Automated Mapping of Regular Communication Graphs on Mesh Interconnects

Abhinav Bhatelé, Gagan Raj Gupta, Laxmikant V. Kalé
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
E-mail: {bhatele, gagan, kale}@illinois.edu

I-Hsin Chung
IBM Research
Thomas J. Watson Research Center
Yorktown Heights, NY 10598, USA
E-mail: ihchung@us.ibm.com

Abstract— **Network contention has a significantly adverse effect on the performance of parallel applications with increasing size of parallel machines. Machines of the petascale era are forcing application developers to map tasks intelligently to job partitions to achieve the best performance possible. This paper presents a framework for automated mapping of parallel applications with regular communication graphs to two and three dimensional mesh and torus networks. This framework will save much effort on the part of application developers to generate mappings for their individual applications.**

One component of the framework is a process topology analyzer to find regular patterns and if found, to determine the dimensions of the communication graphs of applications. The other component is a suite of heuristic techniques for mapping 2D object grids to 2D and 3D processor meshes. The framework chooses the best heuristic from the suite for a given object grid and processor mesh pair based on the *hop-bytes* metric. We show performance improvements using the framework, for a 2D Stencil benchmark in MPI and the Weather Research and Forecasting model running on the IBM Blue Gene/P. We also compare our algorithms with others discussed in literature.

I. Introduction

Should parallel applications be written taking the interconnect topology into account? The glib and naïve answer may be "Of course, yes!". Yet, since the mid 1980's until just a few years ago, the correct answer was "no". It is significantly more challenging (and certainly more work) to consider the network topology while developing an application; so understanding the nuanced answer to this question is important. Most interconnects since the mid 1980's have used wormhole routing or its variants. Instead of storing and forwarding the entire message or even large packets, at each hop along the route, very short *flits* are forwarded along a route set up by the first flit. As a result, for a message beyond a few hundred bytes in size, the latency is not significantly affected by the number of hops it traverses. This, combined with the relatively small number of nodes used, made it a safe assumption to ignore the interconnect topology.

With the advent of large machines with toroidal topologies, such as the IBM Blue Gene/P at Argonne National Lab (ANL) that has $40,960$ nodes, and Cray XT4/5 (Jaguar) at Oak Ridge National Laboratory ($12,141$ nodes), a new and somewhat subtle issue has made topology important again, at least for some applications. While the benefits of wormhole routing remain relatively unaffected, messages that traverse a large number of hops occupy a larger fraction of the available bandwidth, and thereby increase the chance of contention in the network. However, contention might not affect overall application performance if communication is a small fraction of the total execution time or if the application is latency tolerant. So, contention is a second order effect and not all applications are affected by it. But for the applications that are affected, the impact can be dramatic. We and a few other researchers, have demonstrated this effect, and have brought it to the attention of the community [1]–[4]. It is therefore necessary now to undertake a research program aimed at topology aware mapping of tasks to processors.

It should be noted that the level of efficacy of mapping is hardware dependent but looking at the current trends of some supercomputers, bandwidth available per flop is decreasing. For example, the advertised bandwidth per flop for the Cray XT3, XT4 and XT5 machines is 8.77, 1.36, 0.23 bytes per flop respectively [5], [6]. Hence, in situations where bandwidth is insufficient and applications are communication bound, we should see some benefit from mapping.

Modern supercomputers are forcing application developers to map tasks intelligently to job partitions to achieve the best performance possible. This paper presents a framework for automated mapping of parallel applications. This framework will save much effort on the part of application developers to generate mappings for their individual applications. The paper takes on a specific aspect of topology aware mapping: how to assign tasks with two-dimensional (2D) near-neighbor communication graphs, to the prevailing three-dimensional (3D) mesh and torus topologies. Several classes of parallel applications fall under this category of regular communication graphs, a few examples being MIMD Lattice Computation (MILC) [7], Parallel Ocean Program (POP) [8] and Weather Research and Forecasting Model (WRF) [9].

We present several mapping algorithms, some new and some known via the literature, and compare their performance using the *hop-bytes* metric. The hop-bytes metric is the weighted sum of message sizes where the weights are the number of hops (links) traveled by the respective messages. Hop-bytes is an indication of the *average* communication load on each link on the network. Of course, using this metric as a measure of contention is tantamount to assuming that the application generates nearly uniform traffic over all links in the partition. The metric does not give an indication of hot-spots generated on specific links on the network. However, it is an easily derivable metric and correlates well with actual application performance. Also, it is often better than the prevailing metric *maximum dilation*.

The main contribution of the paper, is the framework for automatic mapping of a wide class of MPI and other applications with regular communication graphs. We consider point-to-point communication in this paper because collective operations involving all ranks in a program cannot be optimized by a remapping of ranks. From analysis of communication patterns, to using different heuristics in different situations for mapping solutions, everything is handled by the framework. Parallel applications can get performance improvements using mapping solutions from this framework without any changes to the code base. We demonstrate the use of this framework for performance improvements of two parallel applications: a 2D Stencil benchmark in MPI and the Weather Research and Forecasting (WRF) program.

This paper is organized as follows: Section II builds on the introduction, motivates the work further and places it in perspective. The automated mapping framework and the evaluation metric *hop-bytes* are discussed in Section III. The two steps for automatic mapping are presented in the subsequent sections: Section IV presents algorithms for identifying communication patterns from profiling data. Sections V and VI present mapping algorithms for 2D object grids to 2D and 3D processor meshes respectively. Section VII presents results on mapping of applications with regular communication.

II. Previous Work and Motivation

Research on topology aware mapping originated in the fields of graph embedding, circuit design and parallel computing. Techniques that embed rectangular 2D grids into square 2D grids were proposed to optimize VLSI circuits and significant results were obtained [10]–[12]. Techniques from mathematics and circuit design are not always applicable to parallel computing. For example, mapping research motivated by reducing the total area of circuit layouts tried to minimize the length of longest wire [10]. Longest edge dilation might not be the best metric for parallel machines.

In parallel computing also, the problem of mapping parallel programs onto parallel systems has been studied extensively. Significant research was done on topology aware mapping to restrict communication to near-neighbors and optimize performance [13]–[15]. Techniques ranging from physical optimizations to heuristic approaches were developed. Most of these techniques (heuristic techniques especially) were developed specifically for hypercubes, shuffle-exchange networks or array processors.

In recent times, several groups of application developers have used mapping techniques to improve performance for their codes [16]–[21]. The motivation for the work accomplished in this paper is to relieve the application developers from doing the mapping. We are trying to build an intelligent mapping framework which can automate the process of mapping of applications to parallel machines. In this paper, we chose applications which have a regular communication pattern. Several classes of applications in high-performance computing fall in this category ranging from ocean simulations, weather simulations to lattice QCD simulations.

The general mapping problem is computationally equivalent to the graph isomorphism problem which belongs to NP, neither known to be solvable in polynomial time nor NP-complete. Hence, we have developed various heuristics to generate mapping solutions. We hope that a wide class of applications can benefit from the mapping framework and obtain performance optimizations.

III. Automatic Mapping Framework

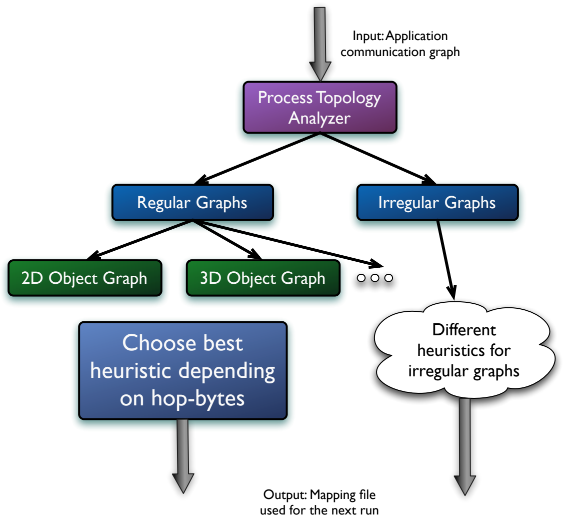
The previous section mentioned that several application groups have realized the importance of topology aware mapping and used it to improve the performance of individual codes. A general mapping framework which takes the communication graph of an application as input and outputs efficient mapping solutions would relieve the application developers of the mapping burden. A library with several mapping heuristics for different communication graphs would be a great asset to the parallel computing community.

All parallel applications can be divided into two categories depending on their communication graph – regular and irregular. Regular graphs refer to those where the number of edges from all the nodes is the same and there is a certain pattern to the communication. Examples of regular communication are 2D and 3D stencil-like communication and structured mesh computations. Graphs with varying number of edges from different nodes and all others which do not fall under the regular category can be labeled as irregular. Examples of irregular communication are unstructured mesh computations. This paper takes on one specific aspect of mapping: how to assign tasks with two-dimensional (2D) near-neighbor communication graphs, to the prevailing three-dimensional (3D) mesh and torus topologies.

The process of automating the mapping of applications can be divided in to two steps:

1) Obtain the communication graph for an application and identify specific communication patterns in the communication graph.
2) Apply heuristics in different cases depending on the communication patterns to obtain mapping solutions.

The communication graph for an application gives information about the number of bytes exchanged between tasks or processes in the program. For example, in case of MPI, the nodes of this graph are the MPI ranks or processes in the program and edges exist between two nodes if the corresponding MPI ranks communicate through messages. A test run of the application is performed to obtain a $n \times n$ matrix of communication bytes exchanged between different pairs where n is the total number of MPI ranks.



Input: Application communication graph
Process Topology Analyzer
Regular Graphs
Irregular Graphs
2D Object Graph
3D Object Graph
Choose best heuristic depending on hop-bytes
Different heuristics for irregular graphs
Output: Mapping file used for the next run

Fig. 1. Schematic of the automatic mapping framework

The communication graph is obtained by profiling libraries such as those in the IBM High Performance Computing Toolkit (HPCT) [22] and information from one run can be used to develop mapping solutions for subsequent similar runs. This approach assumes that the communication graph of an application running with a certain set of input parameters does not change from run to run. As of now, the situation where the graph changes with time, within one run, can not be handled either, since migrating MPI tasks at runtime is not possible. However, some programming models such as CHARM++ provide the capability of profiling applications as they are running [23]. In such cases, the information can be used for dynamic mapping (and even load balancing if the need arises).

Figure 1 presents a schematic of the automatic mapping framework. There are two inputs to the framework:

1) The application communication graph which is used by the process topology analyzer (referred to as the object graph or grid in the rest of the paper).
2) The processor topology graph (referred to as the processor mesh) which is used by the mapping algorithms.

The framework first searches for regular communication patterns. Depending on the communication patterns identified by the process topology analyzer, the framework chooses the best heuristic from the suite for a given object grid and processor mesh pair, based on the *hop-bytes* metric. If there is no regular pattern, we assume the graph to be general and use heuristics for irregular graphs. Mapping of irregular graphs is out of scope of this paper and will be discussed in a future publication. The framework outputs a mapping solution in the form a file to be used for a subsequent run (by providing it to the job scheduler).

A. Evaluation Metric

The mapping solutions derived by the framework will be evaluated by the *hop-bytes* metric which is the weighted sum of message sizes where the weights are the number of hops (links) traveled by the respective messages. Hop-bytes is an indication of the average communication load on each link on the network. If a significant percentage of all messages travel multiple links, the total hop-bytes increase which suggests increased likelihood of contention. Hop-bytes can be calculated by the equation,

$$HB = \sum_{i=1}^{n} d_i \times b_i \tag{1}$$

where d_i is the number of links traversed by message i and b_i is the message size in bytes for message i and the summation is over all messages sent.

Hop-bytes is a measure of the bytes the network has to deliver for an application to run to completion. This assumes that the application generates nearly uniform traffic over all links in the partition. It is also important to remember that hop-bytes is an approximate indication of the contention created by an application since it does not capture the specific loads on each link. The metric does not give an indication of hot-spots generated on specific links on the network but is an easily derivable metric and correlates well with actual application performance.

The rest of the paper discusses various components of the mapping framework and performance improvements achieved by using it with different applications.

IV. IDENTIFYING COMMUNICATION PATTERNS

Automatic topology aware mapping, as we shall see in the next few sections, uses heuristics for fast scalable runtime solutions. Heuristics can yield more efficient solutions if we can derive concrete information about the communication graph of the application and exploit it. For this, we need to look for identifiable communication patterns, if any, in the object graph. Many parallel applications have relatively simple and easily identifiable 2D, 3D or 4D communication patterns. If we can identify such patterns, then we can apply better suited heuristic techniques for such scenarios. We use relatively simple techniques for pattern identification. We believe that these can be extended to detect more complex patterns based on similar work in literature [24], [25]. Some performance analysis tools also provide communication pattern identification and visualization for identifying performance problems in parallel applications [26].

In some cases, the process topology can be presented by the application user to the mapping framework. However, there are two reasons for adding the process topology analyzer: 1. Often the application developer would know the application topology but the application end user might not, and 2. The process topology discovery is a part of the automation of the mapping process. This rules out any intervention by the user in the mapping process. Also, we need profiling to know which MPI rank communicates with which other ranks where the topology analyzer can be useful.

Here, we explain the algorithm for identifying if the communication in an application has a near-neighbor stencil-like pattern with four neighbors in 2D. Algorithms for doing the same in 3D and 4D are similar. We first begin by ensuring that the number of communicating neighbors for each MPI rank is the same and is 5 or less. For a 2D communication pattern, a given rank would typically have four communicating neighbors and may have some communication (through global operations) with rank 0. Broadcasts from and reductions to rank 0 cannot be optimized by reordering of ranks and hence we ignore that. We also ignore any communication edges whose weight is less than a threshold, say 20% of the average across all edges in the graph. This is a heuristic and the threshold can be changed.

For a 2D communication pattern, if there is no wraparound, ranks on the boundaries may have fewer neighbors. Filtering these aberrations, we choose a random rank and find its "distance" (difference between the ranks IDs) from its four neighbors. The distance from two of its neighbors (left and right) would be 1 and from its top and bottom neighbors would be one of the dimensions of the 2D grid. This assumes that ranks are ordered in a row or column major order. Then, for all other ranks, we ensure that the distances from their respective neighbors are either 1 or the value of distance obtained for the previously chosen random rank. If this holds true for all other ranks, then the communication is indeed a uniform 2D near-neighbor pattern.

Algorithm 1 Pseudo-code for identifying regular communication graphs

```
Input: $CM_{n,n}$ (communication matrix)
Output: $isRegular$ (boolean, true if communication is regular)
        dims[ ] (dimensions of the regular communication graph)
for $i = 1$ to $n$ do
    find the maximum number of neighbors for any rank in $CM_{i,n}$
end for
if max neighbors $\leq 5$ then
    // this might be a case of regular 2D communication
    select an arbitrary rank $start_{pe}$ find its distance from its neighbors
    $dist$ = difference between ranks of $start_{pe}$ and its top or bottom neighbor
    for $i := 1$ to $n$ do
        if distance of all ranks from their neighbors == 1 or $dist$ then
            $isRegular$ = true
            dim[0] = $dist$
            dim[1] = $n/dist$
        end if
    end for
end if
```

Algorithm 1 shows the pseudo code for identifying one possible 2D communication pattern. Currently, this algorithm can only identify a 5-point stencil pattern. However, the algorithm can be extended trivially so that it can identify other regular patterns such as a 9-point stencil or a communication with all 8 neighbors around a rank in 2D. The algorithms for identifying 3D and 4D near-neighbor patterns are similar. Once the information about communicating ranks has been extracted and identified, the mapping algorithms can use it to map communicating neighbors on nearby physical processors.

V. 2D TO 2D MAPPING ALGORITHMS

Let us say that processes in an application have a 2D stencil-like communication pattern where each task communicates with four neighbors, two in each direction. So the communication graph for this application is a planar graph which resembles a 2D grid. We want to map this 2D grid to a 2D processor mesh. For heuristics in this paper, we assume that all edges in the communication graph have the same weight. Further, we are targeting MPI applications, so, the number of ranks (nodes in the communication graph) is the same as the number of processors (in the mesh). Therefore, we can assume that the number of nodes in the two graphs is the same. Of course, if the object grid has the same dimensions as the processor mesh, the best mapping is trivial. Heuristic strategies are needed when the aspect ratios are different. We describe five heuristics to map a 2D object grid to a 2D processor mesh. All of these heuristics are designed to optimize different cases and as we shall see in the results section, they perform best for grids of different aspect ratios.

Heuristic 1 - Maximum Overlap: This heuristic attempts to find the largest possible area of the object grid which overlaps with the processor mesh and maps it one-to-one. For the remaining area of the object grid and the processor mesh, we then make a recursive call to the algorithm. The intuition behind this heuristic is that we get the best hop-bytes for a large portion of the grid, although for a few nodes at the boundaries of the recursive calls, we might have longer hops. However, the hope is that the average hop-bytes for the entire object grid will be low and a few distant messages will not affect performance.

Figure 2 illustrates this mapping technique, where an object grid of dimensions 9×8 is to be mapped to a processor mesh of dimensions 6×12. We first map the largest possible sub-grid with dimensions 6×8 to the processor mesh. Once this is done, a recursive call is made for the object grid of size 3×8 to be mapped onto a processor mesh of size 6×4.

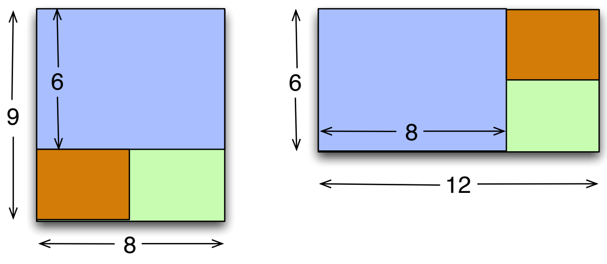


Fig. 2. Maximum Overlap (MXOVLP)

Algorithm 2 presents the pseudo-code for this heuristic (referred to as MXOVLP in figures and tables). O_x and O_y refer to the x and y dimensions of the object grid and P_x and P_y refer to the x and y dimensions of the processor mesh.

Heuristic 2 - Maximum Overlap with Alignment: This is similar to Heuristic 1 but it tries to align the longer dimension of the object grid with that of the processor mesh (referred

Algorithm 2 Maximum Overlap Heuristic (MXOVLP) for 2D to 2D mapping

```
procedure MXOVLP($O_x$, $O_y$, $P_x$, $P_y$)
    if $O_x == P_x$ then
        do a one-to-one mapping and return
    end if
    if $O_x > P_x$ then
        map the area $P_x \times O_y$
        MXOVLP($O_x - P_x$, $O_y$, $P_x$, $P_y - O_y$)
        copy the mapping into the main array and return
    else
        map the area $O_x \times P_y$
        MXOVLP($O_x$, $O_y - P_y$, $P_x - O_x$, $P_y$)
        copy the mapping into the main array and return
    end if
end procedure
```

to as MXOV+AL in figures and tables). This realignment is done at each recursive call and yields a better mapping than MXOVLP in most cases. Heuristics 1 and 2 lead to dilation at each recursive call at the boundaries where the object grid is split during recursion. However, as long as the average hop-bytes is low, we should obtain a good mapping. All the following heuristics (including Heuristic 1 above) also do an initial alignment of the longer dimensions of the object grid and the processor mesh.

There are some optimization possibilities for Heuristics 1 and 2 which will be explored in future work – After the mapping for recursively smaller sub-graphs is complete, at the end of each recursive call, it is possible to rotate the mapping for the sub-graph by 180 degrees or flip it. There are several possibilities at each recursive call leading to a combinatorial explosion of arrangements. Hence we will not discuss this post-rotation in this paper.

Heuristic 3 - Expand from Corner: In this algorithm, we start at one corner of the object grid and order all other objects by their increasing Manhattan distance from the chosen corner. We also order the processors in a similar fashion starting from one corner. Then the objects are placed in order, starting from the chosen corner of the processor grid (pseudo-code for EXCO in Algorithm 3). The intuition is that objects which communicate with one another will be close in the new ordering based on the Manhattan distance and hence, will get mapped nearby. Figure 3 shows how we start from the upper left corner of the object grid and map objects successively starting from the corresponding corner of the processor mesh.

Heuristic 4 - Corners to Center: This is similar to Heuristic 3 but in this case, we start simultaneously from all four corners of the 2D object grid and move towards the center. The objects are again picked based on their Manhattan distance from the corner closest to them (this heuristic is referred to as COCE in figures and tables). This modification to Heuristic 3 achieves better proximity for a larger number of objects since we start simultaneously from four directions. However, for certain aspect ratios, as we move closer to the center, objects may be placed farther from their communicating neighbors leading to larger hop-bytes.

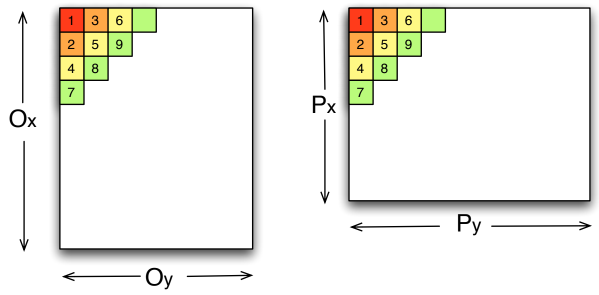


Fig. 3. Expand from Corner (EXCO)

Algorithm 3 Expand from Corner (EXCO) Heuristic for 2D to 2D mapping

```
procedure EXCO($O_x, O_y, P_x, P_y$)
    $no_x = no_y = np_x = np_y = 0$
    for $i := 1$ to $O_x \times O_y$ do
        $< co_x, co_y > = < no_x, no_y >$
        $< cp_x, cp_y > = < np_x, np_y >$
        Map[$co_x$][$co_y$] = $cp_x \times P_y + cp_y$
        $< no_x, no_y >$ = findNearest2D($co_x, co_y, O_x, O_y$)
        $< np_x, np_y >$ = findNearest2D($cp_x, cp_y, P_x, P_y$)
    end for
end procedure
```

Heuristic 5 - Affine Mapping: The idea is to stretch/shrink the object grid in both dimensions and align it to the processor mesh. It is expected that such a mapping will preserve the relative orientations of the objects, thereby minimizing the dilation. A destination processor is calculated for each object based on its position, (x, y) in the communication graph:

$$(x, y) \rightarrow (\lfloor P_x \times \frac{x}{O_x} \rfloor, \lfloor P_y \times \frac{y}{O_y} \rfloor) \qquad (2)$$

Algorithm 4 Affine Mapping (AFFN) Heuristic for 2D to 2D mapping

```
procedure AFFN($O_x$, $O_y$, $P_x$, $P_y$)
    for $i := 1$ to $O_x$ do
        for $j := 1$ to $O_y$ do
            $af_x = \lfloor P_x \times \frac{i}{O_x} \rfloor$
            $af_y = \lfloor P_y \times \frac{j}{O_y} \rfloor$
            $< free_x, free_y >$ = findNearest2D($af_x, af_y, P_x, P_y$)
            Map[$i$][$j$] = $free_x \times P_y + free_y$
        end for
    end for
end procedure
```

Since the coordinates are constrained to be integers, it is possible that two objects may be mapped to the same processor. To resolve this, we use the function findNearest2D which returns an unused processor closest to (af_x, af_y). It should be noted that this mapping is not strictly affine since we use findNearest to resolve conflicts for the same processor (see Algorithm 4). This mapping is referred to as AFFN in figures and tables.

Heuristic 6 - Step Embedding: This algorithm is an imple-

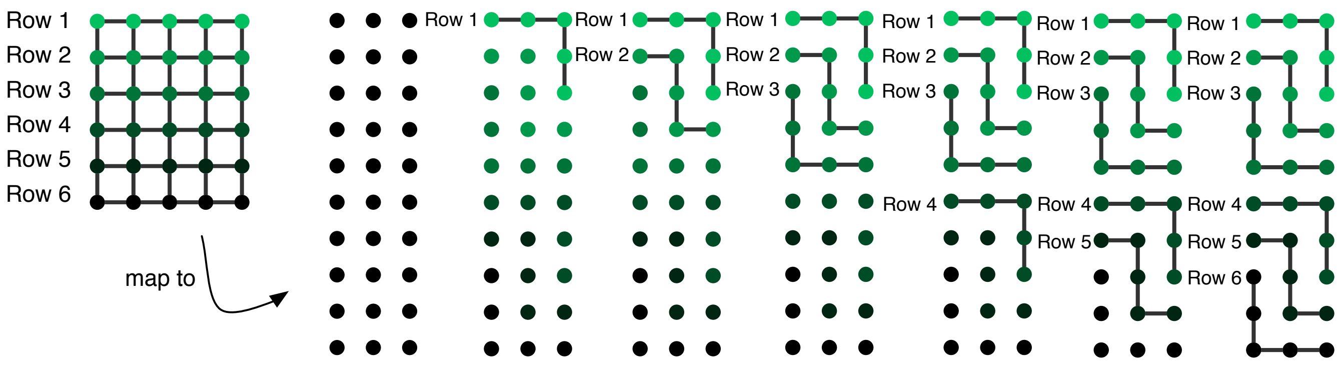


Fig. 4. Mapping of a 6×5 object graph to a 10×3 processor graph using the `STEP` algorithm

mentation of the step embedding technique (`STEP`) presented in [10]. Techniques in [10] were written to optimize chip layout and hence they try to minimize the length of the longest wire. The paper presents ways to "square up" an arbitrary rectangular grid. However, unlike our mapping algorithms, for `STEP`, the number of nodes in the processor mesh can be greater than that in the object grid. We borrow the idea of visualizing the mappings for the object grids from this paper.

Heuristic 7 - Pairwise Exchanges: Several research papers in the past have used the technique of pairwise exchanges by itself or with an intelligent initial mapping [15], [27]. In this technique (`PAIRS`), we start with a chosen initial mapping, choose two objects randomly and swap the processors they are placed on. We retain this swap if a chosen metric improves otherwise we discard it. This is continued until the improvement in the chosen metric falls below a certain threshold. The pairwise exchange algorithm with probabilistic jumps, presented in [15] has a $\mathcal{O}(n^3)$ time complexity. This algorithm is too expensive to be practical. However, in absence of the knowledge of the true optimal mapping, we use this algorithm to produce an approximation of the optimal mapping, to which other (faster) heuristic strategies can be compared. We use a simpler implementation for our purposes to obtain values for the hop-byte metric which are close to the optimal solution.

A. Time Complexity

Among the five heuristics presented in this chapter, `MXOVLP` and `MXOV+AL` visit each node in the object grid only once and decide on its mapping. Hence, they have a time complexity of $\mathcal{O}(n)$ where n is the number of objects to be mapped. However, `EXCO`, `COCE` and `AFFN` algorithms use the `findNearest2D` function, which in the worst case can take $\mathcal{O}(n)$ time. Hence the worst case running time of these three algorithms is $\mathcal{O}(n^2)$. In the era of petascale machines with hundreds of thousands of cores, it is crucial to use linear or linearithmic running time algorithms for mapping and all presented heuristics adhere to that on the average.

B. Quality of Mapping Solutions: Hop-bytes

This section evaluates the mapping algorithms presented above. We use the *hop-bytes* metric to compare across them. For better intuition, the graphs in this section present the average hops per byte for each algorithm. The ideal value for average hops per byte is 1 and so the closer to 1 we get, the better the mapping algorithm is. In an effort to find the hops per byte for close to optimal solutions, we also implemented the $\mathcal{O}(n^3)$ algorithm of pairwise exchanges (`PAIRS`) used in literature [15], [27].

Visualization of the mapping of an object graph to a processor graph helps understand mapping algorithms better and also helps in fixing potential problems with them. Figure 4 shows a step-by-step process of mapping individual rows of the object graph on to the processor graph. We used the `STEP` algorithm for this figure which maps rows one by one. Each dot represents a node in the graph and the edges are communication arcs (object graph on extreme left). The 7 graphs with dimensions 10×3 show how individual rows of the object graph are mapped on to the processor mesh. Similar diagrams are used to show the mapping of individual rows for other heuristics even though they do not map graphs by rows. It helps one compare across the various mapping solutions in a graphical manner.

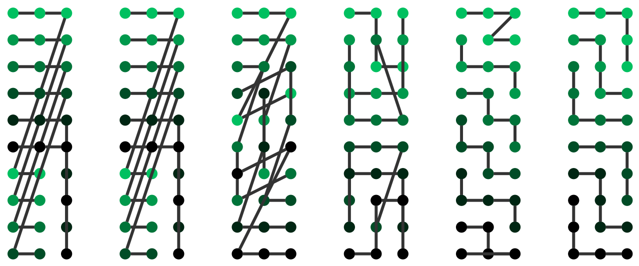


Fig. 5. Mapping of a 6×5 grid to 10×3 grid using `MXOVLP, MXOV+AL, EFC, COCE, AFFN` and `STEP` heuristics

Figure 5 presents mappings of a 6×5 to 10×3 grid to a

10×3 grid pictorially. The six 2D grids in the figure illustrate mappings of the object graph onto the processor graph based on the six heuristic algorithms: `MXOVLP`, `MXOV+AL`, `EFC`, `COCE`, `AFFN` and `STEP`. We can see that the first three heuristics stretch some edges significantly while the rest try to minimize both hop-bytes and maximum dilation.

Figure 6 compares the hops per byte for the different algorithms assuming that communication is 2D near-neighbor and regular. The representative object grids and processor meshes were chosen so as to cover a wide range of aspect ratios. The maximum overlap with alignment heuristic (`MXOV+AL`) gives the best solution in most cases. The `AFFN` heuristic which does a affine inspired mapping also performs quite well. In the case of mapping of a 100×40 grid to a 125×32 mesh, `AFFN` does considerably better than the other algorithms.

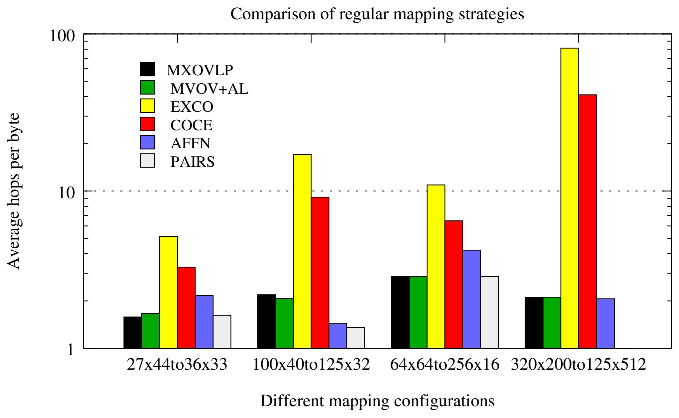


Fig. 6. Hop bytes for different techniques compared to the lower bound

VI. 2D to 3D Mapping Algorithms

Some of the largest and fastest supercomputers in the Top500 [28] list today have a 3D torus or mesh interconnect. So, in order to use our mapping algorithms on these real machines, we need to develop algorithms to map 2D communication graphs to 3D processor topologies. We now present algorithms for mapping 2D grids to 3D processor meshes. Some algorithms presented in this section use the 2D to 2D mapping algorithms developed in the previous section. The significance of 2D to 2D mapping heuristics is in using them as "base cases" for the 2D to 3D mapping because different heuristics perform best for different aspect ratios.

Heuristic 1: Stacking: The general idea is to use the algorithms developed in the previous section for mapping 2D object grids to 2D processor meshes. We find the longer dimension of the 2D object grid and split the object graph along it, into several smaller grids (subgrids). The number of subgrids equals the length of the smallest dimension of the 3D processor mesh. We then take the first subgrid and map it onto a plane perpendicular to the smallest dimension of the 3D processor mesh. The mapping framework chooses the best heuristic to map the 2D subgrid to the 2D processor mesh. A simple translation is used to map the remaining subgrids to other planes of the processor mesh. For example, if we wish to map a 16×32 object grid onto a $8 \times 8 \times 8$ processor mesh, we split the longer dimension (32) into 8 pieces (the smallest dimension of the processor mesh) and then map a 16×4 object subgrid to a 8×8 processor mesh.

Heuristic 2: Folding: If the processor topology is a 3D mesh, then in the previous heuristic, elements at the boundaries of the subgrids are separated by a large distance in the processor mesh. To avoid this, we use a more general strategy where we fold the 2D object grid like an accordion folder and place the folded parts perpendicular to the smallest dimension. To achieve this, once we obtain the mapping for the first subgrid, instead of a simple translation, we flip the mapping for every alternate subgrid by 180 degrees. Figure 7 shows the stacking and folding heuristics to map a 2D grid to a 3D torus or mesh.

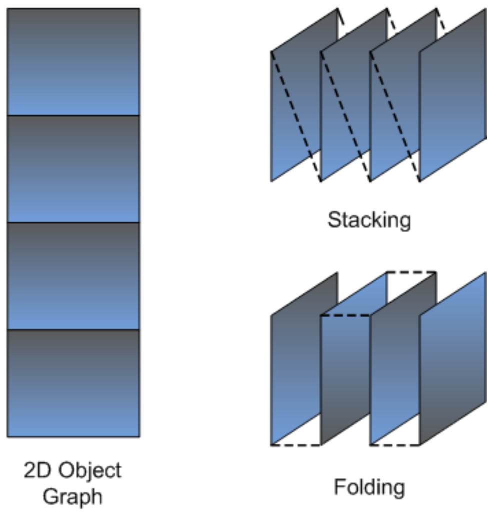


Fig. 7. Stacking and folding of 2D graph to 3D

Heuristic 3: Space Filling Curve: A space filling curve is a continuous functions whose domain is the unit interval [0,1]. These curves (discovered by Peano in 1890 [29], [30] and generalized later on) can be used to fill the entire n-dimensional Euclidean space (where n is a positive integer). We use a space filling curve to map the 2D object grid to a 1D line and another space filling curve to map the 1D line to a 3D processor mesh. Space filling curves preserve locality and hence we expect the dilation and hop bytes to be small under this construction.

VII. Application Results

Using the algorithms developed in the previous sections, we attempted topology aware mapping of two applications: a 2D Stencil benchmark in MPI and the Weather Research and Forecasting (WRF) program. All performance runs were done on the IBM Blue Gene/P machines at Argonne National Laboratory and Watson Research Center. We use `TXYZ` mapping as the default mapping. This means that MPI ranks are mapped in order on the four cores of a node first, then along the increasing X dimension, then Y and Z respectively.

A. 2D Stencil

Many scientific applications [8], [9] have a communication structure similar to a five-point stencil. We use a representative code where there is some computation followed by the exchange of data with neighbors in every iteration. Each element communicates with four neighbors in the 2D grid.

We study the weak scaling behavior of the application. In other words, the amount of computation and communication on each processor remains the same as we scale to more and more processors. The effect of congestion in the network, however, may be different with varying partition sizes (number of processors) owing to the difference in topology and corresponding mapping. We compare the performance of the default mapping with the "folding" scheme presented in Section VI.

# Cores	Torus	Stencil	2D to 2D Map
256	$(4,4,4)$	$(32,8)$	$(4,4)$ to $(4,4)$
512	$(4,4,8)$	$(16,32)$	$(8,4)$ to $(4,8)$
1024	$(8,4,8)$	$(64,16)$	$(8,8)$ to $(8,8)$
2048	$(8,8,8)$	$(32,64)$	$(16,4)$ to $(8,8)$
4096	$(8,8,16)$	$(32,128)$	$(16,8)$ to $(8,16)$
8192	$(8,8,32)$	$(64,128)$	$(32,8)$ to $(8,32)$
16384	$(8,16,32)$	$(64,256)$	$(32,16)$ to $(16,32)$

TABLE I
DIMENSIONS OF THE 2D STENCIL, 3D TORUS AND ASPECT RATIOS OF THE 2D TO 2D BASE CASES

The results from this experiment are presented in Figure 8. The aspect ratios of the Stencil at different core counts and the dimensions of the torus partitions are given in Table I. Each processor holds a 2D array of doubles (size 64×64) and exchanges 4 messages (containing 64 doubles corresponding to the first and last rows and columns). Topology aware mapping leads to performance improvements for most processor counts, the maximum being 11% at $16,384$ cores (refer to the line plots in Figure 8). A peculiar observation is the reduction of time per step from $1,024$ to $2,048$ cores for the default TXYZ mapping. This is because the torus links become available only at $2,048$-core and larger allocations. The availability of these links reduces the congestion significantly. We observe that as the application is scaled to larger number of cores, the congestion in the network and hence the time per step keeps increasing for the TXYZ Mapping. On the other hand, by using the topology aware mapping, the time per iteration for all the runs remains practically unchanged. We conclude that topology aware mapping can lead to performance benefits and improve scaling for this class of applications.

It is interesting to compare the improvement in performance (line plots) in Figure 8 with the improvement in hops per byte (bar graph in the same figure) in each of the above cases. It is evident that decreasing the hops per byte ratio leads to decreased congestion in the network. As demonstrated by the $1,024$ and $16,384$-core runs, a larger reduction in hop-bytes translates into larger performance improvements from

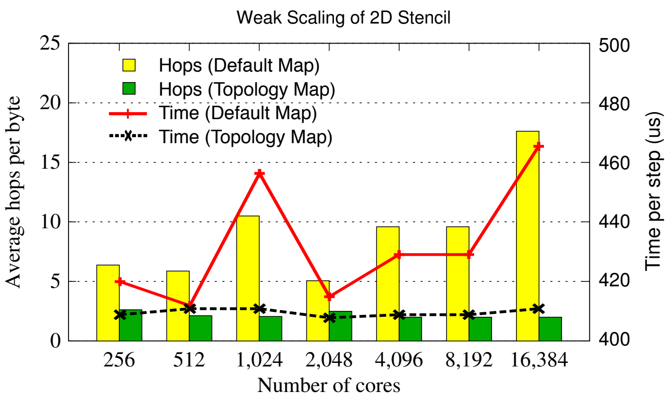


Fig. 8. Weak scaling experiment results for 2D Stencil

topology aware mapping.

We also study the effect of varying the ratio of computation and communication for this application. This is motivated by the following factors:

- Different supercomputers have varying processing and communication characteristics. For example, comparing the network bandwidth available per floating point operation, Blue Gene/P can transfer 0.375 bytes per flop whereas Cray XT4 can transfer 1.357 bytes per flop to the network. The general trend for building faster supercomputers is to increase the number of cores per node and use faster processors. As a result, computation will tend to be faster and communication will be a bottleneck.
- The same application may use a bigger stencil (e.g. a nine-point stencil) for the purpose of achieving faster convergence at the cost of doubling the communication and communication per time step. Increased communication will lead to an increased congestion in the network.

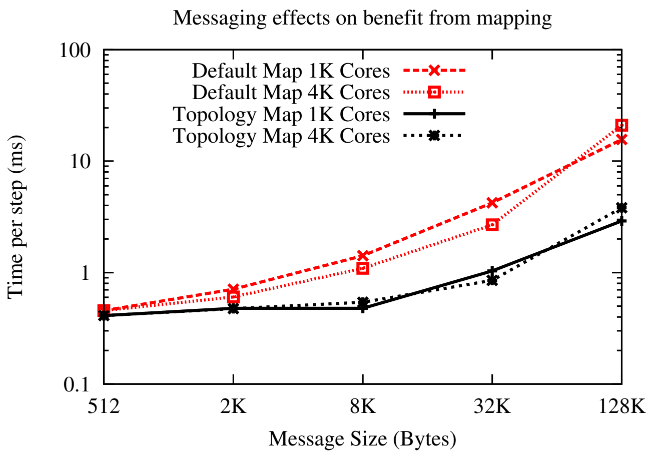


Fig. 9. Effect of increasing communication per step for 2D Stencil

The ratio of computation versus communication was varied by increasing the message size while keeping the amount of computation constant. The results for this experiment are pre-

# **Nodes**	Torus Dimensions	WRF Dimensions	Percentage reduction in		
			Hops	Comm. Time	Total Time
256	$8 \times 4 \times 8$	16×16	33.9	0.7	0.3
512	$8 \times 8 \times 8$	32×16	41.8	-1.4	-1.4
1024	$8 \times 8 \times 16$	32×32	63.2	11.1	16.7
2048	$8 \times 16 \times 16$	64×32	66.3	-19.4	-40.7
4096	$16 \times 16 \times 16$	64×64	60.4	1.5	4.8

TABLE II
PERCENTAGE REDUCTION IN AVERAGE HOPS PER BYTE, COMMUNICATION TIME AND TOTAL TIME USING TOPOLOGY AWARE MAPPING

sented in Figure 9. On the X-axis, we have increasing message sizes for a fixed amount of computation. This experiment was run on two partitions of dimensions $8 \times 4 \times 8$ (1K cores) and $8 \times 8 \times 16$ (4K cores). Notice that the default mapping leads to significant congestion in the network leading to long delays and a considerable increase in the running time of the application. However topology aware mapping minimizes congestion and hence the message delivery times, especially for large message sizes. The performance improvements for 8 KB messages at $1,024$ cores and $4,096$ cores are 66% and 53% respectively.

B. WRF Experiments

WRF stands for Weather Research and Forecasting Model [9]. This code is a next-generation mesoscale numerical weather prediction system that is being designed to serve operational forecasting and atmospheric research. For our experiments, we used the weather data from the 12 km resolution case over the Continental U.S. (CONUS) domain on October 24, 2001. We used profiling tools to obtain the communication graph of WRF which was given as input to the mapping framework. The process topology analyzer found that WRF has a 2D near-neighbor communication pattern (the specific dimensions when running on specific number of nodes are tabulated in Table II.) Based on the process topology findings, the framework output mapping files which were passed as an option to the job scheduler on Blue Gene/P. As an example, for WRF running on 1024 nodes, the application grid is 32×32 and the processor mesh is $8 \times 8 \times 16$. We use folding and the base case requires mapping a subgrid of dimensions 32×4 to a 8×16 mesh.

WRF was run in the SMP mode (using 1 process per node) because it uses OpenMP to create threads on each node. We compare the topology aware mapping results with the default `XYZ` mapping on Blue Gene/P using average hops per byte. The hops per byte are obtained by using IBM's HPCT profiling tools [22]. Figure 10 shows the actual weighted hops (hops per byte averaged over all MPI ranks) obtained from profiling data for WRF. The corresponding percent performance improvements in total communication time and elapsed time are shown in Table II. The empirically obtained values correlate strongly with the algebraically calculated hops. It is evident that topology aware mapping of MPI ranks to physical processors is successful in decreasing the average hops per rank and bringing it close to the lower bound. The average hops for WRF reduce by more than 60% for runs using more than $1,024$ nodes. This is quite significant and should lead to a dramatic drop in the load on the network.

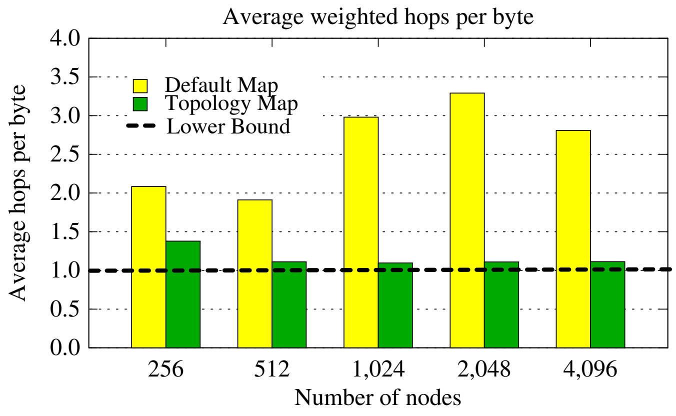


Fig. 10. HPCT data for actual hops per byte for WRF

Using topology aware mapping, we were able to bring the average hops per rank close to 1 (Figure 10). This suggests that most MPI ranks are sending messages only 1 hop away and we should see performance improvements. When running WRF on $1,024$ nodes, the average hops per byte reduced by 63% and communication time reduced by 11%. We see an overall performance improvement of 17%. At $4,096$ nodes, we see a reduction in total execution time by 5%. Such performance improvements can be quite significant for the overall completion time of long running simulations. On $2,048$ nodes, the communication time and overall time increases with topology aware mapping although the hops per byte reduces and we have not been able to analyze this further yet. This information is summarized in Table II above.

It is important to reiterate the point made in Section I that the performance of a parallel application is a complex function of various factors. The routing protocols, latency tolerance of the application and fraction of time spent in communication can affect performance in varying degrees. Hence, a reduction in hop-bytes and a corresponding improvement in the communication behavior of an application may not always lead to an overall performance improvement.

VIII. Conclusion

This paper presents a framework for automatic mapping of parallel applications with regular communication graphs onto parallel machines with 3D mesh and torus networks. The two steps involved in automating the process are: 1. obtaining the communication graph and identifying 2D, 3D or higher-dimensional regular patterns, 2. intelligent and fast heuristic solutions for mapping such graphs. Several heuristic techniques for mapping 2D object grids to 2D and 3D processor meshes have been developed. These heuristics along with pattern identification techniques form a framework, for automatic mapping of a wide class of parallel applications with regular communication graphs. This framework will save much effort on the part of application developers to generate mappings for their individual applications.

There are several directions for future research. The mapping framework can be enhanced with more sophisticated algorithms for process topology discovery and topology aware mapping. The topology analyzer only identifies communication with a small number of neighbors. Other cases for consideration are complex communication patterns such as many-to-many and multicasts. The mapping framework does not handle simultaneous multiple communication patterns in an application. The class of applications which has not been targeted in this paper is that with irregular communication. We plan to build on the framework developed in this paper by including techniques for mapping such applications.

Acknowledgments

This work was supported in part by a DOE Grant DE-SC0001845 for HPC Colony II and NSF Grant 0720827 for BigSim. We used running time on the Blue Gene/P at Argonne National Laboratory, which is supported by DOE under contract DE-AC02-06CH11357.

References

[1] Eduardo Huedo and Manuel Prieto and Ignacio Martín Llorente and Francisco Tirado, "Impact of PE Mapping on Cray T3E Message-Passing Performance," in *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*. London, UK: Springer-Verlag, 2000, pp. 199–207.

[2] Brian E. Smith and Brett Bode, "Performance Effects of Node Mappings on the IBM Blue Gene/L Machine," in *Euro-Par*, 2005, pp. 1005–1013.

[3] A. Bhatelé and L. V. Kalé, "Quantifying Network Contention on Large Parallel Machines," *Parallel Processing Letters (Special Issue on Large-Scale Parallel Processing)*, vol. 19, no. 4, pp. 553–572, 2009.

[4] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *Cluster Computing, 2008 IEEE International Conference on*, October 2008, pp. 116–125.

[5] Deborah Weisser, Nick Nystrom, Chad Vizino, Shawn T. Brown, and John Urbanic, "Optimizing Job Placement on the Cray XT3," *48th Cray User Group Proceedings*, 2006.

[6] Cray Inc., "Cray XT Specifications," http://www.cray.com/Products/XT/Specifications.aspx, 2009.

[7] C. Bernard, T. Burch, T. A. DeGrand, C. DeTar, S. Gottlieb, U. M. Heller, J. E. Hetrick, K. Orginos, B. Sugar, and D. Toussaint, "Scaling tests of the improved Kogut-Susskind quark action," *Physical Review D*, no. 61, 2000.

[8] J. K. Dukowicz and R. D. Smith, "Implicit free-surface method for the Bryan-Cox-Semtner ocean model," *Journal of Geophysics Research*, vol. 99, pp. 7991–8014, Apr. 1994.

[9] Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, "The Weather Research and Forecast Model: Software Architecture and Performance," in *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, October 2004.

[10] Aleliunas, R. and Rosenberg, A. L., "On Embedding Rectangular Grids in Square Grids," *IEEE Trans. Comput.*, vol. 31, no. 9, pp. 907–913, 1982.

[11] Ellis, J.A., "Embedding rectangular grids into square grids," *Computers, IEEE Transactions on*, vol. 40, no. 1, pp. 46–52, Jan 1991.

[12] Melhem, Rami G. and Hwang, Ghil-Young, "Embedding Rectangular Grids into Square Grids with Dilation Two," *IEEE Trans. Comput.*, vol. 39, no. 12, pp. 1446–1455, 1990.

[13] P. Sadayappan and F. Ercal, "Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes," *IEEE Trans. Computers*, vol. 36, no. 12, pp. 1408–1424, 1987.

[14] S. Wayne Bollinger and Scott F. Midkiff, "Processor and Link Assignment in Multicomputers Using Simulated Annealing," in *ICPP (1)*, 1988, pp. 1–7.

[15] Shahid H. Bokhari, "On the Mapping Problem," *IEEE Trans. Computers*, vol. 30, no. 3, pp. 207–214, 1981.

[16] A. Bhatelé, E. Bohm, and L. V. Kalé, "A Case Study of Communication Optimizations on 3D Mesh Interconnects," in *Euro-Par 2009, LNCS 5704*, 2009, pp. 1015–1028.

[17] A. Bhatelé and L. V. Kalé, "Benefits of Topology Aware Mapping for Mesh Interconnects," *Parallel Processing Letters (Special issue on Large-Scale Parallel Processing)*, vol. 18, no. 4, pp. 549–566, 2008.

[18] A. Bhatelé, L. V. Kalé, and S. Kumar, "Dynamic topology aware load balancing algorithms for molecular dynamics applications," in *23rd ACM International Conference on Supercomputing*, 2009.

[19] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, and M. C. Pitman, "Blue matter: Approaching the limits of concurrency for classical molecular dynamics," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM Press, 2006.

[20] Kei Davis and Adolfy Hoisie and Greg Johnson and Darren J. Kerbyson and Mike Lang and Scott Pakin and Fabrizio Petrini, "A Performance and Scalability Analysis of the Blue Gene/L Architecture," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2004, p. 41.

[21] L. Oliker, A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier, and T. Goodale, "Scientific Application Performance on Candidate PetaScale Platforms," in *Proceedings of IEEE Parallel and Distributed Processing Symposium (IPDPS)*, March 2007.

[22] H. Wen and S. Sbaraglia and S. Seelam and I. Chung and G. Cong and D. Klepacki, "A Productivity Centered Tools Framework for Application Performance Tuning," in *QEST '07: Proceedings of the 4th International Conference on the Quantitative Evaluation of Systems*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 273–274.

[23] G. Zheng, "Achieving high performance on extremely large parallel machines: performance prediction and load balancing," Ph.D. dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

[24] R. Preissl, T. Köckerbauer, M. Schulz, D. Kranzlmüller, B. R. de Supinski, and D. J. Quinlan, "Detecting Patterns in MPI Communication Traces," *Parallel Processing, International Conference on*, vol. 0, pp. 230–237, 2008.

[25] D. J. Kerbyson and K. J. Barker, "Automatic Identification of Application Communication Patterns via Templates," in *In Proc. Int. Conf. Parallel and Distributed Computing Systems (PDCS)*, Las Vegas, NV, August 2005.

[26] N. Bhatia, F. Song, F. Wolf, J. Dongarra, B. Mohr, and S. Moore, "Automatic experimental analysis of communication patterns in virtual topologies," *Parallel Processing, International Conference on*, vol. 0, pp. 465–472, 2005.

[27] Soo-Young Lee and J. K. Aggarwal, "A Mapping Strategy for Parallel Processing," *IEEE Trans. Computers*, vol. 36, no. 4, pp. 433–442, 1987.

[28] "Top500 supercomputing sites," http://top500.org.

[29] G. Peano, "Sur une courbe, qui remplit toute une aire plane," *Mathematische Annalen*, vol. 36, no. 1, pp. 157–160, 1890.

[30] D. Hilbert, "Ueber die stetige Abbildung einer Line auf ein Flächenstück," *Mathematische Annalen*, vol. 38, pp. 459–460, 1891.