Optimizing the performance of parallel applications on a 5D torus via task mapping

Abhinav Bhatele[†], Nikhil Jain[*,†], Katherine E. Isaacs[§,†], Ronak Buch[*], Todd Gamblin[†],
Steven H. Langer[†], Laxmikant V. Kale[*]

[†]*Lawrence Livermore National Laboratory, Livermore, California 94551 USA*
[*]*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 USA*
[§]*Department of Computer Science, University of California, Davis, California 95616 USA*
E-mail: [†]{*bhatele, tgamblin, langer1*}*@llnl.gov,* [*]{*nikhil, rabuch2, kale*}*@illinois.edu,* [§]*keisaacs@ucdavis.edu*

***Abstract*—Six of the ten fastest supercomputers in the world in 2014 use a torus interconnection network for message passing between compute nodes. Torus networks provide high bandwidth links to near-neighbors and low latencies over multiple hops on the network. However, large diameters of such networks necessitate a careful placement of parallel tasks on the compute nodes to minimize network congestion. This paper presents a methodological study of optimizing application performance on a five-dimensional torus network via the technique of topology-aware task mapping. Task mapping refers to the placement of processes on compute nodes while carefully considering the network topology between the nodes and the communication behavior of the application. We focus on the IBM Blue Gene/Q machine and two production applications – a laser-plasma interaction code called pF3D and a lattice QCD application called MILC. Optimizations presented in the paper improve the communication performance of pF3D by 90% and that of MILC by up to 47%.**

***Keywords*-task mapping, 5D torus, performance, congestion**

I. Motivation

The scale of large parallel machines at the disposal of computational scientists is unprecedented. The top five machines on the Top500 list [1] have over half a million cores. These machines have complex on-node architectures and their compute nodes are connected together by high-speed interconnection networks. The rapid increase of on-node computational power in proportion to network speeds suggests that optimizing communication is important to ensure high efficiency and optimal scaling of parallel applications.

Task mapping is a technique to optimize communication of parallel applications on the interconnection network without having to modify the source code [2]. The technique places tasks or processes on compute nodes based on a careful consideration of the interconnection topology between the nodes and the communication behavior of the application. This is especially important on torus networks because their large diameters can require messages to travel multiple hops to reach their destination, thereby increasing the burden on the shared links. A torus or mesh network topology has been commonly used to connect compute nodes since the Cray T3D machine was developed twenty years ago. Six of the ten fastest supercomputers in 2014 use a torus network for message passing between compute nodes. The K computer uses the Tofu interconnect which is a six-dimensional (6D) mesh/torus. The IBM Blue Gene/P architecture uses a three-dimensional torus and its successor, the IBM Blue Gene/Q, uses a five-dimensional (5D) torus. Several generations of Cray machines (T3D, T3E, XT3/4/5, XE6, XK7) from 1993 to the present have used a three-dimensional torus as their communication backbone.

In this paper, we focus on the 5D torus network deployed in the IBM Blue Gene/Q (BG/Q) machines. Task mapping is an NP-hard problem [3]. The complexity of conceptualizing a 5D torus (as opposed to a 3D one) makes it even more challenging to develop near-optimal mapping heuristics. We use Rubik, a tool developed at LLNL to map applications with structured/Cartesian process grids to arbitrary n-dimensional meshes/tori [4]. We present a step-by-step methodology to improve application performance using topology-aware task mapping, based on our experience with optimizing production codes on BG/Q.

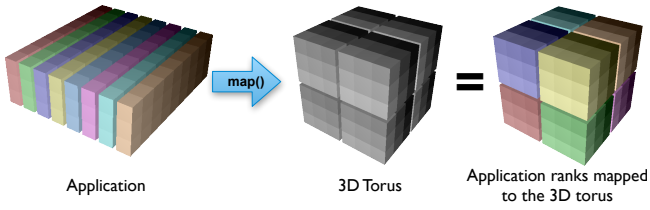
There are three steps involved in improving application performance using topology-aware task mapping: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. We present our experience of applying task mapping to improve the performance of two highly scalable, production applications – pF3D, a laser-plasma interaction code, and MILC, a lattice quantum chromodynamics (QCD) application.

We provide a detailed analysis of the cases in which task mapping improves application performance and the reasons for the performance benefits. We compare the improvements to the change in the average number of hops traveled by messages and the maximum load/congestion on the network links. Task mapping improves the communication performance of pF3D by 2.8× on 131,072 processes and that of MILC by 47% on 65,536 processes. An easily correctable performance bug found when profiling pF3D for this study gives an additional 3.9× improvement. We believe that the methodology presented here will be useful for improving the performance of a broad class of computational science and engineering applications on torus architectures.

II. Background

Task mapping of an HPC application requires generating an *assignment* of MPI task IDs or *ranks* to the cores and nodes in the torus network. Traditionally, programmers have written custom scripts to generate such assignments from scratch. This process is tedious and error-prone, especially with many tasks and high-dimensional networks. We developed *Rubik* [4], a tool that abstracts several common mapping operations into a concise syntax. Rubik allows complex mappings to be generated using only a few lines of Python code. It supports a wide range of permutation operations for optimizing latency or bandwidth, and we only describe a relevant subset here. The full range of transformations possible with Rubik is covered in [4].

Partitioning: Figure 1 shows a Rubik script that describes the application's process grid (a $9 \times 3 \times 8$ cuboid) and a Cartesian network (a $6 \times 6 \times 6$ cube) by creating a "box" for each. Each box is divided into sub-partitions using the `tile` function, resulting in eight $9 \times 3 \times 1$ planes in the application and eight $3 \times 3 \times 3$ sub-cubes in the network. Rubik provides many operations like `tile` for partitioning boxes, allowing users to group communicating tasks. These partitioning operations can also be applied hierarchically.




```
app = box([9,3,8]) # Create application grid
app.tile([9,3,1]) # Create eight sub-planes

network = box([6,6,6]) # Create network topology
network.tile([3,3,3]) # Create eight sub-cubes

network.map(app) # Map app. planes to proc. cubes
```

Figure 1: Mapping 2D sub-partitions to 3D shapes in Rubik

Mapping: The `map` operation assigns tasks from each sub-partition in the application box to corresponding sub-partitions in the network box. Partitions can be mapped to one another if they have the same size, *regardless of their dimensions*. This means we can easily map low-dimensional planes to high-dimensional cuboids, changing the way in which communicating tasks use the network. Thus, the user is able to convert high-diameter shapes of the application, like planes, into compact, high-bandwidth shapes on the network, like boxes.

Permutation: In addition to partitioning and mapping operations, Rubik supports permutation operations that reorder ranks within partitions. The `tilt` operation takes hyperplanes in a Cartesian partition and maps them to diagonals. Tilting is a bandwidth-optimizing operation – if tasks are laid out initially so that neighbors communicate with one another (e.g., in a stencil or halo), tilting increases the number of routes between communicating peers. Successive tilting in multiple directions adds routes in additional dimensions. Tilting can be applied at any level of the partition hierarchy – to specific partitions or to an entire application grid.

III. Mapping, Congestion and Performance

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

A. Performance debugging

Application scientists are often unaware of the reason(s) for performance issues with their codes. It is important to determine if communication between parallel tasks is a scaling bottleneck. Performance analysis tools such as mpiP [5], HPCToolkit [6], and IBM's MPI trace library [7] can provide a breakdown of the time spent in computation and in communication. They also output times spent, message counts and sizes for different MPI routines invoked in the code. Some advanced tools can also calculate the number of network hops traveled by messages between different pairs of tasks. The first step is to collect performance data for representative input problems (weak or strong scaling) on the architecture in question.

Performance data obtained from profiling tools can be used to determine if communication is a scaling bottleneck. As a rule of thumb, if an application spends less than 5% of its time in communication when using a large number of tasks, there is little room for improving the messaging performance. If this is not the case, we can attempt to use topology-aware task mapping to improve performance and the scaling behavior. As we will see in the application examples, task mapping can even be used to reduce the time spent in collective operations over all processes.

B. Performance optimization

There are several tools and libraries that provide utilities for mapping an application to torus and other networks [4], [8], [9], [10], [11], [12]. We use Rubik, described in Section II, to generate mappings for pF3D and MILC. Since the solution space for mappings is so large, there are several factors to consider when trying out different mappings:

- Are there multiple phases in the application with conflicting communication patterns?
- Is the goal to optimize point-to-point operations or collectives or both?
- Is the goal to optimize network latency or bandwidth?

- Is it beneficial to consolidate communication on-node or spread communication on the network?

The previous performance debugging step can provide answers to the questions above and guide us in pruning the space of all possible mappings. Once we have identified a few mappings that lead to significant improvements, it is crucial to understand the cause of the performance improvements, which is the next step in the process.

C. Performance analysis

Performance analysis tools can also be used to dissect the communication behavior of the application under different mapping scenarios. Several metrics have been used in the literature to evaluate task mappings and correlate them with the network behavior – dilation [13], [14], hop-bytes [15], [16] and maximum load or congestion on the network [11]. A more detailed analysis on correlating different task mappings with different metrics can be found here [17].

Comparing the communication and network behavior under different mappings can enable us to understand the root cause of performance benefits and help us in finding near-optimal mappings. In this work, we explore three different metrics that influence communication performance:

- Time spent in different MPI routines
- The average and maximum number of hops traveled over the network
- The average and maximum number of packets passing through different links on the network

All three metrics reflect the state of the network and congestion to different extents and correlate, to differing degrees, with the messaging performance of the application. An iterative process of trying different mappings and analyzing the resulting performance can bring us closer to the optimal performance attainable on a given architecture.

IV. Experimental setup

We use Vulcan, a 24,576-node, five Petaflop/s IBM Blue Gene/Q installation on the unclassified (OCF) Collaboration Zone network at Lawrence Livermore National Laboratory (LLNL) for all the runs in this paper. The BG/Q architecture uses 1.6 GHz IBM PowerPC A2 processors with 16 cores each, 1 GB of memory per core, and the option to run 1 to 4 hardware threads per core. The nodes are connected by a proprietary 5D torus interconnect with latencies of less than a microsecond and unidirectional link bandwidth of 2 GB/s. Ten links, two in each direction (A, B, C, D, and E), connect a node to ten other nodes on the system. The E dimension has length two, so the bandwidth between a pair of nodes in E is twice the bandwidth available in other directions. When running on Vulcan, the shape of the torus and the connectivity for a given node count can change from one job allocation to another. The jobs shapes that were allocated for most of the runs in this paper are summarized in Table I.

#nodes	A	B	C	D	E	Torus or Mesh
128	1	4	4	4	2	Torus in all directions
256	4	4	4	4	1	Torus in all directions
512	4	4	4	4	2	Torus in all directions
1,024	4	4	4	8	2	Mesh in D, Torus in rest
2,048	4	4	4	16	2	Torus in all directions
4,096	4	8	4	16	2	Torus in all directions

Table I: Shape and connectivity of the partitions allocated on Vulcan (Blue Gene/Q) for different node counts

Both pF3D and MILC were run on 128 to 4,096 nodes. Based on our previous experience with the two applications, the performance sweet spot for hardware threads is at 2 threads per core for pF3D and 4 threads per core for MILC. Both applications were run in an MPI-only weak scaling mode, keeping the problem size per MPI task constant. We used mpiP [5] to obtain the times spent in computation and communication in different MPI routines. We used a tracing library by IBM designed specifically for the BG/Q to obtain the average and maximum number of hops traveled by all messages. An in-house library for accessing network hardware counters was used to collect the packet counts for different torus links.

We compare different partitioning and permutation operations from Rubik with two system provided mappings on BG/Q. ABCDET is the default mapping on BG/Q in which MPI ranks are placed along T (hardware thread ID) first, then E, D, and so on. This mapping fills the cores on a node first before moving on to the next node. In the TABCDE mapping, T grows slowest which is similar to a round-robin mapping. MPI ranks are assigned to a single core of each node before moving onto the next core of each node.

V. Mapping study of pF3D

pF3D [18] is a scalable multi-physics code used for simulating laser-plasma interactions in experiments conducted at the National Ignition Facility (NIF) at LLNL. It solves wave equations for laser light and backscattered light. With respect to communication, the two main phases are: 1) wave propagation and coupling and 2) light propagation. The former is solved using fast Fourier transforms (FFTs) and the latter is solved using a 6th order advection scheme.

A 3D Cartesian grid is used for decomposing pF3D's domain among MPI processes. For the input problem that we used in this paper, the X and Y dimensions of the process grid are fixed at 32 and 16, respectively. As we scale the application from 4,096 to 131,072 processes, the number of planes in Z increases from 8 to 256. In the wave propagation and coupling phase, the 2D FFT is broken down into several 1D FFTs, one set involving processes with the same X and Z coordinate and another involving processes with the same Y and Z coordinate. The advection messages are exchanged in the Z direction between corresponding processes of the XY planes. `MPI_Alltoall`s over sub-communicators of

size 32 and 16 are used for the FFT phase and `MPI_Send` and `MPI_Recv` are used for the advection phase.

A. Performance debugging: Baseline performance

We start with profiling pF3D using mpiP to understand the relative importance of the two phases described above and the contribution of communication to the overall time. Figure 2 shows the average, minimum, and maximum time spent in messaging by MPI processes on different node counts. The percentage labels on top of each vertical bar denote the contribution of communication to the overall runtime of the application. For a weak scaling study, we would expect the communication time to be constant, but it continues to grow, especially beyond 1,024 nodes, and adds up to 46% of the total time at 4,096 nodes.

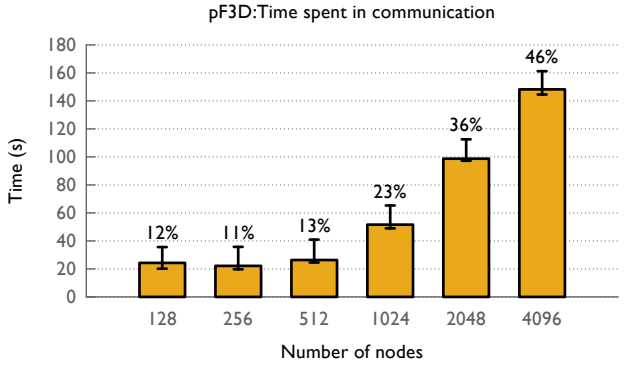


Figure 2: Average, minimum, and maximum time spent in communication by pF3D for weak scaling on Blue Gene/Q

A careful look at the breakdown of this time into different MPI routines (Figure 3) shows that the messaging performance is dominated by three MPI routines – `MPI_Alltoall` from the FFT phase, `MPI_Send` from the advection phase and `MPI_Barrier`. The all-to-alls are over sub-communicators of size 32 and 16 and the message sizes between each pair of processes are 4 and 8 KB, respectively. The advection send messages are of size 256 and 384 KB. We spend ∼200 ms in each send, which is much higher than expected. At 4,096 nodes, we also spend a significant amount of time in an `MPI_Barrier`. We believe communication imbalance due to network congestion manifests itself as processes waiting at the barrier. We hope that an intelligent mapping can reduce this time as well.

B. Performance optimization: Mapping techniques

We now know that for pF3D, the all-to-all and send messages are a scaling bottleneck and any mappings that we develop should try to optimize these two operations. The first two mappings that we tried are ABCDET and TABCDE. ABCDET keeps the all-to-alls in the X direction within the node. However, this mapping is very inefficient for the Sends because 32 tasks on one node try to send messages

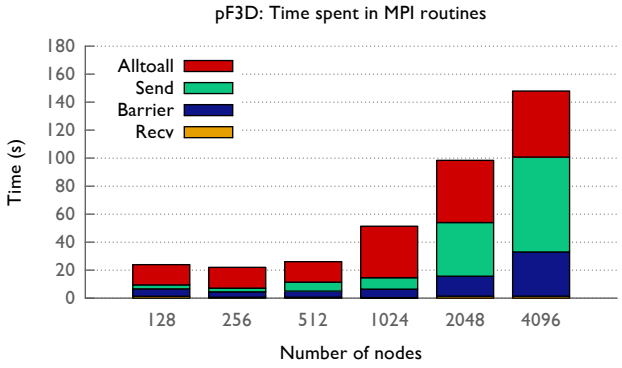


Figure 3: Average time spent in different MPI routines by pF3D for weak scaling on Blue Gene/Q

to corresponding tasks on a neighboring node over a single link. The TABCDE mapping spreads the pF3D XY planes on the torus thereby reducing the congestion and time spent in both the all-to-all and the send. The first and second bar in the plots of Figure 5 show the reduction in time of those two operations, 78% and 52% respectively on 4,096 nodes (ABCDET is referred to as Default and TABCDE is referred to as RR for round-robin in all the figures).

```
from rubik import *

# processor topology -- A x B x C x D x E x T
torus = autobox(tasks_per_node=32)
numpes = torus.size

# application topology -- mp_r x mp_q x mp_p
mp_r = torus.size / (16*32)
app = box([mp_r, 16, 32])

ttile = [int(sys.argv[i]) for i in range(1, 7)]
torus.tile(ttile) # tile the torus

atile = [int(sys.argv[i]) for i in range(7, 10)]
app.tile(atile) # tile the application

# map MPI ranks to their destinations
torus.map(app)

f = open('mapfile', 'w') # write out the mapfile
torus.write_map_file(f)
f.close()
```

Figure 4: A Rubik script to generate tiled mappings for pF3D

The next mapping operation that we try with pF3D is tiling which can help group communicating tasks together on the torus. The entire code for doing this is shown in Figure 4. Rubik obtains the torus dimensions for the allocated partition automatically at runtime (we only need to specify the number of MPI tasks per node). Then we tile the torus and the application and finally call the map operation.

The various tile sizes that we tried for pF3D at different node counts can be handled as inputs by the same script. We tried four different combinations of tile sizes for the torus

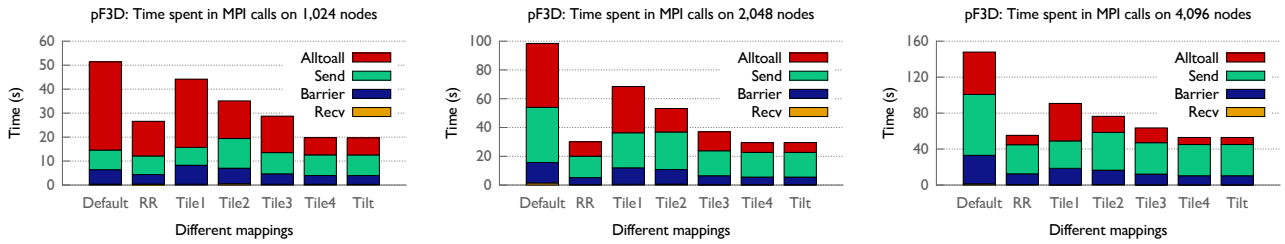


Figure 5: Reduction of time spent in different MPI routines by using various task mappings for pF3D running on 1,024, 2,048 and 4,096 nodes of Blue Gene/Q (Note: y-axis has a different range in each plot)

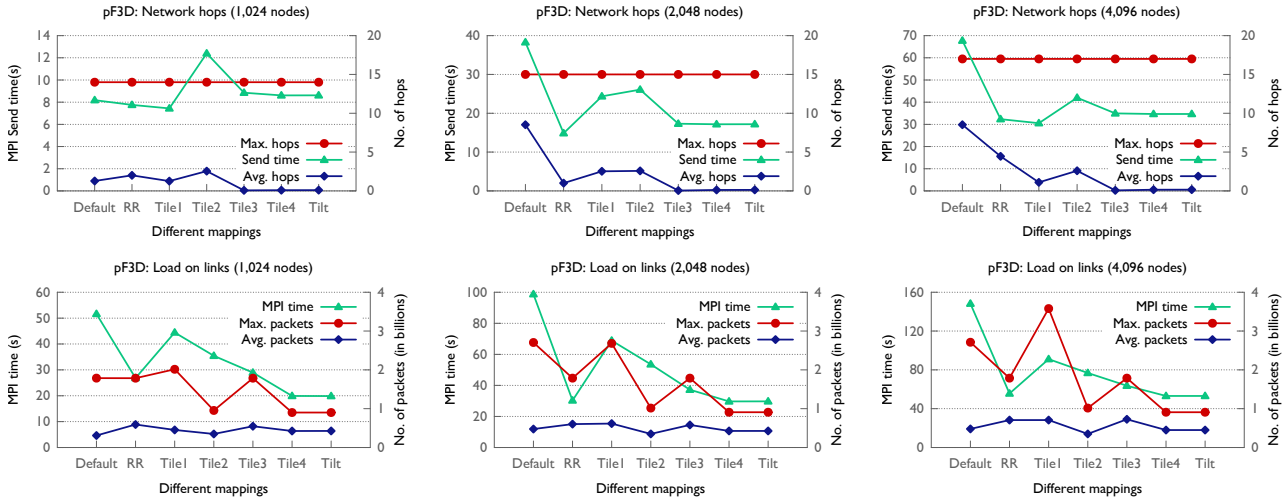


Figure 6: pF3D plots comparing the time spent in point-to-point operations with average and maximum hops (top) and the MPI time with average and maximum load on network links (bottom) (Note: y-axis has a different range in each plot)

and the application which are listed in Table II. *Tile1* and *Tile3* use as few dimensions of the torus as possible. *Tile2* and *Tile4* use as many torus dimensions as possible which results in a $4 \times 4 \times 4 \times 4 \times 2 \times 1$ tile. In *Tile1* and *Tile2*, we make cubic tiles out of the pF3D process grid and in *Tile3* and *Tile4*, we tile by XY planes in the application.

Mapping	Torus tile ($A \times B \times C \times D \times E \times T$)	pF3D tile
Tile1	Use fewest possible dimensions	$8 \times 8 \times 8$
Tile2	$4 \times 4 \times 4 \times 4 \times 2 \times 1$	$8 \times 8 \times 8$
Tile3	Use fewest possible dimensions	$32 \times 16 \times 1$
Tile4	$4 \times 4 \times 4 \times 4 \times 2 \times 1$	$32 \times 16 \times 1$

Table II: Tile sizes used for the Blue Gene/Q 5D torus and pF3D in different mappings

C. Performance analysis: Comparative evaluation

We now compare the performance of various mappings across different node counts with respect to the reduction in time spent in MPI routines and the amount of network traffic that they generate. Figure 5 shows the MPI time breakdown for seven different mappings on 1,024, 2,048 and 4,096 nodes. The first six mappings have already been described above; in the *Tilt* mapping, we create 3D tiles in the 5D torus partition and tilt BC planes in the 3D sub-tori along B. This operation led to significant performance benefits on Blue Gene/P [4] but does not seem to help on BG/Q.

We can make several observations from these scaling plots. The first trend we notice is that an intelligent tiling of the application on to the torus reduces the time in both the all-to-all and the send operation. We also see a reduction in the time spent in the barrier which suggests reduced congestion on the network and/or less communication imbalance. In the case of pF3D, torus tiles that use all the dimensions of the torus perform better than cubic tiles. This is because the messages, especially the all-to-alls, can use more directions to send their traffic. Finally, the time for sends decreases with better mappings but levels off after a certain point – more analysis on this is described in Section V-D. Overall, *Tile4* gives the best performance by reducing the time spent in communication by 52% on 1,024 nodes and 64% on 4,096 nodes as compared to the default ABCDET mapping.

In Figure 6, we use the output from the IBM MPI profiling tool and the network hardware counters library to understand the traffic distribution on the network for different mappings. In the top figures, we see that the maximum number of hops traveled is constant for different mappings (it is also

higher as compared to MILC as we will see in Figure 10). The time spent in the MPI_Send calls closely follows the average number of hops. This suggests that if the application primarily does point-to-point messaging, then reducing the average number of hops is a good idea. The bottom figures plot the average and maximum number of packets passing through any link on the torus network. We notice that the trends for the total MPI time and the maximum load are similar which suggests that it is important to minimize hot-spots or links with heavy traffic on the network.

D. Performance refinement: Iterative process

The plateau in the MPI_Send time reduction (Figure 5) prompted us to look further into the problem. We looked at the stack trace to find the origin of these calls in the source code. These calls are made in the syncforward and syncbackward functions which are a part of the advection phase.

A closer look at the MPI standard and BG/Q's implementation of large sends revealed that the use of MPI_Send followed by MPI_Recv resulted in an unintended serialization of the advection messages. For large messages, MPI_Send uses a direct copy to the receive buffer and returns after the data is transferred to the destination. However, the location of the receive buffer is known only when an associated MPI_Recv is posted. Hence, when the MPI processes in the rightmost XY plane (which do not have to send any data) post their receives, actual transfer of data begins from the MPI processes in the XY plane penultimate to it. When this transfer is completed, the sends posted by the MPI processes in the penultimate plane return and their receives are posted. At this point, the data transfer from the processes in the plane to its left begins. Such inefficient serialized transfer of data continues till we reach the leftmost XY plane.

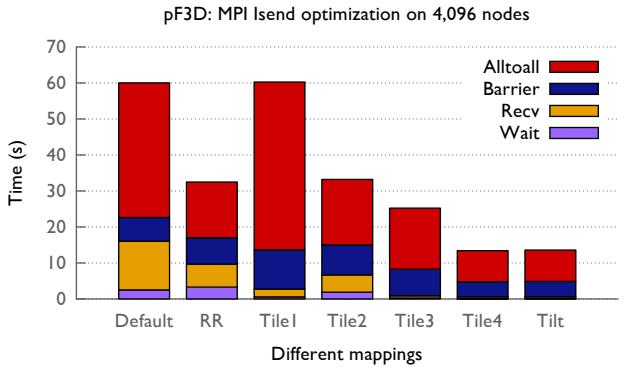


Figure 7: Average time spent in different MPI routines by pF3D on 4,096 nodes (includes MPI_Isend optimization)

The solution is simple – use a non-blocking send, post receives, and then wait on completion of posted sends. We replaced the MPI_Send calls with MPI_Isend and observed significant improvement in the rates for advection messages. Figure 7 shows the new distribution of the time spent in different MPI routines and we can see that most of the time spent in MPI_Send has been eliminated. This also has a positive effect on mapping – the same mappings now lead to higher performance benefits compared to the default. For example, *Tile4* reduces the communication time by 77% w.r.t the default mapping as compared to 64% before.

VI. Mapping study of MILC

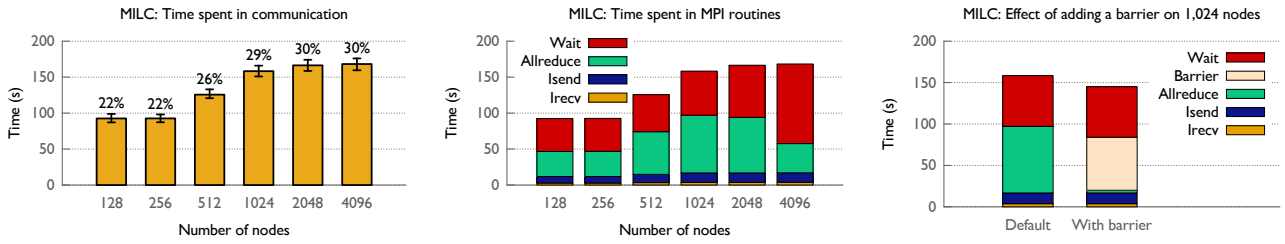
MILC [19], developed by the MIMD Lattice Computation collaboration, is a widely used parallel application suite for studying quantum chromodynamics (QCD), the theory of strong interactions of subatomic physics. It simulates four dimensional SU(3) lattice gauge theory in which strong interactions are responsible for binding quarks into protons/neutrons and holding them together in the atomic nucleus. We use the MILC application su3_rmd, distributed as part of the NERSC-8 Trinity Benchmark suite [20]. In su3_rmd, the quark fields are defined on a four-dimensional (4D) grid of space time points. This grid is mapped onto a 4D grid of MPI processes. In every simulation step, each MPI process exchanges information related to its quarks with its nearest neighbors in all four dimensions. Thereafter, computation (primarily a conjugate gradient solver) is performed to update the associated states of the quarks. Global summations are also required by the conjugate gradient solver.

In order to obtain the best performance, MILC was executed on BG/Q using 4 hardware threads per core. As a result, when running from 128 to 4,096 nodes, the number of MPI processes ranges from 8,192 to 262,144 respectively. The grid size per MPI process is kept constant at $4\times4\times8\times8$, which leads to a weak scaling of the global grid as the node count increases. The dimensions of the MPI process grid for different node counts are decided by the application.

A. Performance debugging: Baseline performance

As stated in Section III-A, the first step in our methodology is to evaluate the communication characteristics of the application. Figure 8a shows that MILC spends between 22% and 30% of its execution time performing communication. As the node count increases from 128 to 4,096, the overhead of communication increases by 83% (from 92 to 168 seconds). Given the weak-scaling nature of these experiments, this increase in MPI time is unexpected and has a negative impact on the overall performance.

The next step is to obtain a detailed profile of MILC to find the predominant MPI routines. Figure 8b reveals that MPI_Wait (following an MPI_Isend/MPI_Irecv pair) and MPI_Allreduce over all processes are the key MPI calls. These results are not very encouraging w.r.t using task mapping for performance optimization. While mapping may help reduce the wait time, it is typically not useful for improving the performance of global collectives (over MPI_COMM_WORLD).



(a) Average time spent in communication (b) Time spent in different MPI routines (c) Imbalance appears as all-reduce time

Figure 8: Evaluation of the baseline performance of MILC with the default mapping (ABCDET) on Blue Gene/Q

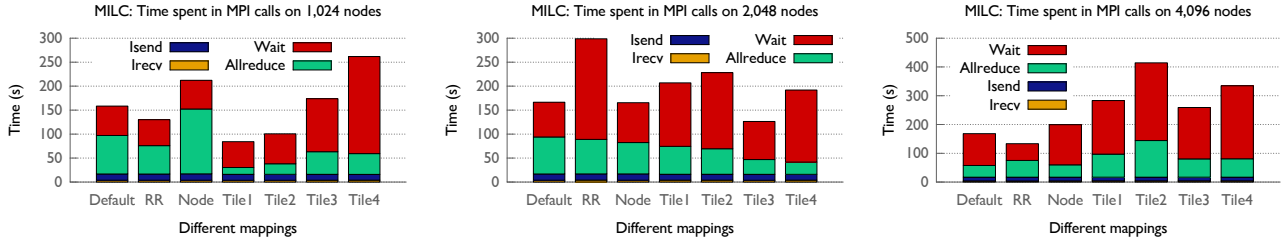


Figure 9: Reduction of time spent in different MPI routines by using various task mappings for MILC running on 1,024, 2,048 and 4,096 nodes of Blue Gene/Q (Note: y-axis has a different range in each plot)

Reason for the apparently slow all-reduce: While the MPI profiles show that a significant time is spent in the all-reduce, the data exchanged per all-reduce per process is only 8 bytes. The long time spent in the all-reduce is puzzling because a typical 8-byte all-reduce on $1,024$ nodes of BG/Q takes only 77 microseconds. A possible explanation is that MILC suffers from either computational or communication imbalance which leads to an increased time spent in the all-reduce, a blocking call that causes global synchronization as a side effect. In order to verify this hypothesis, we did multiple runs in which an `MPI_Barrier` was inserted just before the all-reduce call.

Figure 8c compares the profile for the default case with an execution that has a barrier inserted before the all-reduce on $1,024$ nodes. Most of the reported all-reduce time in the former case shifts to the barrier time in the latter version. This is also observed on other node counts, which supports our hypothesis. Further, since MILC does not have dynamic computational load imbalance and the all-reduce time of *all* processes is high (as confirmed by per process profiling), we can confidently attribute the high all-reduce time to communication-induced variations. This is possibly due to imbalanced or congested point-to-point communication, and can probably be reduced via task mapping.

B. Performance optimization: Mapping techniques

As described in the previous section, MILC spends a significant fraction of its execution time in point-to-point communication which can possibly be reduced by mapping its tasks carefully. The simplest variation to try is the TABCDE mapping. Figure 9 presents the comparison of the communication time for the default ABCDET mapping (*Default*) and TABCDE (*RR*). Depending on the node count, we observe contrasting effects — for 1,024 and 4,096 nodes, TABCDE reduces the communication time by 20%, whereas for 2,048 nodes, it increases the communication time by 80%. The difference shows up in the time spent in wait and all-reduce, both of which can be attributed to point-to-point communication (Section VI-A).

Another mapping which we called the *Node* mapping blocks communicating MPI processes into sub-grids and places them on hardware nodes (64 MPI processes per node). This is a natural choice for mapping of structured applications such as MILC. Surprisingly though, such blocking does not improve the performance; for most cases, *Node* mapping increases the communication time. As a result, we avoid blocking and instead, scatter nearby ranks (as in TABCDE) when generating tile-based mappings with Rubik.

For the tile-based mappings, we attempt to map sub-grids of MILC to similar sub-grids on the BG/Q torus. For example, *Tile1* maps a $4 \times 4 \times 4 \times 4$ sub-grid of MILC to a $4 \times 4 \times 4 \times 4$ sub-grid of BG/Q along its first four dimensions (A, B, C, D). Other mappings (*Tile2*, *Tile3* and *Tile4*) perform similar tilings on different symmetric and asymmetric sub-grid sizes. Most of these choices were guided by the observed performance and profiling information for these experiments (summarized in Figure 9).

In a manner similar to *RR* and *Node* mappings, we observe significant variations in the impact of tile-based mappings on the communication time. *Tile1* and *Tile2* reduce the communication time significantly on 1,024 nodes, but increase the communication time on other node counts. Similar observations about other mappings can be made.

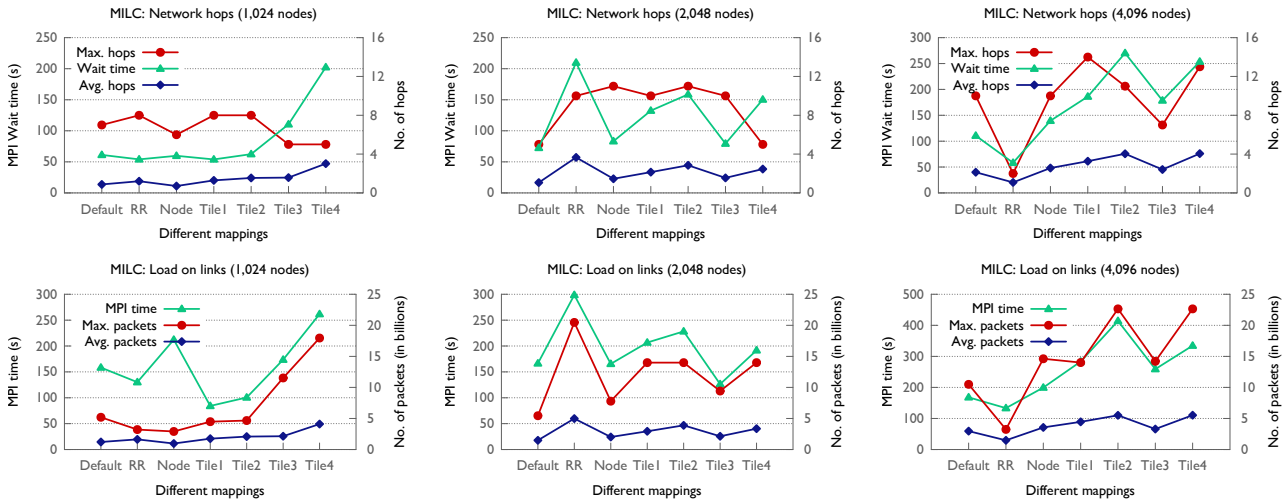


Figure 10: MILC plots comparing the time spent in point-to-point operations with average and maximum hops (top) and the MPI time with average and maximum load on network links (bottom) (Note: y-axis has a different range in each plot)

None of the tile-based mapping we attempted were able to reduce the communication time on 4,096 nodes.

C. Performance analysis: Comparative evaluation

In this section, we attempt to find the cause for varying impact of different mappings on various node counts presented in Section VI-B. Figure 10 (top) shows the time spent in `MPI_Wait` and the average and maximum number of hops traveled by point-to-point messages. The average and maximum number of 512-byte packets on the network links and the total MPI time are shown in Figure 10 (bottom). For MILC, since most of the communication volume is due to the point-to-point communication, the average hop curves are very similar to the average packet curves.

Overall, no correlation is observed between the wait times and maximum #hops. This is expected since the messages in MILC are a few KBs in size and hence, not latency bound. It also suggests that maximum #hops is not a good indicator of network congestion. However, the average #hops and maximum #packets follow trends similar to the wait time and MPI time respectively, with a few aberrations. These observations are in line with our previous results on correlating performance and metrics [17].

For the default mapping, similar values for average #hops and maximum #packets are observed on 1,024 and 2,048 nodes which translates into similar wait and MPI times. At 4,096 nodes, the average #hops doubles which results in increased wait time. However, the MPI time remains the same due to reduced all-reduce time, indicating a better communication balance. In contrast, for TABCDE, both the average #hops and maximum #packets are significantly higher on 2,048 nodes. As a result, TABCDE has a very high wait time and MPI time on 2,048 nodes. On other node counts, use of TABCDE halves the maximum #packets in comparison to the default mapping, and hence also reduces communication time.

On 2,048 and 4,096 nodes, the *Node* mapping provides higher average and maximum #packets in comparison to the default mapping. As a result, it also shows higher wait time and MPI time. At 1,024 nodes, we do not see similar trends. This needs a more detailed study, along the lines of [17]. The four tile-based mappings follow similar trends: a mapping that provides lower average hops and maximum packets on a node count shows lower wait and MPI times.

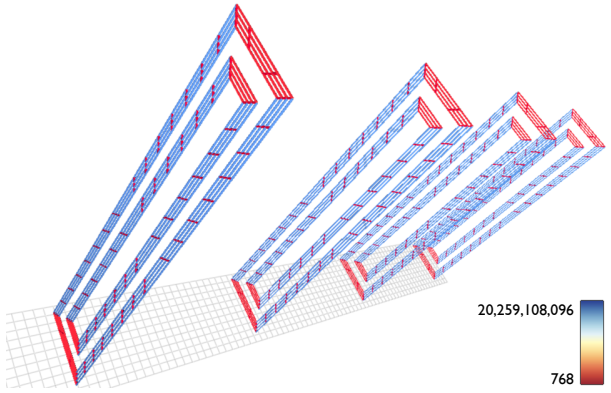


Figure 11: Four sub-tori showing the D (blue), C (red, long), and B (red, short, diagonal) links for the same E on BG/Q

D. Network visualization of packets

We explore network traffic in more detail using the BG/Q visualization module in Boxfish [21]. Boxfish projects the links of an n-dimensional torus into 2D planes to reduce occlusion. By changing which torus directions compose the planes, the links of different directions can be examined. Figure 11 shows an example focusing on the C and D torus

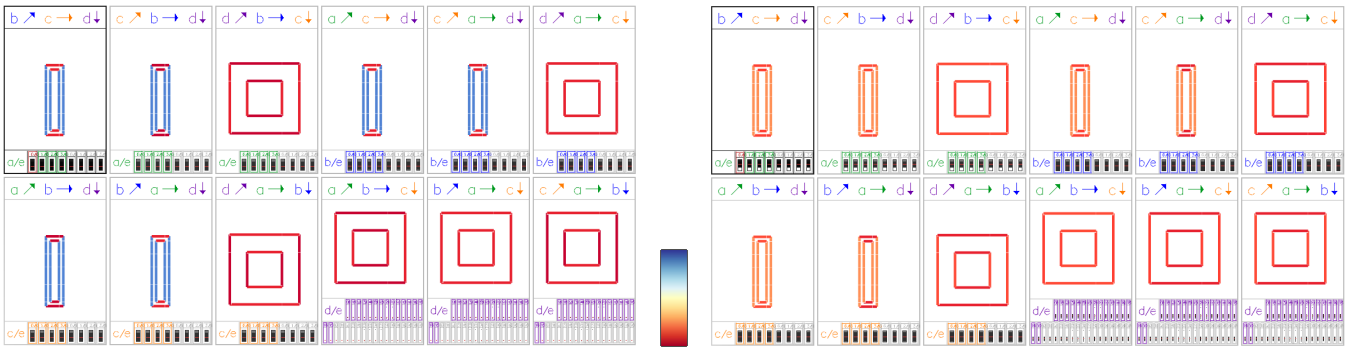


Figure 12: MILC on Blue Gene/Q: Minimaps showing aggregated network traffic along various directions for the TABCDE (left) and *Tile3* mapping (right, the colors represent aggregated traffic flowing in different directions)

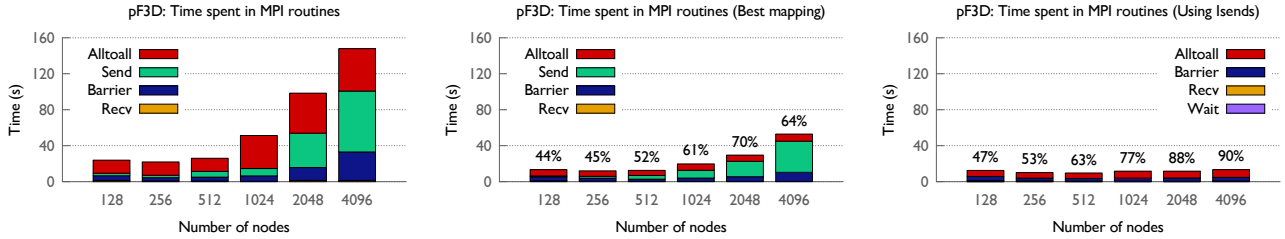


Figure 13: pF3D on Blue Gene/Q: A scaling comparison of the time spent in different MPI routines with the default mapping (left), best mapping discovered (center) and with the best mapping using the Isend optimization (right)

directions for the TABCDE mapping using $2,048$ nodes. The displayed traffic is the network hardware counters data (number of packets) obtained from MILC runs.

We first look at previews of the planes ('*minimaps*') which are colored by aggregated packets taken across all but two torus dimensions. This provides an overview of link behavior in all directions. As each minimap shows two directions aggregated along a third (and the short E direction), they are twelve in total. Figure 12 shows the minimaps for the TABCDE and *Tile3* mappings of MILC on 2,048 nodes. Traffic in the D direction for the TABCDE mapping is high while traffic in all other directions is low. This holds for all minimaps showing the D direction, indicating that this is true for all D links and is not affected by other directions. We also examined the individual links (Figure 11) and verified that there is no significant variation. In comparison, while the *Tile3* mapping has heavier traffic in the D direction, it is still relatively low. These observations are consistent with our findings in Figure 10, which shows higher maximum traffic for the TABCDE mapping than the *Tile3* mapping. Further, the visualization reveals that the maximum traffic is not due to outliers, but is caused by uniformly heavy use of a single torus direction.

VII. Discussion and Summary

This paper presents a step-by-step methodology to optimize application performance on 5D torus architectures through the technique of topology-aware task mapping. We have learned several lessons in the process of performance analysis and optimization of pF3D and MILC on IBM Blue Gene/Q using this methodology. These are some interesting observations that might be useful to others scientists optimizing their codes on BG/Q:

- The default mapping (ABCDET), which blocks MPI tasks on the compute node may not yield the best performance even for near-neighbor codes.
- Mappings that spread traffic all over the network such as TABCDE may lead to better performance for some bandwidth-bound parallel applications.
- Also, for such applications, using all directions to distribute network traffic may provide better performance rather than confining traffic to a few directions.
- Sometimes, computational or communication imbalance or delays due to network congestion can manifest themselves as wait time or time spent in a global collective. Careful mappings can reduce this time.
- It can take several iterations to improve the performance of a parallel code. Sometimes, when the scaling bottleneck is elsewhere, it may appear that intuitive mappings are not leading to expected performance gains.

Finally, in the process of analyzing different mappings and the corresponding network behavior, we were able to improve the performance of pF3D and MILC significantly. Plots in Figure 13 show the time spent by pF3D in different MPI routines for the default mapping, the best mappings we found, and the best mappings combined with the Isend optimization. The labels in the center and right plot denote

the percentage reduction in communication time compared to the default ABCDET mapping. The center plot shows that the best mappings can reduce the time spent in all-to-all, sends and barrier. Tiled mappings improve the communication performance of pF3D by 2.8× on 4,096 nodes and the Isend modification improves it further by 3.9×. Figure 14 shows the performance improvements obtained with the best mappings for a scaling run of MILC (21% reduction in MPI time at 4,096 nodes).

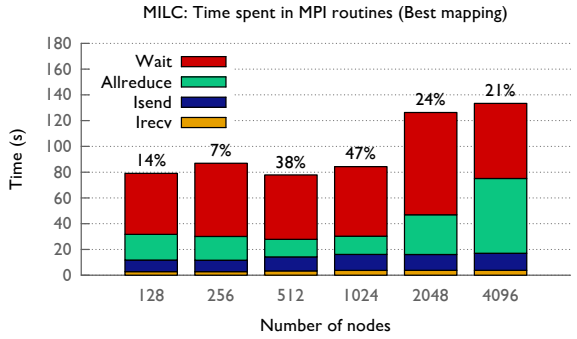


Figure 14: A scaling comparison of the benefits of task mapping for MILC on Blue Gene/Q

ACKNOWLEDGMENT

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work was funded by the Laboratory Directed Research and Development Program at LLNL under project tracking code 13-ERD-055 (LLNL-CONF-655465).

REFERENCES

[1] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. (2009) "Top500 Supercomputer Sites". http://www.top500.org.

[2] A. Bhatele, "Automating Topology Aware Mapping for Supercomputers," Ph.D. dissertation, Dept. of Computer Science, University of Illinois, August 2010, http://hdl.handle.net/2142/16578.

[3] Shahid H. Bokhari, "On the Mapping Problem," *IEEE Trans. Computers*, vol. 30, no. 3, pp. 207–214, 1981.

[4] A. Bhatele, T. Gamblin, S. H. Langer, P.-T. Bremer, E. W. Draeger, B. Hamann, K. E. Isaacs, A. G. Landge, J. A. Levine, V. Pascucci, M. Schulz, and C. H. Still, "Mapping applications with collectives over sub-communicators on torus networks," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. IEEE Computer Society, Nov. 2012, LLNL-CONF-556491.

[5] J. Vetter and C. Chambreau, "mpiP: Lightweight, Scalable MPI Profiling," http://mpip.sourceforge.net.

[6] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "Hpctoolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[7] I.-H. Chung, R. E. Walkup, H.-F. Wen, and H. Yu, "Mpi performance analysis tools on blue gene/l," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[8] 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*. New York, NY, USA: ACM, 2006, p. 116.

[9] 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.

[10] A. Bhatele and L. V. Kale, "Heuristic-based techniques for mapping irregular communication graphs to mesh topologies," in *Proceedings of the Workshop on Extreme Scale Computing APplication Enablement - Modeling and Tools*, ser. ESCAPE '11, Sep. 2011, LLNL-CONF-491311.

[11] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the international conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 75–84.

[12] M. Deveci, S. Rajamanickam, V. J. Leung, K. Pedretti, S. L. Olivier, D. P. Bunde, U. V. Çatalyürek, and K. Devine, "Exploiting geometric partitioning in task mapping for parallel computers," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '14. IEEE Computer Society, May 2014.

[13] Aleliunas, R. and Rosenberg, A. L., "On Embedding Rectangular Grids in Square Grids," *IEEE Trans. Comput.*, vol. 31, no. 9, pp. 907–913, 1982.

[14] S.-K. Lee and H.-A. Choi, "Embedding of complete binary trees into meshes with row-column routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, pp. 493–497, May 1996.

[15] F. Ercal and 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, pp. 210–221.

[16] 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*, April 2006.

[17] N. Jain, A. Bhatele, M. P. Robson, T. Gamblin, and L. V. Kale, "Predicting application performance using supervised learning on communication features," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. IEEE Computer Society, Nov. 2013, LLNL-CONF-635857.

[18] S. Langer, B. Still, T. Bremer, D. Hinkel, B. Langdon, and E. A. Williams, "Cielo full-system simulations of multi-beam laser-plasma interaction in nif experiments," *CUG 2011 proceedings*, 2011.

[19] C. Bernard, T. Burch, T. A. DeGrand, C. DeTar, S. Gottlieb, U. M. Heller, J. E. Hetrick, K. Orginos, B. Sugar, and D. Toussaint, "Scaling tests of the improved Kogut-Susskind quark action," *Physical Review D*, no. 61, 2000.

[20] "NERSC-8: Trinity Benchmarks." [Online]. Available: http://www.nersc.gov/users/computational-systems/nersc-8-system-cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks

[21] A. G. Landge, J. A. Levine, K. E. Isaacs, A. Bhatele, T. Gamblin, M. Schulz, S. H. Langer, P.-T. Bremer, and V. Pascucci, "Visualizing network traffic to understand the performance of massively parallel simulations," in *IEEE Symposium on Information Visualization (INFOVIS'12)*, Seattle, WA, October 14-19 2012, LLNL-CONF-543359.