Automated Mapping of Structured Communication Graphs onto Mesh Interconnects

Abhinav Bhatelé [†], I-Hsin Chung [⋆] and Laxmikant V. Kalé [†]

[†] Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
E-mail: bhatele@illinois.edu, kale@illinois.edu

[⋆] IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
E-mail: ihchung@us.ibm.com

Abstract

Network contention has an increasingly 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 structured communication graphs to two and three dimensional mesh networks. We present several heuristic techniques for mapping 2D object graphs to 2D and 3D processor graphs and compare their performance with other algorithms in literature.

We use the *hop-bytes* metric to evaluate and compare across different mapping strategies and justify that it is more important to reduce the average hop-bytes than *maximum dilation*. We test our algorithms on three scientific applications, MILC, POP and WRF and present performance improvements of more than 15% in some cases on IBM's Blue Gene/P machine. The main contribution of this paper, is the automated mapping framework for a wide class of MPI applications with structured communication graphs. This framework will save much effort on the part of application developers to generate mappings for their individual applications.

1 Introduction

Should application programs be written taking the interconnection topology into account? The glib and naïve answer may be "Of course, yes!". Yet, since the mid 1980's until just a year or two ago, the correct answer was "no". It is significantly more challenging (and certainly more work) to take the interconnection topology into account; so understanding the correct 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 of the hops 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 hundreds of bytes in size, the latency is insignificantly 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 interconnection 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 on the network 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.

It is therefore necessary now to undertake a research program aimed at topology aware mapping of tasks to processors. Even though there has been significant research in related areas in early 1980's, it was based on the metric of longest dilation; this metric is not a good match for the current technological context, since contention is the main issue now. This paper demonstrates the utility of *hop-bytes*, a metric we have been using. Secondly, this 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) grid and torus topologies. It presents several algorithms, some new and some known via the literature, and compares their performance using the *hop-bytes* metric.

The main contribution of the paper, is a framework for automatic mapping of a wide class of MPI applications with structured communication graphs. From pattern recognition of communication graphs 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. This framework will save much effort on the part of application developers to generate mappings for their individual applications.

This paper is organized as follows: Section 2 expands on the introduction, motivates the work further and places it in perspective. *Hop-bytes* is established as a good metric for evaluation of quality of mapping algorithms in Section 3. The two important steps for automatic mapping are presented in the subsequent sections: Section 4 presents algorithms for identifying communication patterns from profiling data. Sections 5 and 6 present mapping algorithms for 2D object graphs to 2D and 3D processor meshes. Section 7 presents results on three applications with structured communication: MIMD Lattice Computation (MILC), Parallel Ocean Program (POP) and Weather Research and Forecasting Model (WRF).

2 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 [1, 7, 11]. 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 [1]. As we shall show in subsequent sections, 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 [14, 15, 17]. 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 such techniques to improve performance for their codes [3, 4, 5, 8, 10, 13]. 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 structured 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.

Another important contribution of this paper is the insight that total communication over the network (quantified by *hop-bytes*) is a better metric for performance than maximum dilation of messages on the network. Algorithms presented in this paper were developed in light of this idea. Our work was also influenced by Yu et al. [18] and we compare the performance of our algorithms with the ones in that paper. For completeness, we also implemented the step embedding algorithm presented in [1] for VLSI circuits to compare with the algorithms developed by us.

3 Hop-bytes as an Evaluation Metric

The volume of inter-processor communication can be characterized 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 lot of messages travel multiple links, the total hop-bytes increase which suggests increased likelihood of contention. Hop-bytes can be calculated by the equation,

$$HB = \sum_{i=1}^{n} d_i \times b_i \qquad \text{... (1)}$$

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.

In VLSI circuit design and early parallel computing work, emphasis was placed on minimizing the longest length of the wire which is referred to as the maximum dilation in mathematical terms. We claim that reducing the largest number of links traveled by any message is not as critical as reducing the average hops across all messages. An MPI benchmark was created to justify this claim. Every pair talks with its six neighbors (two in each dimension) to create some background communication. We then add one of these patterns: 1. Each rank sends two messages, one to a neighbor three hops away and another to a neighbor six hops away ("avg: 1.88 max: 6" line on the

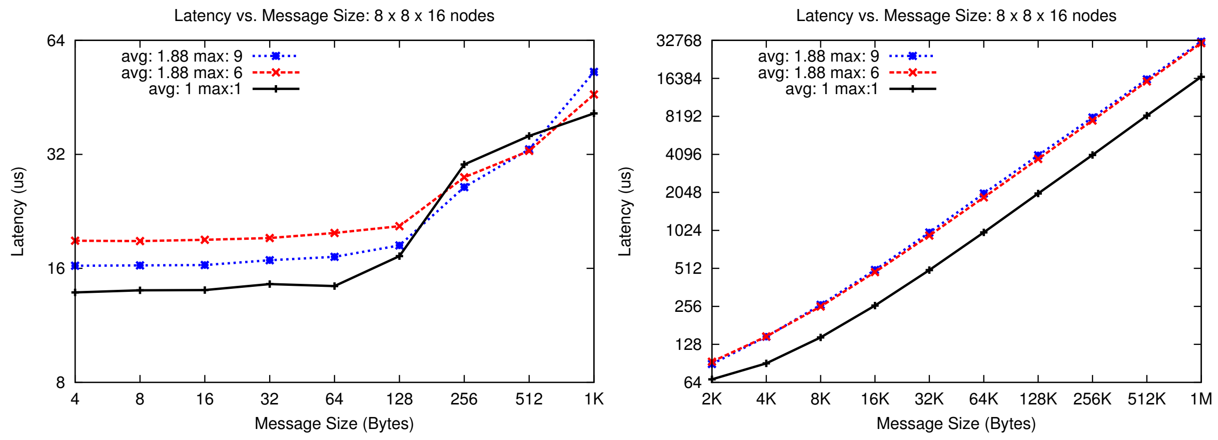


Figure 1: Plots showing that average hops is an important factor guiding performance. Runs were done on a $8 \times 8 \times 16$ partition of BG/P

plots), or 2. Each rank sends a message to a neighbor nine hops away ("avg: 1.88 max: 9" line on the plots). The "avg: 1 max: 1" line due to six near-neighbor messages is plotted as a baseline.

While the difference in the maximum hops between the two cases is three, the total hop bytes is the same and hence, we expect to see similar performance. Figure 1 shows the results obtained from running this benchmark. We ran the benchmark on ANL's Blue Gene/P (BG/P) on $1,024$ nodes which form a $8 \times 8 \times 16$ torus. Message size is varied from 4 bytes to 1 MB and multiple trials are done for each point on the plot. Time for the entire communication pattern to finish is recorded between barriers. For small messages, the max: 9 case does better than max: 6 cases because only one message is sent in the first one whereas two are sent in the second case (alluding to per-message overheads). For large messages, we see that the lines coincide confirming our predictions that hop-bytes is a more important metric than dilation.

However, it is 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. It is a measure of the bytes the network has to deliver for an application to run to completion. So, information about specific hot-spots on the network is not expressible through this metric. Actual load on different links depends on the actual routing protocols used on a specific machine, hence it is not usable as a metric to evaluate algorithms without doing actual runs.

We created an MPI benchmark to demonstrate that hop-bytes is a good metric overall but we should still consider other factors such as routing protocols and hot-spots. In this benchmark, each MPI rank is paired with a partner and all pairs send messages simultaneously. Both partners in a pair call `MPI_Irecv`, `MPI_Send` and then `MPI_Wait`. The pairs always have the same X and Y coordinate on the torus. The benchmark is run in different modes depending on how the ranks are grouped into pairs:

- *avg: 1 max: 1 hop*: Every rank is paired with one which is exactly one link away from it along the Z direction.
- *avg: 2 max: 2 hops*: Every rank is paired with one which is exactly two links away from it in the Z direction.
- *avg: 2 max: 8 hops*: Most ranks are paired with a partner which is one link away but one

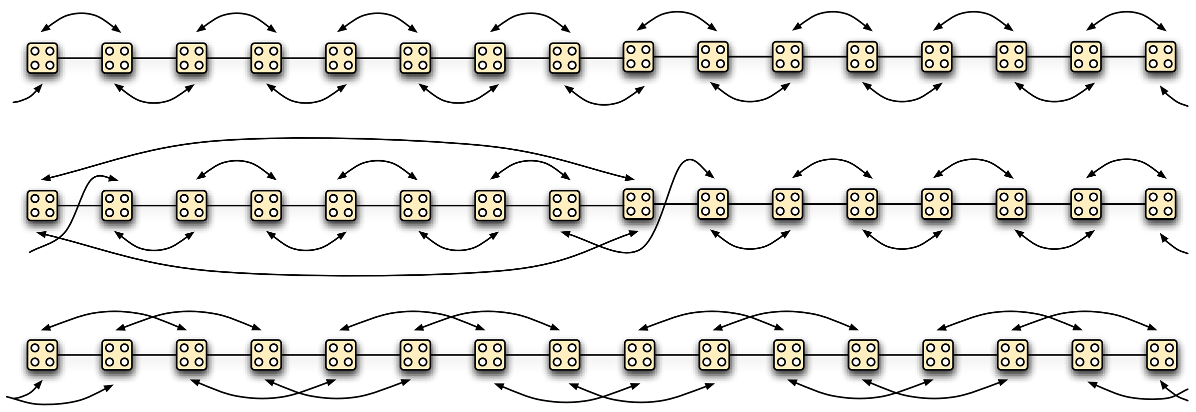


Figure 2: Communication patterns along the Z dimension in the artificial benchmark

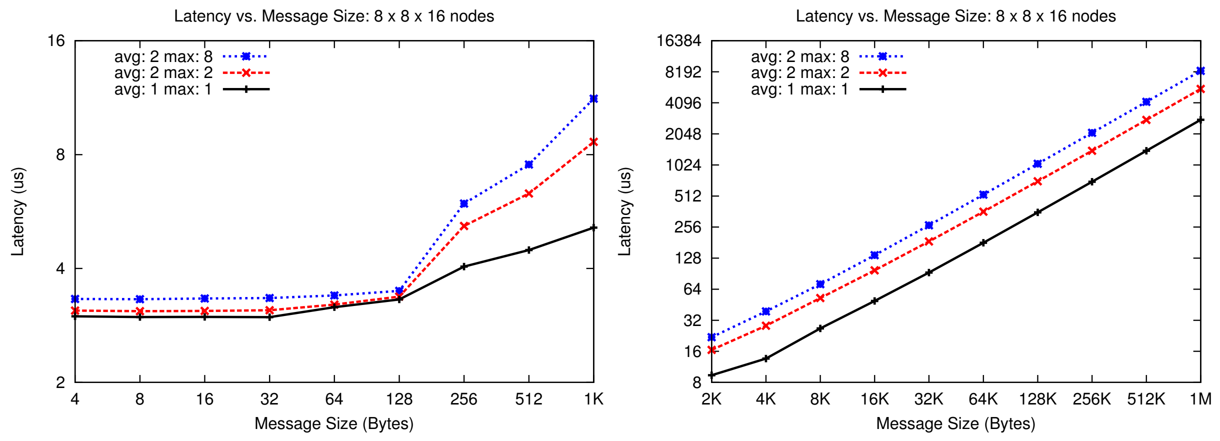


Figure 3: Plots showing that maximum dilation can also impact performance. Runs were done on a $8 \times 8 \times 16$ partition of BG/P

pair is such that the distance between the partners is 8 links or hops. The average hops is 2.

Figure 2 shows the pairing of 16 nodes along the Z dimension. Figure 3 shows the results from running the benchmark on $1,024$ nodes of BG/P. There are several observations to be made from these plots. Small messages (less than 128 bytes) are not affected severely because there is negligible contention. For messages greater than 256 bytes, a clear trend is that as the average number of hops increases, the time increases significantly (note, the difference between avg: 1 and avg: 2 lines and that the y-axis has a logarithmic scale). So average hop-bytes is a good indicator of the contention created on the network.

Furthermore, when the maximum hops is different between two cases (avg: 2), the performance worsens further. For a further diagnosis of this situation, we used the IBM Performance Monitor library [16] to obtain information about the number of bytes passing through each node. Figure 4 shows the data obtained from `BGP_TORUS_ZP_32BCHUNKS` and `BGP_TORUS_ZM_32BCHUNKS` which gives the number of 32 byte chunks passing through the $+Z$ and $-Z$ links of each node. Since there is no communication in the X and Y direction, counters for those are zero. For these profiling runs, only $1,024$-byte messages were used.

The figure shows that when the maximum hops is 8, more packets are sent along the $+Z$ links than the $-Z$ links which leads to a degradation in performance. This exercise shows that subtle routing choices can affect contention and lead to performance degradation. Since the maximum hops is not captured by the hop-bytes metric, it is sometimes relevant to consider that as a factor affecting performance. That said, since hop-bytes is still a good indicator of the communication traffic on the network, we will use it as a basis for evaluation of mapping algorithms in the subsequent sections.

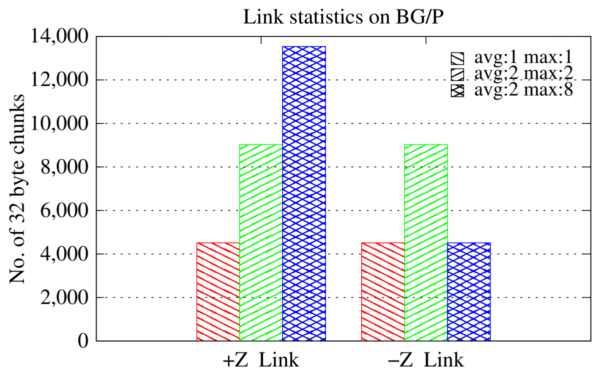


Figure 4: HPM Counters Data for Z links

The next few sections will describe the various components of the automated mapping framework. Two important steps in mapping of a parallel application are: 1. Obtaining the communication graph through profiling and identifying specific communication patterns in the communication graph, 2. Depending on the communication patterns identified, using different heuristics in different cases to obtain mapping solutions.

4 Identifying Communication Patterns

Automatic topology aware mapping, as we shall see in the next few sections, uses heuristics for fast scalable runtime solutions. Since different heuristics apply to different communication scenarios, we need to look for identifiable communication patterns, if any, in the object graph. The first step is obtaining the communication graph for an MPI application. 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. The communication graph is obtained by profiling libraries in the IBM High Performance Computing Toolkit (HPCT) [9]. 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.

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 will now discuss an algorithm we have developed which takes a communication graph as input and if it identifies a 1D, 2D, 3D or 4D communication pattern, outputs the dimensions of the graph. We can also visualize the object graphs as we shall see in Section 7.

We will explain the algorithm for identifying if the communication in an MPI application is 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 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. 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

Algorithm 1: Pseudo code for identifying structured communication graphs

```
begin
    Data: $CM_{n,n}$ (communication matrix)
    Result: isStrc (boolean, true if communication is structured)
            dims[ ] (dimensions of the structured communication graph)

    for $i \leftarrow 1$ to $n$ do
        Find the maximum number of neighbors for any rank in $CM_{i,n}$;

    if max neighbors $\leq$ 5 then
        // this might be a case of structured 2D communication
        // select an arbitrary rank $start_{pe}$ find its distance from its neighbors
        $distance$ = difference between ranks of $start_{pe}$ and its top or bottom neighbor
        for $i \leftarrow 1$ to $n$ do
            // if the distance of all ranks from their neighbors is 1 or the same as
            // calculated above, then
            isStrc = true;
            dim[0] = $distance$;
            dim[1] = $n/distance$;

end
```

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. Then we ensure for all other ranks 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 shows the pseudo code for identifying one possible 2D communication pattern. Currently, this algorithm can only identify a stencil-like structured computation, but we plan to enhance it, so it can identify other structured patterns such as communication with all 8 neighbors around a rank in 2D. The code for identifying 3D and 4D near-neighbor patterns is similar. Once the information about communicating neighbors has been extracted and identified, the mapping algorithms can use it to map communicating neighbors on nearby physical processors.

5 2D to 2D Mapping Algorithms

This section describes different heuristics to map a 2D communication (object) graph to a 2D processor graph (mesh). We assume that all arcs in the communication graph are of the same weight and that the number of nodes in the two graphs is the same. Since 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) and hence the assumption. We will describe five different heuristics to map a 2D object graph to a 2D processor graph. 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 graph which can overlap with the processor graph and maps it one-to-one. For the remaining area of the object graph and processor graph, we then make a recursive call to the

Algorithm 2: Maximum Overlap Heuristic (`MXOVLP`) for 2D to 2D: mapping_algo1

```
begin
    Data: $obj_x$ (x dimension of the object graph)
          $obj_y$ (y dimension of the object graph)
          $proc_x$ (x dimension of the processor graph)
          $proc_y$ (y dimension of the processor graph)
    Result: $Map_{obj_x,\, obj_y}$ (mapping of objects to processors)

    if $obj_x == proc_x$ then
        do a one-to-one mapping and return;

    if $obj_x > proc_x$ then
        map the area $obj_y \times proc_x$;
        mapping_algo1($obj_x - proc_x, obj_y, proc_x, proc_y - obj_y$);
        copy the mapping into the main array and return;
    else
        map the area $proc_y \times obj_x$
        mapping_algo1($obj_x, obj_y - proc_y, proc_x - obj_x, proc_y$);
        copy the mapping into the main array and return
end
```

algorithm. For example, in Figure 5, an area of $M \times L$ of the object graph can be mapped directly to an area of the same dimensions in the processor graph. Once this is done, a recursive call is made for the object graph of size $(K - M) \times L$ to be mapped onto a processor graph of size $M \times (N - L)$. Algorithm 2 presents the pseudo code for this heuristic (referred to as `MXOVLP` in figures and tables).

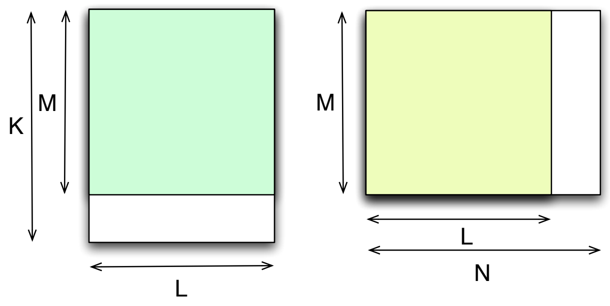


Figure 5: Maximum Overlap (`MXOVLP`)

Heuristic 2 - Maximum Overlap with Alignment: This is similar to Heuristic 1 but it tries to align the longer dimension of the object graph with that of the processor graph (referred to as `MXOV+AL` in figures and tables). This is done at each recursive call and yields a better mapping in most cases. Heuristic 1 and 2 lead to dilation at each recursive call but as per our claim in Section 3, as long as the average is low, we should obtain a good mapping. There are some optimization possibilities in these two heuristics which will be explored in future work: After the mapping for sub-graphs is done, 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 and it leads to a combinatorial explosion. Hence this is out of the scope of this paper.

Heuristic 3 - Expand from Corner: In this algorithm, we start at one corner of the object grid and enumerate all other objects by their increasing Manhattan distance from the chosen corner. We then place these objects in the same order starting from the same corner of the processor grid and moving in the same fashion from nearest to farthest neighbors (pseudo code for `EXCO` in Algorithm 3 on the next page). The code for finding the next processor to place an object on is fairly complicated but it is irrelevant to the discussion in this paper. Figure 6 shows how

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

```
begin
    Data: $obj_x$ (x dimension of the object graph)
          $obj_y$ (y dimension of the object graph)
          $proc_x$ (x dimension of the processor graph)
          $proc_y$ (y dimension of the processor graph)
    Result: $Map_{obj_x,\, obj_y}$ (mapping of objects to processors)

    int cox, coy, nox, noy, cpx, cpy, npx, npy, i;
    nox = noy = npx = npy = 0;
    for $i \leftarrow 1$ to $obj_y \times obj_x$ do
        cox = nox; coy = noy;
        cpx = npx; cpy = npy;
        $Map_{cox,\, coy} = cpy \times proc_x + cpx$;
        nextElem(cox, coy, objX, objY, &nox, &noy);
        nextElem(cpx, cpy, procX, procY, &npx, &npy);
end
```

we start from the upper left corner of the object grid and map objects successively onto the processor grid starting from the corresponding corner.

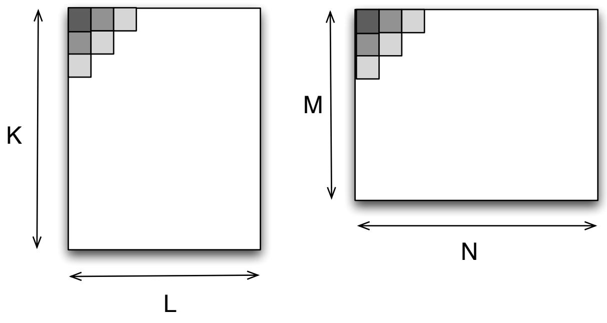


Figure 6: Expand from Corner (`EXCO`)

Heuristic 4 - Corners to Center: This is similar to the third heuristic but in this case we start simultaneously from the four corners of the object grid and move towards the center (referred to as `COCE` in figures and tables). This achieves a better proximity for a larger number of objects since we start simultaneously from four directions. However, from some aspect ratios, as we move closer to the center, objects may get placed further from their communicating neighbors.

Heuristic 5 - Step Embedding: This algorithm is an implementation of the step embedding technique (`STEP`) presented in [1]. Techniques in [1] were written to optimize chip layout and hence try to minimize the length of the longest wire. Although that is not the main motivation for other mapping algorithms, we wanted to compare our algorithms to this one in literature. Results showed that the step embedding algorithm does well for some aspect ratios (Section 7).

5.1 Time Complexity

All the algorithms presented in the previous subsection visit each node in the object graph only once and decide on its mapping. Hence, all of these heuristics have a time complexity of $O(n)$ where n is the number of objects to be mapped. In the era of petascale machines with hundreds of thousands of cores, it is crucial to use linear running time algorithms for mapping and all presented heuristics adhere to that.

Object Grid	**Proc Grid**	MXOVLP	MXOV+AL	EFC	COCE	STEP	OTHER
6×5	10×3	296	296	352	260	284	244
8×4	4×8	256	208	304	272	256	208
5×9	9×5	392	304	432	444	432	584
14×3	6×7	320	292	620	432	328	304
8×9	12×6	692	704	848	692	708	516

Table 1: Hops for different object and processor grids using different heuristics

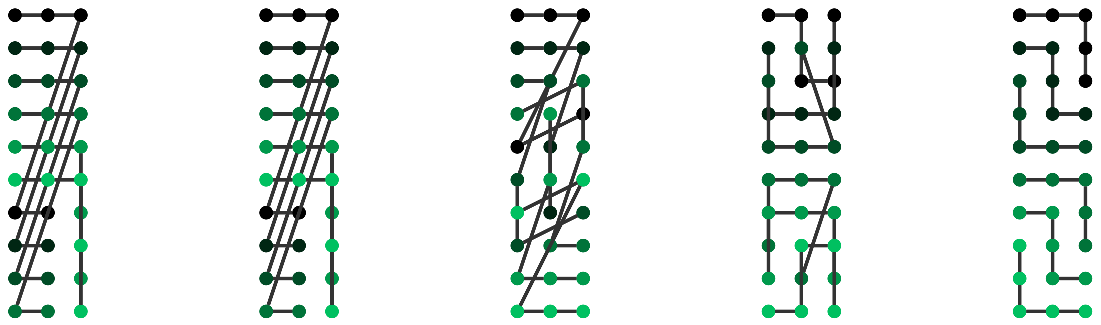


Figure 7: Mapping of a 6×5 to 10×3 grid using MXOVLP, MXOV+AL, EFC, COCE, STEP

5.2 Mapping Solutions

This section evaluates the mapping algorithms presented above. We use the *hop-bytes* metric to compare across them. For simplicity, we assume that all messages are of the same size and hence, we drop the message size term and just use the total number of hops traversed. We compare the four heuristics with the step embedding technique (STEP) in [1] and the folding techniques (OTHER) in [18].

Table 1 shows the total theoretical hops for the different algorithms assuming that communication is 2D near-neighbor and structured. We chose some representative object and processor grids with different aspect ratios which cover a lot of different possible cases. The maximum overlap with alignment heuristic (MXOV+AL) gives the best solution in most cases. The fourth heuristic, COCE which expands from all corners towards the center, does a lot better than others for the first and fifth case. The step embedding technique from VLSI design is worse than the at least one heuristic that we have developed for all the cases above. The folding techniques from [18] perform better than the new heuristics in the first and last case and we will try to incorporate those ideas into our techniques.

Figures 7, 8 and 9 present mappings of the first three cases pictorially. The five 2D grids in each figure are the mappings of the object graphs onto the processor graphs based on the five heuristic algorithms: MXOVLP, MXOV+AL, EFC, COCE and STEP. The dots connected by lines represent one row each of the object graph and how it gets mapped to the processor graph.

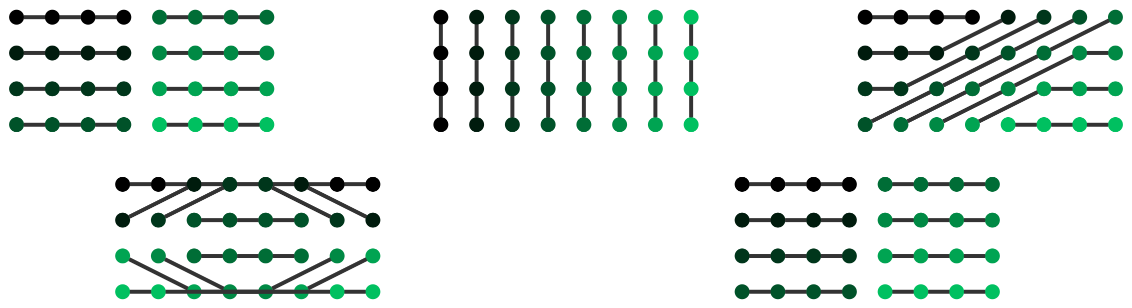


Figure 8: Mapping of a 8×4 to 4×8 grid using `MXOVLP, MXOV+AL, EFC, COCE, STEP`

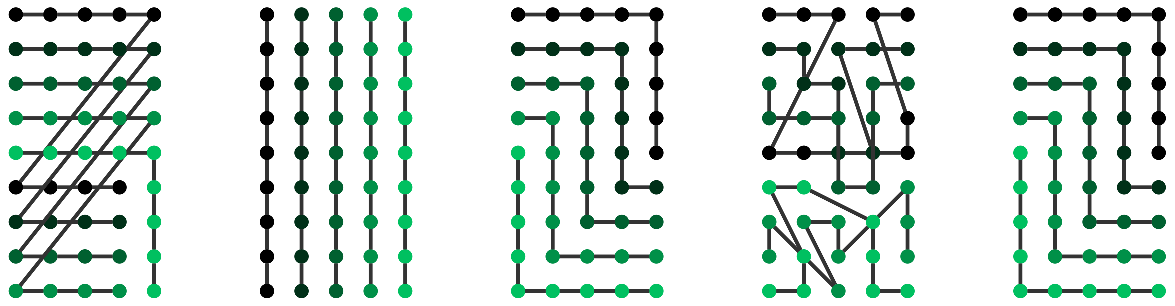


Figure 9: Mapping of a 5×9 to 9×5 grid using `MXOVLP, MXOV+AL, EFC, COCE, STEP`

6 2D to 3D Mapping Algorithms

Some of the largest and fastest supercomputers in the top500[§] 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. Algorithms presented in this section use the 2D to 2D mapping algorithms developed in the previous section. IBM Blue Gene machines allow the user to specify a mapping file with the job submission script which can be used to explicitly place MPI ranks on processors.

We present two heuristics for mapping 2D communication graphs to 3D processor graphs. One is more general than the other, in that it is suitable for both mesh and torus topologies.

Heuristic 1: Stacking: The general idea is to use the algorithms developed in the previous section to map 2D object graphs to 2D topologies. We find the longer dimension of the 2D object graph and split the object graph along that into a number of smaller ones. The number of smaller graphs equals the smallest dimension of the 3D processor graph. We then

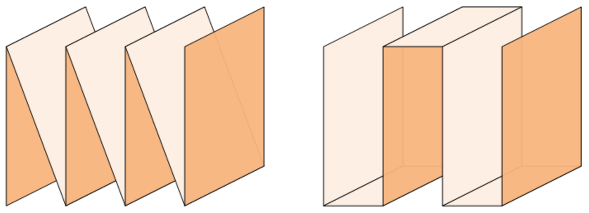


Figure 10: Stacking and Folding

§ http://top500.org/lists/2009/06

take the first piece and map it onto a plane perpendicular to the smallest dimension of the 3D graph. For this mapping, we use the best heuristic for mapping the specific 2D graph to the 2D grid. We then use a simple translation to map all pieces to consecutive planes. For example if we wish to map a 32×8 object grid onto a $8 \times 8 \times 4$ processor grid, we split the longer dimension 32 into 4 pieces (the smallest dimension of the processor graph) and then map a 8×8 object graph to a 8×8 processor graph.

Heuristic 2: Folding: If the processor topology is a 3D mesh, then in the previous heuristic, elements at the boundaries of splitting are separated in the processor graph. To avoid this, we use a more general strategy where we always fold the 2D graph like a paper and place the folded parts perpendicular to the smallest dimension. To achieve this, once we obtain the mapping for one plane, instead of a simple translation, we flip the mapping by 180 degrees for every alternate plane. Figure 10 shows the stacking and folding heuristics to map a 2D graph to a 3D torus or mesh.

7 Application Results

Using the algorithms developed in the previous sections, we attempted topology aware mapping of a few scientific applications. The three applications we used are MILC, POP and WRF. All performance runs were done on the IBM Blue Gene/P machines at Argonne National Laboratory and IBM T. J. Watson Research Center.

7.1 Applications

A brief description of the applications and input sets used in the paper follows. MILC represents a set of codes developed by the MIMD Lattice Computation (MILC) collaboration used to study quantum chromodynamics (QCD), the theory of strong interactions of subatomic physics [2]. It performs simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines. We used `ks_imp_rhmc` which is the Dynamical RHMC code for staggered fermions. We used an input grid of dimensions $32 \times 32 \times 32 \times 32$ on 256 processors.

Parallel Ocean Program (POP) [6] is an ocean circulation model developed by Los Alamos National Laboratory to solve three dimensional fluid motion equations on a sphere under hydrostatic and Boussinesq approximations. The input we used is similar to what is usually used in production runs of CCSM (Community Climate System Model). The input has a horizontal resolution of one degree (a 320×384 grid). This code spends a significant amount of time in communication. At 512 processors, roughly 80% of the time is spent in communication and `MPI_Allreduce` is responsible for the majority of this time.

WRF is a Weather Research and Forecasting Model [12]. 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. The benchmark simulates the weather for 3 hours using the data from a restart file.

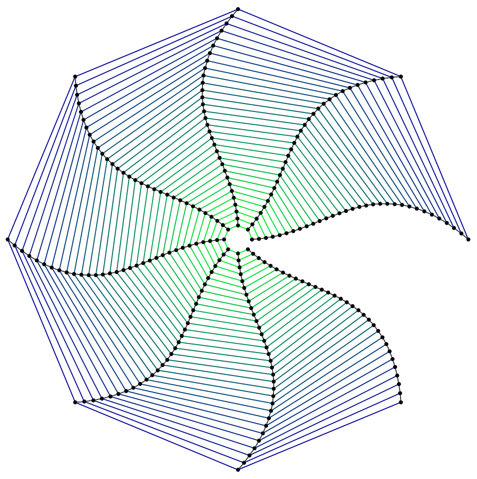
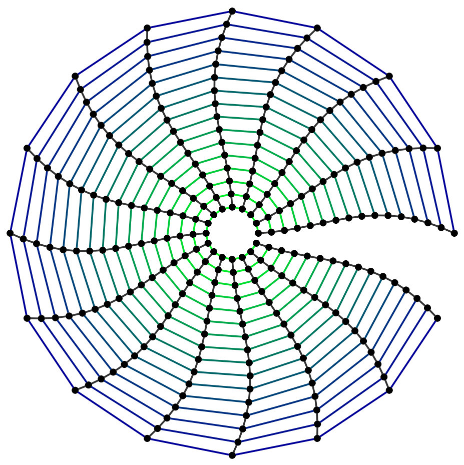



Figure 11: Structured Communication Graphs of POP and WRF for 256 processors

		VN mode			**SMP mode**		
Application	**Graph**	`XYZT`	`TXYZ`	`3DMAP`	`XYZ`	`FOLD`	`3DMAP`
MILC	$4 \times 4 \times 4 \times 4$	2304	2304	2304	5376	4096	5376
POP	8×32	2208	1632	672	1120	1056	1056
POP	32×16	4032	2176	1088	3456	2560	2200
WRF	16×16	1728	1152	832	1856	1344	1096
WRF	32×32	9216	4096	2688	11648	4608	4376

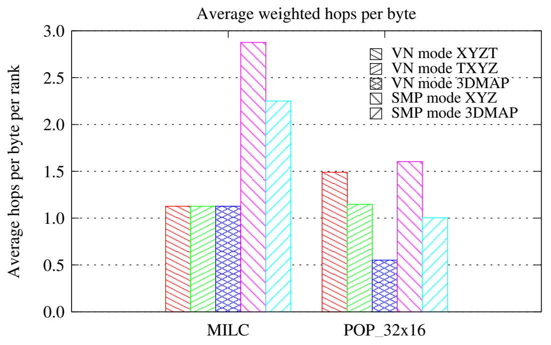
Table 2: Total theoretical hops for the three applications on 256, 512 and $1,024$ processors

7.2 Identifying Communication Patterns

The pattern matching algorithm was successful in identifying communication patterns for all three applications. MILC has a 4D communication pattern and the graph is $4 \times 4 \times 4 \times 4$ on 256 processors. POP has a 2D communication graph of 8×32 on 256 processors and 32×16 on 512 processors. WRF also has a 2D graph, of dimensions 16×16 on 256 processors and 32×32 on $1,024$ processors. Figure 11 shows visualizations of the 2D communication graphs as output by the pattern matching library for POP and WRF. The radial and tangential directions in the graph show the two dimensions of the object graph.

7.3 Quality of Mapping Algorithms

The hop-bytes metric is used to evaluate the quality of mapping algorithms developed in this paper. We compare the mappings with the default `XYZT` and `TXYZ` mapping on IBM's BG/P and with the results from the algorithms in [18]. This comparison is done both by calculating the hops theoretically and by using IBM's HPCT profiling tools [9] to get actual values by running the applications. Table 2 shows the theoretical values of the total hops obtained for the various algorithms assuming near-neighbor communication: `XYZT` is the default mapping on BG/P, `TXYZ` is the other layout possible on BG/P, `3DMAP` refers to the techniques developed in this paper and `FOLD` refers to the



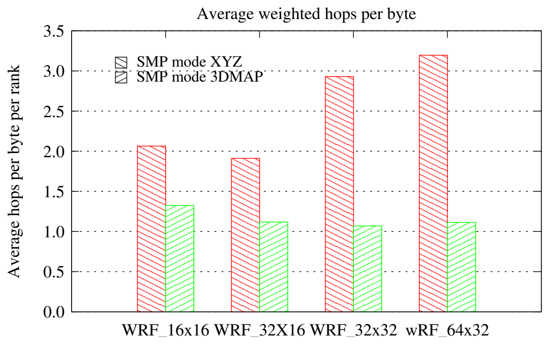


Figure 12: HPCT data for actual hops per byte for MILC, POP and WRF

algorithms presented in [18]. FOLD does not provide solutions for multiple cores per node and hence we could only compare with it, in SMP mode.

Runs on BG/P were done both in the VN (using 4 cores per node) and SMP (using 1 core per node) mode. Figure 12 shows the actual weighted hops (hops per byte averaged over all MPI ranks) obtained from profiling data for MILC, POP and WRF. We can see that they correlate exactly with the calculated theoretical hops. It is clear that topology aware mapping of MPI ranks to physical processors is successful in decreasing the average hops per rank for all the applications. In case of POP, we see an reduction of 63% in average hops from XYZT to 3DMAP in VN mode. WRF was only run in the SMP mode because it uses OpenMP to create threads on each node. The average hops for WRF reduce by 64% on $1,024$ and $2,048$ nodes which is quite significant and should lead to a dramatic drop in the load on the network.

Now, it remains to be seen if the reduction in hops leads to an improvement in performance. Let us study this on a per application basis. There is a reduction of 20% in the average hops per rank for MILC in SMP mode and we see a corresponding reduction of 15% in the communication time. This can be attributed to the reduction in time for MPI_Wait. Communication forms only 5% of the execution time of MILC at this scale and hence, we do not see an improvement in overall performance. Although there is more than 60% reduction in the hops for POP in VN mode from XYZT to 3DMAP, we do not see an improvement in POP's performance. Communication accounts for 80% of the total execution time in POP. However, most of it is spent in MPI_Allreduce which cannot be optimized by a simple rearrangement of ranks on the torus. Optimizing this requires optimization of the implementation of such global operations in MPI.

For the third application, WRF, using topology aware mapping, we were able to bring the average hops per rank close to 1 (Figure 12). This suggests that most MPI ranks are sending messages only 1 hop away and we should see performance improvements. When running on 256 nodes, we were able to reduce the average hops by 35% and the time spent in MPI_Wait by 4%. This leads to a 2% reduction in overall communication time. This does not lead to any overall performance improvement. The results on 512 nodes are similar in spite of the reduction in hops. However, when we ran WRF on $1,024$ nodes, average hops per rank diminished by 64% and communication time reduced by 45% (attributed to reduction in time for MPI_Bcast by 95%). At $1,024$ nodes, communication is roughly 45% of the total time and hence we see an overall performance improvement of 17%. On $2,048$ nodes, there is similar improvement in hops but the communication time increases by 12%. We still obtain a performance improvement of 8%, probably due to better utilization overall. Such performance improvement scan be quite significant for the overall completion time of long running simulations. We expect that the gains when running WRF on large installations will be even more.

It is important to reiterate the point made in Section 1 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 various 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. However, learning from the WRF results, we expect that as we run on larger partitions, communication time will be a significant fraction of the overall execution time and hence the benefits from topology aware mapping will increase. We hope to have more scaling performance results to large processor counts for the final version of the paper.

8 Conclusion

This paper presents a framework for automatic mapping of parallel applications with structured 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 4D patterns, 2. intelligent and fast heuristic solutions for mapping such graphs. We presented several heuristic techniques which are comparable to and often better than other techniques in literature. They also work on newer machines with multiple cores per node.

This paper makes several important contributions. We claim, aided by micro-benchmark results that *hop-bytes* is a good evaluation metric for mapping algorithms. *Maximum dilation* which was considered as the more important metric in VLSI design and early parallel computing work is not as important on parallel machines though it can still impact performance in certain cases. Several heuristic techniques for mapping 2D object graphs to 2D and 3D processor graphs have been developed. These heuristics along with pattern identification techniques form a framework for automatic mapping of a wide class of MPI applications with structured communication graphs. This framework will save much effort on the part of application developers to generate mappings for their individual applications.

Currently, we do not consider the actual amount of bytes communicated between ranks in when designing our mapping heuristics which will be considered in the future. Of course, the quest for better heuristics for near-optimal mapping is never-ending. The class of applications which has not been targeted in this paper is that with unstructured communication. Examples of such applications are molecular dynamics codes and unstructured mesh applications. We plan to build on the framework developed in this paper by including techniques for mapping applications with irregular communication.

Acknowledgments

This work was supported in part by a DOE Grant B341494 funded by Center for Simulation of Advanced Rockets. 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] Aleliunas, R. and Rosenberg, A. L. On Embedding Rectangular Grids in Square Grids. *IEEE Trans. Comput.*, 31(9):907–913, 1982.

[2] 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*, (61), 2000.

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

[4] A. Bhatelé and L. V. Kalé. Benefits of Topology Aware Mapping for Mesh Interconnects. *Parallel Processing Letters (Special issue on Large-Scale Parallel Processing)*, 18(4):549–566, 2008.

[5] 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.

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

[7] Ellis, J.A. Embedding rectangular grids into square grids. *Computers, IEEE Transactions on*, 40(1):46–52, Jan 1991.

[8] 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, 2006. ACM Press.

[9] 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*, pages 273–274, Washington, DC, USA, 2007. IEEE Computer Society.

[10] 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*, page 41. IEEE Computer Society, 2004.

[11] Melhem, Rami G. and Hwang, Ghil-Young. Embedding Rectangular Grids into Square Grids with Dilation Two. *IEEE Trans. Comput.*, 39(12):1446–1455, 1990.

[12] Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. The Weather Reseach 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.

[13] 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.

[14] P. Sadayappan and F. Ercal. Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes. *IEEE Trans. Computers*, 36(12):1408–1424, 1987.

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

[16] V. Salapura, K. Ganesan, A. Gara, M. Gschwind, J. Sexton, and R. Walkup. Next-Generation Performance Counters: Towards Monitoring Over Thousand Concurrent Events. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 139 – 146, April 2008.

[17] Shahid H. Bokhari. On the Mapping Problem. *IEEE Trans. Computers*, 30(3):207–214, 1981.

[18] H. Yu, I.-H. Chung, and J. Moreira. Topology mapping for Blue Gene/L supercomputer. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 116, New York, NY, USA, 2006. ACM.