

# Scalable Topology Aware Object Mapping for Large Supercomputers

Abhinav Bhatel †

Department of Computer Science,  
University of Illinois at Urbana-Champaign, Urbana, IL 61801.  
E-mail: [bhatele@illinois.edu](mailto:bhatele@illinois.edu)

**Advisor:** Laxmikant V. Kal 

## Abstract

The largest and fastest supercomputers in the top500 list deploy a scalable, three-dimensional torus or mesh interconnect. For these machines, the number of links (hops) traversed by a message has a direct effect on the time required to reach the destination. This is especially true in presence of bandwidth congestion, when multiple messages share links from source to destination. For large parallel machines, with a significant diameter, this can become a serious performance bottleneck. Traditionally, application developers have neglected this fact because of the advantages of virtual cut-through and wormhole routing for most message sizes on small machines. This might not be true any longer due to the large diameters of machines.

This dissertation will demonstrate the effect of network contention on message latencies and propose and evaluate techniques to minimize communication traffic and hence, bandwidth congestion on the network. This would be achieved by topology-aware mapping of tasks in an application. By placing communication tasks on processors which are in physical proximity on the network, communication can be restricted to near neighbors. This reduces link sharing among messages and leads to a better utilization of the available bandwidth. Our aim is to minimize *hop-bytes*, which is a weighted sum of the number of hops between the source and destination for all messages, the weights being the message sizes. This can minimize the communication time and hence, lead to significant speed-ups for parallel applications and also remove scaling bottlenecks in certain cases. The dissertation will involve developing a general automatic topology-aware mapping framework which takes the task graph and processor graph as input, and outputs near-optimal mapping solutions.

## 1 Introduction

The network topology of the largest and most scalable supercomputers today is a three-dimensional (3D) torus. Some examples are Cray’s XT family (XT3 and XT4) and IBM’s Blue Gene family (Blue Gene/L and Blue Gene/P). For large installations of such machines, the diameter of the network can be large (somewhere between 20 to 60 hops for Blue Gene/P and XT4.) This can have a significant effect on message latencies. When multiple messages start sharing network resources, this effect becomes more pronounced due to network congestion. Thus, it becomes necessary to consider the topology of the machine while mapping tasks to processors.

This dissertation will demonstrate that it is not wise to assume that message latencies are independent of the distance a message travels. This assumption has been supported all these years by the advantages of virtual cut-through and wormhole routing suggesting that the message latency

---

†<http://charm.cs.uiuc.edu/~bhatele/phd>

is independent of the distance in absence of blocking [1, 2, 3, 4, 5, 6, 7, 8]. When virtual cut-through or wormhole routing is deployed, message latency is modeled by the equation,

$$\frac{L_f}{B} * D + \frac{L}{B} \quad (1)$$

where  $L_f$  is the length of each flit,  $B$  is the link bandwidth,  $D$  is the number of links (hops) traversed and  $L$  is the length of the message. In absence of blocking, for sufficiently large messages (where  $L_f \ll L$ ), the first term is very small compared to the second. But with large diameters of very large supercomputers, this is no longer true for small to medium-sized messages. Let us say that the length of the flit is 32 bytes and the total length of the message is 1024 bytes. Now, if the message has to traverse 8 links, the first term is not negligible compared to the second one (it is one-fourth of the second term). Message sizes around 1 KB are found in several applications which deal with strong scaling to tens of thousands of processors [9, 10]. Hence, for such fine-grained applications, we should not neglect the dependence of message latencies on hops.

Equation 1 models message latencies in the absence of contention. In the case where multiple messages share the same network links, the situation becomes more complex. The bandwidth available per link per message is reduced since it is now being shared. The phenomenon of network resource sharing leading to contention can be explained with a simple example. Let us consider a 3D torus network of size  $8 \times 8 \times 8$ . The total number of uni-directional links on the system is  $512 \times 6 = 3072$ . The diameter of this network is  $4 + 4 + 4 = 12$  and hence, if messages travel from one random node to another, they will traverse 6 hops on average. Now, if we have four processors per node and every processor sends a message at the same time, all these messages require  $512 \times 4 \times 6 = 122884$  links in total and hence, every link will be used for four messages on average. This leads to contention for each link and therefore increases message latencies. Describing this scenario in terms of bandwidth requirements, in order to operate at minimum latency, we need four times the total raw bandwidth available. However, this is not the case and thus the delivered bandwidth is one-fourth of the peak.

Another assumption made by application developers today is that supercomputers like Cray XT3/XT4 do not have bandwidth congestion because of their fast interconnect and thus would not benefit from topology mapping. It is often assumed that contention is inconsequential on these faster interconnects and hence, application developers should not have to worry about network latencies and hence about topology-aware optimizations. This is evident from the fact that job scheduling on Cray XT machines is not topology-aware (unlike Blue Gene/L or Blue Gene/P, where users are allocated complete tori for their jobs). Also, there is no easy mechanism to obtain topology information on XT machines and for the same reason the `MPI_Cart` functions are not implemented efficiently.

The first part of this dissertation will use simple MPI benchmarks for an extensive study of message latencies and their dependence on distance (hops) on several machines – Cray’s XT family, IBM’s Blue Gene family and clusters like Ranger. As we shall see in Section 3.1, in absence of contention, message latencies depend on hops for small and medium-sized messages. In presence of contention, this dependence becomes more striking, especially for large-sized messages. Hence, it is important to consider topology of the machine, especially of 3D torus/mesh interconnects, to obtain the best performance. As we shall see, this holds true for both IBM and Cray machines. This study will enhance our understanding of the reasons for contention for network resources and will benefit the development of topology-aware mapping algorithms.

We will also demonstrate, with simple benchmarks and production codes that, topology mapping can significantly improve performance and scaling of communication-bound applications (Section 3.3.) The metric we will employ to assess the success of topology-aware schemes will be

*hop-bytes*. Hop-bytes are the weighted sum of the number of hops between the source and destination for all messages, with the weights being the message size. This metric gives an indication of the total communication traffic on the network due to an application. Reducing the total hop-bytes reduces link sharing and contention, thereby keeping message latencies close to the ideal.

Building on our successes at mapping individual applications, we aim to develop an automatic, topology-aware mapping framework. Given the communication information of an application and topology of a machine, this framework will produce intelligent mappings to minimize communication traffic. The mapping problem can be reduced to the graph-embedding problem, which is NP-complete. Hence, we will develop a set of heuristics, each dealing with a different scenario, which together will be able to deal with most parallel applications and their communication requirements. In an effort towards making mapping techniques scalable to very large machines, we will also discuss strategies for completed distributed and hybrid mapping strategies.

## 2 Related Work

The problem of topology-aware mapping has been studied extensively and proved to be NP-complete [11, 12, 13]. Pioneering work in this area was done by Bokhari in 1981, where he used pairwise exchanges between nodes to arrive at good solutions [11]. Subsequent research in the 80s can be divided into two categories – physical optimization techniques and heuristic approaches. Physical optimization techniques involve simulated annealing [14, 15], graph contraction [16, 17] and genetic algorithms [18]. Among the techniques mentioned above, [11, 12, 14] use pairwise exchanges which can take  $O(N^3)$  time where  $N$  is the number of nodes in the graph. Physical optimization techniques take a long time to arrive at the solution and hence, cannot be used for a time-efficient mapping during runtime. They are almost never used in practice. Heuristic techniques such as pairwise exchanges [12] and recursive mincut bipartitioning [19] are theoretical studies with no results on real machines. Also, most of these techniques (heuristics techniques especially) were developed specifically for hypercubes, shuffle-exchange networks or array processors.

Recent emergence of very large parallel machines has led to the necessity of topology mapping again. Most work from the 80s cannot be used in the present context because of unscalable techniques and different topologies from the ones being used today. As mentioned earlier, increasing effect of the number of hops on message latencies has fuelled such studies again. The Cray T3D and T3E supercomputers were the first to raise such issues in the late 90s and the problem of congestion and benefit from PE mapping were re-evaluated [20, 21, 22]. Newer line of supercomputers from Cray with faster interconnects, such as the XT3 and XT4, relieved the application writers of such problems. Opposing popular belief that these machines do not stand to benefit from topology mapping, this dissertation proves otherwise and provides detailed methods and results for performance improvements from topology mapping on Cray machines. Although researchers at Pittsburgh Supercomputing Center (PSC) have demonstrated the benefit of topology-aware job scheduling schemes on their Cray XT3 [23], we believe there has been no published research reporting network contention or quantifying it for the Cray XT family. This dissertation is pioneering work on topology-aware research on the Cray machines. One contribution of this dissertation is an API for providing topology information on Cray machines and demonstrating that topology-aware mapping can lead to improvements on Cray machines also.

Contrary to Cray, IBM systems like Blue Gene/L and Blue Gene/P acknowledge the dependence of message latencies on distance and encourage application developers to use topology of these machines to their advantage [24, 25, 26]. On Blue Gene/L, there is a 89 nanoseconds per hop latency attributed to the torus logic and wire delays. This fact has been used both by system

developers [27, 28] and application developers to improve performance on Blue Gene/L [29, 30, 31, 32]. Bhanot et. al [30] have developed a framework for optimizing task layout on BG/L. Since it uses simulated annealing, it is quite slow and the solution is developed offline. Embedding of tasks onto nodes using simple graph embedding schemes (for rings, meshes etc) has been discussed in [31, 29]. These can be used in MPI Topology functions. This work is not as useful when multiple objects have to be mapped to each physical processor (as in CHARM++.) Several application writers have also realized the importance of topology mapping on this machine and have used it to their benefit for speeding up their codes [28, 33, 34]. This dissertation shows similar successes for some CHARM++ applications [10, 35].

This dissertation is among the first to discuss the effects of contention on Cray and IBM machines and to compare across multiple architectures. We believe that the set of benchmarks we have developed for quantifying message latencies would be useful for the HPC community to assess latencies on a supercomputer and to determine the message sizes for which number of hops makes a significant difference. The effective bandwidth benchmark in the HPC Challenge benchmark suite measures the total bandwidth available on a system but does not analyze the effects of distance or contention on message latencies [36].

Our experience in developing mapping algorithms for production codes and insights discussed in this dissertation will be useful to individual application writers trying to scale their codes to large supercomputers. We believe that the proposed automatic mapping framework will be applicable to a wide variety of communication scenarios and will relieve the application writers from the burden of finding correct mapping solutions for their codes. Unlike most of the previous work, this dissertation handles both cardinality and topological variations in the graphs. It also handles various architectures and different kinds of communication scenarios (through the use of heuristics). Therefore, it will be useful to a large body of applications running on large parallel machines.

### 3 Progress So Far

Progress on the dissertation has been within two fronts: 1. study of different interconnects to understand the effects of distance on message latencies (Section 3.1), and 2. topology-aware mapping of specific benchmarks and applications. The second part involves two things - 1. development of a uniform API for obtaining topology information on IBM and Cray machines at runtime (Section 3.2), and 2. application specific mapping using this API (Section 3.3.) The next three sub-sections discuss these.

#### 3.1 Study of Message Latencies

Two different benchmarks were written to quantify message latencies in the absence and presence of contention. Both have been written in MPI and they use `MPI_Send` and `MPI_Recv` calls to send and receive messages.

**Without Contention (WOCON):** In this benchmark, one particular node is chosen from the allocated partition to control the program flow. We will call this node the master node (or master rank.) It sends  $N$  messages of  $M$  bytes to every other node in the partition, one at a time and expects a same-sized message in return. For machines with multiple cores per node, this benchmark places just one MPI task per node to avoid intra-node messaging effects. The sending of messages to each node is done sequentially, one message at a time. The size of the message,  $M$  is varied and for each value of  $M$ , the average time for sending a message to every other node is recorded. In effect, this benchmark records message latencies for varying hops in absence of contention. Since

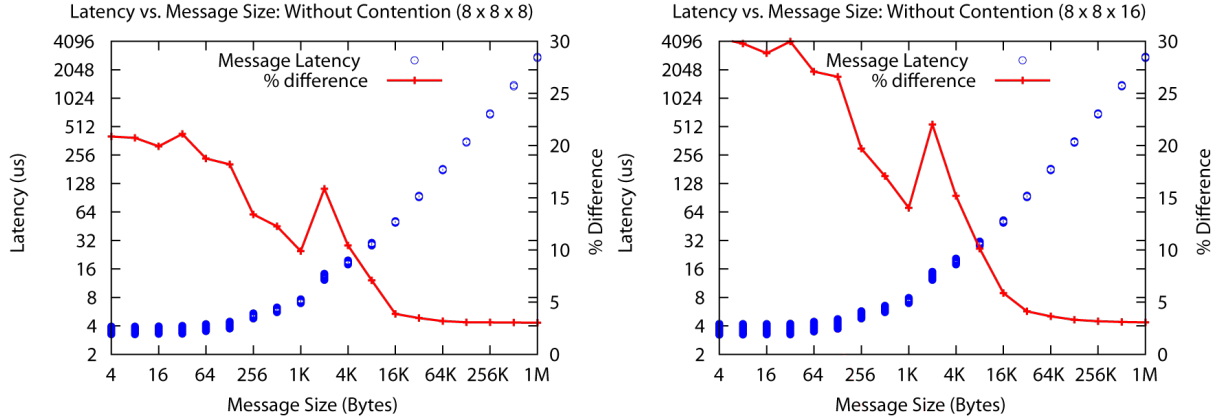


Figure 1: Plots showing the effect of hops on message latencies in the absence of contention (for torus sizes  $8 \times 8 \times 8$  and  $8 \times 8 \times 16$  on Blue Gene/P (top) and XT3 (bottom), Benchmark: WOCON)

the hop distance from the master node to other nodes varies, we should see different message latencies depending on the distance. As suggested by wormhole routing, the effect should be more pronounced for small messages. This benchmark will be used to find the range in which message latencies are affected by distance.

**With Contention (WICON):** The second benchmark is used to quantify message latencies in presence of contention, which is a regime not handled by the basic model of wormhole routing discussed earlier. It should be noted that unlike WOCON, this benchmark places one MPI task on each core to create as much contention as possible. All MPI tasks are grouped into pairs and the smaller rank in the pair sends  $k$  messages of size  $M$  in a loop to its partner and awaits a reply. The average time for the send and reply is recorded. To quantify the effect of hops on latencies, this benchmark is run in two modes:

- Near Neighbor Mode (NN): The ranks which form a pair only differ by one. This ensures that everyone is sending messages only 1 hop away (in a torus).
- Random Processor Mode (RND): The pairs are chosen randomly and thus they are separated by a random number of links.

The experiment is repeated for different values of the message size  $M$  and different values of  $k$ . We expect that with increasing message sizes and distance (diameter of the network), latencies for the RND mode should also increase.

**Findings:** Wormhole routing suggests that message latencies are independent of distance in the absence of contention, for most message sizes. The benchmark WOCON was used to verify if the number of hops has an effect on messages of any size. Figures 1 and 2 presents the results obtained from running WOCON on BG/P and XT4. There are two patterns on each plot: 1. For each message size on the x-axis, the circles represent the time for a message send from the master rank to different nodes on the allocated partition. Note that the vertical bars are actually a cluster of circles, one each for a message send to a different node; 2. Each point on the line represents the percentage difference between the minimum and maximum time for a message send with respect to the minimum time for a particular message size.

The important observation from these plots is that the difference is significant for message sizes up to 16 KB ( $\sim 6\%$  for 16 KB in the 1024 nodes plot). For the same plot, the difference is as much

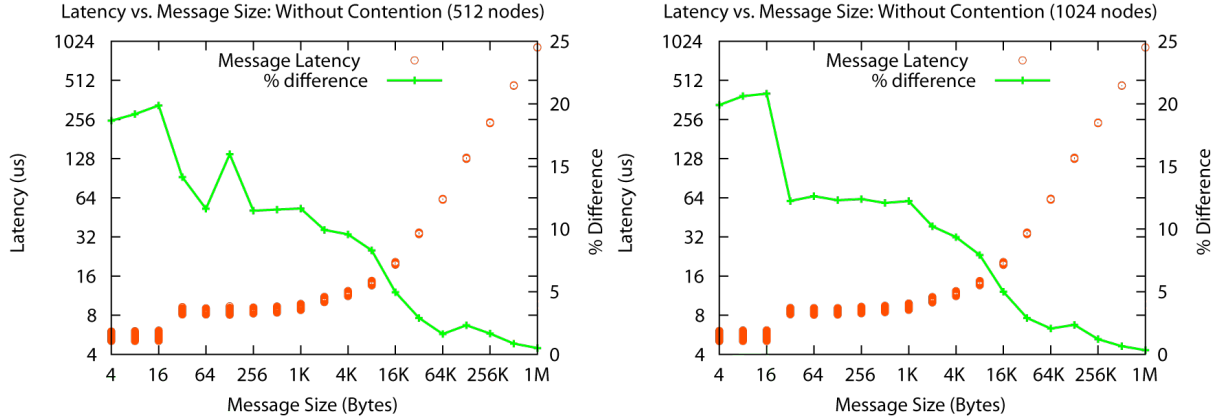


Figure 2: Plots showing the effect of hops on message latencies in absence of contention (for torus sizes  $8 \times 8 \times 8$  and  $8 \times 8 \times 16$  on Blue Gene/P (top) and XT3 (bottom), Benchmark: WICON)

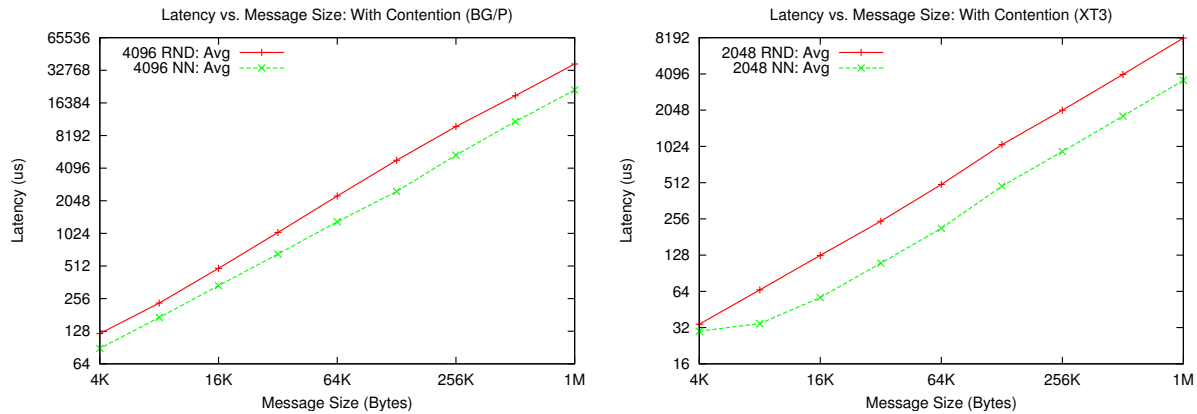


Figure 3: Plots showing the effect of hops on message latencies in presence of contention (on Blue Gene/P and XT3, Benchmark: WICON)

as 30% for 4 byte messages and remains significant at 10% for 8 KB messages. Most fine-grained applications use messages which fall in this range and hence it is not wise to blindly assume that latencies do not depend on hops for most practical message sizes. Strong scaling of problems to very large number of processors also elevates us to this range of message sizes. Another observation is that the difference increases for a particular message size with the increase in diameter of the partition.

Figure 3 shows the results for the WICON benchmark on Blue Gene/P and XT3. The first plot shows the results for WICON on 4,096 cores of BG/P. It is clear that the random-processor (RND) latencies are more than the near-neighbor (NN) latencies. This is especially true for larger message sizes, which is what we would expect based on the assertion that hops have a significant impact on the latencies in the presence of contention, which increases with larger messages because of a proportional increase in packets on the network.

The second plot in Figure 3 presents the results for the WICON benchmark on XT3. It is noteworthy that no previous studies report contention and the effect of hops on latencies on the XT family of machines. We see a significant difference between the NN and RND lines for all torus sizes. Just as an example, at 16 KB messages, the difference is nearly two times as much. This is not unexpected and a quantum chemistry code has shown huge benefits (up to 60%) from

topology-aware mapping on XT3 [37]. Only a subset of results from the machines Blue Gene/P, Blue Gene/L, XT3 and XT4 have been reported here. Detailed results and analysis can be found in a technical report [38].

### 3.2 Topology Information

Topology-aware mapping requires information about two things – the processor graph and the object graph (or the communication graph). The idea is to obtain this information automatically at runtime. This helps towards the goal of automatic mapping by the runtime which is hidden to the application writer. In addition, this information can also be used by the application for communication optimizations.

Information about the topology of the machine is needed to map objects or VPs to processors (such as the dimensions of the 3D mesh/torus). The application should be able to query the runtime to get information like the dimensions of the allocated processor partition, mapping of ranks to physical nodes, et cetera. However, the mapping interface should be simple and should hide machine-specific details from the application. We have implemented a “Topology Manager” interface which gives useful information for torus interconnects like Blue Gene/L, XT3 and Blue Gene/P and for n-way SMP machines like NCSA’s Abe and TACC’s Ranger.

We now describe an API which we call the *Topology Manager* which can be used by any application for mapping of objects to processors. In this paper, the API is used in the context of CHARM++ applications; however, it is generic and can be used in any parallel program. Mapping of object graphs onto the processor graph requires information about the machine topology at runtime. The application should be able to query the runtime to get information like the dimensions of the allocated processor partition, mapping of ranks to physical nodes etc. However, the mapping interface should be simple and it should hide machine-specific details from the application.

The API provides different functions, which can be grouped into the following categories:

1. **Size and properties of the allocated partition:** At runtime, the application needs to know the dimensions of the allocated partition (`getDimNX`, `getDimNY`, `getDimNZ`), number of cores per node (`getDimNT`) and whether we have a torus or mesh in each dimension (`isTorusX`, `isTorusY`, `isTorusZ`).
2. **Properties of an individual node:** The interface also provides simple calls to convert from ranks to physical co-ordinates and vice-versa (`rankToCoordinates`, `coordinatesToRank`).
3. **Additional Functionality:** Mapping algorithms often need to calculate number of hops between two ranks or pick the closest rank to a given rank from a list. Hence, the API provides functions like `getHopsBetweenRanks`, `pickClosestRank` and `sortRanksByHops` to facilitate mapping algorithms.

We now discuss the process of extracting this information from the system at runtime and why is it useful to use the Topology Manager API on different machines:

**IBM Blue Gene machines:** On Blue Gene/L and Blue Gene/P [39], topology information is available through system calls to the “BGLPersonality” and “BGPPersonality” data structures, respectively. It is useful to use the Topology Manager API instead of the system calls for two reasons. First, the system calls can be expensive (especially on Blue Gene/L) and so it is advisable to limit their use. The API does a few system calls to obtain enough information so that it can independently construct the topology information. Once an instance of the *TopoManager* class has

been created, it does not make any further system calls. The second reason is that on Blue Gene/L and Blue Gene/P, there is a limit to the smallest size of a partition which can be allocated (32 nodes on the Watson BG/L and 64 nodes on the ANL BG/P). If fewer nodes than this smallest unit are requested, the smallest partition will be allocated though only a subset of the nodes in it are used by the application. In these cases, the lower level system calls give information about the entire booted partition and not the actual nodes being used. Our API calculates which portion of the allocated partition is being used when you use fewer nodes than the allocated partition and gives the correct information.

**Cray XT machines:** Cray machines have been designed with a significant overall bandwidth, and possibly for this reason, documentation for topology information was not readily available at the installations we used. (We thank Shawn Brown from PSC, Larry Kaplan from Cray Inc. and William Renaud at ORNL for helping us obtain topology information through personal communication). There is no documentation or publication which provide a user with system calls to obtain topology information on these machines. We hope that the information provided here will be useful to other application programmers.

Obtaining topology information on XT machines is a two step process: 1. Getting the node ID (`nid`) corresponding to a given MPI rank (`pid`) which tells us which physical node a given MPI rank is on. This can be done through different system calls on XT3 and XT4: `cnos_get_nidpid_map` available through “`catamount/cnos_mpi_os.h`” and `PMI_Portals_get_nidpid_map` available from “`pmi.h`”. These calls provide a map for all ranks in the current job and their corresponding node IDs. 2. The second step is obtaining the physical coordinates for a given node ID. This can be done by using the system call `rca_get_meshcoord` from “`rca.lib.h`”. Once we have information about the physical coordinates for all ranks in the job, the API derives information such as the extent of the allocated partition by itself (this assumes that the machine has been reserved and we have a contiguous partition). Using the size of the partition and the size of the total machine ( $11 \times 12 \times 16$  for BigBen and  $21 \times 16 \times 24$  for Jaguar), the API can tell if we have a mesh or torus in each direction. Again, once the TopoManager object is instantiated, it stores all this information and does not make system calls again.

The Topology Manager API provides a uniform interface to the application developer and hence the application just knows that it is a 3D torus or mesh topology. Application specific task mapping decisions require no architecture or machine specific knowledge (BG/L or XT3 for example) as there is no need for it to use the lower level system calls for topology information. Our API provides a very easy-to-use wrapper for this functionality.

Obtaining the object graph is not as simple. We have used two methods for this until now – (1) using the information available from the application writers directly and, (2) using the load balancing framework in CHARM++ to instrument and record this information. Obtaining this information automatically is a part of the proposed work.

### 3.3 Application-specific Mapping

The efforts for topology-aware mapping were started for specific applications to begin with. This requires understanding the communication patterns of the application and then trying to optimize the mapping of its objects on to the processor topology. We will discuss two applications here – OPENATOM and NAMD. Initial work on 2D and 3D Stencil applications can be found here [40, 41].

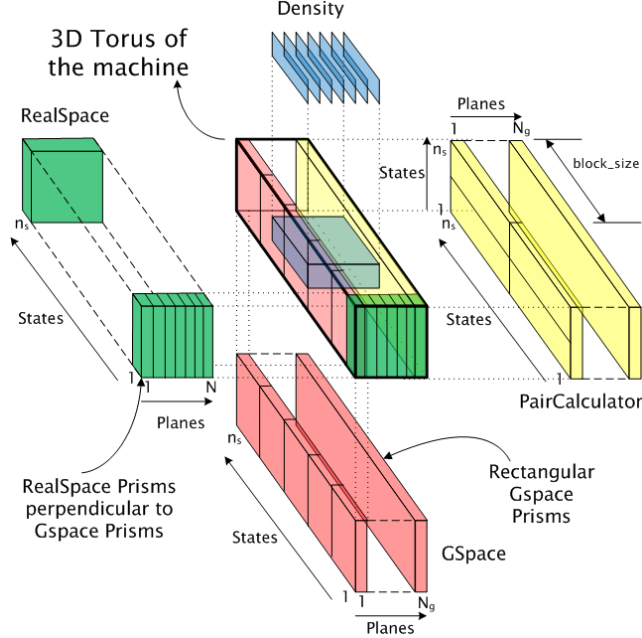


Figure 4: Mapping of different arrays to the 3D torus of the machine

### 3.3.1 OpenAtom

OPENATOM is a fine-grained parallelization of the CPAIMD method [42] to understand dynamics of atoms at a quantum scale [10]. Computation in OPENATOM is divided into a large number of objects, enabling scaling to tens of thousands of processors. Calculating the electrostatic energy involves computing several terms. Hence, CPAIMD computations involve a large number of phases with high inter-processor communication. These phases are discretized into a large number of objects, which generate a lot of communication, but ensures efficient interleaving of work. The entire computation is divided into ten phases, which are parallelized by decomposing the physical system into fifteen *chare arrays*. For a detailed description of this algorithm please refer to [10].

Since multiple chare arrays interact among one another, the communication dependencies are complex and mapping is a challenging task. Here, we will briefly discuss the object graphs between two pairs of chare arrays and propose mapping schemes which take both into account. OPENATOM provides us with a scenario where the load on each object is static (under the CPAIMD method) and the communication is regular and clearly understood. Hence, it should be possible to intelligently map the arrays in this application to minimize inter-processor communication and maintain load balance.

GSpace and RealSpace are 2D *chare arrays* with states in one dimension and planes in the other. These arrays interact with each other through transpose operations, where all planes of one state in GSpace,  $G(s, *)$  talk to all planes of the same state,  $R(s, *)$  in RealSpace (state-wise communication). The number of planes in GSpace is different from that in RealSpace. GSpace also interacts with the PairCalculator arrays. Each plane of GSpace,  $G(*, p)$  interacts with the corresponding plane,  $P(*, *, p)$  of the PairCalculators (plane-wise communication) through multicasts and reductions. So, GSpace interacts state-wise with RealSpace and plane-wise with PairCalculators. If all planes of GSpace are placed together, then the transpose operation is favored, however, if all states of GSpace are placed together, the multicasts/reductions are favored. To strike a balance between the two extremes, a hybrid map is built, where a subset of planes and states of these three arrays

Cores	WATER_32M_70Ry		WATER_256M_70Ry	
	Default	Topology	Default	Topology
<b>512</b>	0.248	0.205	-	-
<b>1024</b>	0.188	0.127	10.78	6.70
<b>2048</b>	0.129	0.095	6.85	3.77
<b>4096</b>	0.114	0.067	4.21	2.17
<b>8192</b>	-	-	3.52	1.77

Table 1: Execution time per step (in seconds) of OPENATOM on Blue Gene/P (VN mode)

Cores	WATER_32M_70Ry		WATER_256M_70Ry		GST_BIG	
	Default	Topology	Default	Topology	Default	Topology
<b>256</b>	0.226	0.196	-	-	-	-
<b>512</b>	0.179	0.161	7.50	6.58	6.28	5.06
<b>1024</b>	0.144	0.114	5.70	4.14	3.51	2.76
<b>2048</b>	0.135	0.095	3.94	2.43	2.90	2.31

Table 2: Performance of OPENATOM on XT3. The numbers represent time per step in seconds.

is placed on one processor.

**Mapping GSpace and RealSpace Arrays:** Initially, the GSpace array is placed on the torus and other objects are mapped relative to GSpace’s mapping. The 3D torus is divided into rectangular boxes (which will be referred to as “prisms”) such that the number of prisms is equal to the number of the planes in GSpace. The longest dimension of the prism is chosen to be same as one dimension of the torus. Each prism is used for all states of one plane of GSpace. Within each prism for a specific plane, the states in  $G(*, p)$  are laid out in increasing order along the long axis of the prism. Figure 4 shows the GSpace prisms (box at the bottom) being mapped along the long dimension of the torus (box in the center). Once GSpace is mapped, the RealSpace objects are placed. Prisms perpendicular to the GSpace prisms are created, which are formed with the inclusion of processors holding all planes for a particular state of GSpace,  $G(s, *)$ . These prisms (box on the left) are perpendicular to the GSpace prisms and the corresponding states of RealSpace,  $R(s, *)$  are mapped on to these prisms.

**Mapping PairCalculator Arrays:** Since PairCalculator and GSpace objects interact plane-wise, the aim is to place  $G(*, p)$  and  $P(*, *, p)$  nearby. *Chares* with indices  $P(s1, s2, p)$  are placed around the centroid of  $G(s1, p), \dots, G(s1 + block\_size, p)$  and  $G(s2, p), \dots, G(s2 + block\_size, p)$ . This minimizes the hop-count for the multicast and reduction operations. The result of this mapping co-locates each plane of PairCalculators (box on the right in Figure 4) with its corresponding plane of GSpace objects within the GSpace prisms.

The mapping schemes discussed above substantially reduce the hop-count for different phases. They also restrict different communication patterns to specific prisms within the torus, thereby reducing contention and ensuring balanced communication throughout the torus. State-wise and plane-wise communication is confined to different (orthogonal) prisms. This helps to avoid scaling bottlenecks.

As shown in Table 1, performance improvements from topology-aware mapping for Blue Gene/P (BG/P) can be quite significant. As the number of cores and likewise, the diameter of the torus

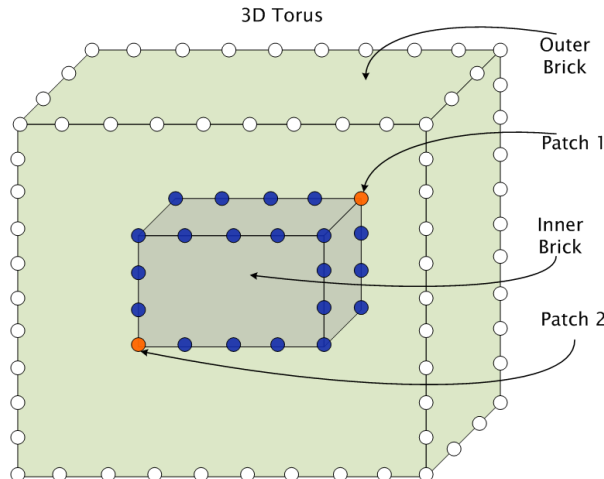


Figure 5: Topological placement of a compute on a 3D torus/mesh of processors

grows, the performance impact increases until there is 40% improvement for WATER\_32M\_70Ry at 4096 and 50% for WATER\_256M\_70Ry at 8192 cores. The improvements from topological awareness on Cray XT3, presented in Table 2, are comparable to those on BG/P. There is an improvement of 20% on XT3 for WATER\_256\_70Ry at 1024 cores, compared to the improvement of 38% on BG/P at 1024 cores.

### 3.3.2 NAMD

NAMD is a production Molecular Dynamics application used for simulation of bio-molecules [35, 9, 43]. NAMD is parallelized by use of CHARM++ objects called patches and computes. The simulation box is spatially divided into smaller cells called patches. The force calculation for every pair of patches is assigned to a different compute. Thus, communication in NAMD consists of section multicasts from patches to computes and back. Every patch multicasts its atom data to multiple computes, whereas each compute receives data from only two patches. Patches are statically assigned to a few processors during start-up and computes are distributed evenly by a load balancer. Let us now see the deployment of topology-aware techniques in the static placement of patches and the load balancers.

**Topology placement of patches:** Since patches form a geometric decomposition of the simulation space, they constitute a 3D group of objects which can be mapped nicely onto the 3D torus of machines. An ORB of the torus is used to obtain partitions equal in number to the patches and then, a one-to-one mapping of the patches to the processor partitions is done. This is described in detail in [35].

**Topology-aware Load Balancers:** Once patches have been statically assigned onto the processor torus, computes which interact with these patches should be placed around them. We will now discuss modifications to the load balancing algorithm that try to achieve this heuristically [44]. The three steps for choosing a suitable processor to place a compute on (for both the comprehensive as well as refinement load balancer) are modified as follows:

*Step I:* If the compute gets placed on a processor with both the patches, then no heuristic can

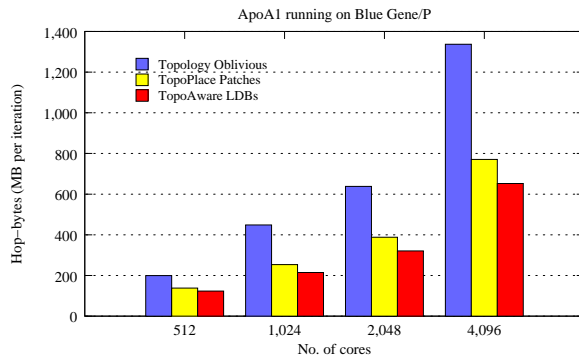


Figure 6: Hop-bytes for different schemes on IBM Blue Gene/P

No. of cores	512	1024	2048	4096	8192	16384
<b>Topology Oblivious</b>	13.93	7.96	5.40	5.31	-	-
<b>TopoPlace Patches</b>	13.85	7.87	4.57	3.07	2.33	1.74
<b>TopoAware LDBs</b>	13.57	7.79	4.47	2.88	2.03	1.55

Table 3: Performance of NAMD (ms/step) on IBM Blue Gene/P

do better than that, because both messages are local to the processor (and no new communication paths are added). However, if we are searching for a processor with proxies for both patches, we can give topological preference to certain processors. Consider Figure 5, which shows the entire 3D torus on which the job is running. When placing a compute, it should be placed topologically close to the two processors that house the patches it interacts with. The two patches define a smaller brick within the 3D torus (shown in dark grey in the figure). The sum of distances from any processor within this brick to the two patches is minimum. Hence, if we find two processors with proxies for both patches, we give preference to the processor that is within this inner brick defined by the patches.

**Step II:** In this case too, we give preference to a processor with one proxy or patch which is within the brick defined by the two patches that interact with the compute.

**Step III:** If Step I and II fail, we are supposed to look for any underloaded processor to place the compute on. Under the modified scheme of things, we first try to find an underloaded processor within the brick and if there is no suitable processor, we spiral around the brick to find the first underloaded one.

Figure 6 shows the *hop-bytes* for all messages per iteration when running NAMD on Blue Gene/P on different sized partitions. A standard benchmark used in the MD community was used for the runs: 92,227-atom ApoLipoprotein-A1 (ApoA1). As we would expect, hop-bytes consistently increase as we go from a smaller partition to a larger one. The three strategies compared are: topology oblivious mapping of patches and computes (Topology Oblivious), topology-aware static placement of patches (TopoPlace Patches) and topology-aware placement for both patches and load balancing for computes (TopoAware LDBs).

Topology aware schemes for the placement of patches and the load balancer help in reducing the hop-bytes for all processor counts. Also, the decrease in hop-bytes becomes more significant

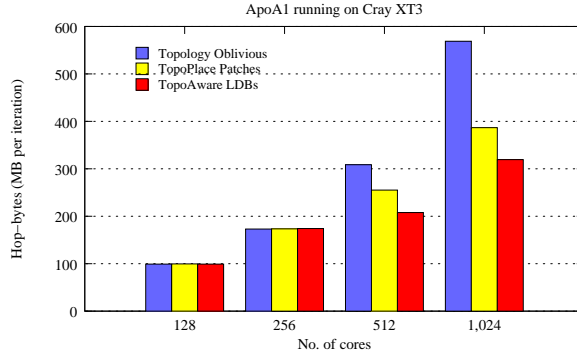


Figure 7: Hop-bytes for different schemes on Cray XT3

No. of cores	128	256	512	1024
<b>Topology Oblivious</b>	17.43	8.83	5.14	3.08
<b>TopoPlace Patches</b>	17.50	8.88	5.34	3.15
<b>TopoAware LDBs</b>	17.47	8.78	5.10	3.01

Table 4: Performance of NAMD (ms/step) on Cray XT3

as we go to larger-sized partitions. This is due to the fact that the average distance traveled by each message increases as we increase the partition size in the case of default mapping; however, it becomes controlled when we do a topology-aware mapping. Since the actual performance of the load balancers depends on several metrics, the question remains that as to whether the reduction in hop-bytes leads to an actual improvement in performance. As it turns out, we also see a reduction in the number of proxies and in the *max-to-average* ratio for topology-aware load balancers, which is reflected in the overall performance of NAMD on Blue Gene/P (Table 3). The topology oblivious scheme stops scaling around 4,096 cores and hence, we did not obtain numbers for it beyond that. We see an improvement of 10% at 16,384 cores with the use of topology-aware load balancers.

Similar tests were done on Cray XT3 to assess if a faster interconnect can hide all message latencies and make topology-mapping unnecessary. Figure 7 shows the hop-bytes for all messages per iteration when running NAMD on Cray XT3 on different sized partitions. We could only run on up to 1024 nodes (1 core per node) on XT3 and as a result, we do not see a huge benefit on the lower processor counts. However, if we compare the 512 processor runs on XT3 with 2048 processor (512 node) runs on Blue Gene/P, we see a similar reduction in hop-bytes. It is also reflected in a slight improvement in performance at this point (Table 4).

## 4 Proposed Work

The dissertation proposes research in two directions. One direction would be to enhance our understanding of the communication characteristics of different interconnects using MPI benchmarks. Understanding specific applications and their communication behavior using performance counters and application-specific mapping would also be an effort in this direction. The other direction would be to utilize this experience, in developing an automatic framework which can develop topology-aware mapping solutions. Finally, we would also look at scalable techniques for load balancing, which will involve distributed and hierarchical techniques.

## 4.1 Experiments and Analysis

We plan to extend the work on quantifying message latencies on different networks. It would be interesting to understand the per message (alpha) and per byte (beta) costs and to develop models which closely simulate message latencies in presence of contention. This would lead to a better understanding of how contention affects message latencies and techniques can then be developed to avoid such problems. We would also like to consider hybrid interconnects and how topology-aware mapping affects them. If the details of machines like Blue Gene/Q and Blue Waters become public in the time frame of this dissertation, we would be interested in working with new topologies too.

Understanding the communication characteristics of an application using performance analysis tools and performance counters is also important before we attempt to map them topologically. Such analysis is also useful in making comparisons between different topology-aware mapping schemes and the default schemes. Using performance counters to monitor the traffic on different links can also help improve the mapping for specific scenarios. This would also allow us to take simple routing strategies on specific machines into account.

## 4.2 Further Application-specific Mapping

Application-specific mapping will enhance our experience and also give us a baseline for comparison of automatic mapping solutions with the best manual mapping possible. We plan to study the following applications:

- **Matrix Multiplication:** A parallel matrix multiplication algorithm that uses a 3D decomposition for 2D matrices [45] is a good test case for mapping. Our implementation divides the input matrices A and B among a 3D array of chares. Before each chare can do its local computation, A should be replicated along the Z dimension of the array (chosen arbitrarily) and B along the X dimension. For this, every chare sends its portion of A and B to other chares, which share its Z and X coordinate respectively. Once the computation is done, C needs to be distributed among the chares and hence every chare communicates with others having the same Y coordinate. As a result, we have communication along different dimensions of the array and arranging the objects on a 3D mesh can be a challenge.
- **MILC:** MILC stands for MIMD Lattice computation and is used for large scale numerical solutions to study quantum chromodynamics [46]. For benchmarking, the commonly used application *ks\_imp\_dyn*, which is used for simulating full QCD with dynamical Kogut-Susskind fermions. This code has a 4D grid which is to be mapped on to the 3D torus/mesh. Communication in this code is with near-neighbors and so we expect that it would benefit greatly from an intelligent topology-aware mapping.
- **Unstructured Mesh Computation:** An unstructured mesh application is more challenging as a mapping problem because the number of communicators for each object is variable and they are not near-neighbors. Heuristics techniques for such applications would enable us to develop patterns about which heuristics work better in which scenarios.

This is not a fixed set of applications. We will possibly include more applications and understand which of them are communication intensive and latency intolerant. Such applications stand to benefit from topology-aware mapping techniques. This work might also involve the use of the BigSim [47] simulation framework to induce artificial latencies and understand the communication behavior of different applications.

### 4.3 Automatic Mapping Framework

Building on the experience gained from mapping of specific applications, we propose to develop an automatic topology-aware mapping framework. This framework will obtain topology information about the machine at runtime and gather information about the communication in the application. Using various heuristics, it will try to arrive at a near-optimal solution for the mapping of objects to processors.

As presented in Section 3.2, an API to obtain topology information at runtime has already been developed. The remaining efforts towards an automatic mapping framework have to be on two fronts:

**Object Graph Information:** The object graph is the other important input to the topology mapping algorithm. The object graph can be obtained at compile time (if the code is written in a static data-flow language) or at runtime (e.g. by instrumenting the code and saving the communication information). The load balancer framework in CHARM++ already provides this information at runtime. Similar ideas can be applied by using AMPI when running MPI programs. Various communication scenarios are possible and different strategies will be required depending on the specific case. Before we decide on the strategies, we would have to use some techniques to identify commonly used communication patterns.

**Finding an Optimal Mapping:** Depending on the object graph, different heuristic strategies will be deployed:

1. Regular static communication, as in 3D Stencil or 3D implementation of matrix multiplication (point-to-point messages or multicasts to specific members of the 3D array). In this case, we can use pattern matching to find regular communication patterns and use simple embedding techniques for regular graphs.
2. Irregular static communication, as in meshing applications. In case of arbitrary static graphs, we can do an intelligent initial mapping based on a combination of embedding and heuristics techniques.
3. Regular or Irregular dynamic object graph, as in MD applications. If the object graph is dynamic, then we can use different topology-aware heuristics at runtime (for example, in the load balancing framework.)

The above set of problems can get complicated when we have multiple object graphs which interact with one another. OPENATOM is one such example. It becomes a harder problem to track communication arcs between objects in different graphs and co-locate them together. Such cases require more information about the application such as which communication is the worst bottleneck and which communication can be neglected over another.

### 4.4 Scalable Load Balancing

Most work on topology-aware mapping was done when the number of processors were on the order of tens or hundreds. As a result, the algorithms developed then were serial and executed on a single processor. With tens of thousands of processors and millions of objects to be mapped, we will face both memory constraints and longer run times for the algorithms. Therefore, we want to make an effort towards moving from centralized mapping strategies to distributed and hierarchical approaches. Centralized strategies have a global view of the problem and can obtain the best

solutions but they are costly in terms of their running time. Distributed strategies have a limited view of the object graph but can be significantly faster. We plan to develop strategies which do not have a global view and hence run fast but still output solutions which are close to optimal.

## 5 Merit and Impact

Significant work was done in the 80s on topology-aware mapping but it was directed towards interconnect topologies like hypercubes and shuffle-exchange networks which are not used in practice today. Work done in those years was directed towards machines consisting of a few hundred processors. The proposed work is highly relevant for the machines of the petascale era, such as Blue Gene and XT. Apart from MPI applications, it is directed towards applications with virtualization (having multiple objects per physical processor), which has not been explored much earlier. As opposed to earlier research in this area, this work is directed towards high scalability and fast runtime solutions.

This dissertation will demonstrate that topology-aware mapping is important for communication bound applications. Results quantifying message latencies in presence of contention and application-specific mapping will demonstrate to be useful to application writers trying to optimize their codes. The TopoManager API is already being used by other application groups (such as the US Lattice QCD collaboration) since information on Cray machines is not readily available and this API provides a single wrapper for different machines. We hope that this API will be gradually used to implement `MPI_Cart_create` and other virtual topology functions on Cray machines.

The work on the automatic mapping framework will relieve the application writers of doing the mapping themselves. We also foresee acceptance of the idea that applications running on Cray XT machines will benefit as much as those on Blue Gene machines. This might even lead to modification of the batch schedulers on these machines to allocate contiguous 3D mesh partitions for jobs.

## 6 Plan

Work on several of the topics mentioned in the previous section will proceed simultaneously. However, here is a rough outline of expected completion of various tasks:

- Experiments and Analysis: Spring 2009
- Work on Applications: Spring 2009
- Automatic Mapping Framework: Spring 2010
- Scalable Load Balancing: Fall 2009

## References

- [1] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.
- [2] Dilip D. Kandlur and Kang G. Shin. Traffic routing for multicomputer networks with virtual cut-through capability. *IEEE Trans. Comput.*, 41(10):1257–1270, 1992.
- [3] James W. Dolter, P. Ramanathan, and Kang G. Shin. Performance analysis of virtual cut-through switching in harts: A hexagonal mesh multicomputer. *IEEE Trans. Comput.*, 40(6):669–680, 1991.

- [4] Ronald I. Greenberg and Hyeong-Cheol Oh. Universal wormhole routing. *IEEE Transactions on Parallel and Distributed Systems*, 08(3):254–262, 1997.
- [5] Prasant Mohapatra. Wormhole routing techniques for directly connected multicomputer systems. *ACM Comput. Surv.*, 30(3):374–410, 1998.
- [6] Loren Schwiebert and D. N. Jayasimha. On measuring the performance of adaptive wormhole routing. *hipc*, 00:336, 1997.
- [7] Mohamed Ould-Khaoua and Hamid Sarbazi-Azad. An analytical model of adaptive wormhole routing in hypercubes in the presence of hot spot traffic. *IEEE Transactions on Parallel and Distributed Systems*, 12(3):283–292, 2001.
- [8] H. H. Najaf-abadi and H. Sarbazi-Azad. An accurate combinatorial model for performance prediction of deterministic wormhole routing in torus multicomputer systems. pages 548–553, 2004.
- [9] Abhinav Bhatele, Sameer Kumar, Chao Mei, James C. Phillips, Gengbin Zheng, and Laxmikant V. Kale. Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*, 2008.
- [10] Eric Bohm, Glenn J. Martyna, Abhinav Bhatele, Sameer Kumar, Laxmikant V. Kale, John A. Gunnels, and Mark E. Tuckerman. Fine Grained Parallelization of the Car-Parrinello ab initio MD Method on Blue Gene/L. *IBM Journal of Research and Development: Applications of Massively Parallel Systems*, 52(1/2):159–174, 2008.
- [11] Shahid H. Bokhari. On the mapping problem. *IEEE Trans. Computers*, 30(3):207–214, 1981.
- [12] Soo-Young Lee and J. K. Aggarwal. A mapping strategy for parallel processing. *IEEE Trans. Computers*, 36(4):433–442, 1987.
- [13] P. Sadayappan and F. Ercal. Nearest-neighbor mapping of finite element graphs onto processor meshes. *IEEE Trans. Computers*, 36(12):1408–1424, 1987.
- [14] S. Wayne Bollinger and Scott F. Midkiff. Processor and link assignment in multicomputers using simulated annealing. In *ICPP (1)*, pages 1–7, 1988.
- [15] S. Wayne Bollinger and Scott F. Midkiff. Heuristic technique for processor and link assignment in multicomputers. *IEEE Trans. Comput.*, 40(3):325–333, 1991.
- [16] Francine Berman and Lawrence Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4(5):439–458, 1987.
- [17] N. Mansour, R. Ponnusamy, A. Choudhary, and G. C. Fox. Graph contraction for physical optimization methods: a quality-cost tradeoff for mapping data on parallel computers. In *ICS '93: Proceedings of the 7th international conference on Supercomputing*, pages 1–10. ACM, 1993.
- [18] S. Arunkumar and T. Chockalingam. Randomized heuristics for the mapping problem. *International Journal of High Speed Computing (IJHSC)*, 4(4):289–300, December 1992.
- [19] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. In *Proceedings of the 3rd conference on Hypercube concurrent computers and applications*, pages 210–221. ACM Press, 1988.
- [20] M. Muller and Michael Resch. Pe mapping and the congestion problem in the t3e. In *Proceedings of the Fourth European Cray-SGI MPP Workshop*, Garching, Germany, 1998.

- [21] Eduardo Huedo, Manuel Prieto, 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*, pages 199–207, London, UK, 2000. Springer-Verlag.
- [22] Thierry Cornu and Michel Pahud. Contention in the cray t3d communication network. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II*, pages 689–696, London, UK, 1996. Springer-Verlag.
- [23] 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.
- [24] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49(2/3), 2005.
- [25] G. Almasi, C. Archer, J. G. Castanos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, B. D. Steinmacher-Burow, W. Gropp, and B. Toonen. Design and implementation of message-passing services for the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3), 2005.
- [26] IBM System Blue Gene Solution. Blue gene/p application development redbook, 2008. <http://www.redbooks.ibm.com/abstracts/sg247287.html>.
- [27] George Almasi, Siddhartha Chatterjee, Alan Gara, John Gunnels, Manish Gupta, Amy Henning, Jose E. Moreira, and Bob Walkup. Unlocking the Performance of the Blue Gene/L Supercomputer. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 57. IEEE Computer Society, 2004.
- [28] Kei Davis, Adolfo Hoisie, Greg Johnson, Darren J. Kerbyson, Mike Lang, 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.
- [29] Brian E. Smith and Brett Bode. Performance Effects of Node Mappings on the IBM Blue Gene/L Machine. In *Euro-Par*, pages 1005–1013, 2005.
- [30] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J. C. Sexton, and R. Walkup. Optimizing task layout on the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3):489–500, 2005.
- [31] Hao Yu, I-Hsin Chung, and Jose 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.
- [32] Tarun Agarwal, Amit Sharma, and Laxmikant V. Kalé. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.
- [33] Leonid Oliker, Andrew Canning, Jonathan Carter, Costin Iancu, Michael Lijewski, Shoaib Kamil, John Shalf, Hongzhang Shan, Erich Strohmaier, Stephane Ethier, and Tom Goodale. Scientific Application Performance on Candidate PetaScale Platforms. In *Proceedings of IEEE Parallel and Distributed Processing Symposium (IPDPS)*, 2007.
- [34] Blake G. Fitch, Aleksandr Rayshubskiy, Maria Eleftheriou, T. J. Christopher Ward, Mark Giampapa, and Michael 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.

- [35] Sameer Kumar, Chao Huang, Gengbin Zheng, Eric Bohm, Abhinav Bhatele, James C. Phillips, Hao Yu, and Laxmikant V. Kalé. Scalable Molecular Dynamics with NAMD on Blue Gene/L. *IBM Journal of Research and Development: Applications of Massively Parallel Systems*, 52(1/2):177–187, 2008.
- [36] Jack Dongarra and P Luszczek. Introduction to the hpc challenge benchmark suite. Technical Report UT-CS-05-544, University of Tennessee, Dept. of Computer Science, 2005.
- [37] Abhinav Bhatele, Eric Bohm, and Laxmikant V. Kale. Topology Aware Task Mapping Techniques: An API and Case Study. Technical Report 08-06, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, January 2008.
- [38] Abhinav Bhatele and Laxmikant V. Kale. An Evaluation of the Effect of Interconnect Topologies on Message Latencies in Large Supercomputers. Technical Report 08-07, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2008.
- [39] IBM Blue Gene Team. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development*, 52(1/2), 2008.
- [40] Abhinav Bhatele and Laxmikant V. Kale. Application-specific Topology-aware Mapping for Three Dimensional Topologies. In *Proceedings of Workshop on Large-Scale Parallel Processing (held as part of IPDPS '08)*, 2008.
- [41] Abhinav Bhatel  and Laxmikant V. Kal . Benefits of Topology Aware Mapping for Mesh Interconnects. *submitted to Parallel Processing Letters (LSPP special issue)*, 2008.
- [42] M. E. Tuckerman. Ab initio molecular dynamics: Basic concepts, current trends and novel applications. *J. Phys. Condensed Matter*, 14:R1297, 2002.
- [43] Klaus Schulten, James C. Phillips, Laxmikant V. Kale, and Abhinav Bhatele. Biomolecular modeling in the era of petascale computing. In D. Bader, editor, *Petascale Computing: Algorithms and Applications*, pages 165–181. Chapman & Hall / CRC Press, 2008.
- [44] Abhinav Bhatel  and Laxmikant V. Kal . Dynamic Topology Aware Load Balancing Algorithms for MD Applications. *submitted to Philosophical Transactions of the Royal Society A*, 2008.
- [45] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39(5):575–582, 1995.
- [46] Claude Bernard, Tom Burch, Thomas A. DeGrand, Carleton DeTar, Steven Gottlieb, Urs M. Heller, James E. Hetrick, Kostas Orginos, Bob Sugar, and Doug Toussaint. Scaling tests of the improved Kogut-Susskind quark action. *Physical Review D*, (61), 2000.
- [47] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kal . Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *18th International Parallel and Distributed Processing Symposium (IPDPS)*, page 78, Santa Fe, New Mexico, April 2004.