

OPTIMIZATION OF FFT COMMUNICATION ON 3-D
TORUS AND MESH SUPERCOMPUTER NETWORKS

BY

ANSHU ARYA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Adviser:

Professor Laxmikant V. Kalé

ABSTRACT

The fast Fourier transform (FFT) is of intense interest to the scientific community. Its utility in a vast range of parallel scientific simulations warrants investigating its efficiency on a leading class of supercomputers that employ 3-D torus and mesh networks. Given the divide-and-conquer approach of the popular Cooley-Tukey approach, a parallel FFT can use a highly optimized serial algorithm for intra-node calculations, but the communication patterns between nodes reveal a potential for improvement.

In this work, the scalability and communication patterns of global 1-D and 3-D parallel FFTs and how they map to 3-D torus and mesh networks are investigated. As a starting point, a naive implementation of a 1-D FFT is modified and scaled to 65,536 cores of a BlueGene/P machine. Insights into the machine architecture and network limits gained from the 1-D FFT are then applied to improve the performance of a 3-D FFT. Default mappings of a 3-D FFT onto a torus result in low network utilization and sub-optimal communication costs. The communication graph of a 3-D FFT is formulated and graph partitioning techniques are used to find a new mapping that greatly improves performance. Data migration techniques are then used to alleviate possible link contention associated with the new mappings. Migration techniques are discussed and modeled. It is discovered that specific migration patterns are able to reduce link contention without increasing the overall data movement (hop-bytes). With both mapping and migration techniques combined, a decrease of nearly 20% in the running time compared to the default 3-D FFT mapping is observed on 512 nodes, 2048 cores of a BlueGene/P machine.

Table of Contents

Chapter 1 Introduction	1
1.1 1-D FFT as a Benchmark	2
1.2 3-D Volumetric FFT for Science	2
1.3 Binary Exchange versus Transpose	3
Chapter 2 Parallel 1-D FFT	4
2.1 Communication and Computation Phases	4
2.2 Implementation	4
2.3 Results	5
Chapter 3 Parallel 3-D Volumetric FFT	9
3.1 Communication and Computation Phases	9
3.2 Mapping	9
3.3 Graph Partitioning	11
3.4 Migration	13
3.5 Results	15
Chapter 4 Conclusions and Future Work	21
4.1 Summary	21
4.2 Future Work	22
References	24

Chapter 1

Introduction

Since its inception, the fast Fourier transform (FFT) algorithm has become the predominant method for computing the discrete Fourier transform of N complex points. The method became prevalent after the seminal work done by Cooley and Tukey in 1965 and has since been applied to virtually every domain where the Fourier transform is required [1]. Their divide-and-conquer algorithm reduced the time complexity from a naive implementation of $O(N^2)$ to $O(N\log(N))$ for serial computation, and enabled a new generation and scale of science simulations.

The variety of applications and an in-depth discussion of the serial algorithm are not warranted here, as this work focuses on the parallelization of the FFT, and its performance on modern-day supercomputers with torus and mesh network topologies. Given that the serial implementation remains the same for parallel and multi-dimensional FFTs, the results here focus primarily on communication patterns, network features, and other scalability concerns.

The scope of this work is limited to supercomputing networks with torus and mesh topologies because such computers represent a leading class of general-purpose machines available to the academic and scientific community. At the time of this work, over half of the top ten ranked supercomputers in the world use a torus or mesh network [2]. It is

assumed the reader already understands the connectivity of multi-dimensional mesh and torus networks. Specifically excluded from the discussion are computers with a butterfly or hypercube network for targeted FFT performance.

1.1 1-D FFT as a Benchmark

Due to the prevalence of the FFT in large scale science simulations, such as molecular dynamics, quantum chemistry, and plasma-laser interactions, the FFT has become a common benchmark for supercomputing centers [3, 4, 5]. A parallel, global 1-D FFT has been included as a workload in the High Performance Computing Challenge (HPCC) benchmark suite since the benchmarks were introduced, and results from supercomputers around the world are made available on an annual basis. The systems benchmarked vary from modest clusters of a few hundred CPU cores, up to machines with over half a million cores. The design and optimization of a global 1-D FFT meeting HPCC specifications was a starting point for this work [6].

1.2 3-D Volumetric FFT for Science

Although a parallel 1-D FFT is a widely used benchmark and useful for analysis of FFT performance, 3-D volumetric FFTs are a typical use case for molecular and particle interaction simulations [3, 4]. Higher dimensional FFTs add computation phases with interesting communication patterns, and expose another vector for optimization that can significantly impact the performance of the algorithm. The specific mapping and decomposition of the data onto a torus network becomes vital to the efficiency of the algorithm. At the scale of thousands of cores, previous work has shown that a variety algorithms benefit from targeted mapping schemes, and with a high communication volume, FFT is a prime candidate for a similar analysis [7, 8, 9, 10, 11].

1.3 Binary Exchange versus Transpose

A parallel FFT may be performed using two different communication patterns, a binary exchange or a matrix transpose. The data layout and efficiency of each vary considerably. Numerous theoretical studies have been performed describing and comparing binary exchange and transpose algorithms for parallel FFT so the descriptions will be omitted here. Most notably in the work by Gupta and Kumar it was concluded that machine parameters and problem size dictate if a binary exchange or transpose algorithm will be more efficient on a hypercube network [12]. However, as the problem size increases for a given computer, a 2-D transpose algorithm scales better than binary exchange, even on a hypercube network where binary exchange has a natural mapping [13]. In light of that, many scientific codes and reference implementations of parallel FFTs focus on the transpose algorithm [14, 15, 16]. For similar reasons, this study, too, focuses on a 2-D transpose algorithm to achieve scalability. Since a transpose operation maps directly to an all-to-all communication pattern, the terms are used interchangeably.

Chapter 2

Parallel 1-D FFT

2.1 Communication and Computation Phases

For parallel computation, data is split evenly across P processors. Processors are numbered linearly and for N^2 input points, each processor holds N^2/P consecutive points. The computation of a parallel 1-D FFT using a transpose algorithm requires data to be logically laid out in a column-major 2-D matrix and four steps: (1) serial FFT on columns of matrix, (2) multiplication by twiddle factors, (3) matrix transpose, (4) serial FFT on columns of matrix. A N^2 size 1-D FFT with a data layout of an $N \times N$ matrix results in N serial FFTs of size N in steps (1) and (4).

2.2 Implementation

A parallel 1-D FFT was implemented using Charm++, a parallel communication library. Specific features of Charm++, such as its domain-specific language SDAG, made the implementation short and efficient [17, 18]. Because Charm++ is written in C++, the implementation naturally ordered the input in a row-major fashion. Therefore, additional transposes at the beginning and end of the program were necessary to compute the complete FFT and

unscramble the data. In total, three transposes, two serial FFT phases, and a twiddle factor computation are included in all timings and performance data. Each phase and the data layout is shown in figure 2.1. For the purposes of this study, any FFT computed within a multicore node and that did not perform communication over the network, was treated as a self-contained serial FFT. The IBM ESSL library was used for a fast serial or multithreaded FFT for intra-node calculations. This implementation is competitive in performance to implementations with other parallel libraries submitted to the HPCC Class II competition as evidenced by it winning 1st place in 2011 and becoming a finalist in 2012 [19, 20].

Two versions of the code were written. First, a naive implementation that uses point-to-point communication to perform the transpose operation. This results in P^2 messages flooding the network as each processor communicates with every other in an all-to-all pattern at each transpose phase. Then, a version that leveraged the Charm++ software routing Mesh-Streamer library, which combines and moves messages through each dimension of the network in stages. This results in fewer, larger messages and a predetermined software routing of all data.

2.3 Results

Results summarized in figure 2.2 and table 2.1 were collected on the Intrepid IBM BlueGene/P (BG/P) machine with a 3-D torus interconnect at Argonne National Lab. Initially the algorithm scaled well to 2048 cores of the machine. This corresponds to a midplane of 512 nodes configured in an 8x8x8 torus. A midplane is the smallest unit on a BlueGene/P that provides a full torus network topology in each dimension. Below or above one midplane, one or more of the network dimensions may only be a mesh and the network topology will generally not result in a cube (e.g. 8x8x16 for two midplanes of 4096 cores) [21]. Therefore, the drop-off in scalability was attributed to inefficient routing of the point-to-point messages and contention during the all-to-all transpose steps, especially over elongated dimensions.

Input Size	Cores	GFlops
32768 ²	256	19.4
46080 ²	512	39.9
65536 ²	1024	69.1
92160 ²	2048	229.2
131072 ²	4096	284.1
180224 ²	8192	454.4
262144 ²	16384	867.9
360448 ²	32768	1430.8
524288 ²	65536	2708.9

Table 2.1: Problem sizes and performance results for 1-D FFT on BlueGene/P using Mesh-Streamer and an input size that filled 25% of machine memory, as per HPCC FFT benchmark specifications.

To overcome this barrier, a second version of the code which employed software routing was used. The software routing algorithm in the Mesh-Streamer extension of Charm++ was configured to aggregate messages, and enforce dimension-order routing. The routing option forces messages to move in the X, then Y, then Z dimensions sequentially. Messages with the same target processor that meet at an intermediate destination along the path are combined, thus reducing the overall message load on the network and reducing network link contention. The software routed version eliminated the scalability barrier hit at the midplane and allowed increasing performance on up to 65536 cores of the machine. Equally significant is that below the midplane limit, software routing had no major impact on performance, suggesting the network was sufficiently able to handle the traffic of P^2 point-to-point messages for $P < 2048$ on this particular machine. The peak at the midplane point of 2048 processors simply shows the advantage of the fully symmetric torus network configuration, which effectively cuts maximal message travel distance by half the dimension length and is the same length in each dimension.

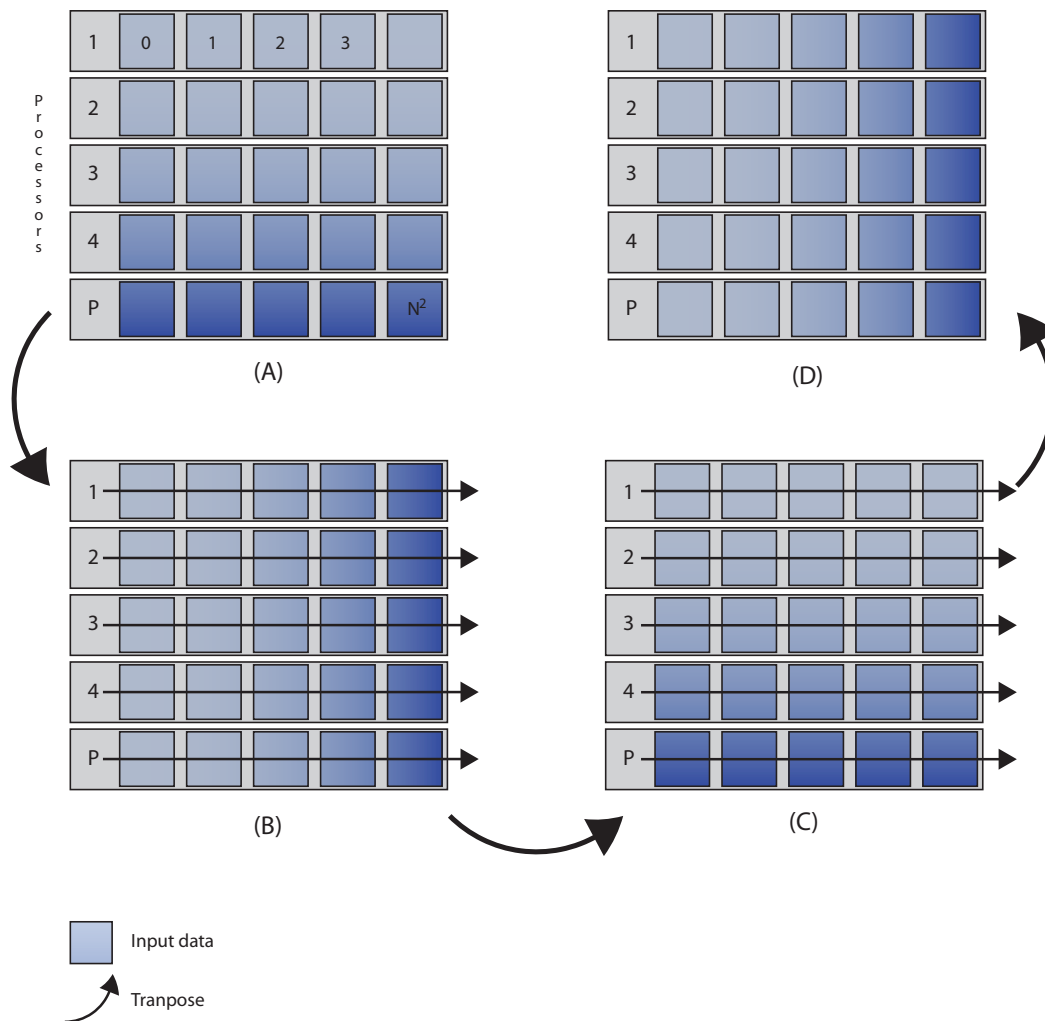


Figure 2.1: 1D FFT: (A) Data layout in row major format spread evenly amongst P processors. (B) After a transpose, a serial FFT is performed on each processor along the columns of the matrix. Twiddle factors are also calculated. (C) Another transpose and serial FFT. (D) After a final transpose the data is unscrambled and in the correct location.

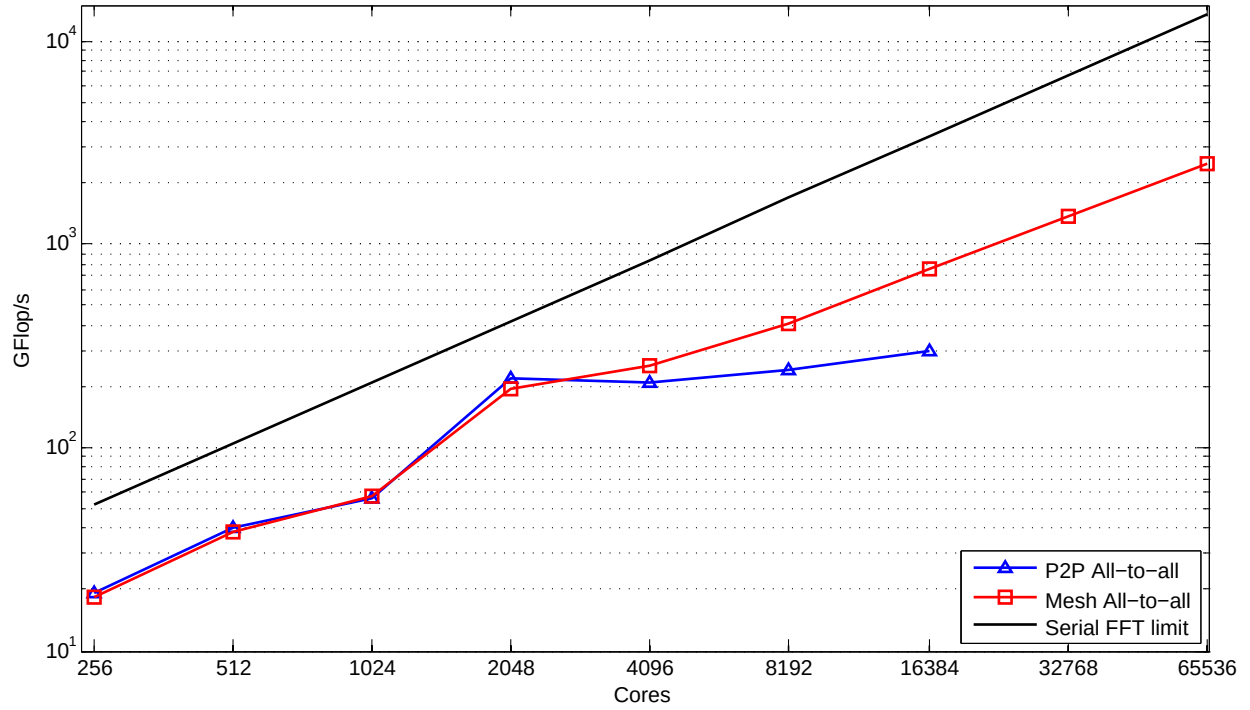


Figure 2.2: Results for global parallel 1-D FFT on a BlueGene/P. P2P All-to-all represents a naive implementation using point-to-point messages for the transpose. Mesh All-to-all represents an implementation using the Mesh-Streamer library in Charm++. Serial FFT limit is the theoretical peak performance of a serial FFT on the machine.

Chapter 3

Parallel 3-D Volumetric FFT

3.1 Communication and Computation Phases

Similar to the 1-D case, in a 3-D FFT, an N^3 point FFT splits data evenly amongst P processors, resulting in each processor storing N^3/P data. In order to perform a 2-D transpose algorithm, the processors are assigned a 2-D coordinate based on their location in a real or logical 3-D mesh/torus network and store a volume of the 3-D input data based on their coordinates. The data is logically laid out in a 2-D matrix with the third dimension flattened. Ignoring twiddle factor multiplications, which are a purely intra-node serial computation, a 3-D FFT decomposed into a 2-D transpose algorithm may be generalized into five steps: (1) serial FFT in the Z dimension, (2) transpose in the X dimension, (3) serial FFT in the X dimension, (4) transpose in the Y dimension, (5) serial FFT in the Y dimension.

3.2 Mapping

Multi-dimensional FFTs present an opportunity to try various mapping configurations, however the target machine topology can limit the options. Mapping a 2-D transpose to a BG/P midplane with an 8x8x8 configuration of nodes in the network gives rise to a configuration of

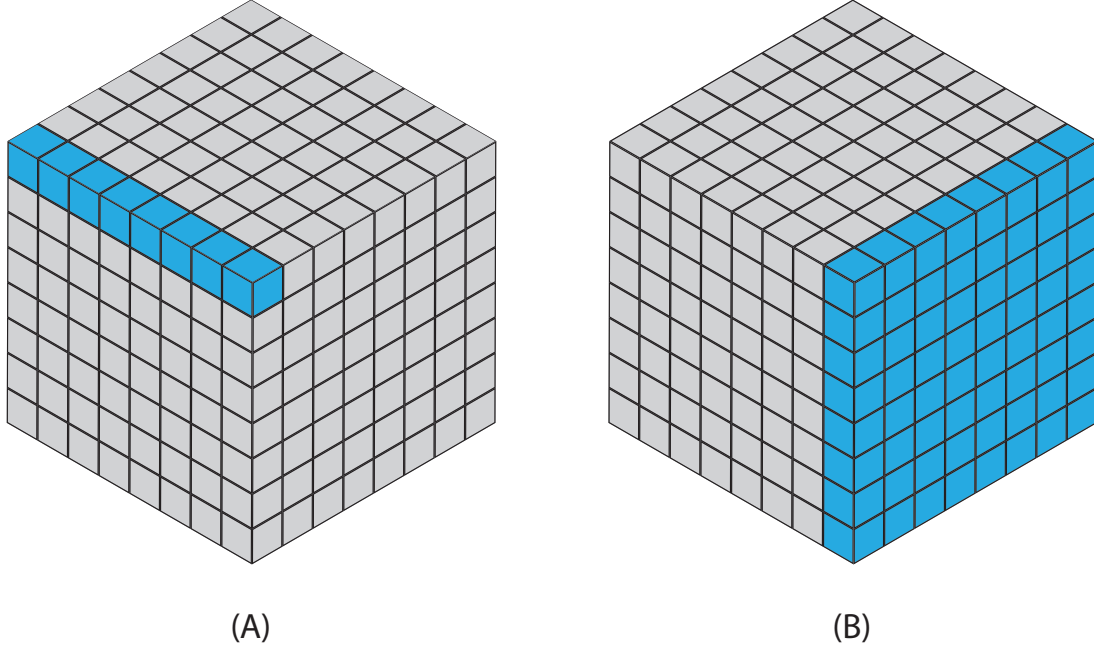


Figure 3.1: Two all-to-all phases in a 3-D FFT. These shapes result from mapping a 2-D transpose of size 8×64 to an $8 \times 8 \times 8$ torus. (A) 1×8 pencils of the first transpose. (B) 8×8 planes of the second transpose.

8×64 by simply collapsing one dimension. By default, this results in the transposes of steps (2) and (4) to occur in a 1×8 pencils and a 8×8 planes, respectively. Figure 3.1 illustrates the pencils and planes. Note that in step (2) sixty-four simultaneous 1×8 pencil all-to-all operations will occur, and in step (4) eight simultaneous 8×8 plane all-to-all operations.

Intuitively the 1×8 and 8×8 shapes provide simple axis and planes for the all-to-all operation performed during the transpose. However, the network links are underutilized. For example, when each 1×8 pencil is communicating along a single dimension, say X, the Y and Z links between nodes will be under-utilized. Similarly, when each 8×8 plane is communicating along two dimensions, one entire dimension of links in the network will be ignored. The question then arises: is there a mapping that will better utilize the network?

3.3 Graph Partitioning

Finding quality mappings is not always intuitive. To investigate the possibility of more efficient mappings, the problem was formulated as a communication graph and partitioned using the recursive bi-partitioning methods available in SCOTCH, a popular graph partitioning library [22].

In general a graph partitioner divides a graph $G(v, e)$ into k parts such that the number of vertices, v , in each part are approximately equal and the number of edges, e , that span distinct parts is minimized. Extending the idea to graphs with weighted vertices and edges, a partitioner will attempt to equalize the weight of the vertices and the summed weight of the edges between each distinct part. Although it is an NP-complete problem, a number of quality heuristics are implemented in modern partitioners.

The recursive bi-partitioning technique in SCOTCH simultaneously partitions an input graph and a target graph, and outputs a mapping between them. This additional optimization feature also has a target function that attempts to minimize the cost,

$$c = \sum_{e \in E} w_e d_e$$

where e is an edge, w_e is the weight of the edge, and d_e is the dilation of the edge. The dilation of an edge represents the number of “hops” or intermediate links/edges a message must traverse before reaching its destination. The cost function is equivalent to the commonly used hop-bytes metric [23].

The input graph partitioned by the recursive bi-partitioning method was the communication graph of a 3-D FFT decomposed into a 8x64 2-D problem, and the target graph was the symmetric 8x8x8 torus of a BlueGene/P midplane. For the input graph, each vertex was given an equivalent weight, representing an equivalent amount of computation performed by each processor, and multiple edges for the all-to-all operations, weighted in accordance

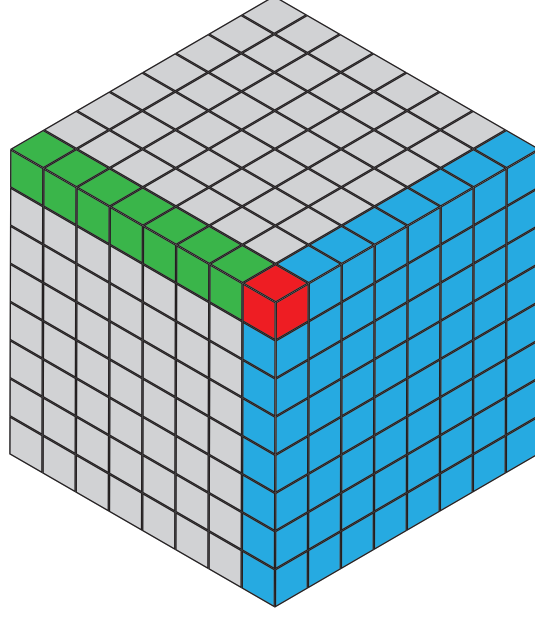


Figure 3.2: A representation of the communication graph of the 3-D FFT decomposed into a 2-D transpose used as the input graph into SCOTCH. The red vertex has edges to 7 green vertices with weight $N^3/P/8$ and edges to 63 blue vertices with weight $N^3/P/64$ where N^3 is the size of the input and P is the number of processors or vertices.

with the amount of data sent per edge. For the first transpose along a 1×8 pencil, each processor sends $N^3/P/8$ data points to 7 other processors. For the second transpose across an 8×8 plane, each processors sends $N^3/P/64$ data points to 63 other processors. Figure 3.2 illustrates the vertex and edges of the input graph.

The resultant mapping is shown in figure 3.3. The 1×8 pencils are folded into $2 \times 2 \times 2$ blocks. Consequently, the 8×64 planes become interleaved but span an entire $7 \times 7 \times 7$ cube. Further investigation reveals that spanning the edge with a weight of $N^3/P/8$ across a 1×8 pencil has a severe impact on the hop-bytes, so the partitioner wraps the pencils into tight units at the cost of interleaving and spreading the 8×64 planes. Note that the graph partitioning heuristics are not designed to increase link utilization. The poor link utilization the original pencils and planes had was not sufficient for the partitioner to fold the pencils and interleave the planes; however the problem appears to be alleviated by the new mapping. This mapping is analogous to the mapping found by Jagode and Hein, but was found

automatically [24].

3.4 Migration

Having arrived at a mapping that improved performance by shrinking one all-to-all communication phase into a tight cube while spreading another out across the entire set of processors raised another question: is it worth migrating data between the all-to-all phases to shrink the large interleaved all-to-all into a tight communication pattern?

Consider a 1-D interleaved all-to-all, as shown in figure 3.4. Each processor performs an exchange with another of the same color. In this scenario, each message has a hop-byte of three, and all six messages in the system must use the bi-directional link between processors 2 and 3. Similarly the links between processors 1-2 and 3-4 are used by multiple messages. This causes contention on those links as multiple messages compete for resources. The maximum link contention in this scenario is three, as three messages traverse the center link in each direction.

In figure 3.5, the first step migrates data between processors with larger messages; but now there is no link contention since they are bi-directional links. The second step then performs the all-to-all with new neighbors. Again, no link contention. Furthermore, the sum of the migrate and exchange procedure results in the same hop-bytes as the interleaved all-to-all in figure 3.4. The contention increases with the number of interleaved all-to-alls, so the benefit from migration may improve with problem size. It is also clear the migration step can always be performed contention-free along each dimension.

Extending the idea into three dimensions, the cost of performing an all-to-all operation on a small 2x2x2 cube as we increase its sparsity, s , as shown in figure 3.6, can be modeled with the following equations:

(1) hops in a 2x2x2 all-to-all

$$hops_{a2a(s)} = [(s + 1) * 3 + (s + 1) * 2 * 3 + (s + 1) * 3] * 8$$

(2) hops to migrate a sparse 2x2x2 into a tight block

$$hops_m = s * 3 * 4$$

(3) total hops to migrate a sparse 2x2x2 block and then perform a tight all-to-all

$$hops_{m+a2a(s=0)} = hops_m + hops_{a2a(0)} = s * 12 + 12 * 8$$

(4) hop-bytes for 2x2x2 all-to-all

$$hopbytes_{a2a(s)} = hops_{a2a(s)} * bytes_{msg}$$

(5) hop-bytes to migrate sparse 2x2x2 block and then perform tight all-to-all

$$hopbytes_{m+a2a(s=0)} = hops_m * bytes_{msg} * 8 + hopbytes_{a2a(0)}$$

where $s = 0$ represents the case of a tight all-to-all with adjacent neighbors and $bytes_{msg}$ is the bytes per message of the transpose and is determined by the input size. These equations show that the ratio of hops required for an interleaved all-to-all versus a migration with a tight all-to-all is always greater than or equal to one, i.e. $hops_{a2a(s)} \geq hops_{m+a2a(s=0)}$ for $s \geq 0$. More importantly the hop-bytes measurement remains equivalent:

$$\begin{aligned} \frac{hopbytes_{a2a(s)}}{hopbytes_{m+a2a(s=0)}} &= \frac{hops_{a2a(s)} * bytes_{msg}}{hops_m * bytes_{msg} * 8 + hopbytes_{a2a(0)}} \\ &= \frac{hops_{a2a(s)} * bytes_{msg}}{hops_m * bytes_{msg} * 8 + hops_{a2a(0)} * bytes_{msg}} \\ &= \frac{hops_{a2a(s)}}{hops_m * 8 + hops_{a2a(0)}} = \frac{[(s+1) * 3 + (s+1) * 2 * 3 + (s+1) * 3] * 8}{s * 3 * 4 * 8 + 12 * 8} \end{aligned}$$

$$= \frac{(s+1) * 12 * 8}{s * 96 + 12 * 8} = \frac{s * 96 + 96}{s * 96 + 96} = 1$$

The equations show that performing a sparse interleaved all-to-all will result in more hops than a migration followed by a tight interleave, but the overall hop-bytes for both strategies is equivalent. This suggests that if the bandwidth and latency cost of the migration is less than the effect of link contention from the interleaved transposes, migration can result in improved performance. Migrations can be done with minimal contention by performing a dimensional collapse as shown in figure 3.7.

3.5 Results

Results are summarized in figure 3.8 and were collected on the Surveyor IBM BlueGene/P machine with a 3-D torus interconnect at Argonne National Lab. Results were limited to a midplane of 512 nodes configured in an 8x8x8 torus to avoid the effects of elongated dimensions, to avoid communication inefficiencies that were revealed in the 1-D FFT case when scaled past a midplane, and to run within resource constraints.

The baseline values represent the default mapping of a 3-D FFT using an 8x64 2-D decomposition. The improvement from the mapping derived from graph partitioning techniques is significant, ranging from 4-18% speedup depending on problem size and number of cores used per node. Gains from mapping increase as the core count per node increases and as the problem size decreases. This trend can be attributed to the amount of communication versus computation. As more processors are used for the same problem size, the relative gain is higher because serial computation is faster and communication represents a larger portion of the runtime. Similarly, as the problem size decreases, less serial work is performed while the number of messages sent remains the same, shifting more time spent into communication.

The improvements from migration are not so clear or significant. The migration tech-

nique can only speedup the second all-to-all, limiting the scope of its capabilities. It sends large messages when migrating all the data from a processor, results in an additional buffer read/write, and adds latency by adding another communication phase before the second transpose can occur. The only benefit gained is a decrease in link contention. At the tested problem and machine size, it is clear the benefit is minimal, but still measurable. The improvement follows the same trend as the mapping improvements, smaller problem sizes and more cores per node result in increased performance for the same reasons as previously stated.

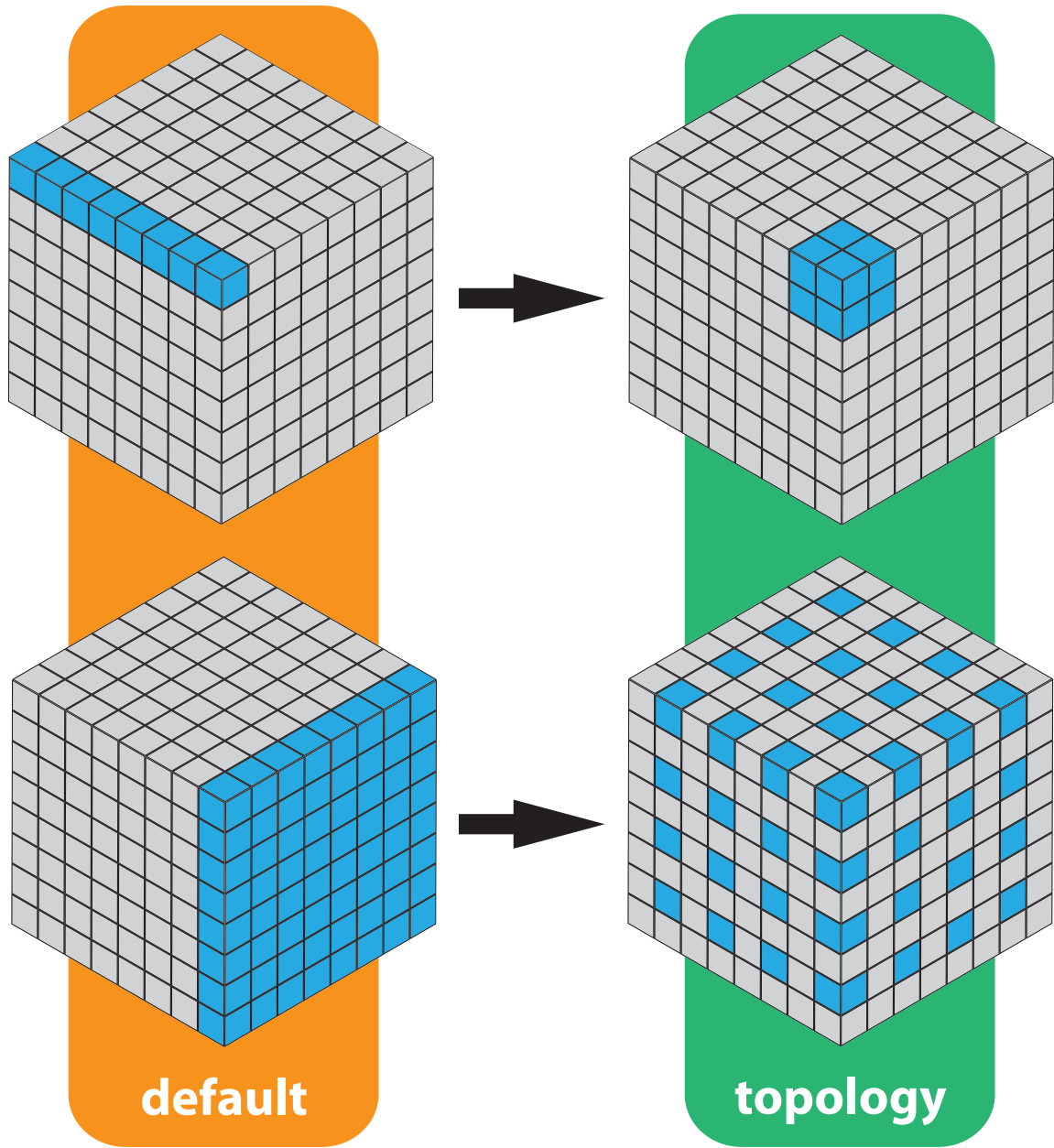


Figure 3.3: Default mapping and topology aware mapping output by SCOTCH.

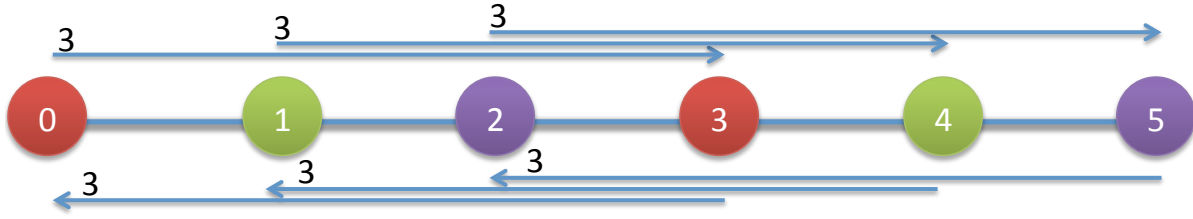


Figure 3.4: 1-D interleaved all-to-all. Colored vertices represent processors in a 1-D mesh with bi-directional links. Similar-colored processors perform an all-to-all within their group. Arrows represent messages, annotated with the number of hop-bytes assuming each processor holds 2 units of data. Total hop-bytes in the system is the summation of all messages.

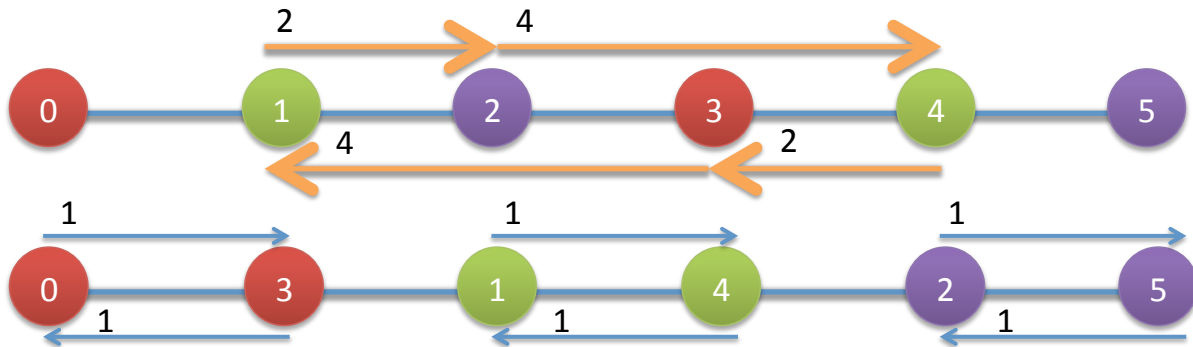


Figure 3.5: 1-D interleaved all-to-all broken into a migration step and a near-neighbor exchange. In step one, thick arrows represent migration messages. All data on a given processor is migrated to the destination. Annotations on the arrows represent the hop-bytes of the message assuming each processor holds 2 units of data. The second step begins with similar-colored processors adjacent. The all-to-all exchange can then occur with neighbors. Total hop-bytes in the system is the summation of all messages across both steps.

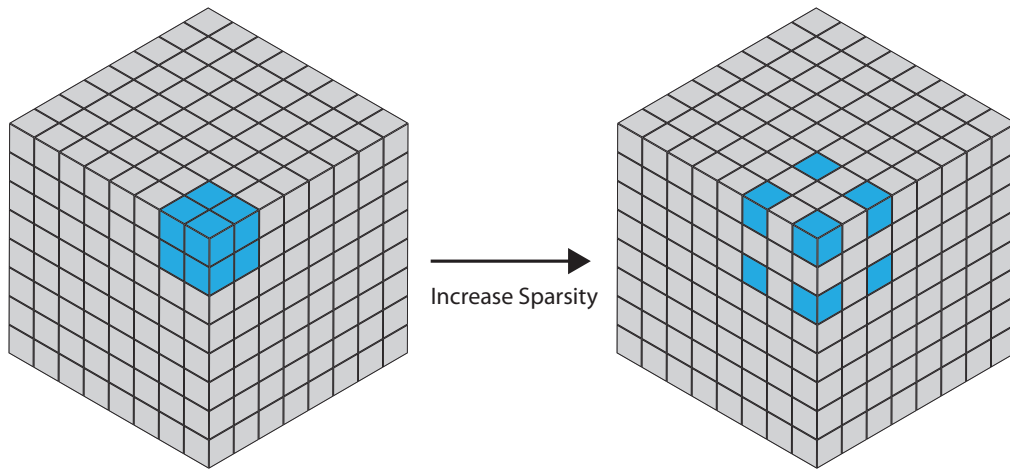


Figure 3.6: Increasing sparsity by pulling a tight 2x2x2 all-to-all apart.

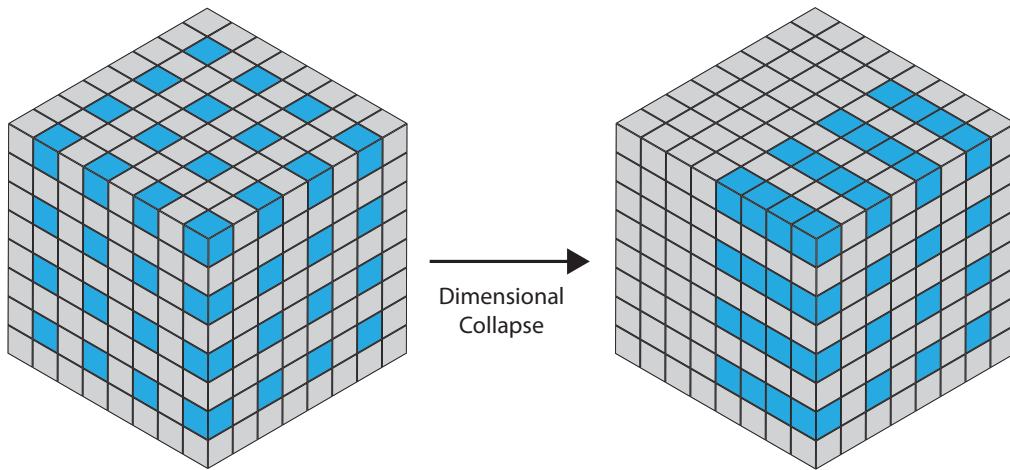


Figure 3.7: Migration by collapsing a dimension

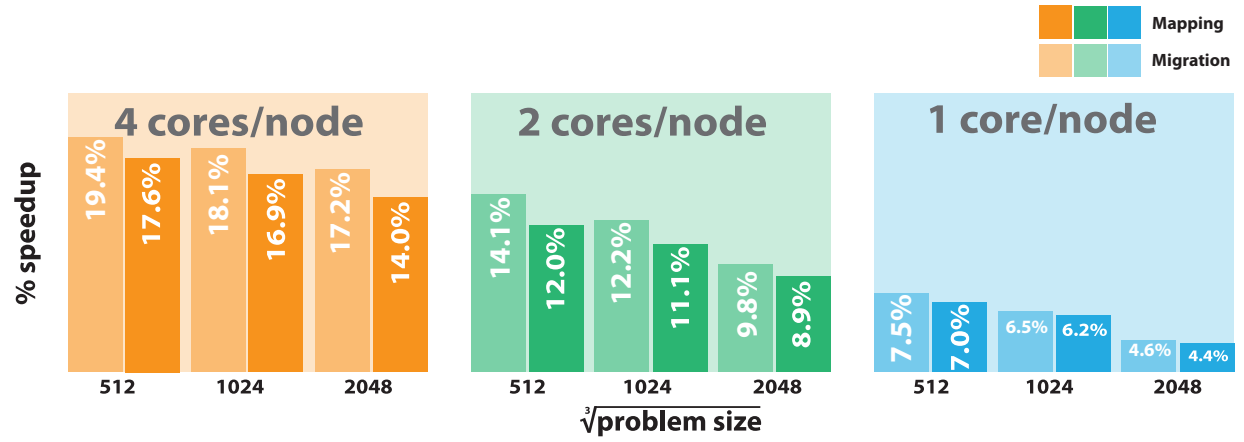


Figure 3.8: Results relative to the default mapping on an 8x8x8 midplane of a BlueGene/P machine. Darker bars represent improvement from re-mapping, lighter shade bars represent the improvement gained from migration and mapping combined.

Chapter 4

Conclusions and Future Work

4.1 Summary

Results showed that FFT performance on 3-D mesh and torus networks can be greatly improved by altering communication strategies and default data layouts. Experiments with a naive implementation of a 1-D FFT showed that when running on more than a tightly-wired midplane of the BlueGene/P architecture, a simple point-to-point implementation does not scale. Dimension-ordered software routing and message agglomeration are required to achieve high performance. Charm++ and its Mesh-Streamer library provide the necessary capability to allow efficient scaling up to at least 65536 cores of the BG/P architecture.

Parallel 3-D FFT communication patterns are significantly more complex than 1-D FFTs. The default mapping on a symmetric 8x8x8 torus results in unused network links and a sub-optimal value for hop-bytes. Graph partitioning techniques revealed that reshaping the pencil and plane transposes of the default mapping can lower the total hop-bytes. The results showed more gains in the case of strong scaling; that is, running a small fixed problem size on more cores. Because the mapping strategy only improves communication efficiency, these techniques mostly benefit small problem sizes and high processor counts where communication time is a large portion of the overall running time. On an 8x8x8 midplane of

BG/P, speedups of up to 17.6% were observed when using all 2048 cores on a problem size of 512^3 complex input points.

The new mapping, although a marked improvement over the default scheme, resulted in an interleaved all-to-all pattern for one of the transpose phases. Data migration to de-interleave the transpose was investigated. It was discovered that by migrating data between the two transposes of the 3-D FFT, link contention could be lowered while the overall hop-bytes of the algorithm remained constant. Migration provided only a minor benefit compared to the mapping while adding latency to the computation by introducing another communication phase which detracted from its overall effectiveness. The most significant result of a 19.4% reduction in run-time – with both mapping and migration effects combined – was observed when using all 2048 cores of a BG/P midplane with a problem size of 512^3 complex input points.

4.2 Future Work

Although this work focused on 3-D torus and mesh networks, a new class of supercomputers employ 5-D and 6-D torus networks [2]. Although the extra dimensions can be folded and this work may be applied directly, future work should investigate how to efficiently utilize the additional links provided by a higher dimension torus.

Furthermore, although the benefits of migration appeared to be marginal, a full scale study of the technique was not completed due to time and resource constraints. Theoretically the link contention during the interleaved transpose increases as the problem size and the processor and node count grow. Using the migration technique on a larger torus network may increase its effectiveness.

Future work also includes the possibility of investigating higher dimension FFTs, although use cases for FFTs higher than 3-D are sparse, they would present an interesting challenge for mapping to the latest 5-D and 6-D torus networks. Modeling the migration phase and

further analysis of the hop-bytes versus link contention could lead to useful insights on the tradeoffs between contention effects, message latency, routing, and bandwidth.

References

- [1] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex fourier series,” *Mathematics of Computation*, vol. 19, pp. 297–297, May 1965.
- [2] “top500.org,” June 2013.
- [3] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, “Scalable molecular dynamics with namd,” *Journal of Computational Chemistry*, vol. 26, pp. 1781–1802, Dec 2005.
- [4] R. V. Vadali, Y. Shi, S. Kumar, L. V. Kale, M. E. Tuckerman, and G. J. Martyna, “Scalable fine-grained parallelization of plane-wave-based ab initio molecular dynamics for large supercomputers,” *Journal of Computational Chemistry*, vol. 25, pp. 2006–2022, Oct. 2004.
- [5] S. H. Glenzer, D. H. Froula, L. Divol, M. Dorr, R. L. Berger, S. Dixit, B. A. Hammel, C. Haynam, J. A. Hittinger, J. P. Holder, and et al., “Experiments and multiscale simulations of laser propagation through ignition-scale plasmas,” *Nature Physics*, vol. 3, pp. 716–719, Sep 2007.
- [6] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, “Introduction to the hpc challenge benchmark suite,” tech. rep., 2005.
- [7] 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.
- [8] Y. Sun, G. Zheng, C. M. E. J. Bohm, T. Jones, L. V. Kalé, and J. C. Phillips, “Optimizing fine-grained communication in a biomolecular simulation application on cray xk6,” in *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, (Salt Lake City, Utah), November 2012.
- [9] E. Solomonik, A. Bhatele, and J. Demmel, “Improving communication performance in dense linear algebra via topology aware collectives,” in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’11*, (New York, NY, USA), pp. 77:1–77:11, ACM, 2011.

- [10] A. Bhatele, G. Gupta, L. V. Kale, and I.-H. Chung, “Automated Mapping of Regular Communication Graphs on Mesh Interconnects,” in *Proceedings of International Conference on High Performance Computing (HiPC)*, 2010.
- [11] A. Bhatele, *Automating Topology Aware Mapping for Supercomputers*. PhD thesis, Dept. of Computer Science, University of Illinois, August 2010. <http://hdl.handle.net/2142/16578>.
- [12] A. Gupta and V. Kumar, “The scalability of fft on parallel computers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 922–932, 1993.
- [13] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing (2nd Edition)*. Addison-Wesley, 2003.
- [14] D. Pekurovsky, “P3dfft: A framework for parallel computations of fourier transforms in three dimensions,” *SIAM Journal on Scientific Computing*, vol. 34, pp. C192–C209, Jan 2012.
- [15] M. Pippig, “Pfft: An extension of fftw to massively parallel architectures,” *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. C213–C236, 2013.
- [16] R. Schulz, “3d fft with 2d decomposition,” *CS Project Report, Oakridge National Laboratory*, 2008.
- [17] L. Kalé and S. Krishnan, “CHARM++: A Portable Concurrent Object Oriented System Based on C++,” in *Proceedings of OOPSLA’93* (A. Paepcke, ed.), pp. 91–108, ACM Press, September 1993.
- [18] L. V. Kale and M. Bhandarkar, “Structured Dagger: A Coordination Language for Message-Driven Programming,” in *Proceedings of Second International Euro-Par Conference*, vol. 1123-1124 of *Lecture Notes in Computer Science*, pp. 646–653, September 1996.
- [19] “Hpcchallenge.org,” July 2013.
- [20] L. Kale, A. Arya, A. Bhatele, A. Gupta, N. Jain, P. Jetley, J. Lifflander, P. Miller, Y. Sun, R. Venkataraman, L. Wesolowski, and G. Zheng, “Charm++ for productivity and performance: A submission to the 2011 HPC class II challenge,” Tech. Rep. 11-49, Parallel Programming Laboratory, November 2011.
- [21] S. Miller, M. Megerian, P. Allen, and T. Budnik, “A flexible resource management architecture for the blue gene/p supercomputer,” *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 438, 2007.
- [22] F. Pellegrini and J. Roman, “Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs,” in *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, HPCN Europe 1996*, (London, UK, UK), pp. 493–498, Springer-Verlag, 1996.

- [23] C. Sudheer and A. Srinivasan, “Optimization of the hop-byte metric for effective topology aware mapping,” in *High Performance Computing (HiPC), 2012 19th International Conference on*, pp. 1–9, IEEE, 2012.
- [24] H. Jagode and J. Hein, “Custom assignment of mpi ranks for parallel multi-dimensional ffts: Evaluation of bg/p versus bg/1,” *International Symposium on Parallel and Distributed Processing with Applications*, vol. 0, pp. 271–283, 2008.