Application-specific Topology-aware Mapping for Three Dimensional Topologies

Abhinav Bhatelé and Laxmikant V. Kalé

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
bhatele2@uiuc.edu, kale@uiuc.edu

Abstract

The fastest supercomputers today such as Blue Gene/L and XT3 are connected by a 3-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 for a fine-grained application with a high communication to computation ratio, topology-aware mapping has 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 and 2,048 processors of Cray's XT3 support the ideas discussed in the paper.

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 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. For older machines with flat topologies, such as fat-tree, the only choice was between local and remote communication. The emergence of large machines with non-flat topologies brings in issues of latency and contention on the network. In the absence of contention, the effect of latency in sending messages is insignificant. However, when contention is present, latency becomes significant and increases with the distance travelled by messages. 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 [12, 20] 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 [2]. **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. 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++ [15, 16] 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. 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 [14, 22] 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 computationally equivalent to the graph embedding problem [8]. A lot of research was done in this area in the era of connection machines in the 80s [8, 9, 19, 21]. Most of the work focused on hypercubes which were the popular networks then [9, 13, 19]. As messaging latencies decreased with more efficient interconnects, topology of the machine became less important. With the emergence of large machines like Blue Gene/L and XT3, application writers and system developers have started studying such issues again [3, 11].

Bhanot et. al [4] use initial heuristic mapping and simulated annealing to arrive at efficient mappings for Blue Gene/L. Yu [25] and Smith [23] 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 [7, 2, 11, 18]. Our work is however one of the first for Cray XT3. Weisser at al. [12] 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).

2 CHARM++ and Stencil

We use the CHARM++ RTS to facilitate topology-aware mapping on a variety of machines. CHARM++ [15] is an object-oriented parallel programming framework based on the idea of virtualization. Virtualization refers to the idea of dividing the problem into 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).

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.

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 (BG/L), this information is available in a data structure called "BGLPersonality" and can be accessed using some system calls. 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. Information about the topology is available to the application through an API [5] in CHARM++. We use this Topology API for running on BG/L and XT3.

2.2 3D and 2D Stencil

Having described the implementation framework, it is time to 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

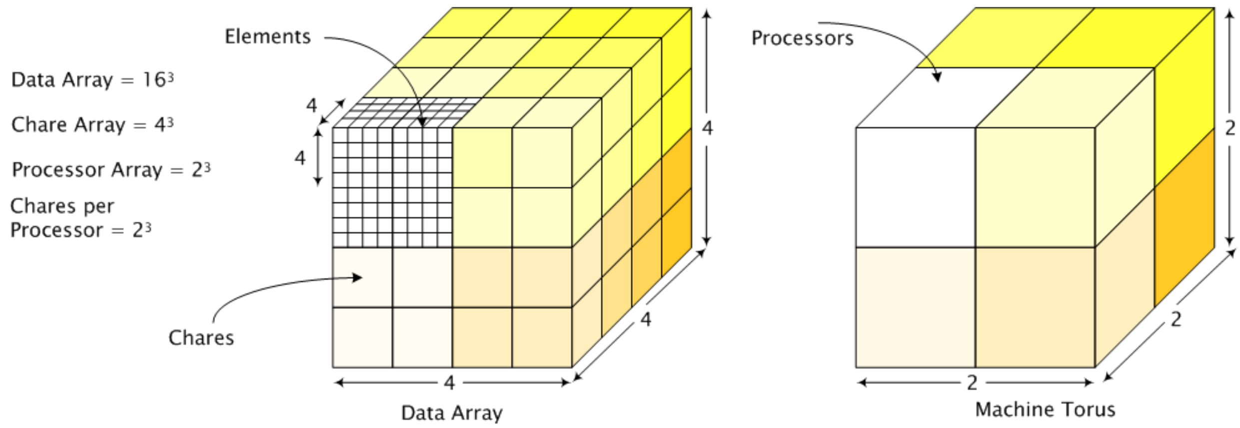

Figure 1. Topology-aware mapping of 3D Stencil's data array onto the 3D processor grid. Different colors (shades) signify which chares get mapped to which processors

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 has two dimensions. 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 prove 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.

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. However, since consecutive ranks get mapped to nearby physical processors, 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 (still keeping the load evenly balanced).

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.

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 map-

Processors	Data Size	Chares RR	Blocks RD	Blocks TO
1024	$1024 \times 512 \times 512$	32.84	36.54	25.79
2048	$1024 \times 1024 \times 512$	32.89	37.29	25.81

Table 1. Execution time (in seconds) of 3D Stencil on Blue Gene/L for different mapping strategies (RR: Round-robin, RD: Random, TO: Topology-aware)

ping 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.

3.1 Evaluation of Mapping Strategies

This section compares three schemes discussed above: 1. Round-robin mapping of chares, 2. Random mapping of blocks, 3. Topology-aware mapping of blocks. Random mapping of chares is a naïve scheme which gives the worst performance and is not used for comparison. The applications were run on Blue Gene/L at IBM T J Watson and Cray XT3 at PSC (BigBen) for results in this paper. Watson BG/L has 20,480 nodes and the nodes are connected by a 3D torus [1]. Each midplane of BG/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 BG/L [6]. BigBen has 2,112 nodes connected into a 3D torus of size $11 \times 12 \times 16$ by a custom SeaStar interconnect [24]. 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.

Table 1 shows the execution time of 3D Stencil for different mapping schemes. Round-robin mapping of chares to processors does better than a random mapping of blocks 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 random and topology-aware mapping of blocks of chares. Topology-aware mapping improves the performance by nearly 30%. Detailed scaling results comparing round-robin and topology-aware mapping are given in Section 4.

The reduction in hop-bytes needs to be quantified, to measure the improvement in performance analytically.

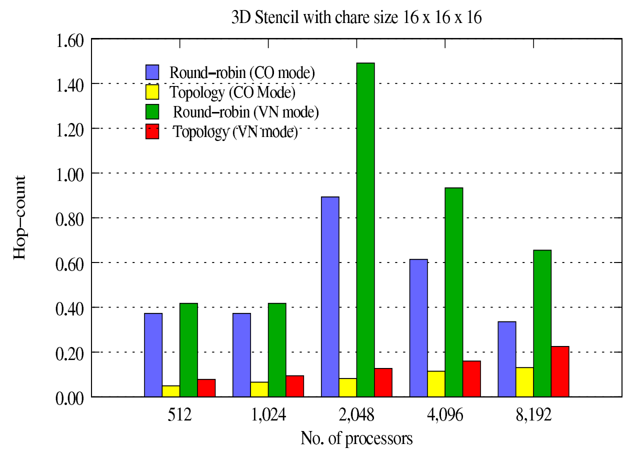

Figure 2. Hop-counts for 3D Stencil running on Blue Gene/L (Data Size: 512 x 512 x 512)

Hop-counts quantify the hop-bytes for Stencil, since the size of each message in this application is same. For each run, the total hops travelled by all messages in one iteration are calculated. Figure 2 shows the hop-counts for running the application on 512 to 16,384 processors of BG/L for one iteration. Hop-counts reduce by 4 to 5 times in general compared to round-robin mapping and at 2,048 processors, they are reduced by 10 times. The strange variation in the hop-counts for the round-robin case is due to the implicit topology-awareness since consecutive ranks end up on contiguous physical processors. This benefits more for certain torus shapes and less for others. In the next section, we shall look at the performance benefits from this technique which motivate its automation in the runtime system.

4 Performance Results

Table 2 compares the time taken for 1000 iterations of 3D Stencil on BG/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

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

Chare Size	$16 \times 16 \times 16$				$32 \times 32 \times 32$			
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

Table 3. Execution time and hop-counts for 3D Stencil on Cray XT3 for different chare sizes (Data Size: fixed at $512 \times 512 \times 512$).

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 4K processors in CO mode, for the 32^3 chare size, there is just one chare per processor. We do not have timings beyond 4K, since once the degree of virtualization becomes one, the problem cannot be parallelized further.

Experiments similar to those on BG/L were repeated on Cray XT3 to test our mapping schemes and the topology interface written in CHARM++ for XT3. Table 3 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 BG/L (Figure 2). We get a reduction of five to six times in the hop-count.

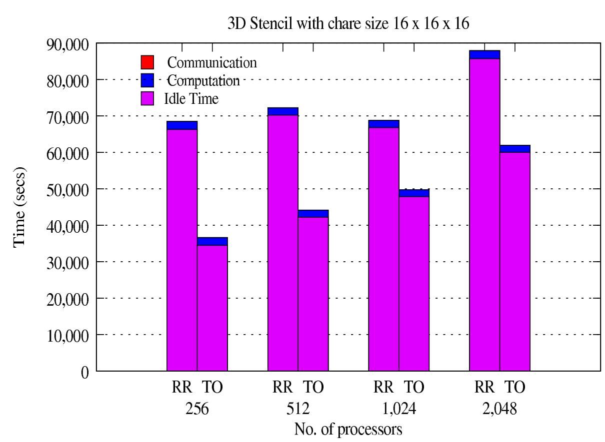

Figure 3. Effect of topology mapping on latency on Blue Gene/L (RR: Round-Robin, T0: Topology, Data Size: 512 x 512 x 512)

Although the improvement in hop-counts is five to six times, improvement in performance is not 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 [17], a part of the CHARM++ distribution. Figure 3 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.

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 BG/L (Table 4). Two chare sizes (128^2 and 256^2) are used. An improvement of more than two times is seen on 8K 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 torus-like machines. Motivated by the study on topology-aware mapping of Stencil, work on a production code has begun which we discuss briefly in the next section.

5 NAMD: Preliminary Study

In an effort to automate the process of topology-aware mapping for "real" applications, a molecular dynamics code has been chosen as the test bed. NAMD [14, 18, 22] is a production molecular dynamics code that is used for simulating small to very large molecular systems on large parallel machines. We first motivate why NAMD is 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. For parallelization, the simulation box is divided into smaller cells (referred to as patches in NAMD) and forces are calculated between them. To use 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.

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 step 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 [18]. 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.

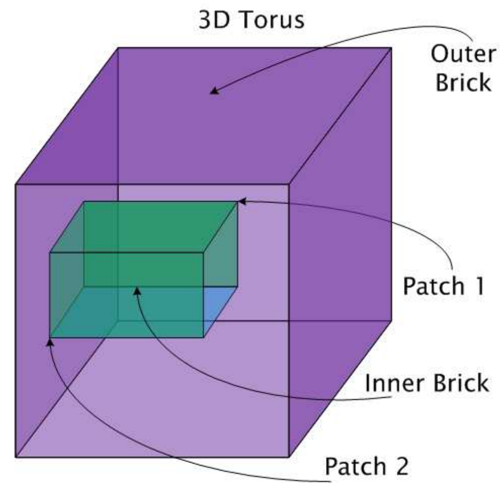

Figure 4. Parallelization of force calculations in NAMD

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 4). 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

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

Table 4. Execution time and hop-counts for 2D Stencil on Blue Gene/L (CO mode) for different decompositions (Data Size: fixed at 1 billion elements).

	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

Table 5. Reduction in hop-bytes for NAMD on Blue Gene/L (Benchmark:ApoA1)

inner brick on the outside and to find the first underloaded processor. The compute is then placed on it.

The topology-aware scheme discussed above reduces the hop-bytes considerably which is one metric for the evaluation of the mapping algorithm. Table 5 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 hope to present detailed performance results in a later publication.

6 Future Work and Conclusion

This preliminary work has shown success in mapping of chare arrays in 3D Stencil. It has also provided us with useful insights on important issues which one might have to face during topology mapping. We wish to extend this idea to do this 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 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.

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. We hope to utilize the insights gained from the study of these applications 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 a DOE Grant B341494 funded by the Center for Simulation of Advanced Rockets and a NSF Grant ITR 0121357 for Quantum Chemistry. This research was supported in part by NSF through TeraGrid resources [10] provided by NCSA and PSC through grants ASC050039N and ASC050040N. 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, Glenn Martyna and Sameer Kumar from IBM for access and assistance in running on the Watson Blue Gene/L.

References

[1] 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.

[2] T. Agarwal, A. Sharma, and L. 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.

[3] G. Almasi, S. Chatterjee, A. Gara, J. Gunnels, M. Gupta, A. Henning, J. E. Moreira, and B. 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.

[4] 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.

[5] A. 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.

[6] M. Blocksome, C. Archer, T. Inglett, P. McCarthy, M. Mundy, J. Ratterman, A. Sidelnik, B. Smith, G. Almasi, J. Castanos, D. Lieber, J. Moreira, S. Krishnamoorthy, V. Tipparaju, and J. 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.

[7] E. Bohm, G. J. Martyna, A. Bhatele, S. Kumar, L. V. Kale, J. A. Gunnels, and M. 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 (to appear)*, 52(1/2), 2007.

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

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

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

[11] K. Davis, A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, S. Pakin, and F. 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.

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

[13] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 210–221. ACM Press, 1988.

[14] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, 151:283–312, 1999.

[15] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng. Programming Petascale Applications with Charm++ and AMPI. In D. Bader, editor, *Petascale Computing: Algorithms and Applications*. Chapman & Hall / CRC Press, 2008.

[16] L. V. Kale and S. Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In G. V. Wilson and P. Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[17] L. V. Kale, G. Zheng, C. W. Lee, and S. 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.

[18] S. Kumar, C. Huang, G. Zheng, E. Bohm, A. Bhatele, J. C. Phillips, H. Yu, and L. V. Kalé. Scalable Molecular Dynamics with NAMD on Blue Gene/L. *IBM Journal of Research and Development: Applications of Massively Parallel Systems (to appear)*, 52(1/2), 2007.

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

[20] 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.

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

[22] K. Schulten, J. C. Phillips, L. V. Kale, and A. 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.

[23] B. E. Smith and B. Bode. Performance Effects of Node Mappings on the IBM Blue Gene/L Machine. In *Euro-Par*, pages 1005–1013, 2005.

[24] J. S. Vetter, S. R. Alam, T. H. D. Jr., M. R. Fahey, P. C. Roth, and P. H. Worley. Early evaluation of the cray xt3. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

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