A Case Study of Communication Optimizations on 3D Mesh Interconnects

Abhinav Bhatelé, Eric Bohm, and Laxmikant V. Kalé

Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{bhatele, ebohm, kale}@illinois.edu

Abstract. Optimal network performance is critical to efficient parallel scaling for communication-bound applications on large machines. With wormhole routing, **no-load** latencies do not increase significantly with number of hops traveled. Yet, we, and others have recently shown that in presence of **contention**, message latencies can grow substantially large. Hence task mapping strategies should take the topology of the machine into account on large machines. In this paper, we present the utility of object-based decomposition to alienate mapping from decomposition. Exploiting the flexibility of mapping objects and the topology of the allocated job partition, we present mapping strategies for a production code, OpenAtom to improve overall performance and scaling. OpenAtom presents complex communication scenarios of interaction between multiple groups of objects. Results are presented for OpenAtom on up to 16,384 processors of Blue Gene/L, 8,192 processors of Blue Gene/P and 2,048 processors of Cray XT3.

1 Introduction

A significant number of the largest supercomputers in use today, including IBM's Blue Gene family and Cray's XT family, employ a 3D mesh or torus topology. With tens of thousands of nodes, messages may have to travel many tens of hops before arriving at their destinations. With the advances in communication technology, especially wormhole routing, it was observed that the *latency* of communication was almost unaffected by the number of hops traveled by a message [1, 2]. However, the fraction of *bandwidth* occupied by the message is proportional to the number of hops (links) traversed. Increased contention for bandwidth might result in longer latencies. Consider a computation in which each processor sends one message to some other processor, in a data permutation pattern (so each processor also receives one message). For a thousand nodes organized in a $10 \times 10 \times 10$ 3D torus, the average number of hops traveled by a random message (i.e. the average inter-node distance) is 7.5. In a torus, a message needs to travel at most 5 hops in each dimension, leading to an average of 2.5 per dimension. So, if we organize our computation so that each message travels only one hop, we will use 7.5 times smaller global bandwidth than in the case of a random communication pattern. This was not as significant an issue when

Table 1. Execution time per step of OpenAtom on Blue Gene/L (CO mode) without topology aware mapping (System: WATER_32M_70Ry)

Cores	512	1024	2048	4096	8192
Time (secs)	0.274	0.189	0.219	0.167	0.129

the number of nodes was relatively small and the processors were slower [1]. But for today's large machines with faster processors, the issue becomes much more significant: on a machine with $64,000$ nodes organized in a $40 \times 40 \times 40$ 3D torus, the average inter-node distance is 30, and that is the ratio of bandwidth used by the two communication patterns. Further, with faster processors, the need for delivered bandwidth is higher. As the communication starts to occupy a large fraction of the available bandwidth, the contention in the network increases and message delivery gets delayed.

In this context, it is important to map computation to the processors to not just minimize the overall communication volume, but also the average number of hops traveled by the bytes communicated. Even though the utility of doing this may be apparent to programmers, the significance of the impact is probably more than most programmers expect. Our methodology builds upon object-based decomposition used in Charm++ [3] and related programming models, including Adaptive MPI (AMPI) [4]. This separates the processes of decomposition from mapping of computation to processors and allows a more flexible mapping based on communication patterns between objects.

In this paper, we first present the abstraction of object-based decomposition in Charm++ and an API which provides a uniform interface for obtaining topology information at runtime on four different machines – Cray XT3, Cray XT4, IBM Blue Gene/L and IBM Blue Gene/P. This API can be used by user-level codes for task mapping and is independent of the programming model being used (MPI, OpenMP, Charm++ or something else). We then demonstrate topology aware mapping techniques for a communication intensive application: OpenAtom, a production Car-Parrinello *ab initio* Molecular Dynamics (CPAIMD) code used by scientists to study properties of materials and nano-scale molecular structures for biomimetic engineering, molecular electronics, surface catalysis, and many other areas [5–7].

Initial scaling studies of OpenAtom on Blue Gene/L uncovered inadequate parallel efficiency for a small system with 32 atoms (Table 1). Further analysis isolated the cause to poor communication performance, which naturally led us to consider the network topology. We consider 3D torus topologies in this paper but not irregular networks or flat topologies. For logarithmic topologies (such as fat trees), the need to pay attention to topology may be smaller because the maximum number of hops between nodes tends to be small. Also, there is no

[1] This fact had made early work on topology aware mapping obsolete. But with large machines, bandwidth effects have again become important.

support for user level derivation of topology for most fat-tree networks so any implementation would be specific to an individual cluster layout.

2 Charm++ Arrays: A Useful Abstraction for Mapping

Parallelizing an application consists of two tasks: 1. decomposition of the problem into a large number of sub-problems to facilitate efficient parallelization to thousands of processors, 2. mapping of these sub-problems to physical processors to ensure load balance and minimum communication. Object-based decomposition separates the two tasks and gives independent control over both of them. In this paper, we use the CHARM++ runtime which allows the application developer to decompose the problem into objects and the CHARM++ runtime does a default mapping of objects to processors.

The basic unit of computation in CHARM++ is called a *chare* (simply referred to as an "object" in this paper) which can be invoked through remote function calls. The application developer decomposes the problem into chares or objects and the CHARM++ runtime does a default mapping of objects to processors. Each processor can have multiple objects which facilitates overlap of computation and communication. This default mapping does not have any information about the topology of the machine. The user can override the default mapping with more intelligent schemes that take the topology of the machine into account.

2.1 Topology Manager API: Runtime Information

Mapping of communication 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 should hide machine-specific details from the application. The Topology Manager API in CHARM++ provides a uniform interface to the application developer and hence the application just knows that the job partition 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).

On Blue Gene/L and Blue Gene/P [8], topology information is available through system calls to the "BGLPersonality" and "BGPPersonality" data structures, respectively. It is useful to use the API instead of expensive system calls throughout the execution. 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.

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

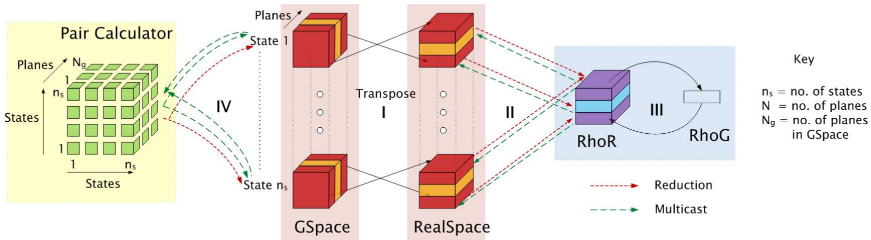

Fig. 1. Decomposition of the physical system into chare arrays (only important ones shown for simplicity) in OpenAtom

"catamount/cnos_mpi_os.h" and `PMI_Portals_get_nidpid_map` available from "pmi.h". 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).

3 OpenAtom: A Case Study

An accurate understanding of phenomena occurring at the quantum scale can be achieved by considering a model representing the electronic structure of the atoms involved. The CPAIMD method [9] is one such algorithm which has been widely used to study systems containing $10-10^3$ atoms. To achieve a fine-grained parallelization of CPAIMD, computation in OpenAtom [10] is divided into a large number of objects, enabling scaling to tens of thousands of processors. We will look at the parallel implementation of OpenAtom, explain the communication involved and then analyze the benefit from topology aware mapping of its objects.

In an *ab initio* approach, the system is driven by electrostatic interactions between the nuclei and electrons. 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].

3.1 Communication Dependencies

The ten phases referred to in the previous section are parallelized by decomposing the physical system into fifteen *chare arrays* of different dimensions (ranging between one and four). A simplified description of five of these arrays (those most relevant to the mapping) follows:

1. **GSpace and RealSpace:** These represent the g-space and real-space representations of each of the electronic states [9]. Each electronic state is represented by a 3D array of "complex numbers". OpenAtom decomposes this data into a 2D *chare array* of objects. Each object holds a plane of one of the states (see Figure 1) . The *chare arrays* are represented by $G(s, p)$ $[n_s \times N_g]$ and $R(s, p)$ $[n_s \times N]$ respectively. GSpace and RealSpace interact through transpose operations (as part of a Fast Fourier Transform) in Phase I and hence all planes of one state of GSpace interact with all planes of the same state of RealSpace.
2. **RhoG and RhoR:** They are the g-space and real-space representations of electron density and are decomposed into 1D and 2D *chare arrays* respectively. They are represented as $G_\rho(p)$ and $R_\rho(p, p')$. RealSpace interacts with RhoR through reductions in Phase II. RhoG is obtained from RhoR in Phase III through two transposes.
3. **PairCalculators:** These 3D *chare arrays* are used in phase IV. They communicate with GSpace through multicasts and reductions. They are represented as $P_c(s, s', p)$ $[n_s \times n_s \times N_g]$. All elements of the GSpace array with a given state index interact with all elements of the PairCalculator array with the same state in one of their first two dimensions.

3.2 Mapping

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. OpenAtom has a default mapping scheme, but it should be noted that the default mapping is far from random. It is the mapping scheme used on standard fat-tree networks, wherein objects which communicate frequently are co-located on processors within the constraints of even distribution. This reduces the total communication volume. It only lacks a model for considering the relative distance between processors in its mapping considerations. We can do better than the default mapping by using the communication and topology information at runtime. We now describe how a complex interplay (of communication dependencies) between five of the *chare arrays* is handled by our mapping scheme.

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, but if all states of GSpace are placed together, the multicasts/reductions are favored. To strike a balance

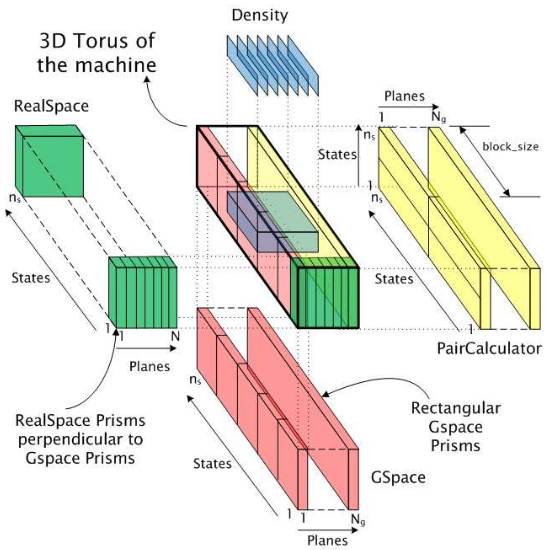

Fig. 2. Mapping of different arrays to the 3D torus of the machine

between the two extremes, a hybrid map is built, where a subset of planes and states of these three arrays are 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 2 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 by including 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 of Density Arrays: RhoR objects communicate with RealSpace plane-wise and hence $R_\rho(p, *)$ have to be placed close to $R(*, p)$. To achieve this, we start with the centroid of the prism used by $R(*, p)$ and place RhoR objects in proximity to it. RhoG objects, $G_\rho(p)$ are mapped near RhoR objects, $R_\rho(p, *)$ but not on the same processors as RhoR to maximize overlap. The density computation is inherently smaller and hence occupies the center of the torus (box

Table 2. Execution time per step (in secs) of OpenAtom on Blue Gene/L (CO mode)

	WATER_32M_70Ry		**WATER_256M_70Ry**		**GST_BIG**	
Cores	**Default**	**Topology**	**Default**	**Topology**	**Default**	**Topology**
512	0.274	0.259	-	-	-	-
1024	0.189	0.150	19.10	16.4	10.12	8.83
2048	0.219	0.112	13.88	8.14	7.14	6.18
4096	0.167	0.082	9.13	4.83	5.38	3.35
8192	0.129	0.063	4.83	2.75	3.13	1.89
16384	-	-	3.40	1.71	1.82	1.20

Table 3. Execution time per step (in secs) of OpenAtom on Blue Gene/P (VN mode)

	WATER_32M_70Ry		**WATER_256M_70Ry**	
Cores	**Default**	**Topology**	**Default**	**Topology**
256	0.395	0.324	-	-
512	0.248	0.205	-	-
1024	0.188	0.127	10.78	6.70
2048	0.129	0.095	6.85	3.77
4096	0.114	0.067	4.21	2.17
8192	-	-	3.52	1.77

on the top in Figure 2).

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 $G(s1, p), ..., G(s1 + block\_size, p)$ and $G(s2, p), ..., G(s2 + block\_size, p)$. 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 2) 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 avoid scaling bottlenecks as we will see in Section 3.3. These maps perform no better (and generally slightly worse) than the default maps on architectures which have more uniform network performance, such as Ethernet or Infiniband.

Although maps are created only once during application start-up, they must still be efficient in terms of their space and time requirements. The memory cost of these maps grows linearly (3 integers per object) with the number of objects, which is a few megabytes in the largest system studied. The runtime cost of creating the most complex of these maps is $O(pn^2 log(n))$ where n is the number

Table 4. Performance (time per step in secs) of OpenAtom on XT3.

	WATER_32M_70Ry		WATER_256M_70Ry		GST_BIG	
Cores	**Default**	**Topology**	**Default**	**Topology**	**Default**	**Topology**
			Single core per node			
512	0.124	0.123	5.90	5.37	4.82	3.86
1024	0.095	0.078	4.08	3.24	2.49	2.02
			Two cores per node			
256	0.226	0.196	-	-	-	-
512	0.179	0.161	7.50	6.58	6.28	5.06
1024	0.144	0.114	5.70	4.14	3.51	2.76
2048	0.135	0.095	3.94	2.43	2.90	2.31

of objects and p the number of processors. Despite this complexity, this time is sufficiently small that generating the maps for even the largest systems requires only a few minutes. As an optimization, once created, maps can be stored and reloaded in subsequent runs to minimize restart time.

3.3 Comparative Analysis of OpenAtom

To analyze the effects of topology aware mapping in a production science code we studied the strong scaling (fixed problem size) performance of OpenAtom with and without topology aware mapping. Two benchmarks commonly used in the CPMD community: the minimization of WATER_32M_70Ry and WATER_256M_70Ry were used. The benchmarks simulate the electronic structure of 32 molecules and 256 molecules of water, respectively, with a standard g-space spherical cutoff radius of $|\mathbf{g}|^2_{cut} = 70$ Rydberg (Ry) on the states. To illustrate that the performance improvements extend beyond benchmarks to production science systems, we also present results for GST_BIG, which is a system being studied by our collaborator, Dr Glenn J. Martyna. GST_BIG consists of 64 molecules of Germanium, 128 molecules of Antimony and 256 molecules of Tellurium at cutoff radius of $|\mathbf{g}|^2_{cut} = 20$ Ry on the states.

Blue Gene/L (IBM T. J. Watson) runs are done in co-processor (CO) mode to use a single core per node. Single core per node runs were chosen to highlight interconnect performance and to facilitate fair comparisons between the two machines. Blue Gene/P (Intrepid at ANL) runs were done in VN mode using all four cores per node. Cray XT3 (BigBen at PSC) runs are done in two modes: single core per node (SN) and two cores per node (VN).

As shown in Table 2, performance improvements from topology aware mapping for Blue Gene/L (BG/L) can be quite significant. As the number of cores and likewise, the diameter of the torus grows, the performance impact increases until it is a factor of two faster for WATER_32M_70Ry at 2048 and for WATER_256M_70Ry at 16384 cores. There is a maximum improvement of 40% for GST_BIG. The effect is not as strong in GST_BIG due to the fact that the

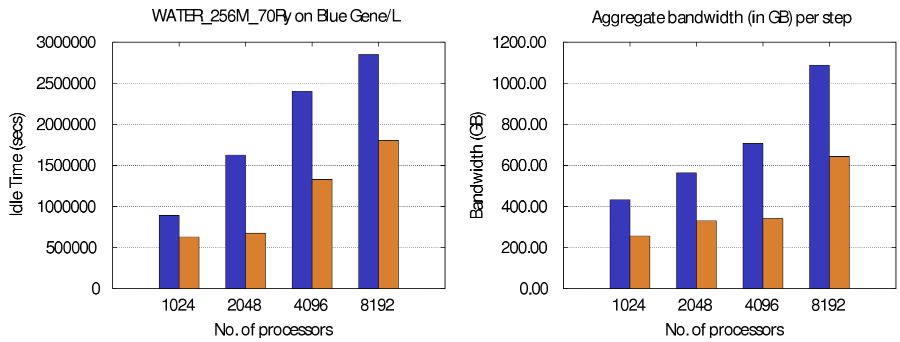

Fig. 3. Effect of topology aware mapping on latency and bandwidth in OpenAtom

time step in this system is dominated by a subset of the orthonormalization process which has not been optimized extensively, but even a 40% improvement represents a substantial improvement in time to solution.

Performance improvements on Blue Gene/P (Table 3) are similar to those observed on BG/L. The improvement for WATER_32M_70Ry is not as remarkable as on BG/L but for WATER_256M_70Ry, we see a factor of 2 improvement starting at 2048 cores. The absolute numbers on BG/P are much better than on BG/L partially because of the increase in processor speeds but more due to the better interconnect (higher bandwidth and DMA engine). The performance for WATER_256M_70Ry at 1024 cores is 2.5 times better on BG/P than on BG/L. This is when comparing the VN mode on BG/P to the CO mode on BG/L. If we use only one core per node on BG/P, the performance difference is even greater, but the higher core per node count, combined with the DMA engine and faster network make single core per node use less interesting on BG/P.

The improvements from topological awareness on Cray XT3, presented in Table 4 are comparable to those on BG/L and BG/P. The improvement of 20% and 18.8% on XT3 for WATER_256_70Ry and GST_BIG at 1024 cores is greater than the improvement of 14% and 12% respectively on BG/L at 1024 cores.

As discussed in prior sections, OpenAtom is highly communication bound. Although Charm++ facilitates the exploitation of the available overlap and latency tolerance across phases, the amount of latency tolerance inevitably drops as the computation grain size is decreased by the finer decomposition required for larger parallel runs. It is important to consider the reasons for these performance improvements in more detail. Figure 3 compares idle time as captured by the Projections profiling system in Charm++ for OpenAtom on BG/L for the default mapping, versus the topologically aware mapping. A processor is idle whenever it is waiting for messages to arrive. It is clear from Figure 3 that the factor of two speed increase from topology awareness is reflected directly in relative idle time and that the maximum speed increase which can be obtained from topology aware mapping is a reduction in the existing idle time.

It is illuminating to study the exact cause for this reduction in idle time. To that end, we ported IBM's libhpm High Performance Monitor [11] for Blue

Gene/P's Universal Performance Counters to Charm++, and enabled performance counters for a single time step in WATER_256M_70Ry in both topology aware and non-topology aware runs. We summed the per node torus counters (BGP_TORUS_*_32BCHUNKS), to produce the aggregate bandwidth consumed by one step across all nodes to obtain the results in Figure 3. It is clear from the figure, that topology aware mapping results in a significant reduction, by up to a factor of two, in the total bandwidth consumed by the application. This more efficient use of the network is directly responsible for the reduction in latency due to contention and decreased idle time.

4 Previous Work

There has been considerable research on the task mapping problem. It has been proven that the problem is NP-complete and computationally equivalent to the general graph embedding problem. Heuristic techniques like pairwise exchange were developed in the 80s by Bokhari [12] and Aggarwal [13]. These schemes, however, are not scalable when mapping millions of objects to thousands of processors. This problem has been handled by others using recursive partitioning [14] and graph contraction [15] by mapping sub-graphs to a subset of processors. Results in the 80s and 90s were not demonstrated on real machines and even when they were, they were targeted towards small sized machines. They also did not consider real applications. With the emergence of large parallel machines, we need to revisit these techniques and build upon them, on a foundation of real machines, in the context of real applications.

A lot of work until the 90s was focused on hypercube networks [16, 13]. The development of large parallel machines like Blue Gene/L, XT3, XT4 and Blue Gene/P has led to the re-emergence of mapping issues. Both application and system developers have evolved mapping techniques for Blue Gene/L [17–21]. Yu [21] and Smith [20] discuss embedding techniques for graphs onto the 3D torus of BG/L which can be used by the MPI Topology functions. Weisser et al. [22] present an analysis of topology aware job placement techniques for XT3. However, our work is a one of the first for task mapping on XT3. This work builds on our previous work [23] demonstrating the effectiveness of topology aware task mapping for a 3D Stencil kernel. It presents a case study for OpenAtom, a production quantum chemistry code, and demonstrates high returns using topology aware schemes. Our earlier publication on OpenAtom [10] demonstrates the effectiveness of such schemes on BG/L. In this paper, we present results for XT3, BG/L and BG/P for multiple systems including a non-benchmark simulation.

5 Conclusion and Future Work

In this paper we demonstrated that topology aware mapping can substantially improve performance for communication intensive applications on 3D torus networks. Significant improvements were shown for the OpenAtom code and the effectiveness of topology aware mapping was shown for both IBM Blue Gene and

Cray XT architectures. Mapping was facilitated by the object-based virtualization in CHARM++ and the availability of the Topology Manager API.

OPENATOM has complex but relatively regular or structured communication. We think that it may be possible to develop general methodologies that deal with such structured communication. Unstructured static communication patterns, as represented by unstructured-mesh computations might need somewhat different mapping techniques. Work in the future would be on the lines of an automatic mapping framework which can work in tandem with the topology manager interface. This would also require a study of a more diverse set of applications with different communication patterns. Further study will be given to characterizing network resource usage patterns with respect to those which are most affected by topology aware task mapping.

References

1. Ronald I. Greenberg and Hyeong-Cheol Oh. Universal wormhole routing. *IEEE Transactions on Parallel and Distributed Systems*, 08(3):254–262, 1997.
2. Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.
3. L.V. Kalé and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In A. Paepcke, editor, *Proceedings of OOPSLA'93*, pages 91–108. ACM Press, September 1993.
4. Milind Bhandarkar, L. V. Kale, Eric de Sturler, and Jay Hoeflinger. Object-Based Adaptive Load Balancing for MPI Programs. In *Proceedings of the International Conference on Computational Science, San Francisco, CA, LNCS 2074*, pages 108–117, May 2001.
5. Pasquarello A, Hybertsen MS, and Car R. Interface structure between silicon and its oxide by first-principles molecular dynamics. *Nature*, 396:58, 1998.
6. De Santis L and Carloni P. Serine proteases: An ab initio molecular dynamics study. *Proteins*, 37:611, 1999.
7. A. M. Saitta, P. D. Soper, E. Wasserman, and M. L. Klein. Influence of a knot on the strength of a polymer strand. *Nature*, 399:46, 1999.
8. IBM Blue Gene Team. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development*, 52(1/2), 2008.
9. M. E. Tuckerman. Ab initio molecular dynamics: Basic concepts, current trends and novel applications. *J. Phys. Condensed Matter*, 14:R1297, 2002.
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. V. Salapura, K. Ganesan, A. Gara, M. Gschwind, J.C. Sexton, and R.E. Walkup. Next-Generation Performance Counters: Towards Monitoring Over Thousand Concurrent Events. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 139 – 146, April 2008.
12. Shahid H. Bokhari. On the mapping problem. *IEEE Trans. Computers*, 30(3):207–214, 1981.

13. Soo-Young Lee and J. K. Aggarwal. A mapping strategy for parallel processing. *IEEE Trans. Computers*, 36(4), 1987.
14. 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*. ACM Press, 1988.
15. Francine Berman and Lawrence Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4(5):439–458, 1987.
16. S. Wayne Bollinger and Scott F. Midkiff. Processor and link assignment in multicomputers using simulated annealing. In *ICPP (1)*, pages 1–7, 1988.
17. 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.
18. Francois Gygi, Erik W. Draeger, Martin Schulz, Bronis R. De Supinski, John A. Gunnels, Vernon Austel, James C. Sexton, Franz Franchetti, Stefan Kral, Christoph Ueberhuber, and Juergen Lorenz. Large-Scale Electronic Structure Calculations of High-Z Metals on the Blue Gene/L Platform. In *Proceedings of the International Conference in Supercomputing*. ACM Press, 2006.
19. 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.
20. 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.
21. 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.
22. 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.
23. Abhinav Bhatelé and Laxmikant V. Kalé. Benefits of Topology Aware Mapping for Mesh Interconnects. *Parallel Processing Letters (Special issue on Large-Scale Parallel Processing)*, 18(4):549–566, 2008.