Parallel Processing Letters

Benefits of Topology Aware Mapping for Mesh Interconnects

Abhinav Bhatelé *

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801, USA

Laxmikant V. Kalé

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801, USA

Received June 20th, 2008
Revised August 25th, 2008
Communicated by Darren J. Kerbyson

ABSTRACT

The fastest supercomputers today such as Blue Gene/L, Blue Gene/P, Cray XT3 and XT4 are connected by a three-dimensional torus/mesh interconnect. Applications running on these machines can benefit from topology-awareness while mapping tasks to processors at runtime. By co-locating communicating tasks on nearby processors, the distance traveled by messages and hence the communication traffic can be minimized, thereby reducing communication latency and contention on the network. This paper describes preliminary work utilizing this technique and performance improvements resulting from it in the context of a n-dimensional k-point stencil program. It shows that even for simple benchmarks, topology-aware mapping can have a significant impact on performance. Automated topology-aware mapping by the runtime using similar ideas can relieve the application writer from this burden and result in better performance. Preliminary work towards achieving this for a molecular dynamics application, NAMD, is also presented. Results on up to 32,768 processors of IBM's Blue Gene/L, 4,096 processors of IBM's Blue Gene/P and 2,048 processors of Cray's XT3 support the ideas discussed in the paper.

Keywords: topology, mapping, mesh, torus, performance, scaling, supercomputers

1. Introduction

Two important aspects of a parallel program are computation and communication which decide its efficiency and performance. The computation needs to be divided evenly among processors to achieve near-optimal load balance. Communication, on the other hand, needs to be minimized across processors to ensure minimum

*The authors can be contacted by electronic mail at: bhatele@illinois.edu

overhead. These two objectives are not independent and have to be performed in tandem. To minimize communication, communicating tasks should be placed on the same physical processor. This is not feasible because many tasks communicate with each other in general and placing all of them on one processor inhibits parallelism. Instead, one can aim at placing communicating tasks on *nearby* processors. For machines with flat topologies, such as a fat-tree, the only choice is between local and remote communication and the concept of physical proximity is non-existent. On the other hand, the emergence of large machines with non-flat topologies brings in issues of physical proximity based on the number of links (hops) between processors. The number of hops traversed by a message can significantly impact its latency in spite of techniques such as wormhole routing and virtual cut-through. The effect becomes more pronounced in the presence of contention. In such cases, placing communicating tasks on nearby processors can lead to performance benefits. This is becoming an increasing concern for better performance and scaling and is the topic of our study in this paper.

Some of the fastest and biggest supercomputers today [1, 2] are connected by a three-dimensional torus interconnect. If the topology of a machine is not flat and visible to the runtime, it is possible to minimize inter-processor communication and balance it evenly across processors. The volume of inter-processor communication can be characterized by hop-bytes which can be defined in terms of hop-count [3]. *Hop-count* is the number of hops (network links) through which a message has to pass to reach one processor from another. *Hop-bytes* are obtained by multiplying the hop-count for a message by its message size. The sum of hop-bytes for all messages in a program gives us its total communication volume. Restricting communicating tasks on nearby processors can reduce the total communication volume on the network. Minimizing inter-processor communication requires two kinds of information during the actual program run: 1. communication graph of the parallel entities in a program and, 2. topology information about the processors being used on the particular machine.

This information is obtained from the Charm++ runtime system which is used to implement the topology interface and the parallel applications used in this paper. The Charm++ [4, 5] parallel language and runtime system is based on an object-oriented parallel programming model. This facilitates mapping at the granularity of virtual objects instead of processes. Charm++ provides us with a framework for obtaining the communication properties of the application and flexibility of mapping tasks to physical processors. It also provides an API to obtain topology information about the parallel machine we are running on. This framework will be discussed in Section 2.

A simple application written in Charm++ is used to demonstrate the effectiveness of topology mapping: a n-dimensional k-point stencil. Two sets of values are used for n and k: a 3-dimensional 7-point stencil and a 2-dimensional 5-point stencil (referred to as 3D Stencil and 2D Stencil henceforth in this paper). Stencil has regular communication to a fixed number of neighbors and hence benefits from a static mapping of its tasks to processors. The techniques developed specifically for Stencil can be generalized into an automated topology-aware mapping framework. This relieves the application writer of this burden and improves performance. In this context, some preliminary work on a production code called NAMD is presented. NAMD [6, 7] is a classical molecular dynamics application and benefits greatly from the load balancing framework in CHARM++. Possible modifications to the load balancing algorithm to consider topology of the machine are discussed.

1.1. *Previous Work*

The task mapping problem is NP-complete and has been proven to be computationally equivalent to the graph embedding problem [8]. A lot of research was done in this area in the 80s [8, 9, 10, 11]. Most of the work focused on hypercubes which were the popular networks then [9, 10, 12]. Research in the 80s was in two directions – physical optimization techniques and heuristic approaches. Physical optimization techniques involve simulated annealing [9, 13], graph contraction [14, 15] and genetic algorithms [16]. 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 [10] and recursive mincut bipartitioning [12] were theoretical studies with no real results on machines. Also, most of these techniques (heuristics techniques especially) were developed specifically for hypercube or linear array topologies.

Mapping techniques lost significance in the 90s with the appearance of more efficient interconnects and deployment of wormhole routing. With the emergence of large machines like Blue Gene/L and XT3, topology-aware mapping has become important 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 [17, 18, 19]. On IBM's Blue Gene/L, these issues have been studied both by system developers [20, 21] and application writers. Bhanot et. al [22] use initial heuristic mapping and simulated annealing to arrive at efficient mappings for Blue Gene/L. Yu [23] and Smith [24] discuss embedding techniques for graphs onto the 3D torus of Blue Gene/L which can be used by the MPI Topology functions. Application writers have also shown improvement by utilizing topology awareness in their codes [3, 21, 25, 26].

Our work is however one of the first for Cray XT3. Weisser at al. [1] discuss the effect of topology on job placement but we do not know of any published work which discusses topology-aware task-mapping on XT3. However, the developers of OpenMPI on Red Storm and Cray XT3 saw considerable promise in utilizing such information on these machines which was a motivation for our work. What differentiates our work from previous machine-specific research is that we have developed a single API which hides the machine level details from the application writer. It can be used on different machines with non-flat topologies (like Blue Gene/L, XT3, XT4 and Blue Gene/P).

This paper is organized as follows: In Section 2, CHARM++ and the stencil benchmarks are introduced. Section 3 discusses topology-aware mapping for the two variants of Stencil. Section 4 provides detailed analysis of the performance benefits achieved by topology mapping of Stencil. NAMD and preliminary work on its load balancers are discussed in Section 5.

2. Charm++ and Stencil

We use the CHARM++ runtime system (RTS) to facilitate topology-aware mapping on a variety of machines. CHARM++ [4] is an object-oriented parallel programming framework based on the idea of virtualization. Virtualization refers to the idea of dividing the problem into a large number of virtual processors (VPs) which are mapped to physical processors (PEs) by an intelligent runtime system. The number of VPs is typically much larger than the number of PEs (making the degree of virtualization greater than one). The benefits of virtualization include an overlap of computation and communication which hides message latencies by doing useful computation.

Tasks or VPs in a parallel application are called "chares" in the context of CHARM++. A *chare* is the basic unit of computation. It can be created on any processor and accessed remotely (through entry methods). A *Chare Array* is an indexed collection of chares. Each element of a chare array is called an array element. The CHARM++ RTS does a default mapping of chares to processors. It also provides the user with the flexibility to decide his own mapping for the chares. We will see how we can use this flexibility to do a better mapping than the system's default. Eventually we plan to automate such mapping in the RTS.

2.1. *Topology Interface in Charm++*

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). On Blue Gene/L, this information is available in a data structure called "BGLPersonality" and can be accessed using some system calls. On Blue Gene/P, there is a similar data structure called DCMF_Hardware_t which can be used for the purpose. Obtaining topology information is not straightforward on Cray XT3. There are no system calls which can provide information about the dimensions of the partition which has been allocated during a run. This information can be derived in several steps. Every node on the XT3 has a unique node ID. A static routing table is available on the machine which has the physical coordinates and neighbors for every node. The CHARM++ RTS reads this file during program start-up. To get the physical coordinates corresponding to a processor rank, the RTS obtains the node ID for the rank through a system call and then gets the coordinates from the routing table. Once it has the coordinates for all processors in an allocation, it can calculate the dimensions of the torus. This information about the topology collected by the RTS is available to the application through an API [27] in Charm++. We use this Topology API for running on Blue Gene/L, Blue Gene/P and XT3.

2.2. *3D and 2D Stencil*

Having described the implementation framework, we now describe the first application which has been used in this paper to demonstrate the benefits of topology-aware mapping. 3D Stencil is an implementation of a 3-dimensional 7-point stencil. It has a three dimensional array of doubles. In every iteration, each element of the array updates itself by computing the average of its six neighbors (two in each dimension) and itself. A 3D chare array is created to parallelize the computation using Charm++. Each element of this chare array is responsible for the computation of some contiguous elements (a 3D sub-partition) of the data array (Figure 1). These chares communicate with their neighbors to exchange the updated data on the boundaries.

2D Stencil is an implementation of a 2-dimensional 5-point stencil. It is similar to 3D Stencil except that it works with a two dimensional data array. In this case, a 2D chare array is used for the computation. Communication is simpler than in 3D Stencil since every element needs data from four neighbors. This example was chosen to show that even though the problem's dimensionality does not match the dimensionality of the machine, it can still benefit from topology mapping.

3. Mapping 3D and 2D Stencil

The technique of topology-aware mapping is discussed in this section which is the central theme of this paper. We shall understand the process of making mapping decisions and then devise a method to analyze the benefit of mapping. To analyze the advantages of topology-aware mapping, it will be compared to random and simple round-robin schemes where equal number of objects are mapped to each processor. These schemes do not have explicit information about the topology of the machine although the round-robin scheme has an implicit knowledge about the topology as we shall see later.

The communication properties of Stencil must be clearly understood to map the chare arrays topologically. 3D Stencil has a fairly simple communication pattern. Each chare object talks to its six neighbors (two in each dimension) to exchange the boundary elements. If these six neighbors can be placed on the same or one of the six neighboring processors, the distance traveled by each message can be minimized. The most simple scheme is to place the individual chares on the processors in a round-robin fashion. When we have multiple cores per node, this scheme does place *some* communicating chares on the same node. Also, since consecutive ranks get mapped to nearby physical processors in a job allocation, this scheme is not completely oblivious of the topology of the machine. Hence for a fair comparison, chares should be mapped randomly on to different processors, while keeping the load evenly balanced. This will be referred to as the random mapping.

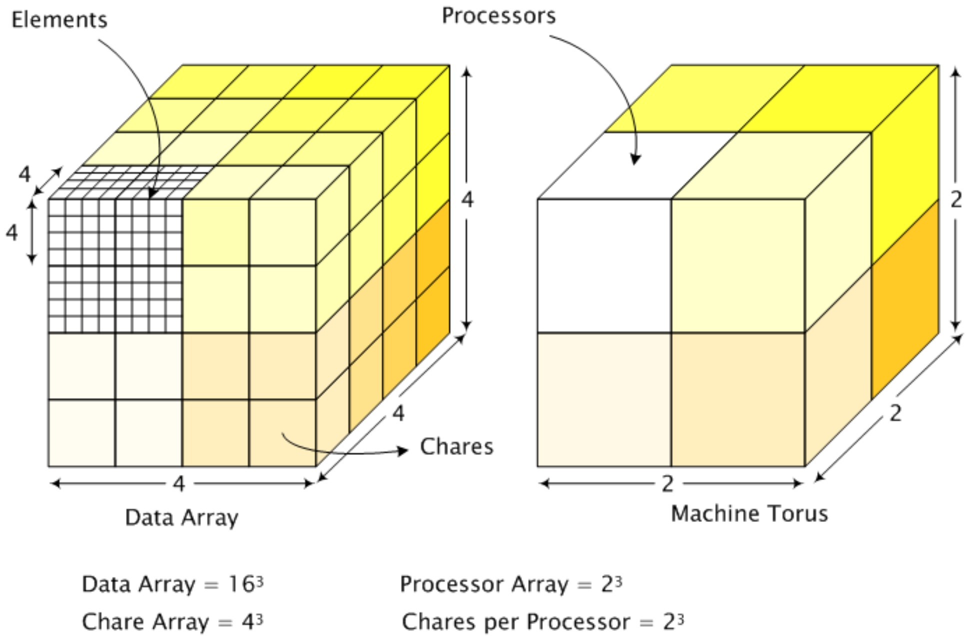

Fig. 1. Topology-aware mapping of 3D Stencil's data array onto the 3D processor grid. Different colors (shades) signify which chares get mapped to which processors

In the two schemes discussed above, no preference is given to local over remote communication. Since, in general there are multiple chares per processor, communicating chares can be placed on the same processor as much as possible. The idea is to divide the 3D chare array into equal-sized boxes and then map those to corresponding processors on the 3D torus. If the dimensions of the chare array are not exactly divisible by torus dimensions, then some processors get an extra row of elements compared to others. An effort is still made to minimize load imbalance. Again as in the case of individual chares, these boxes can be mapped in a random or round-robin fashion. Let us take a concrete example: say, the data array is of size 512^3 and the size of the torus is 8^3 (which gives 512 processors). Let each chare get a partition of size 16^3 from the data array which gives $512^3/16^3 = 32^3$ chares. Thus, a 3D chare array of size 32^3 is to be mapped on a torus of dimensions 8^3. Hence each processor gets $32^3/8^3 = 4^3 = 64$ chares. So, we can map a block of 4^3 chares on each processor.

In a topology-aware placement of chares, our intention is to favor local communication and minimize remote communication. So, the chare array is first divided into equal-sized boxes as discussed above and then boxes which communicate are mapped onto nearby physical processors on the torus. Starting from one end of the chare array, the partitioned boxes are placed on each processor considering the topology of the machine (Figure 1). The mapping is similar to superimposing a 3D object on another. The different shades in the figure show how the data array gets mapped to the torus. For the example in the figure, a cube of eight chares is placed on each processor.

For 2D Stencil, the chare array is folded on to the processor torus. The longer dimension of the array is folded along the third dimension of the torus. The shorter dimension of the chare array is matched up with one of the torus' dimensions (generally the shorter one). The longer dimension of the chare array is split between other two dimensions of the torus. This is similar to splitting the 2D array into 'n' 2D arrays where n is one dimension of the torus and then mapping the n arrays to n planes of the torus. Again, in this case also, chares can be grouped into boxes to favor local communication and the boxes can be mapped topologically to minimize remote communication traffic.

4. Evaluation of Mapping Strategies

This section compares three schemes discussed above: 1. Random mapping of chares, 2. Round-robin mapping of chares, and 3. Topology-aware mapping of chares. The applications were run on Blue Gene/L at IBM T J Watson, Blue Gene/P (Intrepid) at Argonne National Lab (ANL) and Cray XT3 (BigBen) at Pittsburgh Supercomputing Center (PSC) for results in this paper. Watson Blue Gene/L has $20,480$ nodes and the nodes are connected by a 3D torus [28]. Each midplane of Blue Gene/L (512 nodes) is a complete torus in itself of size $8 \times 8 \times 8$. If one processor per node is used, it is called the co-processor (CO) mode and if both processors are used, it is called the virtual node (VN) mode. Instead of using MPI, CHARM++ uses a lower-level one-sided communication library for messaging on Blue Gene/L [29]. The ANL Blue Gene/P machine has $8,192$ nodes and each node has four processors. Just like Blue Gene/L, the nodes are connected by a 3D torus and a midplane of 512 nodes is a complete torus. BigBen has $2,112$ nodes connected into a 3D torus of size $11 \times 12 \times 16$ by a custom SeaStar interconnect [30]. Out of these $2,112$ nodes only $2,068$ can be used as compute nodes. Each node has two Opteron processors. For partitions smaller than the full machine, it is not a torus. Also, one cannot get a contiguous allocation of nodes on XT3 by default. Runs for this paper were done with help from the PSC staff to set up a reservation to get a grid of $8 \times 8 \times 16$ which is 1024 nodes.

4.1. *Comparison of Hop-bytes*

Hop-bytes for an application are an indication of the total volume of communication on the network. More hop-bytes communicated signifies greater contention for each link and hence might result in increases message latencies. So, hop-bytes is one metric used to estimate the success of topology-aware schemes. However, it should be understood that a mere reduction in the total hop-bytes communicated does not guarantee better performance. As we shall see later in the paper, a latency-tolerant application might not benefit from a reduction in hop-bytes.

Table 1 shows the execution time of 3D Stencil for different mapping schemes. All runs in this table were done with a problem size of $2048 \times 2048 \times 2048$ and with 1 chare per processor. Round-robin mapping of chares to processors does better than a random mapping because the round-robin mapping has an implicit benefit of co-locating communication objects on nearby physical processors (since consecutive ranks are mapped to nearby processors by default). The benefit of topology-aware mapping is seen in the comparison between a round-robin and topology-aware mapping of chares. Topology-aware mapping improves the performance by nearly 20% in some cases. Detailed scaling results comparing round-robin and topology-aware mapping are given in Section 4.2.

Table 1. Execution time (in milliseconds) of 3D Stencil on Blue Gene/L (CO mode) for different mapping schemes (Data Size: $\mathbf{2048} \times \mathbf{2048} \times \mathbf{2048}$)

Processors	**Random**	**Round-robin**	**Topology-aware**
512	4072.5	1560.9	1538.5
1024	2705.6	964.8	821.9
2048	1814.5	497.4	421.1
4096	1155.6	256.1	212.7
8192	534.4	106.7	108.1

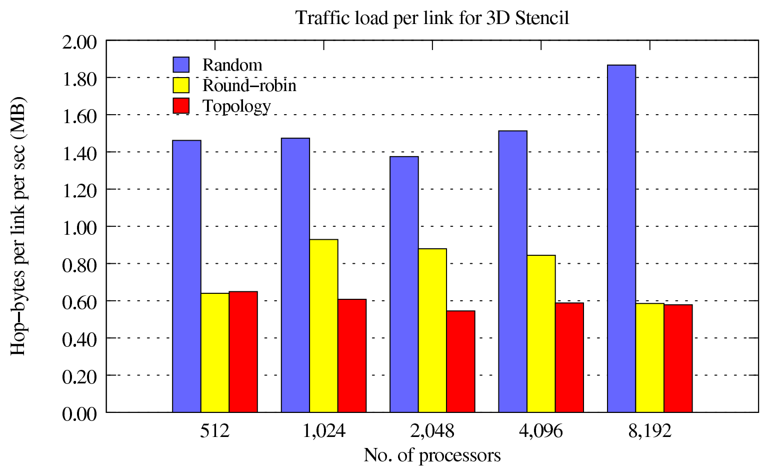

Fig. 2. Hop-counts for 3D Stencil running on Blue Gene/L (Data Size: $2048 \times 2048 \times 2048$)

The reduction in hop-bytes needs to be quantified, to measure the improvement in performance analytically. Hop-counts quantify the hop-bytes for Stencil, since the size of each message in this application is the same. For each run, the total hop-bytes communicated per link per second is calculated. Figure 2 shows the hop-bytes in MB for running the application on 512 to 8,192 processors of Blue Gene/L. At

1,024 processors, hop-bytes reduce by 60% compared to the random strategy and by 35%, compared to the round-robin strategy. The strange variation in the hop-counts for any particular strategy with increasing number of processors is probably due to the shape of the torus for these runs. This benefits certain torus shapes more than others. In the next section, we shall look at the performance benefits from this technique which motivate its automation in the runtime system.

Table 2. Execution time (in milliseconds) of 3D Stencil on Blue Gene/P (VN mode) for different mapping schemes (Data Size: $\mathbf{2048} \times \mathbf{1024} \times \mathbf{1024}$)

Processors	**Random**	**Round-robin**	**Topology-aware**
512	348.18	335.34	334.82
1024	184.70	170.98	170.77
2048	87.39	86.19	86.07
4096	49.46	43.88	43.35

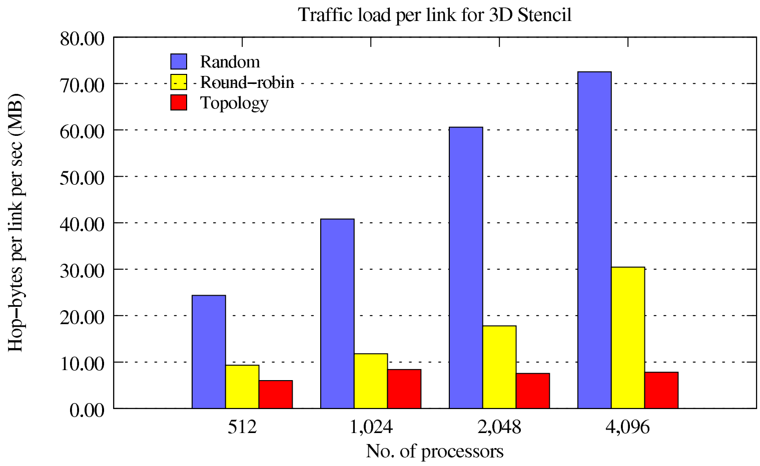

Fig. 3. Hop-counts for 3D Stencil running on Blue Gene/P (Data Size: $2048 \times 1024 \times 1024$)

Table 2 presents similar results for running 3D Stencil on Blue Gene/P for a problem size of $2048 \times 1024 \times 1024$. These runs were also done with 1 chare per processor. We see a smaller impact of topology-awareness on Blue Gene/P if we compare runs at a specific number of processors. This might be due to one or a combination of several reasons: 1. At a given number of cores, the size of the torus on Blue Gene/P is smaller than that on Blue Gene/L. For example at $4,096$ processors, the torus is $8 \times 8 \times 16$ on Blue Gene/P in VN mode and $16 \times 16 \times 16$ on Blue Gene/L in CO mode. Hence, the average number of hops traversed by each message are smaller, 2. Link bandwidth on Blue Gene/P is 2.5 times more than that on Blue Gene/L and message latencies are also smaller, 3. On Blue Gene/P, communication gets offloaded to the DMA engine which does not exist on Blue Gene/L.

On Blue Gene/P, we use strategies similar to those on Blue Gene/L and hence, we see similar considerable reduction in hop-bytes as we go from random to round-robin to topology-aware mapping (Figure 3). The total number of hop-bytes increases for random and round-robin strategies as we increase the size of the torus but remains nearly the same for topology-aware mapping across different torus sizes. In the next subsection, we will analyze the effects of virtualization and also see performance results on Cray XT3.

4.2. *Effects of Virtualization*

A fine-grained decomposition leading to a degree of virtualization greater than one can help in overlapping communication with computation and hence reduce the effects of message latencies. As we will see, topology mapping still benefits performance because it reduces contention by different messages for the same network links. Contention increases with finer decomposition because of greater number of short messages and hence a benefit from mapping. Next, we will compare round-robin versus topology mapping on Blue Gene/L and Cray XT3 for different degrees of virtualization and compare between them.

Table 3. Execution time (in milliseconds) of 3D Stencil on Blue Gene/L for different chare sizes (RR: Round-robin, TO: Topology-aware)

Data Size	**CO mode ($512 \times 512 \times 512$)**				**VN mode ($1024 \times 512 \times 512$)**			
Chare Size	**$16 \times 16 \times 16$**		**$32 \times 32 \times 32$**		**$16 \times 16 \times 16$**		**$32 \times 32 \times 32$**	
Processors	**RR**	**TO**	**RR**	**TO**	**RR**	**TO**	**RR**	**TO**
512	23.06	20.49	29.11	24.86	62.15	51.09	78.44	47.08
1024	11.54	10.23	17.42	14.78	30.72	25.22	38.48	29.86
2048	6.66	5.29	8.29	7.74	23.24	11.23	19.49	17.10
4096	3.15	2.82	4.05	3.43	9.08	5.73	9.34	9.81
8192	1.68	1.51	-	-	4.02	3.25	4.63	5.25
16384	0.89	0.86	-	-	2.07	1.95	-	-
32768	-	-	-	-	2.47	1.23	-	-

Table 3 compares the time taken per iteration of 3D Stencil on Blue Gene/L for round-robin vs. topology-aware mapping. Topology-aware (TO) mapping does better than round-robin (RR) at all processor counts. The maximum improvement is at 2K processors in VN mode for the 16^3 chare size, where the performance improves by nearly two times. This matches the huge reduction in hop-counts in Figure 2. The chare size here refers to the number of data elements per chare which governs the grain-size of the computation. The table also shows the effect of virtualization (number of chares per processor) on performance. A bigger chare size (signifying coarser granularity) helps on 512 processors in VN mode but for all other runs, smaller chare size gives us the best performance. Size of the chares decides the number of chares per processor. Chare size of 16^3 gives 64 chares per processor for a 512-processor run while a chare size of 32^3 gives 8 chares per processor. At $4,096$ processors in CO mode, for the 32^3 chare size, there is just one chare per processor. We do not have timings beyond $4,096$ cores, since once the degree of virtualization becomes one, the problem cannot be parallelized further.

Experiments similar to those on Blue Gene/L were repeated on Cray XT3 to test our mapping schemes and the topology interface written in CHARM++ for XT3. Table 4 shows performance numbers for scaling of 3D Stencil from 256 to $2,048$ processors. As expected, we get a performance improvement of up to 20%. On XT3, the impact of topology-aware mapping is smaller and that this needs further investigation. We can conclude that XT3 is a more forgiving architecture because of higher bandwidth per node. The story of hop-counts is exactly similar to that of Blue Gene/L (Figure 2). We get a reduction of five to six times in the hop-count.

Table 4. Execution time (in ms per iteration) and hop-counts for 3D Stencil on Cray XT3 for different chare sizes (Data Size: fixed at $\mathbf{512 \times 512 \times 512}$)

Chare Size	**16 × 16 × 16**				**32 × 32 × 32**			
Metric	**Time (secs)**		**Hop-count**		**Time (secs)**		**Hop-count**	
Processors	**RR**	**TO**	**RR**	**TO**	**RR**	**TO**	**RR**	**TO**
256	28.05	22.51	296960	53248	16.81	16.65	37888	13312
512	14.51	12.17	335872	69632	9.15	8.98	69632	17408
1024	8.21	7.14	479232	86016	5.51	5.31	94720	21504
2048	6.70	5.65	445952	118784	4.57	4.42	86992	29696

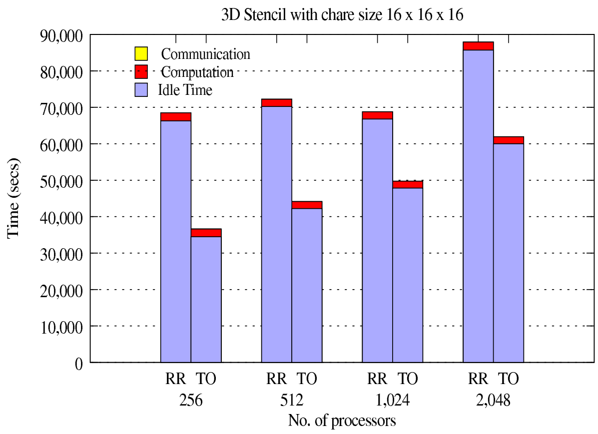

Fig. 4. Effect of topology mapping on latency on Blue Gene/L (RR: Round-Robin, T0: Topology, Data Size: 512 x 512 x 512)

Although the hop-count improves by a factor of five to six, overall performance does not improve as much. Reduction in hop-bytes quantifies the reduction in communication traffic on the network. This reduces the latency of messages over the network and hence processors do not have to wait as long for incoming messages. Hence, the reduction in hop-counts removes the bottleneck of message latency on the network and cannot linearly reflect on the performance improvement achieved. To understand the effect of topology-aware mapping, we used the performance analysis tool Projections [31], a part of the CHARM++ distribution. Figure 4 shows the time spent in communication and computation (added across all processors) for different runs. The time when a processor is waiting for messages to arrive is referred to as "idle time". As is clearly visible, idle time decreases substantially for the case of topology-aware mapping which signifies that message latency is significantly reduced. The reduction is consistent across different number of processors. It should be noted that the benchmark is very fine-grained and has a very high communication-to-communication ratio. But this still signifies that for heavily communication-bound applications, topology-aware mapping has a significant impact on performance.

Table 5. Execution time and hop-counts for 2D Stencil on Blue Gene/L (CO mode) for different decompositions (Data Size: fixed at **1 billion elements**)

Chare Size	**128 × 128**				**256 × 256**			
Metric	**Time (secs)**		**Hop-count**		**Time (secs)**		**Hop-count**	
Processors	**RR**	**TO**	**RR**	**TO**	**RR**	**TO**	**RR**	**TO**
512	55.29	53.03	149504	25600	53.58	46.75	298736	12800
1024	27.23	26.21	149504	34816	28.21	23.24	298736	17408
2048	14.00	12.91	1197824	51200	16.19	11.44	308604	25600
4096	6.98	6.63	672512	83968	10.49	5.77	467972	41984
8192	3.62	3.47	401664	100352	6.00	2.93	806844	50176
16384	1.94	1.89	266240	133120	3.10	1.51	575360	66560

Topology mapping of 2D Stencil was done to prove that the dimensionality of the problem need not be the same as the dimensionality of the torus. Similar performance results are seen for 2D Stencil on Blue Gene/L (see Table 5). Two chare sizes (128^2 and 256^2) are used. An improvement of more than two times is seen on $8,192$ processors in VN mode for the 256^2 chare size. Various results for 3D and 2D Stencil support our claim of the benefit obtainable from topology-aware mapping on 3D mesh/torus machines.

It is observed that for Stencil computations, round-robin mapping does better than random mapping and topology-aware mapping does better than round-robin mapping in general. It is also important to keep in mind that round-robin mapping can only benefit applications where rank i communicates only with rank $i+1$ (Stencil-like near-neighbor communication). For any other complex communication patterns which is what we find for most applications, it becomes imperative to explicitly map objects. We will see in the next section that we need to develop a strategy depending on the communication patterns of the application to get performance improvements.

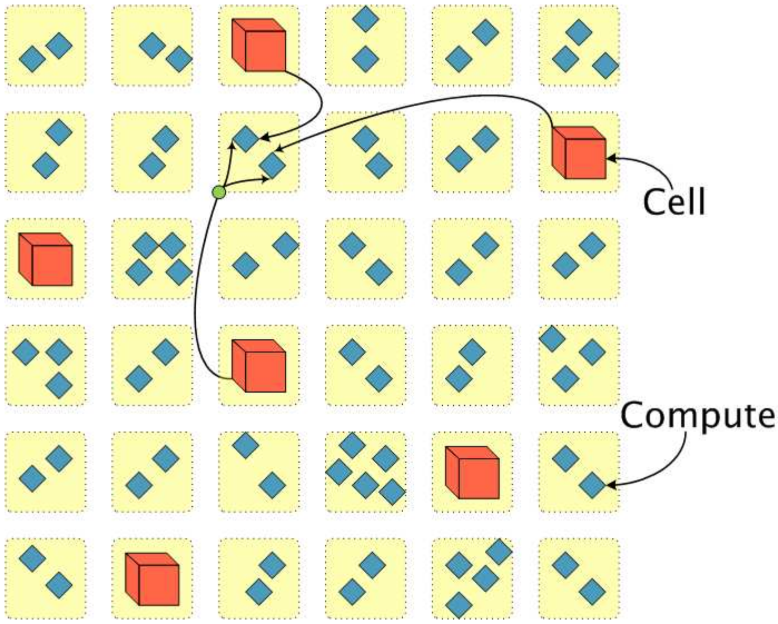

Fig. 5. Parallelization of force calculations in NAMD

5. NAMD: Preliminary Study

In an effort to automate the process of topology-aware mapping for "real" applications, a molecular dynamics code was chosen as the test bed. NAMD [6, 7, 26] is a production molecular dynamics code that is used for simulating small to very large molecular systems on large parallel machines. Let us first motivate why is NAMD a good fit for such schemes. Any molecular dynamics code involves the calculation of forces due to bonds and non-bonded forces on each atom. Non-bonded forces are composed of electrostatic and Van der Waal's forces. A simple parallelization scheme is to divide the simulation box into smaller cells (referred to as patches in NAMD) and to calculate forces between them. To make this scheme scalable beyond more processors than there are cells, NAMD does a hybrid of spatial and force decomposition to combine the advantages of both. For every pair of interacting patches, a chare (called a compute) is created which is responsible for calculating the pairwise forces between the patches. Thus patches hold the information about the atoms and computes do the actual calculation. These chares do not have to necessarily reside on the same processor as the patches they communicate with. As shown in Figure 5, a compute responsible for the force calculation between two patches can be on a third processor. NAMD leverages the load balancing framework in CHARM++ to spread out these computes evenly among all processors. In the following sections, we will see how can topology information be used during load balancing.

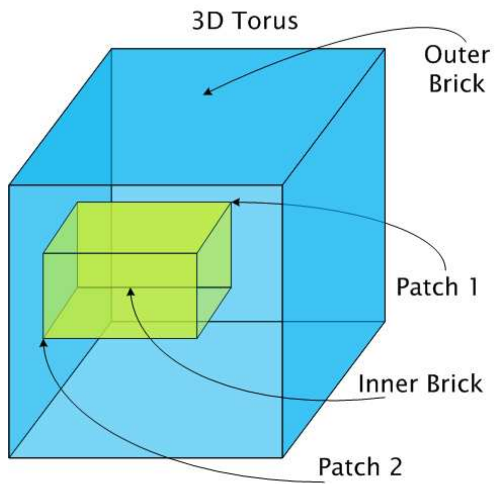

Fig. 6. Topology-aware placement of computes close to their patches

5.1. *Load Balancing*

NAMD depends heavily on the load balancing framework provided by CHARM++ for good performance. Computational load in NAMD is persistent across iterations and hence, load information from previous iterations can be used in future iterations. Every few hundred or thousand iterations, a few iterations are instrumented and the load information from these steps is used during the load balancing phase to unload the overloaded processors. Patches and bonded computes are non-migratable chares which means they do not move from their home processor once assigned. Patches are assigned by a topology-aware Orthogonal Recursive Bisection [26]. Non-bonded computes on the other hand are migratable and can be moved around during load balancing. The load balancing framework records the communication information about the application and the topology API discussed earlier gives us the required information about the machine. With these two things, it should be possible to automate the process of topology mapping in the load balancers.

5.2. *Topology-aware Enhancements*

During load balancing, when the runtime is trying to find a processor to place the compute on, it can consider the topology of the machine. This can reduce the distance (in terms of hops) between the patches and the computes they communicate with. It is best to place a compute on a processor such that the sum of the distance from its two patches is minimized. To ensure this, the coordinates of the two processors which host the patches with which the given compute interacts are obtained. The RTS then tries to find a processor within the region enclosed by these two processors on the torus (see Figure 6). For any point within this region (called the inner brick), the sum of distances from the two patches (at the corners) to the compute is same. Hence the RTS tries to find the least overloaded processor within this brick first. If it fails, then it tries the rest of the torus (called the outer brick). The idea is to spiral around the inner brick on the outside and to find the first underloaded processor. The compute is then placed on it.

Table 6. Reduction in hop-bytes for NAMD on Blue Gene/L (Benchmark:ApoA1)

	CO Mode (MB/iter)		**VN mode** (MB/iter)	
PEs	**Naïve**	**Topology**	**Naïve**	**Topology**
512	202.39	150.10	297.38	307.49
1024	437.01	316.29	505.34	434.62
2048	776.79	514.13	813.06	512.25
4096	1672.93	1193.04	1435.57	1155.41
8192	2920.12	2321.7	2910.08	2290.12

The topology-aware scheme discussed above reduces the hop-bytes considerably which is one metric for the evaluation of the mapping algorithm. Table 6 shows the reduction in hop-bytes as a result of topology-aware mapping of patches and topology-aware load balancing of computes for a molecular system Apolipoprotein-A1 (ApoA1). This system has 92,224 atoms and a patch grid of size $108.86 \times 108.86 \times 77.76$ $\AA$. The numbers given are hop-bytes in MB per iteration added across all processors for all messages. We get nearly 30% improvement (reduction in hop-bytes) at 4K processors in CO mode. Likewise at this point, we also get an improvement in time-step per iteration from 4.68 to 3.88 milliseconds (ms). We just have preliminary performance numbers at this point. We do not get performance improvements at all processor counts in spite of the reduction in hop-bytes. One possible reason for this might be the latency-tolerant nature of the computation in NAMD. This needs more investigation and we hope to present detailed performance results in a later publication.

6. Future Work and Conclusion

This preliminary work has shows benefits from topology-aware mapping of chare arrays in 3D and 2D Stencil. It also provides us with useful insights on important issues which one might have to face during topology mapping. We wish to extend this idea to automate mapping for any application with heavy communication (where such schemes will have an impact). Given the communication dependencies between objects and topology of a machine, the runtime should automatically do an intelligent mapping. This would remove the burden of mapping from the user and give optimized performance compared to a random topology-oblivious mapping.

In this direction, NAMD shows a reasonable improvement in terms of hop-bytes and some improvement in performance for higher processor counts. The strategy developed for NAMD's load balancers can be applied elsewhere quite easily. NAMD is a specific case of section multicasts where each multicast target receives the multicast message from only two sources. Hence, the scheme can be generalized to work for section-multicast and topology-aware load balancers. This is the final goal we are aiming for. This is useful for communication scenarios such as in matrix multiplication.

The work presented in this paper demonstrates the benefit of topology-awareness in mapping of objects on to a parallel machine statically or during load balancing. Results presented for Blue Gene/L, Blue Gene/P and Cray XT3 validate our hypothesis that topology considerations are important on all torus machines. We hope to utilize the insights gained from the study of Stencil and NAMD to create a generalized automatic framework for topology-aware mapping. This would benefit many applications running on three-dimensional machines in the future.

Acknowledgments

This work was supported in part by DOE grants B341494 funded by the Center for Simulation of Advanced Rockets and W-7405-ENG-48 funded by Lawrence Livermore National Laboratory and a NIH Grant PHS 5 P41 RR05969-04 for Molecular Dynamics. This research was supported in part by NSF through TeraGrid resources [32] provided by NCSA and PSC through grants ASC050040N and MCA93S028. We thank Shawn T. Brown and Chad Vizino from Pittsburgh Supercomputing Center for help with system reservations and runs on BigBen. They had to do considerable changes to the batch scheduler to accommodate our runs. We also thank Fred Mintzer and Glenn Martyna from IBM for access and assistance in running on the Watson Blue Gene/L. This research also used running time on Blue Gene/P of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] 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.
[2] M.Blumrich, D.Chen, P.Coteus, A.Gara, M.Giampapa, P.Heidelberger, S.Singh, B.Steinmacher-Burow, T.Takken, and P.Vranas. Design and Analysis of the Blue Gene/L Torus Interconnection Network. *IBM Research Report*, December 2003.
[3] 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.
[4] Laxmikant V. Kale, Eric Bohm, Celso L. Mendes, Terry Wilmarth, and Gengbin Zheng. Programming Petascale Applications with Charm++ and AMPI. In D. Bader, editor, *Petascale Computing: Algorithms and Applications*, pages 421–441. Chapman & Hall / CRC Press, 2008.
[5] L. V. Kale and Sanjeev Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.
[6] 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.

[7] 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.

[8] Shahid H. Bokhari. On the mapping problem. *IEEE Trans. Computers*, 30(3):207–214, 1981.

[9] S. Wayne Bollinger and Scott F. Midkiff. Processor and link assignment in multicomputers using simulated annealing. In *ICPP (1)*, pages 1–7, 1988.

[10] Soo-Young Lee and J. K. Aggarwal. A mapping strategy for parallel processing. *IEEE Trans. Computers*, 36(4):433–442, 1987.

[11] P. Sadayappan and F. Ercal. Nearest-neighbor mapping of finite element graphs onto processor meshes. *IEEE Trans. Computers*, 36(12):1408–1424, 1987.

[12] 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.

[13] S. Wayne Bollinger and Scott F. Midkiff. Heuristic technique for processor and link assignment in multicomputers. *IEEE Trans. Comput.*, 40(3):325–333, 1991.

[14] Francine Berman and Lawrence Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4(5):439–458, 1987.

[15] 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.

[16] S. Arunkumar and T. Chockalingam. Randomized heuristics for the mapping problem. *International Journal of High Speed Computing (IJHSC)*, 4(4):289–300, December 1992.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Kei Davis, Adolfy 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] Abhinav Bhatele. Application specific topology aware mapping and load balancing for three dimensional torus topologies. Master's thesis, Dept. of Computer Science, University of Illinois, 2007. http://charm.cs.uiuc.edu/papers/BhateleMSThesis07.shtml.

[28] N. R. Adiga, G. Almasi, , Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, and A. A. Bright. An Overview of the Blue Gene/L Supercomputer. In *Supercomputing 2002 Technical Papers*, Baltimore, Maryland, 2002. The Blue Gene/L Team, IBM and Lawrence Livermore National Laboratory.

[29] Michael Blocksome, Charles Archer, Todd Inglett, Pat McCarthy, Mike Mundy, Joe Ratterman, Albert Sidelnik, Brian Smith, Gheorghe Almasi, Jose Castanos, Derek Lieber, Jose Moreira, Sriram Krishnamoorthy, Vinod Tipparaju, and Jarek Nieplocha. Design and Implementation of a One-Sided Communication Interface for the IBM eServer Blue Gene Supercomputer. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.

[30] Jeffrey S. Vetter, Sadaf R. Alam, Thomas H. Dunigan Jr., Mark R. Fahey, Philip C. Roth, and Patrick H. Worley. Early evaluation of the cray xt3. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[31] Laxmikant V. Kale, Gengbin Zheng, Chee Wai Lee, and Sameer Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, volume 22, pages 347–358, February 2006.

[32] C. Catlett and et. al. TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications. In Lucio Grandinetti, editor, *HPC and Grids in Action*, Amsterdam, 2007. IOS Press.