs *may* run in sub-optimal conditions, as inter-node distances are longer and communications overlap (jobs are not isolated), but fragmentation is minimized (system utilization is greatly improved) and, at the end, the overall system performance in terms of throughput of jobs *should* be improved. Therefore, different allocation strategies search for different trade-offs between job performance vs. system utilization. Achieved job throughput depends on both factors.

An issue that should not be ignored is the impact, in terms of performance, of the way job tasks are mapped onto the nodes of the allocated partition [19] [18] [3]. The benefits of contiguous strategies are maximized only with good mappings that optimize the inter-application communications [17]. Mappings in which tasks are not physically close, and need to contend for channels with messages of other jobs, are the reason behind the reduced performance of non-contiguous allocation strategies.

We consider in this paper another class of allocation strategies that fit somewhere in the middle between contiguous and non-contiguous as defined above, and are based on Space-filling Curves (SFC) [13]. These SFC strategies “see” the supercomputer as a linear list of nodes, and perform contiguous allocation in this 1D space. Therefore, partitions are sub-lists of consecutive nodes [9] [13]. Then 1D lists are mapped onto a higher-dimensional space, in a way that depends on the selected space-filling curve [1] [11]. These mappings do not guarantee that the resulting partition in the  $nD$  space is consecutive and convex. However, they are designed to keep *locality* between nodes: they are physically close. Compared to pure contiguous mappings, SFC mappings are better in terms of utilization, as internal fragmentation does not occur (the allocator can always search for a 1D list with the required number of nodes) and external fragmentation is less severe. It remains to be verified if the locality guaranteed by SFC allocation is good enough to provide a good execution environment for parallel jobs, matching (or getting close to) that of contiguous allocation.

In summary, in this paper we evaluate how SFC-based allocations trade-off per-partition benefits (locality, isolation) with system-wide benefits (mainly, utilization), when used in supercomputers built around interconnection networks with  $n$ D-mesh shapes. To provide a context, we compare them with a convex, contiguous strategy and with a non-contiguous strategy. For this study we use a diverse collection of metrics. Some are indicators of the quality of the partitions, hinting how well applications would run on them. Others measure the performance of the system-wide scheduling process. The evaluation of all the strategies has been performed using simulation, fed with a large collection of workloads generated synthetically. Our experiments verify the intuitions outlined in this introduction, showing how non-contiguous and SFC based strategies perform very well in terms of system utilization but, for other metrics that consider fitness of partitions to applications, contiguous allocation is better. In order to provide an answer to the question “which strategy is the best in terms of job throughput?”, we only provide a partial answer: it depends on the behavior of the applications that constitute the workload, when executed in differently shaped partitions.

The rest of the paper is organized as follows. Section 2 describes the metrics used to compare allocation strategies. In Section 3 we describe the SFC strategy. In Section 4 we describe more formally the scheduling, allocation and search strategies under evaluation. The workloads used in the experiments are described in Section 5, where we provide additional details about the experimentation set-up. In Section 6 we discuss system-wide results of the different strategies, and we continue in Section 7 with an analysis of the quality of the delivered partitions. Section 8 is devoted to the search of a trade-off between application slowdown (due to the use of non-optimal partitions) and system utilization. Section 9 closes the paper with some conclusions and future lines of research.

## 2 Performance Metrics

We measure allocation strategies using two groups of metrics, the first focused on system utilization, and the second focused on the quality of the partitions.

### 2.1 Scheduling-focused metrics

- **Utilization** indicates the average ratio of active nodes during a measuring time of interest. A node is active if it has been allocated to a running job. Using only utilization to assess system-wide performance can be deceptive, as a strategy with low utilization but that allows faster execution of applications can result in better job throughput [22]. However, it is an excellent indicator of the overhead that results from the use of strategies that search for contiguity or locality.
- **Makespan**: It represents the total time required to process a given input workload. If we do not take into account the effects of partition shape on execution speed, as we do in our experiments, this metric also indicates the cost of looking for contiguity or locality.

Note that these two metrics are related with others not included here, such as **fragmentation** (internal and external). Higher degrees of fragmentation result in lower utilization, and longer makespan.

## 2.2 Partition-focused metrics

The metrics described here depend strongly on the characteristics (topology) of the underlying interconnection network. For the purpose of this evaluation, we focus on  $n$ D meshes (they could be easily extended to tori). Given a partition  $P$  (with an arbitrary shape, convex or not) composed of  $S = |P|$  compute nodes of coordinates  $\mathbf{a} = (a^1, \dots, a^n)$ , being  $n$  the number of dimensions of the network, and being  $d(\mathbf{a}_i, \mathbf{a}_j)$  the Manhattan distance (number of hops) between nodes  $\mathbf{a}_i$  and  $\mathbf{a}_j$  of the partition, we define the following metrics:

1. **Average pairwise distance (APD)**: Average distance between all pairs of nodes in  $P$ .

$$APD = 2 \times \frac{\sum_{i=1}^S \sum_{j=i+1}^S d(\mathbf{a}_i, \mathbf{a}_j)}{(S+1) \times S} \quad (1)$$

2. **Number of affected nodes**: Size of the area covered by the partition, thus the number of nodes that may be participating in the communications. If the partition is not convex, the affected area may include nodes assigned to other running applications.

$$NA = \prod_{i=1}^n \left( \max_{\mathbf{a} \in P} \{a^k\} - \min_{\mathbf{a} \in P} \{a^k\} + 1 \right) \quad (2)$$

where  $\max_{\mathbf{a} \in P} \{a^k\}$  and  $\min_{\mathbf{a} \in P} \{a^k\}$  are the maximal and minimal coordinates in the  $k$ -th dimension of all nodes  $\mathbf{a}$  in the partition.

In Figure 1 we have represented three partitions and the nodes that will be affected by the communications. As we can see, in the first contiguous partition (Figure 1a) all communications remain internal, without affecting neighboring jobs. The second and third non-contiguous partitions (Figures 1b and 1c) show how the affected area extends outside the partition. In Figure 1d, which represents the three partitions put together, we can see how the affected areas of the three partitions are overlapping.

Low values of APD are expected to correlate with reduced execution times of applications running in the partition. However, as explained in [20], this correlation is direct only if the application use an all-to-all communications pattern. The extent of which jobs benefit from good distance-related metrics depends strongly on the characteristics of the application and *the applied mapping*. Also, interference from other applications, that can be severe if all partitions have large numbers of affected nodes, have an important bearing on the performance of the communications [10]. The assessment of application run times falls outside the scope of this paper.

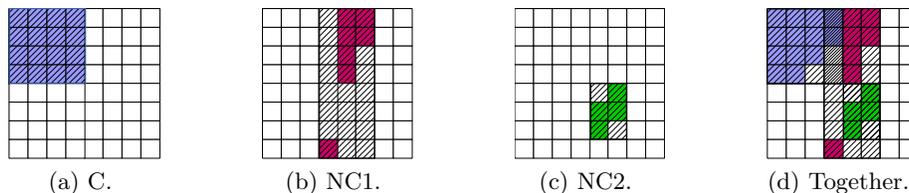


Fig. 1: Nodes affected by the communications of three applications allocated contiguous and non-contiguously.

### 3 Space-filling Curves

A space-filling curve (SFC) maps a one-dimensional list of points onto a  $n$ D hypervolume. The first of this kind of curves was discovered by Hilbert [7], but others have been developed such as the Z-order curve. The first version of the Hilbert curve performed only 1D to 2D mappings, but it was later extended to higher dimensions [2]. The Z-order curve was able to perform multi-dimensional mappings since the beginning.

The idea of using SFCs to map parallel jobs onto network nodes was first introduced in [14]. With this approach, network nodes are ordered using a *rank*. Allocation (search of partitions) is done in this 1D, rank-ordered list, instead of using, for example, the 2D coordinates. A 1D partition is afterwards mapped onto the actual  $n$ D space using the transformations defined by the chosen SFC. Two are the main advantages of these SFC allocation strategies: the search is simple, as it deals with 1D structures, and the resulting  $n$ D partitions are very compact, keeping high levels of locality. In Figure 2 we have represented some examples of mappings from a 1D space to 2D and 3D spaces, using the two different SFCs. In the upper side of the figure, we show the consecutive sets of nodes (partitions) resulted from a 1D allocation. Below we see the same partitions when mapped to 2D and 3D, using the Z-order curve (left) and the Hilbert curve (right). Next we explain how these mappings are performed.

- The **Z-order curve** [16] is a function that maps multi-dimensional points by interleaving the binary representation of their coordinate values. For example, the point (2,4) in 2D would be mapped to the point (1D) with z value 010-100 (010100). The use of this curve preserves locality between points, but does not guarantee contiguity.
- The **Hilbert curve** is a function that traverses the polyhedron vertices of an  $n$ -dimensional hypercube in Gray code order [24]. For example, in 2D the sequence of gray codes (0,0), (0,1), (1,1), (1,0) corresponds to the 1D points: 0, 1, 3, 2. This curve preserves the contiguity and locality between the nodes.

The objective of SFC allocation strategies is to obtain  $n$ D partitions with good locality (to benefit inter-task communications in the interconnection network). This locality has been evaluated in [13] [23] for 2D and 3D networks. Now we extend this study to higher dimensions. We have measured the locality

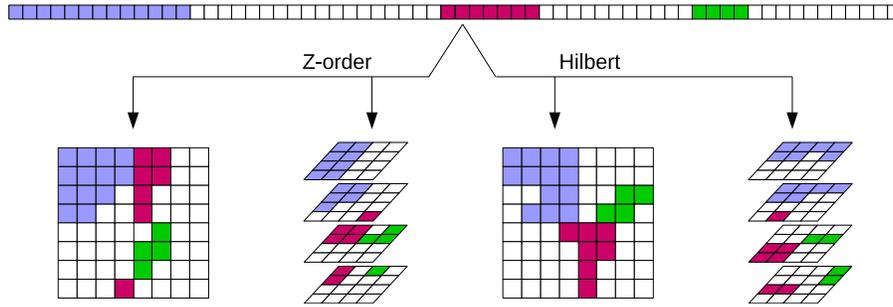


Fig. 2: Mappings of 1D consecutive partitions onto 2D and 3D spaces using Z-order and Hilbert space filling curves.

	1D	2D		3D		4D		5D		6D	
	-	Z	H	Z	H	Z	H	Z	H	Z	H
Set 1	337.24	27.58	22.08	12.35	10.40	8.69	7.64	7.42	6.45	6.55	6.00
Set 2	148.81	19.51	14.50	9.54	7.47	7.18	5.97	6.20	5.14	5.65	4.88
Set 3	81.02	13.97	10.73	7.89	6.26	6.15	5.04	5.42	4.62	5.04	4.34
Set 4	48.70	12.28	8.66	7.13	5.46	5.59	4.57	5.03	4.22	4.70	3.93
Set 5	21.88	7.01	5.55	4.73	3.92	3.98	3.42	3.70	3.20	3.54	3.07

Table 1: Locality, expressed as APD, for five sets of partitions that are consecutive in 1D and then mapped to  $n$ D using the Z-order curve (Z) and the Hilbert curve (H). The higher the set identifier, the smaller the average size of the partitions within.

achieved by both curves when mapping to 2D ( $64 \times 64$ ), 3D ( $16 \times 16 \times 16$ ), 4D ( $8 \times 8 \times 8 \times 8$ ), 5D ( $4 \times 4 \times 4 \times 4 \times 4$ ) and 6D ( $4 \times 4 \times 4 \times 4 \times 4 \times 4$ ) meshes. We generated five sets of 1D contiguous partitions, labeling each set as  $k=1..5$ . Each set  $k$  contains thirty partitions of different sizes, being the maximum size  $\frac{4096}{2^k}$ . This means that Set 1 contains much larger partitions than Set 5.

We first calculated the locality of these 1D sets in terms of APD. Next we did the same after the mapping to  $n$ D, using both Z-order and Hilbert curves. Results are summarized in Table 1. It is clear that locality in the mapped partitions increases with the number of dimensions in the IN. This is due to the increased degree of the network nodes. The results also indicate that the Hilbert curve preserves locality better than the Z-order curve, mapping the points to closer locations, for any number of dimensions. Note that these are static experiments, that neither consider the complete scheduling process nor the way locality affects job's run times. We will explore these issues later in this paper. We want to remark that Hilbert-based allocation is used in the SLURM scheduler [12].

## 4 Scheduling Policies and Allocation Algorithms

In this section we further explore the scheduling process, with focus on allocations algorithms. Scheduling consists of determining which queued job (submitted by a user) will be selected for execution. This is carried out following a established policy [6] such as FCFS, backfilling, Shortest Job First (SJF), etc. The most used policy is backfilling, which tries to avoid a “head-of-line blocking” problem of FCFS. While FCFS selects the jobs in strictly order of arrival, backfilling allows to advance jobs when the job at the head of the queue cannot be executed because the necessary resources are not available. However, a job is allowed to be advanced only when its execution is not expected to delay the starting time of the job at the head of the queue. A requisite to implement backfilling is, thus, an estimation of the run time for the jobs. Normally, users are expected to provide this estimation. Backfilling improves system utilization while respecting submission order.

Once a job has been selected, the allocator is in charge of reserving the set of nodes onto which the tasks of the job will be mapped. In this work we consider contiguous strategies, non-contiguous strategies, and SFC-based strategies.

### 4.1 Contiguous allocation with hyper-rectangular partitions

These kind of strategies look for convex partitions with shapes  $a \times b$ ,  $a \times b \times c$ ,  $a \times b \times c \times d$ , etc. depending on the dimensionality of the underlying IN. These partitions result in optimal values of distance-related metrics. Furthermore, they provide a very desirable property: isolation. This means that partitions do not overlap, and inter-task communications in a job are implemented in the IN without requiring the intervention of nodes in other partitions. In other words, communications in different partitions do not interfere. Both properties together make this kind of partitions the optimal place to execute communication-intensive parallel applications – the ones expected to run in a supercomputer [19] [22]. However, looking for contiguous partitions is not cheap (due to external fragmentation), and the overall system utilization is drastically reduced (when compared with other alternatives).

Searching for a partition of a particular shape requires traversing the system (a data structure representing it) in a particular order. The First-Fit (FF) policy stops the search as soon as the suitable partition has been found – or when the search ends unsuccessfully. In this work we use a search algorithm that implements this policy, called Improved First Fit (IFF) [21]. It searches for hyper-rectangles in multi-dimensional cube networks.

### 4.2 Allocation strategies based on Space-Filling Curves

These strategies are contiguous and consecutive in the 1D representation of the system, but result in non-convex shapes when mapped to higher dimensions. However, as shown in the previous section, they result in partitions with high levels of locality: good values in terms of distance-related metrics. These values

are not as good as those provided by the contiguous strategy, and resulting partitions (communications) do overlap. The achieved benefit of SFC allocation comes in terms of high levels of system utilization.

The search of partitions can be done using First Fit (FF), Best Fit (BF) or other strategy. We consider in this work:

- Strategies based on Z-order curves, searching with FF (ZORD-FF) and BF (ZORD-BF). Note that these strategies take into account locality, but may result in non-contiguous, non-convex partitions.
- Strategies based on Hilbert curves, searching with FF (HILB-FF) and BF (HILB-BF). The resulting partitions are contiguous, but convexity is not guaranteed.

### 4.3 Non-contiguous strategy

We have also evaluated a simple, non-contiguous allocation strategy with FF search (NC-FF). It looks for free nodes using the node identifier, without any special consideration. We will see that it results in excellent results in terms of utilization, but bad per-partition, distance-related metrics.

### 4.4 Mapping tasks onto partitions

We insist in this point: once the scheduler has a job (collection of tasks) and a partition (collection of nodes), it is necessary to map tasks onto nodes. This stage has a huge impact on the performance of applications [19] [3], but evaluating this effect would be very costly and is beyond the scope of this paper. We leave this as future work, and we use here a simple, consecutive mapping strategy: tasks are assigned to nodes consecutively using their identifiers.

## 5 Experimental Set-up and Workloads

In this section we describe the simulation-based evaluation environment used in this work. A fundamental part of experiments with simulators is the collection of workloads used to feed them. Thus, we start describing the workloads.

### 5.1 Workloads

We have used several, synthetically-generated, *workloads*. A workload is defined as a sequence of (parallel) jobs submitted to the system, and includes the following per-job pieces of information:

1. **Size:** The number of nodes requested to run the job.
2. **Shape:** If the scheduler uses a contiguous allocation strategy, then the shape of the requested hyper-rectangle must be supplied. If not specified, the scheduler use the job size and generate a valid hyper-rectangle. Note that if size and shape do not match in terms of number of nodes, there will be internal fragmentation.

3. **Duration:** This value must be provided as part of the workload because we are simulating only the scheduling mechanisms – but it should be the result of the execution of the job in the assigned partition. Thus, for the experiments reported in this work, it simply matches the estimated run time, and does not reflect any effect of the partitioning (and mapping) strategies on execution time.
4. **Estimated run time:** Required when using a scheduler implementing back-filling. In this evaluation, we assume that this time is the real execution time (duration).

As we can see, we do not include in the workload the arrival time of jobs. We consider a situation of maximum input load in which all jobs arrive simultaneously *but ordered* to the waiting queue, thus emulating a production system in which there are always several jobs awaiting. This is to avoid situations of low system utilization due to an empty queue. It also provides a meaning to the makespan metric: the time to consume the full workload.

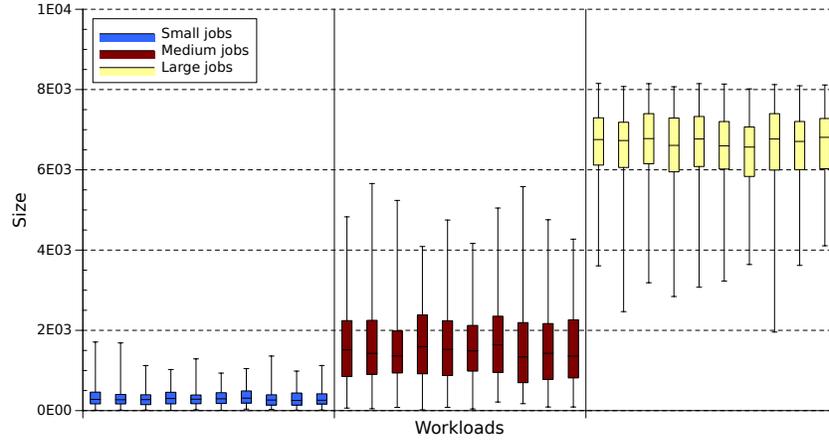
We use beta distributions with different parameters to generate sizes and/or shapes. The generation of hyper-rectangles is not trivial because, for a given job size, several shapes can be valid to contain it. For example, 16 nodes can be arranged as a  $4 \times 4$  or  $8 \times 2$  [22]. Moreover, some sizes such as 7 can only be arranged contiguously as a  $7 \times 1$  partition if internal fragmentation is not allowed. Considering this fact, we have defined two different types of workloads (Figure 3):

- **Unshaped workloads:** The workload includes job sizes, but does not specify shapes. The contiguous scheduler generates a valid shape automatically, as the smallest  $n$ D cube (shaped  $a \times a \times \dots \times a$ ) able to host the job, where  $n$  matches the dimensionality of the IN. Using this criterion, partitions are symmetrical and compact, but internal fragmentation may be severe.
- **Shaped workloads:** The workload includes a per-job shape specification, and the job size is just the number of nodes in this shape. Thus, there is not internal fragmentation. This is assumed to be the best way of running applications, as the user has selected a shape that, supposedly, optimizes inter-task communications.

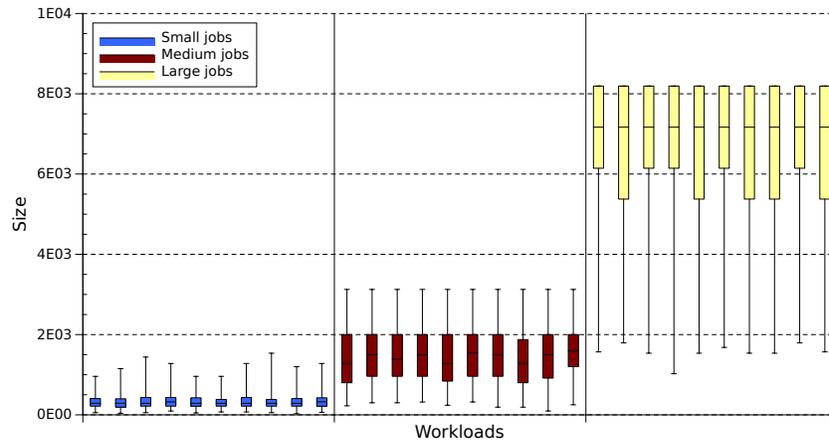
For each type of workload we have generated three sets of 150 jobs, using in each of them a different average job size: small (S), medium (M) and large (L). This has been carried out limiting the maximum size that a partition can have. The resulting size distributions are represented in Figures 3a and 3b respectively. Finally the duration of the jobs must be generated. In this case we generate 10 different durations for a job. Considering all together, we have managed  $2 \text{ types} \times 3 \text{ average sizes} \times 10 \text{ durations} = 60$  workloads of 150 jobs each.

## 5.2 Experimental set-up

We have analyzed the different schedulers using an in-house developed scheduling simulator that takes as input parameters a workload, a scheduling policy (such as



(a) Unshaped workloads.



(b) Shaped workloads.

Fig. 3: Boxplot of the two types of workloads used to evaluate allocation strategies. Each type is composed of three sets (large, medium and small) of 10 workloads each. The figure shows the minimum, the maximum, the median and the first and third quartile.

FCFS or backfilling), an allocation strategy (contiguous, non-contiguous, SFC-based, etc.) and, if required by the experiment, a slow-down factor to be applied to the duration of the jobs specified in the input workload. We will see later the usefulness of this parameter. The simulator’s output reports in a set of metrics, including those explained in Section 2.

In all the experiments we use an implementation of backfilling called “conservative” [6]. Then we consider six different allocation strategies and the 60

workloads defined above. Reported results are averages of the metrics obtained for the 10 sets of job durations.

As underlying INs we consider a 3D ( $32 \times 32 \times 32$ ) and 5D ( $8 \times 8 \times 8 \times 8 \times 8$ ) mesh, both with 32768 nodes. These dimensions have been chosen because they are used by supercomputers currently operational i.e. Blue Gene P (3D) [8], Blue Gene/Q [4] from IBM, Cray systems with the 3D Gemini interconnect [5], etc. We evaluated both meshes and tori but, as the main results are not significantly different, and for the sake of brevity, we only report results with meshes.

## 6 Analyzing System-wide Results

Experiments were designed to understand to what extent the search of contiguity or locality has a bearing on system utilization and makespan. In Figure 4 we have represented the results (utilization and makespan) obtained for the different simulation configurations and workloads. Note that ZORD-FF and HILB-FF are represented together as SFC-FF, because they yield identical results (the same applies to ZORD-BF and HILB-BF, summarized as SFC-BF). This is a direct consequence of the use of the same search strategy over the same 1D space, and the topology of the IN is irrelevant – differences will appear when evaluating the resulting  $n$ D partitions. Results labeled as NC-FF correspond to the non-contiguous strategy and, again, do not depend on the IN topology. CONT3D and CONT5D correspond to the contiguous strategy in the 3D and the 5D mesh respectively.

Let us start focusing on unshaped workloads: those in which the user specifies only a job size. The contiguous scheduler tries to find a  $n$ D rectangular partition for it. This process is expected to hurt performance severely, particularly for high values of  $n$ , due to the effects of internal fragmentation. Results, summarized in Figure 4 show this effect very clearly. Utilization with hyper-rectangular partitions is, in general, very poor, being negatively affected by the dimensionality of the IN and the average job size. The first factor determines the internal fragmentation, and the second has a bearing on external fragmentation. Makespan values confirm these findings. They are longer for workloads with larger average job size, because each job requires more resources and, thus, fewer jobs can run simultaneously.

At the other end of the spectrum, NC-FF yields excellent results, independently of the underlying topology. When we relax all kinds of shape or locality expectations in the partitions, the probability of finding a free set of nodes fitting a job request is drastically increased. This is especially noticeable when dealing with medium to large jobs.

Locality aware, SFC-based strategies show excellent results, close to those of NC-FF for small jobs, although slightly worse for larger jobs. The search strategy does not play a significant role, with FF and BF search performing similarly. Thus, the increased cost of the exhaustive search done by BF does not provide any benefit.

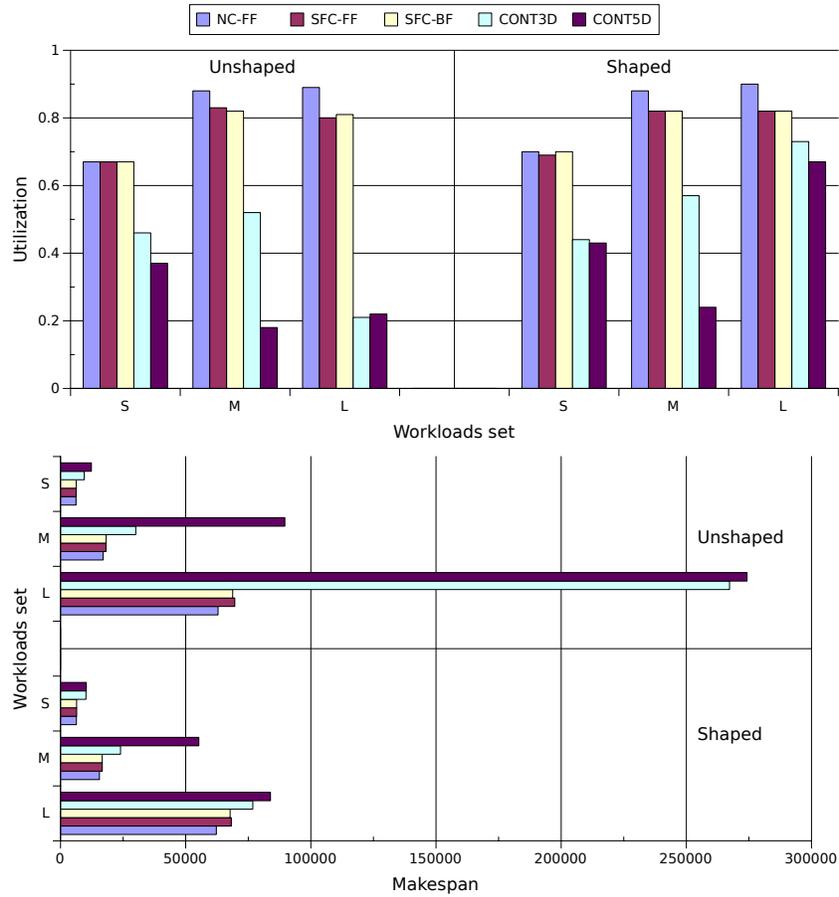


Fig. 4: Representation of the utilization and makespan obtained by the allocation strategies using both unshaped and shaped workloads and both 3D and 5D networks.

Finding specific shapes is more costly than finding arbitrary node sets, and result in higher levels of external fragmentation. We have not measured this effect (fragmentation), but have characterized this cost by measuring how often the scheduler tries to find a partition for a job, and fails. Results are summarized in Table 2. Remember that a workload has 150 jobs. Clearly, the CONT3D and CONT5D schedulers work much harder than the others and fail too often. Not because the required nodes are not there, but because they are not arranged as requested. SFC alternatives have an easier job, as shape restrictions are limited to consecutiveness in a 1D space. The NC-FF strategy fails only when free consecutive nodes are not enough to match the size of the selected job.

The user submitting the job may know the best partition shape to run an application, and we try to reflect this fact with the shaped workloads. The con-

Workloads	CONT3D			CONT5D			SFC-FF			SFC-BF		
	S	M	L	S	M	L	S	M	L	S	M	L
Shaped	4649.80	10356.10	11870.20	7085.60	10791.00	11852.50	94.20	580.8	257.30	79.90	432.80	198.70
Unshaped	4575.50	9835.80	2956.50	5026.00	10656.20	3706.00	126.20	1071.70	194.90	102.10	885.70	161.70

Table 2: Number of allocation failures due to external fragmentation: enough free nodes are available, but not in the required arrangement.

tiguous scheduler will honor the request, searching for the specified shape. In contrast, the non-contiguous scheduler will ignore this request totally, and the SFC-based schedulers will simply look for a set of physically close nodes matching the desired size. Note that, when jobs are specified with a shape, internal fragmentation does not appear, as the size of the assigned partition will be the same of the job. Additionally, note that the contiguous allocator will not necessarily search for regular,  $n$ D hyper-rectangles (where  $n$  is the dimensionality of the IN): they user may request a 2D, planar partition, even if the network is 5D.

Results with these workloads are also summarized in Figure 4. Note that, without the effect of internal fragmentation, the main factor affecting utilization is external fragmentation (not finding the requested shape). The relative performance of NC-FF against SFC-based schedulers remain, as both ignore the requested shapes. However, contiguous strategies offer much better results.

## 7 Analysis of the Quality of the Partitions

Utilization metrics tell us only partial information about the performance of a supercomputer - scheduler combination. Utilization may be low but, if applications run faster, at the end of the day the supercomputer is more productive. With the experiments carried out we cannot verify if SFC based strategies are adequate to run communication-intensive parallel applications. But we can obtain some metrics that can be used as indicators of that adequacy, see Section 2.2.

### 7.1 Average Pairwise Distance

A low value of APD for a partition indicates how compact that partition is. Communications will use short paths, thus (presumably) benefiting communication-intensive applications. We have summarized in Table 3 averages of these metrics for the partitions used by the different schedulers, for both 3D and 5D (mesh-shaped) INs. In all cases, partitions have better (lower) APD in 5D meshes than in 3D meshes. This is due to the topological characteristics of the IN, with higher degree for 5D, that results in shorter distances. Thus, this fact does not require further discussion.

When job requests are unshaped, the cubic partitions used by the contiguous scheduler are very compact, being this allocation strategy the absolute winner. Partitions used by SFC-based allocators are very good, with Hilbert generating partitions almost as compact as the cubes. Non-contiguous partitions exhibit very poor distance-based metrics.

	Unshaped workloads						Shaped workloads					
	3D			5D			3D			5D		
	S	M	L	S	M	L	S	M	L	S	M	L
CONT	<b>6.56</b>	<b>11.34</b>	<b>18.80</b>	<b>4.77</b>	<b>6.88</b>	<b>9.54</b>	9.42	14.37	21.96	<b>4.80</b>	<b>6.76</b>	<b>9.94</b>
HILB-FF	7.10	12.35	20.30	5.02	7.21	10.26	<b>7.33</b>	<b>12.16</b>	<b>20.04</b>	5.15	7.10	10.15
ZORD-FF	9.08	14.88	23.64	5.92	8.29	11.07	9.34	14.79	22.68	6.07	8.27	10.80
HILB-BF	7.09	12.37	20.32	5.01	7.21	10.26	<b>7.32</b>	<b>12.11</b>	<b>20.05</b>	5.15	7.09	10.15
ZORD-BF	9.03	15.05	23.76	5.89	8.36	11.09	9.34	14.71	22.72	6.07	8.23	10.81
NC-FF	21.91	24.86	26.56	9.79	11.04	11.71	22.46	25.19	27.06	9.94	11.2	11.82

Table 3: Average pairwise distance of the partitions found by the allocation algorithms. These results were obtained using two sets of workloads (see Section 5) simulated into two cube-shaped network topologies (3D and 5D).

Results for the workload with shaped job requests may be misleading. In general, the distance-based metrics obtained for all strategies except the contiguous are the same seen for unshaped workloads, see again Table 3, as those strategies ignore the shape request. This is not the case when the partitions are contiguous. In fact, in the 3D mesh, the Hilbert-based SFC allocator provides “better” partitions than those used by the contiguous allocator. In the description of the workloads provided in Section 5, we clearly stated that it is assumed that the user submitting a job will choose the *best* partition shape for it. It may happen that the requested shape is not the same of the underlying IN. For example, a job of size 1024 may request specifically a planar  $32 \times 32$  partition, that fit perfectly in a plane of the 3D network. If the scheduler does not honor the shape request, it could be assigned to a partition with a 3D shape of size  $11 \times 11 \times 11$  (exactly or approximately) with excellent distance metrics but that may not allow optimal inter-task communications. This happens when the *virtual topology* of the application differs from the *physical topology* of the partition [19]. As the Hilbert-based allocator ignores shape requests, prioritizing compactness, this strategy is the best performer in terms of distance-based metrics for 3D networks. For 5D networks, the high degree of the topology shortens the distance-related metrics for all strategies, making this effect less visible.

After seeing these results, we wonder if APD can be considered as a real indicator of performance. As explained in [20], the answer is a clear “no”, because a partition is good only if, after the mapping, it matches the communication demands of the application, and APD does reflect this fact. Furthermore, we should not ignore a side-effect of sharing a supercomputer: the possible interference between applications running simultaneously. However, in general, SFC based strategies will provide compact partitions to execute parallel jobs.

## 7.2 Nodes affected metric

The nodes affected metric tries to reflect the degree of isolation of the partitions. Low values (identical or close to the size of the partition) are indicators of very isolated partitions that share few or none network resources (routers, links) with other partitions. Larger values evidence partitions that require the use of

resources “belonging” to neighboring partitions. It is well known that isolation is highly beneficial for parallel jobs [10] [17] [19] [22].

In Figure 5 we have summarized the results of this metric for the partitions used by the different allocators. They are normalized, being the 100% the results achieved by the contiguous allocator (that guarantees isolation). The NC-FF strategy uses partitions that cover almost the whole IN and, thus, the corresponding numbers would distort the figure. For this reason, they have not been included.

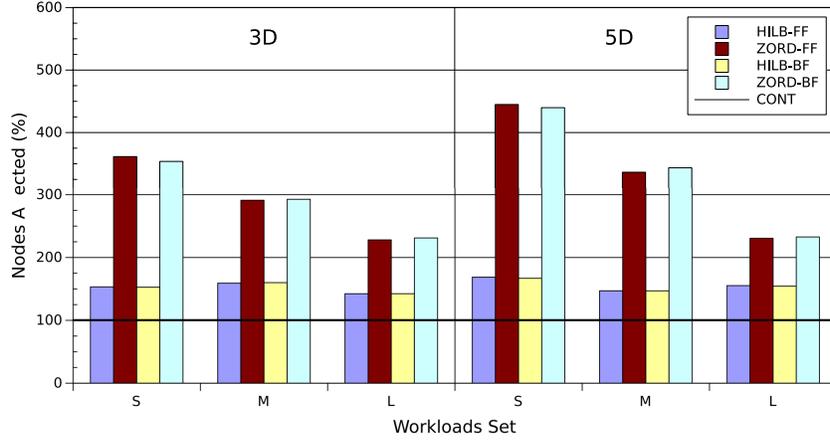
When job requests do not specify shape, SFC-based partitions result in larger numbers of affected nodes, compared to the minimum provided by the cubic partitions (see Figure 5a). In particular, the Hilbert mappings have on average 50% more nodes. As an example, this means that a job of size 7000 (a typical size in the large workload) may be interfering with around 3500 nodes belonging to other applications. The Z-curve mapping is considerably worse, with affected areas 125-350% larger than those corresponding to cubic partitions. Noticeably, this excess area is smaller for larger jobs, and larger for 5D networks than for 3D networks. When dealing with specified shapes, results show a similar pattern, but are slightly worse for SFC-based allocators, see Figure 5b.

In summary, Hilbert-based allocators do a decent job guaranteeing compact and relatively isolated partitions, in addition to provide network utilization values close to those achieved with the non-contiguous allocator. The latter is the winner in terms of utilization, but the cost to pay is the use of scattered partitions with large distances between nodes and intense interference. At the other end, the contiguous allocator provide the best execution environment for applications, but result in severe fragmentation.

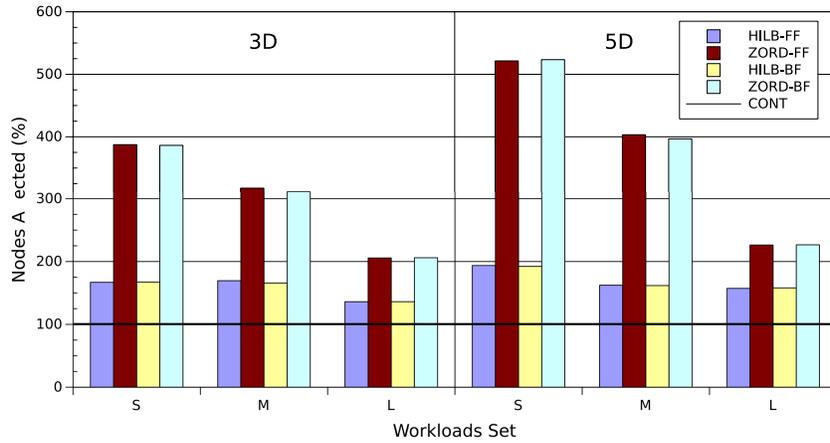
## 8 Trading off costs and benefits of allocation strategies

Without a detailed study of the applications executed by the jobs, and the strategies used to map tasks to nodes, it is simply not possible to make a definite statement about which allocator is the best one. We need to know if applications running in nicely isolated, contiguous partitions do actually execute faster than in scattered partitions. We have evidence that, in fact, they do ([17] [19] [22]), but the degree of improvement depends greatly on the particular application – actually, application set – that conform the workloads. We can take for granted, given the measurements included in the previous sections, that SFC-based allocators based on the Hilbert curve should be preferred to NC-FF, as it yields similar utilization levels while providing much more compact partitions.

It is not clear, though, under which circumstances the contiguous allocator *could* be the one of choice. Here we explore this issue. Let  $t$  be the average job duration in the contiguous and isolated partitions provided by CONT. Let  $s$  be the average slowdown experienced by the same jobs when running in SFC-based partitions that do not guarantee those properties. Thus, the average job duration with HILB-FF would be  $t \times s$ . Note that we are assuming that  $s > 1$ .



(a) Unshaped workloads



(b) Shaped workloads

Fig. 5: % of nodes affected by the partitions obtained by the allocation strategies. Results are normalized, being 100% the result achieved by the CONT strategy.

Similarly, let  $U_H$  be the utilization of the system with HILB-FF, and  $U_C$  the utilization with CONT. Now, we are assuming that  $U_H > U_C$ .

As our workload has  $w$  jobs of size  $n$ , its total computational demand (use of resources) is  $D_C = (w \times n \times t)$  for CONT, and  $D_H = (w \times n \times (t \times s))$  for HILB-FF. The makespan for the workload can be computed as its computational demand divided by the achieved system utilization (actually, utilization  $U$  is computed as  $(D/M)$ ). Thus  $M_C = (D_C/U_C)$  and  $M_H = (D_H/U_H)$ .

Now we are ready to state that HILB-FF is *the preferred choice* over CONT if its makespan for the applied workload is shorter, that is, when  $M_H < M_C$ . This can be expanded as:

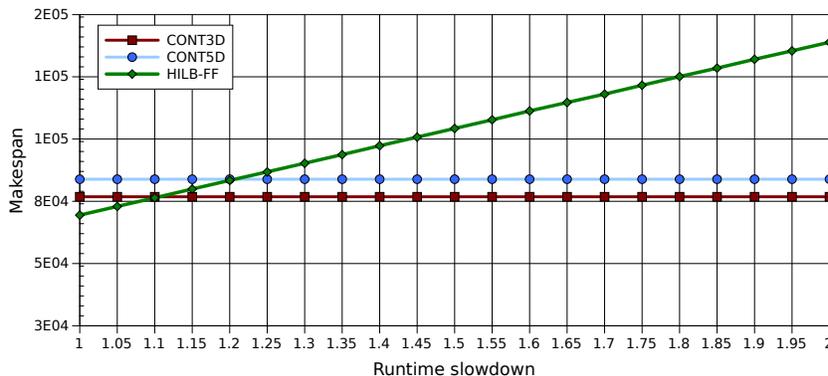


Fig. 6: Makespan of CONT3D, CONT5D and HILB-FF for different slowdown factors.

$$\frac{w \times n \times t \times s}{U_H} < \frac{w \times n \times t}{U_C} \quad (3)$$

The inequality can be simplified, expressing it as:

$$\frac{s}{U_H} < \frac{1}{U_C} \quad (4)$$

Therefore, to make a good choice of allocator we need to know its utilization result (that depends mainly on the allocation strategy and the characteristics of the network and, thus, is application-agnostic) and the average slowdown experienced by applications (which can vary drastically with the specifics of the applications forming the workload).

Equations 3 and 4 can be transformed to obtain answers to this question: which values of slowdown are acceptable for an allocator (compared to CONT) that compensate an increase in per-job run time with a higher utilization and, therefore, a shorter makespan? We have represented this in Figure 6. The horizontal lines represents the baseline makespans for CONT, which are different for 3D and 5D. The raising line corresponds to the makespans achievable by HILB-FF for different values of slowdown  $s$ . These values correspond to the shaped, large workload described before, but the trend is exactly the same for other workloads.

The crossing point is at  $s = 1.1$  for 3D meshes. This means that when applications need on average less than 10% extra time to end when running on HILB-FF generated partitions, then HILB-FF is a good choice of allocator. However, for higher degrees of slowdown, the CONT allocator is the best choice, consuming the workload faster even without fully utilizing all the resources available. For 5D meshes the crossing point is higher, at  $s = 1.2$ , or 20% allowed slowdown for HILB-FF. This is because of the large penalty to pay in terms of fragmentation when using networks of high dimensionality. An exhaustive exploration of the actual values of  $s$  for different, realistic workloads is left as future work. Some

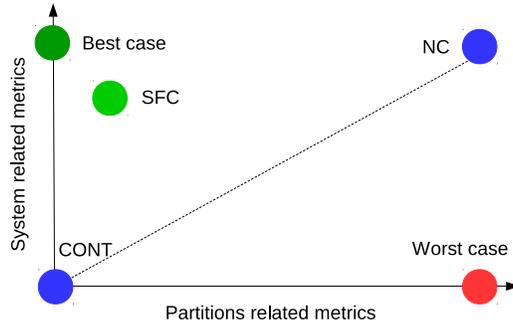


Fig. 7: Visual representation of the relative advantages of different allocation strategies, based on system-level and partition-related metrics.

preliminary work carried out in [17] [22] shows that for some applications we can expect values of  $s$  exceeding 3.

## 9 Conclusions and future work

In this work we have evaluated the quality of SFC based allocation strategies for schedulers of supercomputers. This evaluation has been carried out using synthetic workloads with different characteristics (size, duration, shape), submitted to systems with a 3D and 5D mesh topology. These strategies have been compared with a contiguous (CONT) and a non-contiguous (NC-FF) strategy.

CONT prioritizes the utilization of contiguous and isolated partitions, optimal for the execution of applications. The prize to pay is a large overhead due to fragmentation, that results in low levels of system utilization. On the contrary, NC-FF prioritizes utilization, assigning nodes to partitions independently of their positions in the network. The prize to pay is a collection of sparse and overlapping partitions.

Allocators based on space filling curves have demonstrated excellent properties, in particular when Hilbert is the curve of choice. They perform allocation in a linear (1D) space, which results in utilization levels close to those of non contiguous approaches as represented in Figure 7. The obtained 1D partitions are then mapped to the  $nD$  topology of the actual system, resulting in very compact sets of nodes (see again Figure 7). The partitions achieved this way offer distance metrics similar to those of CONT strategies, although they do not guarantee isolation. However, SFC allocators have characteristics that are close to those of a theoretical, optimum allocator.

The relative merit of the different allocators depend on factors such as the search strategy (FF, BF), the topology of the IN (number of dimensions, related to node degree) and the properties of the workloads submitted to the system. We have studied “shaped” and “unshaped” workloads. The former assume that the user knows and requests the most appropriate shape to run an application, and the CONT scheduler honors this request – at a cost: in busy systems, it is difficult

to find a partition of a specific shape. None of the remaining strategies take this request into consideration, using only the number of nodes in the request to search for a partition.

Finally we have seen that the benefits that SFC-based strategies obtain in terms of utilization, may disappear if we consider the (relative) slowdown that applications would suffer when running in non-optimal, overlapping partitions. If the penalty is over 10% for 3D meshes (20% for 5D meshes), then the CONT strategy is the best, even if utilization figures tell a different story. Otherwise, for lower penalties, SFC based strategies should be the chosen.

As future lines of work, we plan to make a deeper analysis of the impact of non contiguous partitions on applications' slowdown. As we have focused on  $nD$  meshes, we plan to extend this work to other kinds of IN, for example based on trees. Finally, we want to propose and assess a *hybrid* allocation strategy, able to provide contiguous and isolated partitions for those applications requiring them, and other SFC based partitions for less demanding applications, with the objective of achieving high system utilization without penalizing applications.

## Acknowledgments

This work has been partially supported by the Research Groups 2013-2018 (IT-609-13) program (Basque Government), TIN2013-41272P (Ministry of Science and Technology). Jose A. Lozano is also supported by BERC program 2014-2017 (Basque government) and Severo Ochoa Program SEV-2013-0323 (Spanish Ministry of Economy and Competitiveness). Jose Miguel-Alonso is member of the HiPEAC European Network on High Performance and Embedded Architecture and Compilation.

## References

1. J. Alber and R. Niedermeier. On multi-dimensional hilbert indexings. *Theory of Computing Systems*, 33:195–392, 2000.
2. J. Alber and R. Niedermeier. On multidimensional curves with hilbert property. *Theory of Computing Systems*, 33(4):295–312, 2000.
3. E. Balzweit, D. P. Bunde, V. J. Leung, A. Finley, and A. C. S. Lee. Local search to improve coordinate-based task mapping. *Parallel Computing*, 51:67–78, 2016.
4. D. Chen, N. A. Easley, 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*, pages 1–10, New York, NY, USA, 2011. ACM.
5. Cray Inc. <http://www.cray.com/assets/pdf/products/xe/idc.948.pdf>.
6. 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.
7. D. Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. *Annals of Mathematics*, 38:459–460, 1891.

8. IBM Journal of Research and Development staff. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development*, 52(1/2):199–220.
9. C. R. Johnson, D. P. Bunde, and V. J. Leung. A Tie-Breaking Strategy for Processor Allocation in Meshes. In *39th International Conference on Parallel Processing, ICPP Workshops 2010, San Diego, California, USA, 13-16 September 2010*, pages 331–338. IEEE Computer Society, 2010.
10. A. Jokanovic, J. Sancho, G. Rodriguez, A. Lucero, C. Minkenberg, and J. Labarta. Quiet neighborhoods: Key to protect job performance predictability. In *Parallel and Distributed Processing Symposium (IPDPS)*, pages 449–459, May 2015.
11. J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. In *Proceedings of the 17th British National Conference on Databases: Advances in Databases*, pages 20–35. Springer-Verlag, 2000.
12. Lawrence Livermore National Laboratory. Simple linux utility for resource management. <https://computing.llnl.gov/linux/slurm/>.
13. V. Leung, E. Arkin, M. Bender, D. Bunde, J. Johnston, A. Lal, J. Mitchell, C. Phillips, and S. Seiden. Processor allocation on cplant: achieving general processor locality using one-dimensional allocation strategies. pages 296 – 304, 2002.
14. W. Liu, V. Lo, K. Windisch, and B. Nitzberg. Non-contiguous processor allocation algorithms for distributed memory multicomputers. In *Proceedings of ACM/IEEE Conference on Supercomputing*, pages 227–236. IEEE Computer, 1994.
15. V. Lo, K. Windisch, W. Liu, and B. Nitzberg. Noncontiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 8:712–726, 1997.
16. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical Report Ottawa, Ontario, Canada, 1966.
17. 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. IEEE Computer Society, February 2009.
18. J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. Strategies to Map Parallel Applications onto Meshes. In *Distributed Computing and Artificial Intelligence*, pages 197–204. Springer Berlin/Heidelberg, 2010.
19. 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.
20. J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. Application-aware metrics for partition selection in cube-shaped topologies. *Parallel Computing*, 40(5-6):129–139, 2014.
21. J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. A fast implementation of the first fit contiguous partitioning strategy for cubic topologies. *Concurrency and Computation: Practice and Experience*, 26(17):2792–2810, 2014.
22. J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano. Locality-aware policies to improve job scheduling on 3d tori. *The Journal of Supercomputing*, 71(3):966–994, 2015.
23. P. Walker, D. P. Bunde, and V. J. Leung. Faster high-quality processor allocation. In *Proceedings of the 11th LCI International Conference on High-Performance Cluster Computing*, 2010.
24. E. W. Weisstein. Hilbert Curve. <http://mathworld.wolfram.com/HilbertCurve.html>.
25. 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.