PaCMap: Topology Mapping of Unstructured Communication Patterns onto Non-contiguous Allocations

Ozan Tuncer
Boston University
Boston, MA, USA
otuncer@bu.edu

Vitus J. Leung
Sandia National Laboratories
Albuquerque, NM, USA
vjleung@sandia.gov

Ayse K. Coskun
Boston University
Boston, MA, USA
acoskun@bu.edu

ABSTRACT

In high performance computing (HPC), applications usually have many parallel tasks running on multiple machine nodes. As these tasks intensively communicate with each other, the communication overhead has a significant impact on an application's execution time. This overhead is determined by the application's communication pattern as well as the network distances between communicating tasks. By mapping the tasks to the available machine nodes in a communication-aware manner, the network distances and the execution times can be significantly reduced.

Existing techniques first allocate available nodes to an application, and then map the tasks onto the allocated nodes. In this paper, we discuss the potential benefits of simultaneous allocation and mapping for applications with irregular communication patterns. We also propose a novel graph-based allocation and mapping technique to reduce the execution time in HPC machines that use non-contiguous allocation, such as Cray XK series. Simulations calibrated with real-life experiments show that our technique reduces *hop-bytes* up to 30% compared to the state-of-the-art.

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: Parallel Programming—*Topology Mapping*

Keywords

Topology mapping; task mapping; non-contiguous allocation; unstructured communication pattern; performance

1. INTRODUCTION

In high performance computing (HPC), highly parallel applications run on multiple machine nodes for long durations up to several days where each node typically includes multiple processing cores [15]. The size of these applications and the HPC machines have been growing exponentially in the last decade. The number of processing cores in the largest machines has increased from 65536 in 2005 to over 3 million in 2014 [2]. This growth is expected to continue in the forthcoming years, creating a need for scalable workload management solutions to reduce both the application execution times and the energy consumption [14].

One of the most important challenges in HPC workload management is topology mapping, i.e., the placement of an application's tasks to the machine nodes. Placing the highly communicating tasks close to each other can reduce the communication time significantly, and as a result, improve the overall application performance. As most of the HPC applications have a specific communication topology, the problem can be expressed as mapping an application's communication graph onto the machine's network graph.

Mapping of two graphs to each other is shown to be an NP-hard problem [17]. Thus, researchers proposed various heuristics for topology mapping such as recursive graph bisectioning [29], exploiting application's geometric information to partition the communication graph [9, 13], and graph embedding [34]. Most of these techniques focus on HPC machines with contiguous allocation, where each application is assigned to a contiguous block of machine nodes. Although contiguous allocation increases application locality, it decreases the machine utilization [24, 32]. Our work focuses on non-contiguous allocation, which is used in many HPC systems such as Cray XT, XE, and XK series [1]. Note that our technique can also be used for contiguous allocation.

We target HPC applications that have unstructured communication patterns, where each task can have an arbitrary number of neighbors and different communication weight to each neighbor. Examples for such applications include biomolecular simulations and sparse matrix multiplication. Our work is the first to propose a technique for topology mapping of unstructured applications onto machines that use non-contiguous node allocation.

To the best of our knowledge, previous work on topology mapping only considered the mapping of application tasks onto an already-selected set of nodes, which are determined by the system management software without using detailed information on the communication pattern. In this work, we explore the benefits of joint allocation and task mapping based on an application's topology. Our results indicate strong motivation to expand the job submission infrastructures to include detailed communication information.

As application and machine sizes continue to grow and computation performance continues to improve faster than communication performance, the impact of topology mapping on HPC application execution time and power con-

ICS'15, June 8–11, 2015, Newport Beach, CA, USA.

ACM 978-1-4503-3559-1/15/06 ...$15.00
http://dx.doi.org/10.1145/2751205.2751225.

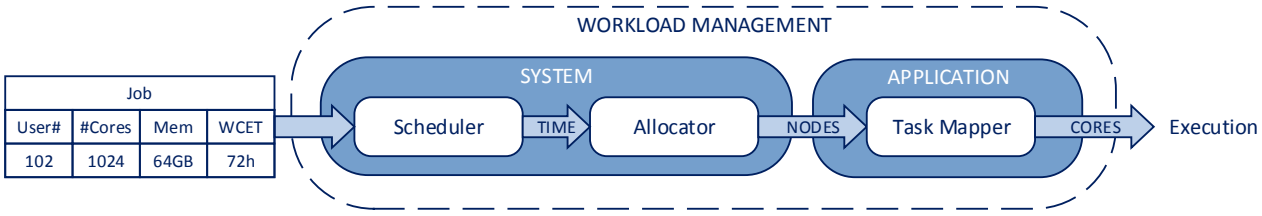


Figure 1: Workload management in conventional HPC machines.

sumption is expected to grow. Our contributions to topology mapping in this paper can be summarized as follows:

- We propose PaCMap (Partitioning And Center MAPping), a graph-based topology mapping algorithm to reduce the execution time of HPC applications with unstructured communication patterns (Section 3). Our algorithm can be used both for complete topology mapping or as a task mapping algorithm for a given set of nodes.
- Using experimental data from a Cray XE6 system, we develop a running time estimation model based on the application's communication pattern as well as the topology mapping decision (Section 4). The model is then used in simulation space to compare various topology mapping algorithms.
- We analyze the potential benefits of joint allocation and mapping (Sections 2 and 5). Simulation results show that topology-aware allocation combined with mapping can decrease *hop-bytes* (i.e., the weighted sum of the traversed network distance for all messages, weighted by the message size) by up to 30% compared to state-of-the-art.

2. TOPOLOGY MAPPING BACKGROUND

In today's HPC systems, topology mapping, i.e., the placement of application tasks onto machine nodes, includes decisions both from the system side and from the application side. The system is responsible for scheduling and allocation, whereas the application side performs task mapping.

As depicted in Figure 1, job submission in conventional HPC systems only supports sending basic requests to the system, such as number of processing cores, memory requirement, and worst case execution time (WCET), where a job refers to a specific instance of an application. Then, as part of the system software, the scheduler determines when to run the job based on the machine availability and the job queue. Once the job is scheduled, the allocator assigns a set of machine nodes for the job. Finally, the application's task mapper places its tasks to the allocated nodes. In this paper, we assume there is no space sharing, i.e., a machine node cannot be shared by multiple jobs. This is common in HPC due to ease of management and security concerns.

Due to the limitations of the job submission framework, the system is unaware of the exact communication pattern of the job for unstructured applications. For task mapping, however, programmers can specify the communication pattern through interfaces such as MPI [27]. Alternatively, the pattern can be profiled during an application's first run and used in the future runs. In addition to the communication pattern, the application side can also discover the physical network topology through system calls [33]. Based on these information, task mapping can make an efficient assignment of the application tasks to the machine nodes to minimize the communication overhead.

The rest of this section explains the problem in more detail and presents various algorithms that are currently used for topology mapping.

2.1 Job Allocation

There are two main considerations for non-contiguous allocation. First, the selected nodes should be close to each other to reduce the communication distances. Second, the allocation should not lead to fragmented machine utilization. The allocation algorithms that disregard the second issue can lead to the segmentation of large empty blocks in the machine, potentially increasing the communication overhead of future jobs. Consider the example shown in Figure 2, where two jobs are scheduled in a 6-node machine with a 2D mesh network topology. In case (a), the first job is allocated to the middle column. Thus, job 2 must be fragmented into smaller parts, increasing the communication distance. However, case (b) assigns job 1 to the side of the machine, leaving sufficient space for job 2.

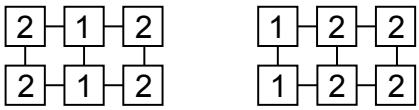


Figure 2: Example for (a) fragmented and (b) non-fragmented job allocation. The boxes represent machine nodes and the numbers represent allocated jobs.

The common techniques for non-contiguous job allocation can be classified into two categories: linear and clustered. In this paper, we use one algorithm from each category.

2.1.1 *Best-fit (linear)*

The best-fit strategy is a combination of the ideas proposed by Lo et al. [24] and Leung et al. [22]. The algorithm first linearly orders the machine nodes along a curve. Then, the free nodes are grouped into intervals along this curve. The job is allocated to the smallest interval that has sufficient nodes. If there is no such interval, the algorithm selects the nodes that minimize the maximum distance along the curve. This strategy is commonly used in real HPC machines due to its simplicity and its small time complexity of $O(M)$ in an M-node machine. In machines with a torus or mesh network topology, linear allocation strategies order the nodes using space-filling curves such as Hilbert curves to improve locality [5].

2.1.2 Mc1x1 (clustered)

The mc1x1 algorithm is a variant of the MC algorithm proposed by Mache et al. [26]. It is a $(2 - 2/k)d$ approximation to the optimal solution for minimizing the average pairwise L1 distance of tasks in a d-dimensional mesh when allocating k processors [7]. The algorithm aims to find a compact cluster of free nodes. For each free node n, mc1x1 calculates an allocation score by counting the number of free nodes in a d-dimensional hypercube centered on node n.

The size of the hypercube is started from a single node and increased until the hypercube contains sufficient free nodes for the job to be allocated. Then, the score of the center node is calculated as follows:

$$NS_{n,J} = \sum_{i=1}^{J} dist_{n,i} \quad (1)$$

where $NS_{n,J}$ is the score of node n for allocating J nodes, and $dist_{n,i}$ is the distance between nodes n and the free node i in the hypercube. The node with the least score is selected as the center node. Although mc1x1 finds clustered nodes, it does not address the fragmented allocation problem and it is not applicable in network topologies other than mesh and torus.

The time complexity of this approach depends on the machine utilization and the machine state. In the worst case, calculating the score of a free node requires checking the availability of all the nodes in the machine, whereas the best case checks only the nearest J nodes of J free nodes. Hence, depending on the machine state, allocating nodes with mc1x1 takes between $O(M^2)$ and $\Omega(J^2)$, where M is the number of nodes in the machine. Note that the node scores can be computed in parallel.

2.2 Task Mapping

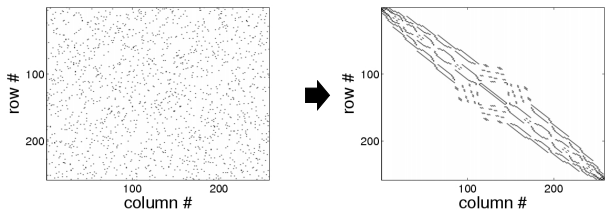
Task mapping considers the assignment of the individual application tasks to the machine nodes, which are selected by the job allocation stage (Section 2.1). Unlike job allocation, task mapping is able to use information provided by the application such as the communication pattern. In this paper, we focus on three different task mapping techniques and use them as baselines for the evaluation of the proposed *PaCMap* algorithm.

2.2.1 Reverse Cuthill-Mckee (RCM)

RCM [11] reduces the bandwidth of a symmetric matrix via permutation, i.e., it reorders the matrix such that the non-zero entries that are far from the diagonal are eliminated as demonstrated in Figure 3. When applied on a task communication matrix, which shows the communication links between the tasks, this corresponds to reducing the maximum distance between the tasks when the tasks are linearly ordered.

In machines with contiguous allocation, RCM can be applied both on the network and the communication graphs with $O(JD \log D)$ complexity, where D is the maximum degree of the application graph. Then, in-order mapping of the tasks to the nodes can effectively reduce the communication distance for sparse communication patterns [8] [17].

In machines that allow non-contiguous job allocation, the machine network cannot be directly used as it results in an unconnected graph. Instead, the tasks are mapped in-order along the curve of a linear allocation algorithm.



(a) Original matrix (b) RCM permutation

Figure 3: RCM applied on a sample sparse matrix

2.2.2 Recursive Graph Bisection (RGrB)

RGrB algorithm uses the task communication graph and the network topology graph. It recursively splits both graphs into equal halves using minimum weighted edge-cuts, and maps the remaining task(s) to the remaining node at the end of the recursion. This algorithm has been used for contiguous allocation by software packages such as LibTopoMap [17] and SCOTCH [29]. As there is no existing RGrB variant specific to non-contiguous allocation, we apply RGrB by building a virtual all-to-all graph for the machine network, where the edges are weighted based on the hop distance between the nodes. Due to the all-to-all graph, RGrB takes $O(J^3)$ for non-contiguous allocation for mapping J tasks [17]. Our implementation of RGrB is based on LibTopoMap, and uses the METIS library [19] for bisectioning. Although this technique demonstrated efficient mappings, it is also shown that it may result in poor p-way partitions [31].

2.2.3 Recursive Geometry Bisection (RGeoB)

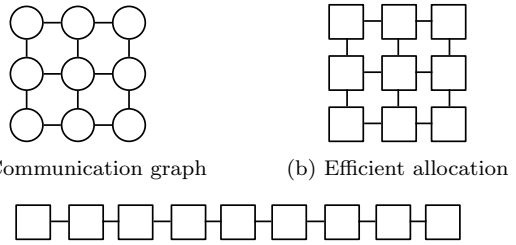
Similar to the RGrB, RGeoB is based on recursive bisections. Instead of using the graphs, however, RGeoB splits the application and the machine geometries into equal halves such that the maximum dimension length is minimized [13]. The application geometry can be inherent to the application, such as the coordinates of an object in a computational fluid dynamics simulation, or it can be generated from the communication graph. Similarly, the machine geometry can be defined as x, y, z coordinates of a 3D mesh network topology. When a generic sorting algorithm with $O(J \log J)$ is used for the geometric bisectioning, the complexity of RGeoB becomes $O(J \log^2 J)$ as shown by the master theorem [10].

The effectiveness of RGeoB strictly depends on how well the communication is represented by the given application geometry. For 3D stencil computations, this technique is shown to perform better than RGrB on a Cray XE6 [13].

2.3 Interplay of Allocation and Mapping

As mentioned earlier in this section, the allocation is unaware of the application's communication pattern in current HPC systems. Hence, the allocation decision can decrease the potential efficiency of the task mapping algorithm. Consider the example in Figure 4. A communication-aware allocation algorithm will allocate a 3x3 mesh (Figure 4b) for an application with a 3x3 stencil communication pattern, so that the task mapping can reduce the average message hop distance to 1. However, if the allocation is unaware of the communication pattern, it can also select the nodes as in Figure 4c, where the minimum average message hop distance that can be achieved is 2.

Similar effects can also be observed in larger scales. Consider a 3D stencil application with dimensions 2x8x32. If



(a) Communication graph (b) Efficient allocation

(c) Inefficient allocation

Figure 4: Communication graph of a 3x3 stencil application, and (b) communication-aware and (c) -unaware allocation examples

this application is allocated to a 2x8x32 mesh, the minimum average hop distance would be 1. However, when it is allocated to a 8x8x8 cubical mesh instead, the RGeoB task mapping algorithm, being the best task mapper for this case, results in an average hop distance of 1.92, increasing the communication overhead.

These example cases create sufficient motivation to develop algorithms that jointly consider allocation and mapping. Section 5 presents further analysis on this issue.

3. PACMAP ALGORITHM

We aim to reduce the execution time of HPC applications through communication-aware topology mapping. We propose PaCMap (partitioning and center mapping), a graph-based algorithm that simultaneously carries out job allocation and task mapping to reduce communication overhead.

As shown in Figure 5, PaCMap first partitions the communication graph into k task groups (TGs) such that each group can fit into a single node in the cluster. This step consolidates the highly-communicating tasks to be executed on the same machine node. After this step, the problem reduces to mapping the TGs into the available machine nodes. In our implementation, the partitioning is done by the multilevel k-way partitioning algorithm from the METIS library [19].

Next, PaCMap selects a *center TG* from the partitioned graph and maps it to a selected *center node* in the cluster. Then, it expands the allocation by picking a node and mapping a task to it based on the network topology and on the communication graph until all tasks are mapped.

PaCMap can also be used only for allocation by ignoring the mapping decision, or only for task mapping by limiting the nodes the algorithm can use. The rest of this section explains in detail how the center node and the center TG are selected and how the expansion is performed, along with a complexity analysis.

3.1 Center Machine Node Selection

As discussed in Section 2, the allocation determines how efficiently the cluster is utilized. The algorithm should select a collected group of nodes and should not lead to fragmented allocation of future jobs. Our solution is a heuristic that addresses both issues.

For each available node n, we look at the other available nodes in the proximity of n and calculate a node score $NS_{n,J}$, where J is the number of nodes to be allocated. To calculate the score, we first create a list of available nodes around n within a communication distance of R using breadth-first expansion. Then, we sort this list with respect to the distance to n. Starting from the closest node, we increase $NS_{n,J}$ for the first J nodes, and penalize it for the remaining extra nodes as follows:

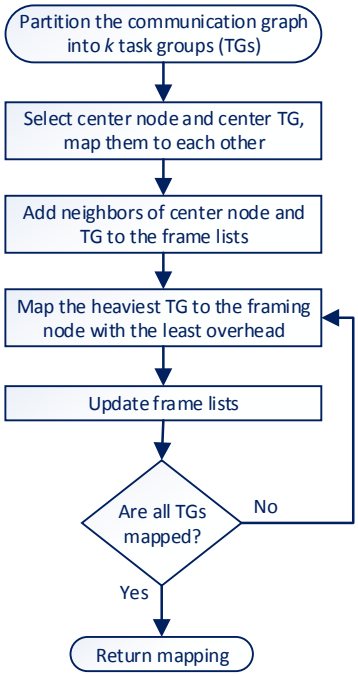


Figure 5: Flowchart of the proposed PaCMap algorithm

$$NS_{n,J} = \sum_{i=1}^{J} f(dist_{n,i}) - \sum_{i=J+1} f(dist_{n,i}) \quad (2)$$

where $f(dist_{n,i})$ is a function of $dist_{n,i}$, which is the communication distance between n and node i in the list. In order to avoid fragmented allocation, R should be selected such that the number of nodes within the network distance of R is larger than J. Additionally, $f(dist_{n,i})$ should prioritize the nodes that are closer to node n to improve locality.

R and $f(dist_{n,i})$ are selected based on the network topology. We use the following heuristics for 3D torus topologies: The maximum number of nodes within a network hop distance of r equals $4r^3/3 + 2r^2 + 8r/3 + 1$ in a 3D torus. Using a pre-calculated look-up table based on this formula, we first find the minimum distance r_{min} that contains J nodes. We then select $R = r_{min} + 2$ to check the excessive availability around the node n. This number can be adjusted for different machines if the above formula becomes invalid due to asymmetrical torus dimensions. We use $f(dist_{n,i}) = 1/(4dist_{n,i}^2 + 2)$ so that the maximum total impact of the nodes equidistant to n on $NS_{n,J}$ is 1. We do not observe a significant change in the allocation performance by the selection of $f(dist_{n,i})$ as long as the nodes closer to n have more impact than others. In other network topologies such as Dragonfly [20], the same methodology can be followed to select R and $f(dist_{n,i})$.

3.2 Center Task Group Selection

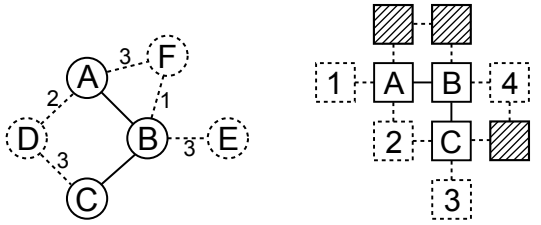
PaCMap expands the allocation from the center node and the center TG, while allocating nodes and mapping TGs at each expansion step. As the center machine node selection stage scores the nodes based on the availability around them without prioritizing any direction, an efficient node selection

requires the expansion to be symmetric in all directions. This can be done by selecting a proper center TG.

The center TG is selected as the one with the minimum cumulative shortest-path distance to all other TGs. To find the center, we run Dijkstra's algorithm on all TGs.

3.3 Expansion

After allocating the center TG to the center node, we create two lists for the expansion: (1) a list of machine nodes that frame the current partial allocation, (2) a list of TGs that frame the currently allocated TGs. The frames are the neighboring nodes/TGs in the corresponding graphs, as demonstrated in Figure 6. The edge weights in the machine graph (the network links) are all 1, whereas the edge weights in the TG graph are given in the figure.



(a) Communication graph (b) Network graph

Figure 6: Partially allocated application. The solid shapes are current allocation/mapping, the dashed shapes are the frames, and striped squares are busy nodes.

For each expansion step, we select the heaviest TG, i.e., the unmapped TG that has the largest communication volume with the currently allocated tasks. In Figure 6a, D would be selected for the next step with a total weight of $2 + 3 = 5$. We map this TG to the node that leads to the least total communication overhead, calculated as follows:

$$overhead_n = \sum_i dist_{n,i} \cdot W_{n,i} \tag{3}$$

where $W_{n,i}$ is the communication weight between the TGs mapped to nodes n and i. In Figure 6b, node #2 would be selected for task D with an overhead of $1 \cdot 2 + 1 \cdot 3 = 5$. If there is a tie between the nodes, we select the closest node to the center to maintain the symmetry of the expansion.

After mapping the heaviest TG to the selected node, we update the frames by adding the neighbors of the allocated node and the mapped TG. If there is no free node among the direct neighbors of the allocated node, we increase the search distance until a free node is found.

3.4 Complexity Analysis

The overhead of the center node selection strictly depends on the underlying network topology. In our implementation, we make use of the coordinate information of the 3D torus topology. For each free node, we check the availability of $O(R^3) = O(J)$ nodes, resulting in a total of $O(MJ)$, where M is the number of nodes in the machine. Computing the node scores is an inherently parallel operation.

Given that most HPC applications are re-run with different parameters or inputs, partitioning of the communication graph into TGs and selection of the center TG need to be done only once per application. The results can be re-used for the future submissions of the same application. Alternatively, this information can be passed to the system along with the job submission. For the sake of completeness, k-way partitioning by METIS is $O(C \log(k))$, where C is the number of edges in the communication graph [19]. This is equal to $O(C \log J)$ as the number of cores per node is bounded. Selecting the center TG by running Dijkstra's algorithm with Fibonacci heap on all TGs is $O(JC + J^2 \log(J))$ [10].

We keep the TG frame in a Fibonacci heap, and the node frame in a linked list. Hence, finding and deleting the heaviest TG for J TGs takes $O(J \log J)$. Selecting the node that leads to the least communication requires calculating the communication overhead for all neighbors of the heaviest TG and for all nodes in the frame. First, assume that there is a bounded number of nodes in the frame. During the entire mapping process, J nodes will be mapped and the communication weight of C links will be calculated, leading to $O(J + C)$ given that the node distances are known. Now, the frame size can be included in the complexity. At any time, there will be a maximum of $O(J)$ nodes in the frame as the number of ports of a router in a 3D torus network is bounded. Hence, selecting nodes during mapping takes a total of $O(J^2 + JC) = O(JC)$ for connected graphs. Note that the best node can be selected in parallel.

Updating the task frame is done by breadth-first search, where each step includes updating the weights of the unmapped neighboring TGs in the Fibonacci heap, leading to a total of $O(J + C)$. The complexity of updating the node frame depends on the machine utilization. If the utilization is close to 100%, finding the closest free node can take worst-case $O(M)$ to search all the nodes in the machine, leading to $O(MJ)$ during the entire mapping process. When the machine utilization is lower, the direct neighbors of the allocated node are likely to be free as the center node is selected with sufficient availability in its proximity. In this case, finding all closest nodes takes $\Omega(J)$ in total.

As a result, the complete allocation and mapping process has a time complexity of $O(MJ + JC)$ for a connected communication graph and for 3D torus machines, assuming that the partitioning and the center task selection is done before the job submission. This complexity is feasible in terms of real-life implementation and scalability, and it is comparable to our best-performing baselines (Section 5).

4. EXPERIMENTAL METHODOLOGY

As real HPC machines do not have the infrastructure to support combined allocation and task mapping, we use simulations for our evaluation. Hence, we also need to create realistic HPC workload traces and communication patterns. To compare allocation and task mapping algorithms, we estimate the execution time of the jobs based on their locality. This section describes our experimental framework.

4.1 Target Machine

Our target machine uses static mapping, i.e., it does not support task migration at runtime. In our evaluation, we use a 3D torus network topology. Torus topologies are commonly used in HPC due to their low cost, ease of design and installation, and high bisection bandwidth (i.e., the total bandwidth of links placed between two equal-sized node sets after partitioning). Example machines that use 3D torus include IBM BlueGene/L [3], Cray XE6 [1], and Cray XK7, which is ranked number 2 on the November 2014 Top500 list [2]. During our simulations, we use static shortest path dimension-ordered (x-y-z) routing in the network.

4.2 Workloads

In order to compare the task mapping algorithms, we need a comprehensive set of unstructured sparse communication graphs as well as application geometries. For this purpose, we use the University of Florida Sparse Matrix Collection [12] as a proxy for communication and geometry information. This collection is commonly used for the evaluation of graph algorithms such as bisectioning (e.g. [18]), and consists of data from real applications in various fields such as circuit simulations, financial modeling, and chemical process simulations. We use the applications in the collection with 2D or 3D geometry information with up to 115K tasks. We assume uniform communication between the tasks as it is the expected case for well-balanced HPC applications.

The comparison of non-contiguous allocation algorithms requires using an already populated machine. Additionally, our analysis should account for the impact of the allocation decision on the performance of future jobs. To address both issues, we use the logs provided in the Parallel Workloads Archive (PWA) [15] as inputs to our simulation, and evaluate entire workload traces. PWA logs are collected from real large scale parallel systems, and provide information on job arrival times, execution times, and job sizes, but no information on the communication. Hence, we assign communication patterns to the jobs by matching them with the proxy communication matrices in the sparse matrix collection. As the job sizes in PWA do not necessarily match with the matrix sizes in the collection, we apply *binning* to the PWA, in which the job sizes are changed to the closest available size in our application set.

Among the logs in the archive, LLNL-Atlas and CEA-Curie traces lead to the most balanced application counts after binning. LLNL-Atlas and CEA-Curie are collected from machines with 9216 CPUs (assumed to be quad-core CPUs) and 93312 cores, respectively. These are the largest machine sizes in the archive after ANL-Intrepid, which leads to an unbalanced binning and biases the results by prioritizing only a few applications in our input set. Logs from the newer and larger machines are not available in the archive due to the explosive growth of machine sizes and the lag in collecting data for research. We use the first two weeks of these two traces in our evaluation as machines like Cray Cielo [25] are usually taken down for maintenance every two weeks. Our LLNL-Atlas and CEA-Curie traces consist of 1001 and 3291 jobs, respectively.

4.3 Performance Model

During our simulations, we modify the execution time of a job based on its communication pattern and mapping. To extract the relationship between communication time and the network-related metrics, we use real-life experiments.

We conduct our real-life experiments on the Cray XE6 Cielo supercomputer located at Los Alamos National Laboratories [25]. Cielo consists of 8944 compute nodes and additional service nodes. Each compute node has a dual AMD Opteron 6136 eight-core "Magny-Cours" socket, providing a total of 16 cores. The nodes are connected using a Cray Gemini 3D torus network topology with the dimensions of 16x12x24 and two nodes per Gemini.

We use the miniGhost application [6], which is a part of the miniapps developed by the DOE community to represent the computational core of various HPC applications. MiniGhost focuses on the nearest neighbor inter-process communication strategy, with computation mainly serving to provide enough data and separation of the boundary exchanges from some computation. Its core is based on CTH, an application for modeling complex multi-dimensional problems that are characterized by large deformations and/or strong shocks [16].

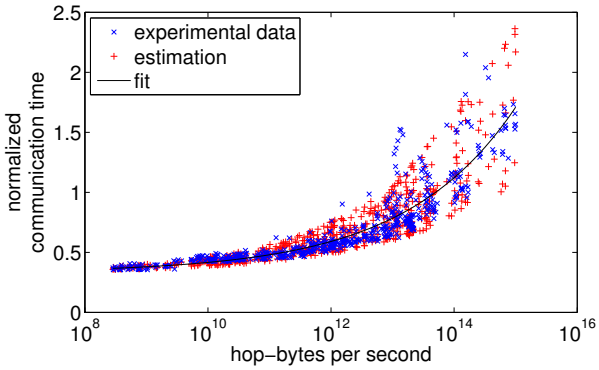


Figure 7: Relationship between communication and hop-bytes. The communication time represents the time spent in communication when the computation time equals 1.

We run miniGhost with sizes of powers of 2, from 64 to 65536. For each size, we run miniGhost under 5 allocation schemes and using 11 task mapping techniques, providing 55 executions for the same application. The task mapping techniques include all algorithms introduced in Section 2.2, all algorithms in LibTopoMap [17], and the system defaults. During the experiments, we collect time spent for communication and computation, number of bytes sent between machine nodes, maximum congestion, and message hop count.

We examine several metrics that have been associated with the communication overhead in the literature. These are maximum dilation (i.e., the maximum network hop distance a message travels), average dilation, average hops per byte, maximum network congestion, and hop-bytes as defined in Equation 4.

$$\textit{hop-bytes} = \sum_{i}^{messages} \textit{hop-distance}_i \cdot \textit{bytes}_i \qquad (4)$$

Hop-bytes represents the total communication volume in the network. Our experimental results show high correlation between hop-bytes and communication time, as shown in Figure 7. Based on the experimental data, we formulate the execution time of a job as follows:

$$T_{exec} = (1 + 0.0019 \cdot \textit{hop-bytes}^{0.16+\tau}) \cdot T_{base} \qquad (5)$$

where T_{base} is the execution time without considering the overhead introduced by topology mapping decision, and τ is a uniformly distributed random number between -0.013 and 0.013 that represents the variation in the experimental results. As the application tasks run on individual processing cores without consolidation, we assume that the computation time is independent of the mapping decision.

4.4 Simulation Environment

We use the Structural Simulation Toolkit (SST), which is an architectural simulation framework designed by Sandia National Laboratories to assist in the design, evaluation and

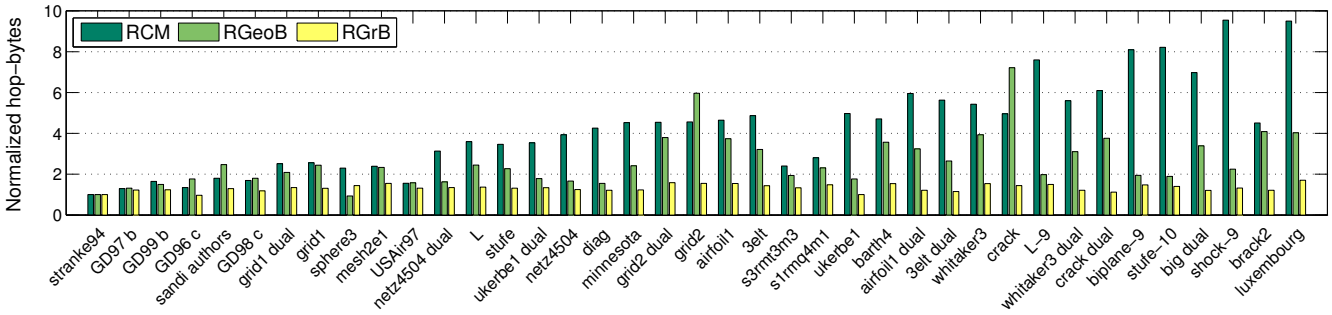


Figure 8: Hop-bytes comparison of task mappers for all applications in our input set. The values are normalized to the hop-bytes resulting from PaCMap.

optimization of HPC architectures and applications [30]. We use macro-SST simulations rather than micro-SST for speed.

For scheduling, we use the "easy scheduler" provided in SST, which is an aggressive backfilling technique that gives a guaranteed start time only to the first job in the queue [23]. We use the existing algorithms in SST for job allocation except for PaCMap. For task mapping, we implement PaCMap as well as the baseline algorithms. We calibrate the running time estimation in the simulator as described in Section 4.3.

5. EVALUATION

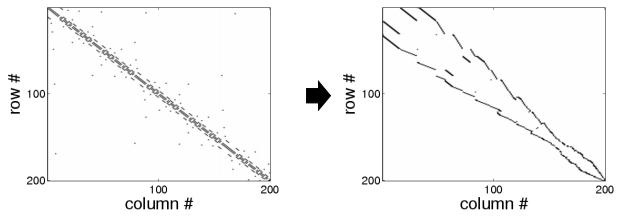
There are many aspects affecting the efficiency of the topology mapping algorithms, making it infeasible to perform a reliable comparison with a single series of experiments. Hence, we first start our evaluation with analyzing the task mappers independently from the allocation algorithms in Section 5.1. For this purpose, we run each application in an empty machine using the same allocation. Based on our results, we summarize how the task mappers perform for various sparse communication patterns. Then, in Section 5.2, we use long workload traces with multiple jobs to observe the impact of allocation on task mapping.

5.1 Single Job Analysis

In order to find the strengths and weaknesses of the task mappers, we run individual applications using the same allocation. We use best-fit allocator with Hilbert curves to provide a fair comparison for RCM, which maps the tasks in-order to the allocated nodes. For each application, we use the smallest empty cubical machine with single core per node, where the machine dimensions are selected as powers of 2 so that efficient Hilbert curves can be generated. We use PaCMap only as a task mapper in this analysis.

Figure 8 shows the resulting hop-bytes for all applications in our input set. The results are normalized with respect to the hop-bytes of PaCMap. The applications in the x-axis are ordered from smallest with 10 tasks (stranke94) to the largest with 115K tasks (luxembourg). Note that all cases use the same allocator so that the benefits of combined allocation and mapping is not exploited. We now present our detailed analysis on each task mapper.

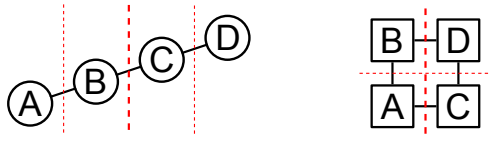
RCM: In the figure, we observe that the RCM performance decreases as the application size increases. The first reason for this scalability problem is that RCM ignores the network links that are not along the curve. Second, the performance of RCM depends on the correlation between the average and the maximum distance in the communication matrix, which typically decreases with the application size. Consider the example given in Figure 9, which is the communication of the first 200 tasks of *diag*. In 9a, the maximum communication distance is 161 and the average distance is 2.37. After applying RCM, the maximum distance reduces to 33; however, the average becomes 8.84, increasing the communication overhead. Analyzing the correlation between the average and the maximum distance during topology mapping is not feasible due to its complexity. In addition to these problems, RCM assumes uniform communication between the task groups, worsening its mapping decision when a node contains multiple cores.



(a) Original matrix (b) RCM permutation

Figure 9: First 200 tasks of the application *diag*. RCM increases the average communication distance while decreasing the matrix bandwidth.

RGeoB: The performance of the RGeoB depends on (1) the similarity between the coordinates and the allocated node structure and (2) how well the geometry represents actual communication. For case 1, consider the example given in Figure 10. Because only coordinates are used during bisectioning, RGeoB places the tasks B and C far from each other and increases the message hop distance. This problem also occurs when the tasks have 2D coordinates but are mapped into a 3D topology. It has a significant impact on the communication overhead in larger scale, where thousands of tasks are placed far apart (e.g. *crack* in Figure 8).



(a) Application Coordinates (b) Network Graph

Figure 10: Example for mismatching coordinates and allocation. The dashed lines represent the bisection cuts.

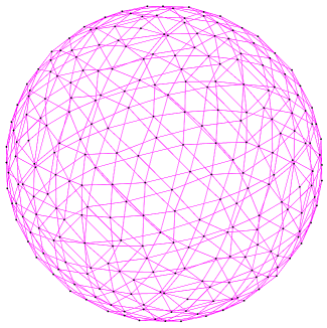


Figure 11: 3D coordinates of *sphere3* [12]. Nodes are the tasks and edges are the communication links.

To avoid this problem in torus/mesh networks for structured communication, researchers have introduced methods such as folding [8]. However, no solution exists on this issue for arbitrary communication patterns. For case 2, the most prominent example is the application *sphere3* with the coordinates given in Figure 11. In the given geometry, the communicating tasks are close to each other, meaning that the communication is represented well in 3D coordinates. As a result, RGeoB leads to the lowest hop-bytes for this application in Figure 8.

RGrB: Note that RGrB uses an all-to-all network graph as the allocation can be non-contiguous and arbitrary. One weakness of RGrB is that the heuristics used for graph bisectioning, which is also an NP-hard problem, perform poorly with all-to-all graphs. Additionally, RGrB can lead to less efficient solutions depending on the nature of the communication and machine graphs as well as how the recursive branches of these two graphs are mapped to each other [28].

PaCMap: The performance of PaCMap is reduced as the allocation is not adapted to the communication graph during this analysis. We compare our technique with others in the subsequent section.

We have performed the same analysis using average message hop distance and per-job congestion metrics to verify that our algorithm do not worsen another network metric. As we obtained very similar results to the hop-bytes metric, we do not present them due to space limitations.

RCM is not a good topology mapping solution for non-contiguous allocation due to its poor scalability. Similarly, RGeoB is preferred only for structured communication patterns. Hence, we will focus on RGrB and PaCMap in the rest of our analysis. Next, we discuss the impact of allocation algorithm on the topology mapping for unstructured communication patterns.

5.2 Workload Trace Analysis

Analyzing how the allocation decision affects the task mapping performance in a machine with non-contiguous allocation requires an already-populated machine. For this purpose, we use entire workload traces and compare the execution time of the jobs. The target HPC machine used in this analysis has 16 processing cores per node as in the real-life experiments we use for calibrations (See Section 4.3). For each input trace, the machine size is selected as the actual machine size the logs are collected from.

Figure 12 shows the cumulative execution time of all jobs that use multiple nodes (i.e., jobs with 16 or more tasks) in the traces with different allocator and task mapper pairs.

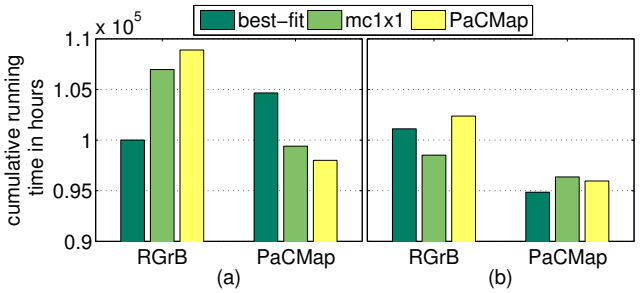


Figure 12: Cumulative running time of the jobs that use multiple nodes in (a) LLNL-Atlas and (b) CEA-Curie traces. The horizontal axis shows different task mappers; whereas bar colors are different allocators.

Each bar group uses a different task mapper, and each bar in a group uses a different allocator. Note that PaCMap can be used as an allocator and/or as a task mapper.

Although intuitively clustered allocations should be more useful for RGrB than curve-following allocations, we observe that for LLNL-Atlas, RGrB leads to 7% less cumulative execution time than mc1x1 with the best-fit allocator. This is because the Hilbert curves provide very high locality in this particular case. However, for CEA-Curie, the RGrB & mc1x1 pair leads to 3% shorter execution than with best-fit. The topology mapping performance not only depends on the allocator and the task mapper, but also on in which order the jobs arrive.

Complete topology mapping (both allocation and task mapping) with PaCMap decreases the cumulative execution time by 2% and 3% for LLNL-Atlas and CEA-Curie, respectively, compared to the best case of RGrB for each trace. Note that this reduction corresponds to 3000 hours of cumulative active node computation time in two weeks, which implies power and energy savings besides the execution time. The improvement is expected to grow with increasing application sizes based on our following analysis.

In order to verify that this difference in the execution time is not due to specific workloads, we compare the average per-application hop-bytes in Figure 13. As we generate the communication patterns by binning the job sizes in the traces to the closest available size in the sparse matrix collection, the traces do not contain all the applications. Thus, some applications are not present in the figures. The values Figure 13 are normalized with respect to the results of PaCMap.

In both traces, RGrB performs better than PaCMap for applications with less than 1K tasks. As the application size increases, however, PaCMap leads to smaller hop-bytes up to 30% compared to the best case of RGrB, excluding *sandi authors* and *mesh2e1* in the CEA-Curie trace. These two applications have very large hop-bytes because they have a few instances in the trace, which are allocated poorly in this particular case.

6. RELATED WORK

Topology mapping based on an application's communication pattern and on the network topology is a well-studied NP-hard problem. Researchers have been investigating various heuristic techniques to minimize the communication overhead of HPC applications through topology mapping. Bhatelé et al. proposed techniques such as step embedding

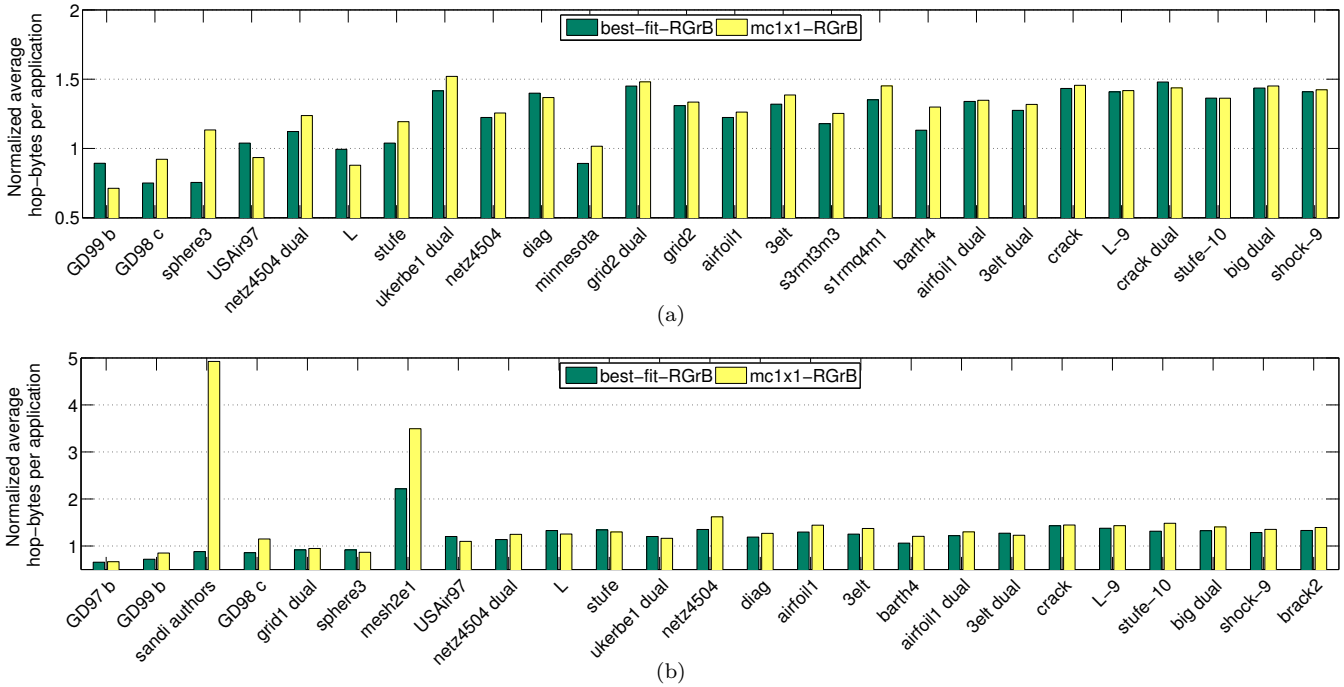


Figure 13: Average per-application hop-bytes in (a) LLNL-Atlas and (b) CEA-Curie traces with different allocator & task mapper pairs. The results are normalized with respect to PaCMap.

and folding to map 2D stencil applications into 2D and 3D mesh machines [8], as well as traversal and affine mapping algorithms to map irregular graphs into meshes [9]. Yu et al. presented graph embedding techniques for structured communication onto IBM Blue Gene/L systems [34]. Hoefler and Snir proposed mapping irregular topologies using graph similarity and graph partitioning [17]. These techniques focus on contiguous allocation, and are integrated in software packages such as LibTopoMap [17] and SCOTCH [29].

For machines with non-contiguous allocation, Krumpke et al. proposed Gen-Alg, which is a $(2 - 2/k)$-approximation allocation algorithm, to minimize the pairwise average L1 distance of the mapped tasks [21], and Lo et al. introduced various linear allocation schemes [24]. Leung et al. improved their solution by using Hilbert curves on the CPlant supercomputer [22]. Albing et al. proposed using different curves to remove the restrictions on the network topology and machine dimensions imposed by Hilbert curves [4]. These solutions limit the allocation performance by ignoring network links outside the given curve. Other researchers have introduced clustered allocation schemes to overcome such limitations. Bender et al. proposed the MM algorithm, a $(2 - 1/2d)$ approximation for d-dimensional grids, and the MC1x1 algorithm [7]. Their techniques are applicable only on mesh and torus topologies.

Previous work on task mapping for non-contiguous allocation has focused on stencil communication patterns. Deveci et al. considered multi-level partitioning of both the machine and application geometries [13]. It is also possible to use certain contiguous mapping algorithms for non-contiguous machines with small modifications (e.g., RCM, RGrB).

Our paper differs from prior work by focusing on the topology mapping of irregular communication patterns into non-contiguous allocation of arbitrary network topologies, including novel topologies such as Dragonfly [20]. Additionally, we analyze the potential benefits of simultaneous job allocation and task mapping.

7. CONCLUSION

Topology mapping of HPC applications have a significant impact on the execution time, especially at the macro-scale, where the applications use more than 1K machine nodes. In this paper, we have presented a novel algorithm, PaCMap, that simultaneously applies job allocation and task mapping to minimize the execution time of applications with unstructured communication patterns. PaCMap is applicable to any network topology, and it can be also used as a mere allocator or task mapper. Furthermore, we have developed an execution time estimation model based on real life experiments on a Cray XE6, which is used to calibrate the simulations of long HPC workload traces. Our results show that PaCMap reduces hop-bytes up to 30% compared to state-of-the-art approaches in HPC machines with non-contiguous allocation. This implies improvement in the machine performance as well as reduction in power and energy consumption.

Acknowledgments

This work has been partially funded by Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL8500.

8. REFERENCES

[1] Cray Inc. http://www.cray.com/.
[2] Top 500 supercomputer sites. http://www.top500.org/.

[3] N. Adiga et al. Blue gene/l torus interconnection network. *IBM Journal of Research and Development*, 49(2.3):265–276, March 2005.

[4] C. Albing et al. Scalable node allocation for improved performance in regular and anisotropic 3d torus supercomputers. In *Proceedings of the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface*, EuroMPI'11, pages 61–70, 2011.

[5] D. Auble and B. Christiansen. SLURM workload manager overview. Presented at the 2014 ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC'14).

[6] R. F. Barrett, C. T. Vaughan, and M. A. Heroux. Minighost: a miniapp for exploring boundary exchange strategies using stencil computations in scientific parallel computing. Technical report, Sandia National Laboratories, Albuquerque, NM, 2012.

[7] M. A. Bender et al. Communication-aware processor allocation for supercomputers: Finding point sets of small average distance. *Algorithmica*, 50(2):279–298, Jan. 2008.

[8] A. Bhatele et al. Automated mapping of regular communication graphs on mesh interconnects. In *International Conference on High Performance Computing (HiPC)*, pages 1–10, Dec 2010.

[9] A. Bhatelé and L. Kalé. Heuristic-based techniques for mapping irregular communication graphs to mesh topologies. In *IEEE 13th International Conference on High Performance Computing and Communications (HPCC), 2011*, pages 765–771, Sept 2011.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition, 2001.

[11] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172.

[12] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, Dec. 2011.

[13] M. Deveci et al. Exploiting geometric partitioning in task mapping for parallel computers. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 27–36, 2014.

[14] J. Dongarra et al. The international exascale software project roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60, Feb. 2011.

[15] D. G. Feitelson, D. Tsafrir, and D. Krakov. Experience with the parallel workloads archive, 2012.

[16] E. S. Hertel et al. CTH: A software family for multi-dimensional shock physics analysis. In *Proceedings of the 19th International Symposium on Shock Waves*, pages 377–382, 1993.

[17] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 75–84, 2011.

[18] M. Holtgrewe, P. Sanders, and C. Schulz. Engineering a scalable high quality graph partitioner. IPDPS'10, pages 1–12, April 2010.

[19] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, Jan. 1998.

[20] J. Kim et al. Technology-driven, highly-scalable dragonfly topology. In *35th International Symposium on Computer Architecture, 2008. ISCA '08.*, pages 77–88, June 2008.

[21] S. Krumke et al. Compact location problems. *Theoretical Computer Science*, 181:238–247, 1996.

[22] V. Leung et al. Processor allocation on cplant: achieving general processor locality using one-dimensional allocation strategies. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 296–304, 2002.

[23] D. A. Lifka. The anl/ibm sp scheduling system. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, IPPS '95, pages 295–303, 1995.

[24] V. Lo, K. J. Windisch, W. Liu, and B. Nitzberg. Noncontiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Trans. Parallel Distrib. Syst.*, 8(7):712–726, jul 1997.

[25] Los Alamos National Laboratory. Cielo supercomputer. http://www.lanl.gov/projects/cielo/index.php.

[26] J. Mache, V. Lo, and K. Windisch. Minimizing message-passing contention in fragmentation-free processor allocation. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems*, pages 120–124, 1997.

[27] MPI Forum. MPI: A message-passing interface standart. Version 3.0, Sept. 2012. http://www.mpi-forum.org/.

[28] F. Pellegrini and J. Roman. Experimental analysis of the dual recursive bipartitioning algorithm for static mapping. Technical report, TR 1038-96, LaBRI, URA CNRS 1304, Univ. Bordeaux I, 1996.

[29] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, HPCN Europe, pages 493–498, 1996.

[30] A. Rodrigues et al. Improvements to the structural simulation toolkit. In *Proceedings of the 5th International Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 190–195, 2012.

[31] H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, sep 1997.

[32] V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnston, and P. Sadayappan. Selective buddy allocation for scheduling parallel jobs on clusters. In *CLUSTER'02*, pages 107–116, 2002.

[33] H. Subramoni et al. Design of a scalable infiniband topology service to enable network topology aware placement of processes. In *SC'12*, pages 70:1–70:12, 2012.

[34] H. Yu, I.-H. Chung, and J. Moreira. Topology mapping for blue gene/l supercomputer. In *SC'06*, pages 52–52, Nov 2006.