RAHTM: Routing Algorithm Aware Hierarchical Task Mapping

Ahmed H. Abdel-Gawad*, Mithuna Thottethodi*, Abhinav Bhatele†

*School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907 USA
†Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California 94551 USA
E-mail: {aabdelga, mithuna}@purdue.edu, bhatele@llnl.gov

***Abstract*—The mapping of MPI processes to compute nodes on a supercomputer can have a significant impact on communication performance. For high performance computing (HPC) applications with iterative communication, rich offline analysis of such communication can improve performance by optimizing the mapping. Unfortunately, current practices for at-scale HPC consider only the communication graph and network topology in solving this problem.**

We propose Routing Algorithm aware Hierarchical Task Mapping (*RAHTM*) which leverages the knowledge of the routing algorithm to improve task mapping. *RAHTM* achieves high-quality mappings by combining (1) a divide-and-conquer strategy to achieve scalability, (2) a limited search of mappings, and (3) a linear programming based routing-aware approach to evaluate possible mappings in the search space. *RAHTM* achieves 20% reduction in the communication time and 9% reduction in the overall execution time for three communication-heavy benchmarks scaled up to 16,384 processes on a Blue Gene/Q platform.

***Keywords*—task mapping, routing, divide-and-conquer, linear programming, torus**

I. Introduction

High performance computing (HPC) is a 11 billion dollar market [1] and is broadly used for scientific modeling, simulation, and data analysis. In addition to high-end compute performance, HPC platforms also rely on high-bandwidth, low latency interconnection networks.

Given an MPI-based HPC application wherein individual processes of the application may be imagined to be vertices in a graph, and the communication among the processes may be imagined to be edges between these vertices, its corresponding *application graph* (using the vertices and edges described above) defines the structure of the application. However, to understand communication performance, the application's communication graph alone is inadequate. The processes of the application are placed on the underlying topology of the HPC platform where the nodes are the compute elements (CPUs/GPUs/Accelerators) and the edges are the network links. The mapping from the application graph to the underlying topology has a significant impact on performance of the communication-heavy applications.

The problem of mapping application processes to nodes so as to maximize communication performance – *task mapping* – has been examined before. There are broadly two classes of prior literature that are related to this challenge. On one hand, there are tools that perform heuristic-based application-dependent mapping [2], [3], [4]. However, these are routing unaware techniques that only consider application communication patterns without considering the impact of routing. As such, they fail to exploit all available information. Moreover, these tools are not general. For example, some target fixed topologies (e.g., only 3D meshes are considered in [2], [3], [4]) and others target collective communication only. On the other hand, there are mapping tools from the networks-on-chip (NoC) domain that achieve optimal, routing-aware mapping [5]. Unfortunately, these techniques use computationally unscalable approaches (e.g., mixed integer linear programming) which precludes their use for HPC mapping.

The challenge is to balance the two concerns; to achieve high quality mappings while still achieving practical mapping times. To address this challenge, we present Routing Algorithm aware Hierarchical Task Mapping (*RAHTM*) — a technique that combines the use of optimal mapping for smaller subproblems on to a subset of the topology, followed by heuristic combining of the smaller subproblems. *RAHTM* achieves higher performance as well as feasible offline mapping times.

RAHTM uses a three-phase approach. First, it achieves clustering of processes based solely on communication volume to handle *concentration factor*[1]. The goal is to maximize intra-cluster communication (to exploit high intra-node bandwidth) and minimize extra-cluster communication with the intention of mapping each cluster to a node. This reduces the traffic that traverses the off-node interconnection network and helps reduce network contention.

Second, *RAHTM* uses a top-down approach of optimally mapping and pseudo-pinning the communication graph of the clusters to a 2-ary n-cube in each level starting from the root level down to the lowest level where each cluster becomes one leaf node. These topology partitions (along with the pseudo-pinned clusters) will be combined hierarchically in the third and final stage of *RAHTM*. Because the partitions solved at the leaf level are small, aggressive, optimal mapping techniques are practical.

Third, *RAHTM* uses a heuristic search to combine the mapping of smaller subproblems to solve the global problem. Our incremental, heuristic approach to combine the smaller subproblems is as follows. We first attempt pairwise combinations of leaf-level solutions (and re-oriented/rotated leaf-level

[1]Concentration factor refers to the number of processes on a single node.

SC14, November 16-21, 2014, New Orleans
978-1-4799-5500-8/14/$31.00

solutions) to identify the leaf-level problems with the maximum resulting channel load – a widely-used metric for link congestion. We then greedily combine the two subproblems with the maximum contention – effectively giving maximum mapping freedom to the most critical processes/nodes. Following that, we grow our solution by iteratively identifying the next leaf-problem to combine. The process is similar; at each step we identify the best choice to combine by using pairwise combinations with the current solution. The above process prioritizes heavy-traffic/contention subproblems for earlier mapping which is also when there is maximum flexibility in mapping nodes. Towards the end, there is limited mapping flexibility, but correspondingly, because the incremental traffic/contention of the added subproblems are low, the overall impact is minimal.

We evaluate our design for communication-heavy applications scaled up to 16,384 (16K = 2^{14}) processes on the Mira Blue Gene/Q (BG/Q) platform at Argonne National Laboratory. We achieve an average of 9% performance improvement as a result of our mapping techniques. Note, the overall performance improvement includes the damping effect of Amdahl's law because we only target communication performance. When isolated to communication performance, *RAHTM* reduces network delays by 20% on average.

In summary, the key contributions of this paper are:

- We design a routing algorithm aware mapping technique, *RAHTM*, which scales to 16K processes.
- We show that a mix of optimal and heuristic techniques can offer performance improvements while also being computationally feasible.
- We prove the efficacy of *RAHTM* by demonstrating 9% average speedup (arising from a 20% communication speedup) on communication-heavy benchmarks on a BG/Q machine.

The remainder of this paper is organized as follows. Section II provides a brief background on the mapping problem and the limitations of some prior approaches. Section III describes *RAHTM*. Our evaluation methodology is detailed in Section IV. Section V presents the experimental results. Section VI discusses the limitations and possible extensions to *RAHTM*. Finally, Section VII concludes this paper.

II. Background and Related Work

We consider the prior work on mapping which suffers from the drawbacks of either non-scalability or routing obliviousness. In addition, we offer a brief background on canonical dimension-by-dimension mapping for multi-dimensional torus networks (similar to BG/Q's 5D torus). This approach is communication-unaware, and thus trivial to achieve.

A. Communication-heavy benchmarks

We profiled the communication of several benchmarks from the SpecMPI and NAS benchmark suites using the IPM profiling tool [6]. We focus on three benchmarks (see Table I) which we found to be communication-heavy with significant bandwidth demands. All three benchmarks display iterative communication patterns. Further, these reconfirm prior findings in the literature that identify these benchmarks as communication-heavy [7]. Later, in Section V we show that communication accounts for 35%-70% of overall execution time for these benchmarks. Note, one of our benchmarks, CG, is a variant of the HPCG [8] benchmark, a recently proposed alternative to ranking performance of HPC systems (in place of HPL/Linpack).

Table I
Benchmarks

Name	Suite	Description
BT	NAS	Block Tri-diagonal solver
SP	NAS	Scalar Penta-diagonal solver
CG	NAS	Conjugate Gradient

Note that though collective communication can be mapping-sensitive, we are currently unable to evaluate our RAHTM technique with collective communication because of limitations of our profiling tools. While our profiling tool can tell us that specific collective communication-calls were used, we cannot infer communication patterns (as yet) which depend on how the collective communication is implemented. But this is not a fundamental problem for our approach. We do plan to augment RAHTM to assume implementation-specific communication patterns in our future work.

B. Prior work on mapping

Before we begin a discussion on mapping, it is important to understand what the objective of mapping is. If one were only concerned about the latency of communication and one was not worried about the bandwidth, then distance between communication nodes is the metric to minimize. However, for communication-heavy workloads, the bandwidth is the important metric. Rather than model bandwidth directly, we maximize throughput by minimizing the maximum channel load (MCL) – a well-known metric for reasoning about link contention. Effectively, the best mapping for our purposes is the one that achieves the lowest MCL.

a) Mapping on the Blue Gene/Q: The MPI runtime allows for arbitrary task-to-node mappings that can be read from a file. There are other simpler options for regular mappings. Let us consider BG/Q's 5D torus topology. The dimensions of the torus can be referred to as A, B, C, D, E. In addition, each node has 16 cores. The mapping of tasks to cores within a node can be thought of as an additional dimension T. The default mapping traverses the space dimension-by-dimension in ABCDET order assigning processes in increasing order of rank. The BG/Q runtime makes it easy to specify alternate mappings in terms of other permutations of dimensions such as TEDCBA or TABCDE. Such dimension permutation does not represent an optimized mapping by itself. However, human-guided mapping optimizations can exploit this flexibility if it is obvious to experts that one order will achieve higher performance than the other. Indeed, there are visualization tools to help experts map tasks in such high dimensional spaces. Finally, there are non-canonical mappings that may traverse the space in non-canonical ways (on a diagonal plane, or as per a space-filling curve for example). There are tools to facilitate such non-standard mappings as well.

b) Small scale mapping in NoCs: There also exist several pre-silicon techniques from the CAD domain that examine mapping of IP cores on to networks-on-chip (NoCs) [9], [10], [11], [12], [13]. The key commonality in the above techniques is that they focus on small-scale applications (fewer than 64 tasks) for routing over NoCs. While appropriate at that scale, the mixed integer linear programming (MILP) approach is intractable at the scale of HPC applications.

c) HPC Mapping: There is significant work on mapping MPI tasks on to topologies. We briefly discuss a few here. MPI includes an interface to support run-time remapping by giving processes the ability to query their neighborhoods to understand connectivity and to rerank processes at run-time [14]. There are tools based on such interfaces to achieve reranking-based run-time mapping [15]. Such approaches to on-line remapping require remapping to occur on every run, which fundamentally means that remapping must be very lightweight. Instead our approach is to consider more heavy-weight mapping techniques that are brought to bear offline with the understanding that the overhead of such mapping will be repeatedly amortized over subsequent runs.

Previous work [16] and [17] discuss generic topology-aware mapping algorithms for regular mesh topologies. Both structured and irregular communication graphs are considered and results are presented for several benchmarks and production applications, but these techniques are limited to 2D and 3D meshes. Our work applies more broadly to many topologies and includes routing awareness, which makes a significant difference for bandwidth-sensitive applications. In [18], the authors develop a tool for mapping bandwidth-sensitive applications with collective communication. While the tool enables a rich space of mappings, it does not discover the mappings. Rather, it provides an easy tool for human experts to specify/evaluate mappings. Our goal is to automate the process without the need for human experts. Moreover, this paper does not examine collective communication and focuses on point-to-point communication instead. Our technique may be easily extended to include collective communication in the future by incorporating the communication patterns of the collective communication implementations.

III. RAHTM

A. Overview

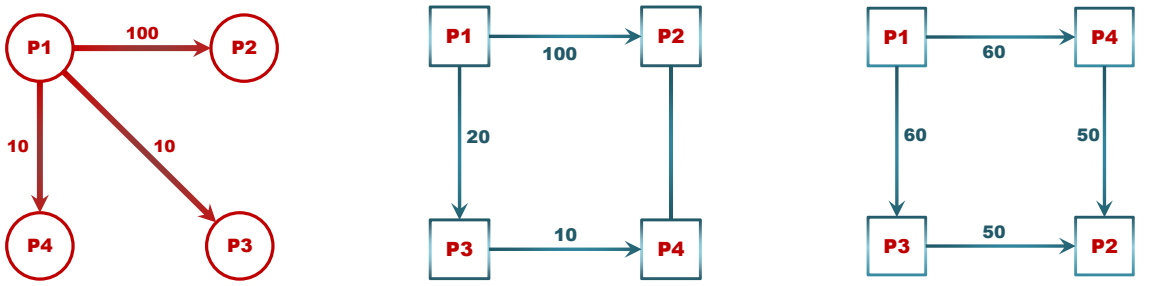
The mapping problem involves assigning each process to a compute node given the communication graph, the topology and the routing algorithm. Although the first two inputs are key to solve the mapping problem, the routing algorithm is crucial to achieve high quality mapping. Previous techniques that neglected the routing algorithm during the mapping determination had to resort to an approximation metric to optimize for in order to maximize the communication performance. In these techniques, this was accomplished via the hop-bytes metric which aims into minimizing the traveled distance for communication flows with heavy weights. However, this turns out to be an inaccurate metric for Blue Gene/Q since it utilizes minimum adaptive routing (MAR) as the routing algorithm. MAR utilizes all available paths from source node to destination node in order to load balance the network.

In Figure 1, we show an example where the hop-bytes metric would perform badly with MAR. Figure 1(a) shows the communication graph to be mapped to a 2×2 network. The communication graph has four process represented as nodes and the communications as edges with the weights indicating the (relative) volumes of these edges. Mapping these processes based on the hop-bytes metric would result into the one shown in Figure 1(b). On the other hand, mapping them based on minimizing the maximum channel load (MCL) given that the employed routing algorithm is MAR would result into the one shown in Figure 1(c). The heavy communication between $P1$ and $P2$ is split across the two possible paths as a consequence to the adaptivity of the routing algorithm. According to this example, the later mapping has better network load balance than the former, thus achieving better communication performance. Actually, the hop-bytes metric works in opposition to the benefits of MAR. Mapping for MAR requires to have pairs of nodes with heavy communication to be on the diagonals of the network to maximize the number of possible paths between each pair. Hop-bytes on the other hand would close the distance between each pair as much as possible, thus creating network bottlenecks.

The objective for the optimization is to improve the network throughput. Alternative heuristics, such as hop-bytes is not the right criterion for throughput; hop-bytes is equivalent to the cost of transporting the data through the network which is related to communication energy. Because my focus is on throughput, minimizing MCL is the appropriate metric as it load balances the network to minimize the maximum contention across all links.

The main design goal for our technique is to be scalable. This implies avoiding to solve the mapping problem for the whole topology directly, given that it is known to be an NP-complete problem [15]. As such, we resort to a bottom-up divide-and-conquer approach to satisfy scalability. Applying this approach to our problem results in clustering both the topology and the communication graph down to small subgraphs which become our leaf-level problems (e.g., mapping a small set of 2^n tasks to a 2-ary n-cube subgraph in a larger k-ary n-torus network). We progressive map and merge the partitions (along with their pseudo-pinned clusters) in a bottom-up fashion. Clustering is also commonly used if the initial communication graph has number of processes larger than the number of nodes in the topology (concentration factor larger than one). However, our clustering serves the dual purposes of achieving concentration factor as well as serving as a basis for small subproblems that can span multiple nodes. All such clustering is achieved in the first phase of *RAHTM*.

In the second phase, we progressively do a top-down mapping of communication graphs of clusters to their associated 2-ary n-cubes till the lowest level where the clusters are the leaf level nodes. These mappings are initial coarse-grain mapping (*pseudo-pinning*) which are later refined and adjusted by reorientation in the third phase. Mapping these subgraphs is crucial to achieve overall high quality mapping since the leaf 2-ary n-cubes contain the local communication at the lowest level while the remaining 2-ary n-cubes in the upper levels orchestrate the global communication overall. We use a mixed integer linear program (MILP) formulation to find the mapping based on an approximation for MAR.



(a) Communication graph (b) Mapping based on hop-bytes metric (c) Mapping based on MCL

Figure 1. Effect of routing algorithm on quality of mapping

In the third phase, we merge the small subgraph mappings in all levels of the hierarchy up to the top level. This is done by incremental merge of the smaller subgraphs, while allowing for each subgraph to be reoriented via rotation and repositioned in the topology. A strict hierarchical approach based solely on clustering offers a recursive mapping structure which is too restrictive. Our twin degrees of freedom of rotation and repositioning significantly enhances the mapping flexibility. At each step, two blocks are merged together based on the best rotation of each block that would give the lowest MCL. We try multiple candidate permutations of rotations of the two blocks and evaluate the MCL of each permutation to be able to rank them. Further, our heuristic choice of the order of block merging enables us to reposition the blocks as we proceed up the hierarchy.

In the remainder of this section, we describe each of the three phases in greater detail along with a single-illustrative example that we use to walk-through each of the phases.

B. Clustering

Consider the twin goals of clustering: achieving the appropriate concentration factor at each node and generating the small subproblems for our bottom-up approach. To satisfy the first requirement, we have to cluster the graph by a factor equal to the concentration factor. Effectively, at the end of this pass, the number of nodes in the topology and the application-graph match. Second, our hierarchical approach requires the dimensions of the topology to be uniform (all dimensions are the same). This is not a serious limitation because topologies that do not satisfy this constraint may be partitioned into smaller partitions where the property holds. (For example, we demonstrate *RAHTM* on Blue Gene/Q where this property is violated because the E dimension is typically has an arity of 2.) We then apply *RAHTM* to each one of the partitions and then merge back the mappings of the subproblems using the third phase of *RAHTM*.

For ease of exposition, we will assume a k-ary, n-torus topology similar to Blue Gene/Q. Later, in Section VI, we will briefly comment as to how these techniques extend to other topologies. We cluster each of the topology and the communication graph such that the leaf-level in each has 2^n nodes. At each level, we merge groups of 2^n nodes to one node in upper level of the hierarchy. For the topology graph, the 2^n nodes are in the form of a 2-ary n-cube at the leaf level (which is straightforward decomposition/clustering that exploits the symmetry of the topology). However, it is not a straight-forward n-ary tree decomposition for the application communication graph because the communication graphs may not be symmetric. As such we try a search of many possible tilings to cluster the 2^n nodes using the inter-tile communication as the metric to choose between possible tile sizes. For example, an 8-node tile in 2-dimensions could search 8×1, 4×2, 2×4, or a 1×8 tile of processor ranks and pick the tile that yields minimal inter-tile communication. We found that such simple tiling based clustering outperformed more sophisticated clustering because they preserved the structure of the communication pattern even after clustering. As such, we did not explore other clustering techniques (e.g., clustering based on min-cuts). We illustrate in Figure 2 the possible tilings for a 16-node communication graph. The tiling that we assume is used repeatedly over the communication graph within a certain level of hierarchy. At each level of hierarchy, the tiling selection is repeated to determine the tile size suitable for that particular level.

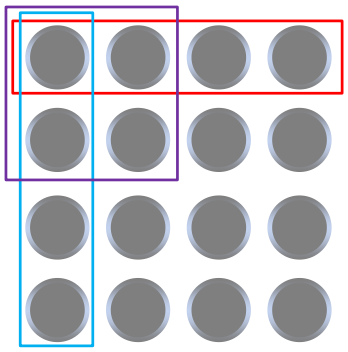


Figure 2. Clustering of communication graph by different tilings

Example of *RAHTM* operation: Clustering
We present a detailed example to illustrate the operation of *RAHTM* in mapping a 16-node communication graph to a 4×4 network. In the first phase, we cluster the 16-node communication graph to 4-node graph shown in the upper level of the hierarchy using a 2×2 tile (see Figure 3; bold circles represent clusters and bold lines represent inter-cluster communication). We also cluster the 4×4 network to a 2×2 network as shown in Figure 4.

C. Hierarchical Mapping of clusters to 2-ary n-cubes

We use a mixed integer linear program (MILP) formulation to find the mapping of the clusters to the 2-ary n-cubes in top-

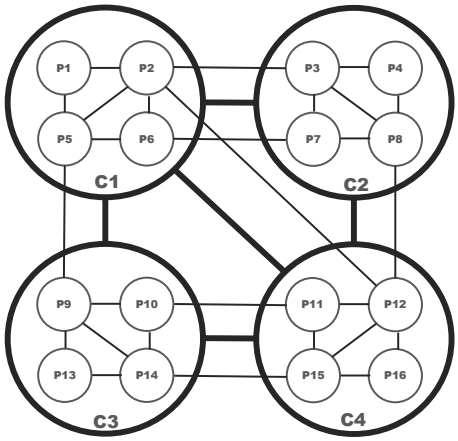


Figure 3. Clustering of communication graph

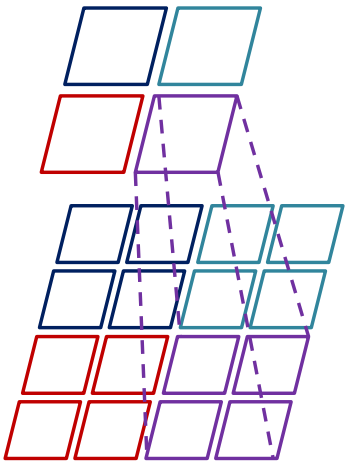


Figure 4. Clustering of topology

down fashion based on the communication graph of the clusters. This is repeated hierarchically down to the lowest level where the clusters become leaf nodes. Our design decision to solve the mapping of these small problems using MILP in order to achieve the best mapping possible is driven by the fact that the scalability issue of MILP formulations is not relevant here since we use it only for very small problems and these subproblems scale only with the number of dimensions which is bounded in practice. Further, we can maximize the use of adaptive paths within the hypercubes to minimize MCL, as stated earlier.

Given the local communication graph of the leaf and the topology (2-ary n-cube), we generate the corresponding MILP formulation and feed it to a MILP solver. The outcome from the solver decides which cluster (process in lowest level) is assigned to which node. Note that this is similar to the approach in [5] although we develop our own formulation based on our prior routing work in TransCom. When the communication graph of clusters (processes) are identical within the same level (i.e., when there is symmetry in the communication graph), which is the common case, the mapping can be reused without the need to solve the MILP formulation repeatedly for each of these communication graphs.

Definition 1: Given a topology graph $G(V, E)$ spanning DM dimensions, where V is the set of vertices (routers) and E is the set of edges (channels), and given an application communication graph $G(A, W)$, where the vertex set A corresponds to the set of clusters/processes and the edge set W corresponds to a set of flows, we must route each of the m flows in $W = \{W_1, \ldots, W_m\}$. Each flow $W_i = (s_i, d_i, l_i)$ consists of source s_i, destination d_i and flow amount l_i. Further, we also define per-channel flow variables $f_i(u, v)$, $i = 1, \ldots, m$, which represents the load carried by the i^{th} flow on the edge $(u, v) \in E$. The mapping is determined using the variables $g_{a,v}$, $a = 1, \ldots, |A|$ and $v \in V$ which are binary variables indicating whether the a^{th} clusters/process is mapped to the vertex v or not. We assume $dim(u, v)$ to represent the dimension which nodes u and v span. It has the property that $dim(u, v) = dim(v, u)$. The direction that a flow occupies is determined by the binary variables $r_{i,dm}$, $i = 1, \ldots$ and $dm = 1, \ldots, DM$.

The objective function and the constraints of the MILP to map the small hypercubes is shown in Table II which aims into minimizing the MCL.

We ensure through the first constraint **(C1)** that each cluster/process is mapped to exactly one vertex (processing node), and that every vertex has at most one cluster/process[2]. The second constraint **(C2)** ensures flow transmission from source to destination. It ensures that a flow is conserved in intermediate vertices in the route except for the source and destination vertices where this condition does not hold, but since the mapping of the source and destination of the flow is floating, the inequalities at the (unknown) source and destination vertices can be converted to equalities by using the $g_{a,v}$ to indicate whether this vertex is a source, destination or an intermediate vertex. We enforce minimal routing by forcing any path to traverse only one direction in any dimension. This constraint is adequate for meshes and not tori. However, for our subproblems, we deal with sub-cubes of the topology that are indeed meshes that lack the wrap-around edge of tori. For our root node, where the 2-ary n-torus indeed has wraparound links, we can still treat the topology as a mesh because a 2-ary n-torus is a special case which is equivalent to a 2-ary, n-mesh with double-wide links. For example, along any one dimension, there are two links (one regular and one wraparound) from one node to the other in the case of a torus as opposed to only one link in the case of a mesh. (While we exploit this coincidence in the case of 2-ary n-tori, it is not central to *RAHTM*; any mechanism to enforce minimal routing is acceptable.) Minimal routing is achieved by the third constraint **(C3)** which enforces flowing through one direction within any dimension. Any non-zero flow variable in one direction within a dimension would zero all flow variables in the opposite direction through the binary variable $r_{i,dm}$ for this dimension dm. In a situation where the application has dense communication, the MILP formulation may end up using minimal routing even without enforcing this constraint. Alternately, if minimal routing is not necessary, this constraint may simply be omitted.

[2]In reality, the number of processes on each node is equal to the concentration factor. However, our clustering stage makes the multiple processes appear to be a single process at this stage.

Table II
RAHTM: MILP FORMULATION FOR FISSION

	Objective Function (Minimization)
O	$Z = \max_{(u,v)\in E} \sum_{i=1}^{m} f_i(u,v)$
	Constraints
C1	$\forall i \quad \sum_{v\in E} g_{s_i,v} = 1$ $\forall v \quad \sum_{i=1}^{m} g_{s_i,v} \leq 1$
C2	$\forall i,\ \forall v,$ $l_i \cdot g_{i,s_i,v} + \sum_{(u,v)\in E} f_i(u,v) =$ $l_i \cdot g_{i,d_i,v} + \sum_{(v,w)\in E} f_i(v,w)$
C3	$\forall i,\ \forall (u,v) \in E, u < v,$ $f_i(u,v) \leq l_i \cdot r_{i,dim(u,v)}$ $f_i(v,u) \leq l_i - l_i \cdot r_{i,dim(v,u)}$

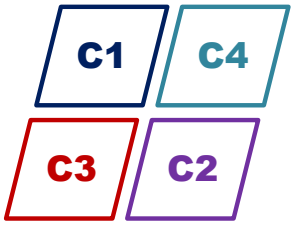


Figure 5. Mapping of root hypercube

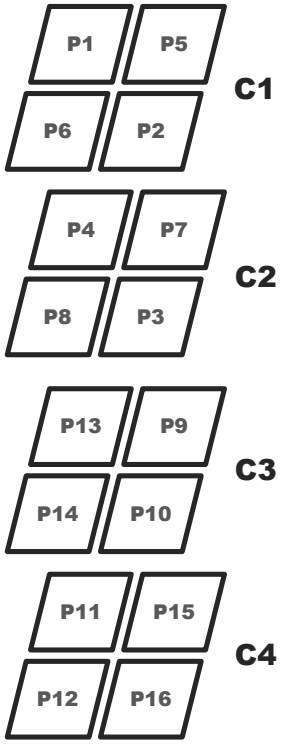


Figure 6. Mapping of leaf hypercubes

Note, that our mapping does not consider other network orchestration issues such as virtual-channel (VC) assignment, which can have some impact on performance. Instead we focus on MCL which models physical channel contention only. Also, adaptive routing techniques may fall back on deadlock-free routing with path restrictions (which are again not captured in our routing algorithm model because the common case behaviour of adaptive routed networks is to operate without path restrictions). However, our experimental evaluation includes all the above real effects (i.e., deadlock-free escape paths for adaptive routing, and virtual channel effects).

Example of *RAHTM* operation: Hierarchical mapping Returning to our example, we start from the top-level of the hierarchy and solve the cluster mapping problem which effectively pseudo-pins clusters of processes to clusters of nodes. The individual mapping is not final because our merging stage may rotate/reorient the mappings to further reduce the MCL. In Figure 5, we show the mapping of the communication graph of the 4 clusters at the top level of the hierarchy to the corresponding 2×2 root network. Then we proceed to the next level of the hierarchy. We map the 4 processes in each of the 4 clusters to one of the 2×2 leaf networks as shown in Figure 6.

D. Merging

To merge the mappings of the subproblems, ideally, one must consider all possible mappings of the subproblems. It is effectively equivalent to considering all possible rotations/orientations of sub-cubes/blocks in space. Because this number of rotations can explode (the total number of orientations is multiplicative in the number of orientations of each block), such exhaustive search is infeasible. To limit the search space, we perform the merging job incrementally, with the following two goals. First, we wish to explore a significant number of rotations and reorientations of individual blocks (though exhaustive search is clearly impractical). Second, we wish to modulate the order of incremental merging to maximize performance.

For the first goal, we restrict reorientations and rotations to pairs of blocks. Even though we consider all possible reorientations and rotations, the pairwise limitation results in a practical number of pairs to consider. For each pair, we evaluate the resulting MCL. We retain information about the lowest MCL achieved by each pair of blocks. For the second goal of order determination, we pick the blocks greedily in decreasing order of average MCLs in their pairwise interactions with other blocks. The rationale here is that the blocks with heavy average interaction must be given maximum flexibility in the mapping stage.

In the first iteration the two blocks with the highest average MCL are picked for consideration. We exhaustively consider all rotations and reorientations of these two blocks and maintain a list of the best N (a tunable parameter in our algorithm which is key to preventing state explosion; we use $N = 64$ in our experiments) orientations that achieve the lowest MCL. In subsequent iterations the block with the next highest average MCL (from the order determining step) is chosen for consideration. Again pairwise interaction of all of the rotations and reorientations of the newly selected block against each of the previous best N merged blocks is computed. At this stage, we again retain the best N merged combinations of three blocks. Proceeding in this way, the algorithm terminates after all blocks have been absorbed with their appropriate rotation/reorientation.

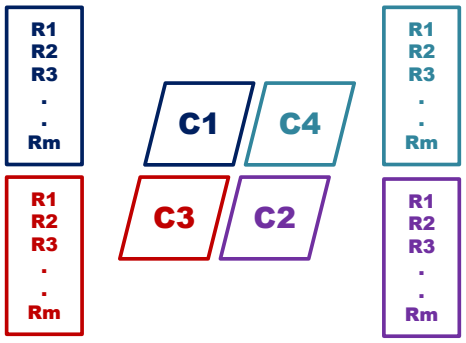
The key insight here is as follows. A purely greedy algorithm that picks a single choice after every merge would be too restrictive and result in poor performance because the algorithm does not satisfy the greedy property. (That is, the best orientation/rotation for merging the first two blocks is not guaranteed to be the best orientation/rotation for the full problem.) On the other hand, exhaustively tracking all rotations/orientations leads to explosive growth. Our decision to prune the list to N maintains a significant number of good solutions while still pruning the search space. Although the search space is being pruned, we do not expect such pruning to be an advantage in reducing the complexity of the problem as my technique is heuristic.

After merging 2^n children blocks of one node in the hierarchy, the mapping can be copied to the neighboring nodes in the same level as long as they have identical local communication graphs; this is a way to exploit symmetry to reduce the computational effort.

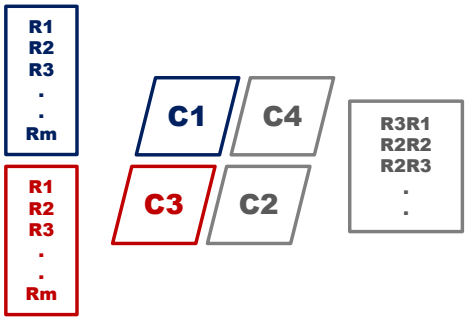
One key step that was abstracted away in the above discussion is the evaluation of any given rotation/reorientation for MCL. The problem of determining the MCL for a given communication pattern for a given oblivious routing algorithm has been solved before [19], [20]. We use their technique with one minor change. Because BG/Q uses adaptive routing (whereas the result from [19], [20] applies only to oblivious routing algorithms), we approximate their routing algorithm by using an oblivious routing algorithm that routes packets over all possible Manhattan paths.

Example of *RAHTM* operation: Bottom-up merging
We perform the merging of the mappings of the subproblem as shown in Figure 7. Initially, we generate the list of possible rotations/reorientations in the space of each 2×2 network ($R1$,$R2$ *etc.*, in Figure 7(a)). We then process them in the order determined by the merging order determination step. For the two 2×2 networks where the two clusters $C4$ and $C2$ were mapped, each of the permutations of rotations between the two lists is evaluated and ranked based on the MCL resulting from the evaluation. These permutations will constitute a new list ordered in ascending order of MCL (descending order of communication performance) for the final merge step of these two subproblems. This is shown in Figure 7(b) where the best solution is the merging of rotation $R3$ of tile $C4$ and rotation $R1$ of the tile $C2$. In figure Figure 7(c), we show the results after the merge of another leaf-level subproblem ($C3$) with the previously merged set of two. Finally in Figure 7(d), we show the result after merging all subproblems. The top ranked permutation represents the best mapping of this final list. The above incremental procedure to merging is vastly simplified compared to a full space search of all possible tiles in all possible rotation/reorientations.

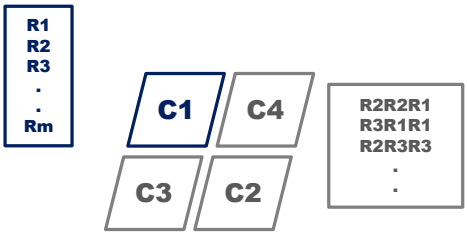
In summary, *RAHTM* reconciles several opposing goals to maximize performance while managing computational complexity. On the one hand, some state exploration is needed to avoid very rigid mapping techniques that lose on performance. On the other hand, maximizing performance via exhaustive search is computationally intractable. *RAHTM*'s use of (1) heuristic clustering based on a limited search of tile shapes, (2) optimal approaches to map (small) cluster-mapping problems, in a hierarchical way, and (3) ordered, incremental merging, combined with pruning the candidate list while maintaining the best candidates, are the design decisions that are geared towards maximizing performance while minimizing computational overhead.



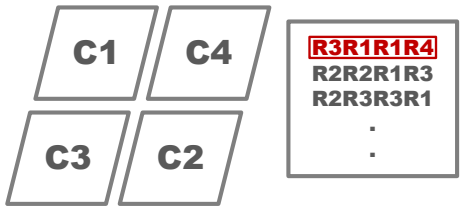
(a) Unmerged hypercubes



(b) After first merge



(c) After second merge



(d) Final mapping

Figure 7. Effect of routing algorithm on quality of mapping

IV. Experimental Methodology

Benchmarks and Evaluation platform: As shown previously in Table I, we use three selected benchmarks from the NAS benchmark suite (`BT`, `CG` and `SP`). We scale these benchmarks to 16K processes and run them on 512 nodes of a Blue Gene/Q machine with 8192 cores. For the benchmarks we used C or D input sets, mainly to achieve reasonable execution times. Our evaluation platform is the Mira BG/Q installation at the Argonne Leadership Computing Facility. BG/Q employs a 5D torus network for communication. We use a partition with $4 \times 4 \times 4 \times 4 \times 2$ dimensions. Each node has 16 cores. We use a higher concentration factor (32) than the number of cores-per-node because these are applications with significant exposed communication.

Other mappings: We compare against the default mapping which assigns process ranks in ABCDET dimension order (as described in Section II). In addition, we chose two other mappings based on similar dimension permutations (ACEBDT and TABCDE). We do not expect such ad-hoc mappings to give us significant performance benefits as they are not chosen based on communication analysis. Further, it is impractical to exhaustively consider all possible $(6!) = 720$ unique permutations of the 6 dimensions. In addition, we also include an adapted Hilbert order mapping. Such space-filling curves have been previously studied for their locality of communication properties. Because Hilbert curves are well-defined in square spaces, we apply Hilbert mapping to the four dimensions that are all 4-nodes long (i.e., ABCD dimensions). For the remaining two dimensions, we map nodes in dimension order (ET order). In addition, we compare against a recent mapping tool – Rubik [18]. The mapping (selected for the best average performance across our benchmarks) is hierarchically tiled using 4x4 tiles from the application space which are mapped to 4x2x2 3D tori in the A, B and E dimensions. We did not include other techniques because either (a) the tools do not work at higher dimensions (e.g., Chaco handles 3 dimensions at most), or may require specific MPI implementations.

MILP and LP solver: To solve the LP and MILP formulations, we use CPLEX v12.5.1 (a commercial LP/MILP solver) to solve such optimization problems on a 64-core (4-socket) AMD Opteron processor-based workstation with 256 GB of memory. The remaining algorithms that constitute *RAHTM* (clustering, greedy incremental combining) used our own implementations.

V. Results

The key results shown by our evaluation are as follows.

- *RAHTM* achieves 9% performance improvement across our benchmarks. It improves performance for all three benchmarks as it reduces communication time by nearly 20% for all three benchmarks.
- The simple, dimension-permutation based mapping techniques do not show uniform improvements. They are better for some benchmarks and worse for others. The net effect (on average) is no better than the baseline.
- The offline mapping computation time is in hours on an individual server i.e., not on an HPC platform. This must be seen in perspective; (1) such routing-aware mapping techniques were previously infeasible at the scales that *RAHTM* enables, and (2) the cost is incurred once for a given scale and does not have to be repeated for individual runs.

A. Overall Performance Improvement

Figure 8 plots the change in execution time relative to the ABCDET mapping (Y-axis) for each benchmark (cluster of bars on the X-axis) for each mapping strategy (individual bars within clusters). In addition, we also include an additional set of bars for the geometric mean performance across the three benchmarks.

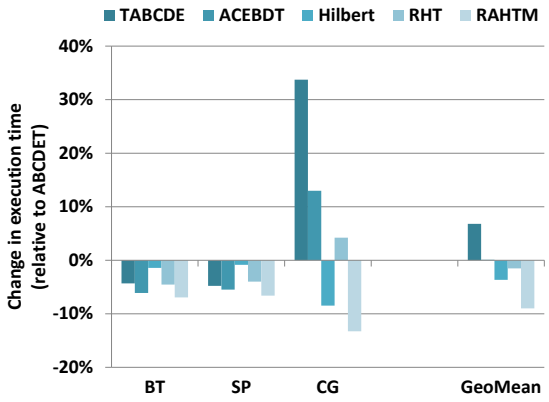


Figure 8. Overall execution time for different mappings

On average, *RAHTM* leads to a mean performance improvement of 9%. *RAHTM* improves performance of all three benchmarks. In contrast, the dimension-permutation based mappings are not uniformly helpful (see `CG`, with 33% and 12% slowdown for TABCDE and ACEBDT mappings, respectively). On an average, the dimension-permutation mappings lead to the same or worse (by 6%) performance as the default mapping. The Hilbert-curve based mapping manages to improve performance modestly (3.7%). The Rubik-based Hierarchical Tiling (RHT) achieves modest improvements for two of the benchmarks, but is worse for the third benchmark(`CG`). On average, RHT performs worse than the Hilbert-curve based mapping. These overall trends in performance improvements due to *RAHTM* can be explained by looking at the amount of communication delay in each of these benchmarks (which represents our opportunity) and the reduction in communication time that *RAHTM* achieves. We expand on these two factors below.

Opportunity: Figure 9 plots the fraction of communication and computation for our three benchmarks, as measured using the IPM tool. Further, the communication fraction is dominated by point-to-point communication because our benchmarks have little collective communication. `CG` is dominated by communication which accounts for over 70% of the total execution time. The other two benchmarks (`SP` and `BT`) also

have a significant (though not dominant at approximately 35%) fraction of communication. Because *RAHTM* is a communication optimization, we expect that the benefit of *RAHTM* will be more obvious in `CG`, as per Amdahl's law.

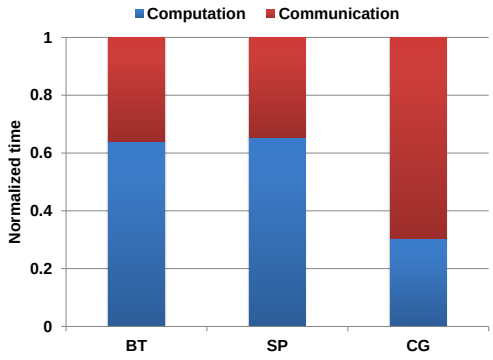


Figure 9. Computation and communication percentages

Communication Reduction: Figure 10 illustrates the two main reasons why our mapping approach outperforms the other mapping techniques. Figure 10 plots the change in communication time relative to the ABCDET mapping (Y-axis) for each benchmark (clusters of bars on the X-axis) for each considered mapping (individual bars within clusters). We include the geometric mean across all benchmarks as an additional set of bars on the extreme right. First, we observe that *RAHTM* consistently improves communication performance whereas the TABCDE and ACEBDT mappings are worse than the baseline for `CG` by significant amounts (48% and 19% worse, respectively). The fact that performance varies so much across applications is a strong case that one cannot *a priori* select a mapping order (a permutation of ABCDET) that is strictly better than our default mapping order. Second, *RAHTM* achieves a consistent 20% reduction in communication time. When read in conjunction with the opportunity measurement, the impact of *RAHTM* on overall execution time is exactly as expected.

B. Optimization Time

Compute time varies from a few minutes (33 minutes for `BT`) to about 35 hours for `CG`. One may think that *RAHTM* is making total time worse (if we consider offline-mapping plus execution time as the total time). However, this view is not appropriate for several reasons. First, given that the offline effort is done once for a given scale and is repeatedly used in multiple runs, it is more directly comparable to compiler optimizations passes than to run-time overheads. As such, adding the mapping time to the execution time is not meaningful. Second, the application occupies the HPC platform (BG/Q in our case) for the application execution time only. The offline mapping computation occurs on a single server. Thus the occupancy of the HPC platform strictly improves.

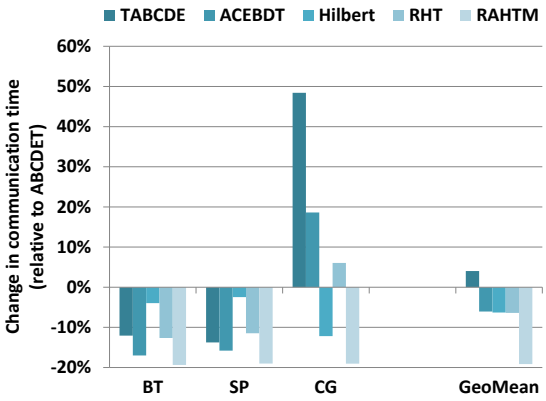


Figure 10. Communication time for different mappings

VI. Discussion

Time for Offline Mapping While *RAHTM* brings routing-aware mapping at scale in to the practical realm, it represents the first step toward scalable routing-aware mapping. We recognize that (a) the mapping times must be further reduced to broaden the appeal of the technique, and (b) that further scaling beyond 16K processes is desirable.

Extension to collective communication Our current implementation of *RAHTM* does not consider collective communication. However, with some additional information on how collective communication is implemented, it is straightforward to extend *RAHTM* to handle collective communication as well. Specifically, *RAHTM* requires the identities of the communicating processes and the (relative) amounts of communication between them. Such communication patterns depend on how the collective communication primitives are implemented. For example, the communication pattern used for a recursive-doubling based implementation of all-gather will be completely different from the pattern used in alternative implementations (e.g., the dissemination all-gather [21]). As such, it is possible to use the communication patterns for known implementations of collective communication primitives to extend *RAHTM* beyond point-to-point communication. We leave such extensions for future work.

Applicability to other topologies While we used the Blue Gene/Q and its 5-dimensional torus topology as the HPC-platform to demonstrate *RAHTM*, the technique itself is topology independent. *RAHTM*'s key ideas are (a) optimal solution for leaf-level subproblems, (b) MCL-based ordered, incremental merging, and (c) pruning of candidate list of rotations/orientations by only retaining the top N candidates. The above basic techniques can be applied to any topology that can be partitioned. The only parts that will change are the parts where (1) we specifically refer to leaf-level subgraphs as hypercubes, and (2) we used the dimensions of the k-ary, n-cube to achieve minimal routing. Note that (1) leaf-level topology partitions can be other structures such as trees in

the case of fat-tree topology, and (2) minimal routing may not necessary in other topologies; and if necessary, there are alternative ways to define minimal routing constraints for other topologies. Further, while symmetries in the topology and/or communication graph can help, the technique broadly applies to other cases as well (at the cost of increased computational complexity). As such, *RAHTM* can be extended to other topologies like fat-trees and dragonfly.

Interaction with application-specific global routing Throughout this paper, we have assumed that the routing algorithm is fixed (i.e., not customized per application beyond simple dimension-order permutation techniques), like in today's HPC systems. If the hardware supports application-specific per-flow routing, there may be interesting co-optimization possibilities by jointly handling mapping and routing (instead of optimizing them independently).

Predictability of Opportunity As observed, some applications may require significant computational effort to optimize mapping. One approach is to use profiling tools which can reveal exposed communication delays. Further, to avoid unnecessary effort for applications that may not have any opportunity, we observed that there are certain qualitative criteria that may be used to identify opportunity. For example, applications with heavy, distant communication seem to offer more opportunity. (Heavy, but largely local communication is relatively easy to handle, even for the baseline.) Alternately, we are also pursuing techniques to reduce the mapping computation without sacrificing the quality of mapping.

VII. Conclusion

There exist many HPC applications that are communication intensive. For such applications, the mapping of processes to nodes is a degree-of-freedom that can be exploited to ameliorate the communication cost. Such mapping can exploit the known communication patterns as well as the routing behavior of the platform. The exploitation of routing behavior ensures that mapping is done in a routing-aware manner to minimize contention on links (and thus maximize bandwidth).

Prior approaches to the mapping problem are either routing-unaware at the larger scales, typical of HPC platforms, or they are routing-aware but only applicable at small scales. The key contribution of this paper is the design and implementation of a routing algorithm aware, hierarchical task mapping (*RAHTM*) technique that scales to 16K tasks. *RAHTM* overcomes the computational complexity of routing-aware mapping by using a mix of optimal techniques and heuristics. *RAHTM*'s bottom-up approach solves leaf-level problems using optimal mapping techniques and a heuristic (greedy) combining technique to merge leaf-level mapping solutions into a larger whole.

RAHTM is an offline mapping tool and its mapping can be used repeatedly in subsequent runs of the application. As such, the cost is not on the critical execution path and does not add to run-time overheads. *RAHTM* achieves 20% reduction in communication time which translates to a 9% reduction in overall execution time for a mix of three communication-heavy benchmarks.

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-653568).

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357 (project allocation: PEACEndStation). We thank the anonymous reviewers for their valuable feedback. We also thank Todd Gamblin and Martin Schulz from LLNL for their early feedback on this approach.

References

[1] E. Joseph, S. Conway, and C. Dekate, "HPC trends, the emerging market for high performance data analysis (HPDA), and the HPC ROI model," International Data Corporation (IDC), Tech. Rep., 2013.

[2] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," in *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '95. New York, NY, USA: ACM, 1995. [Online]. Available: http://doi.acm.org/10.1145/224170.224228

[3] A. Bhatele, L. V. Kalé, and S. Kumar, "Dynamic topology aware load balancing algorithms for molecular dynamics applications," in *Proceedings of the 23rd international conference on Supercomputing*, ser. ICS '09. ACM, Jun. 2009. [Online]. Available: http://doi.acm.org/10.1145/1542275.1542295

[4] A. Bhatele, E. Bohm, and L. V. Kale, "Optimizing communication for Charm++ applications by reducing network contention," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 211–222, Feb. 2011. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/cpe.1637/abstract

[5] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant network-on-chip multiprocessors," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, May 2011, pp. 129–136.

[6] K. Fuerlinger, N. J. Wright, and D. Skinner, "Effective performance measurement at petascale using ipm," in *Proceedings of The Sixteenth IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010)*, Shanghai, China, 2010.

[7] A. Faraj and X. Yuan, "Communication characteristics in the nas parallel benchmarks," in *Proceedings of International Conference on Parallel and Distributed Computing Systems, PDCS 2002*, 2002, pp. 724–729.

[8] J. Dongarra and M. A. Heroux, "Toward a new metric for ranking high performance computing systems," Sandia National Laboratories, Tech. Rep. SAND2013-4744, June 2013.

[9] K. Srinivasan, K. Chatha, and G. Konjevod, "Linear programming based techniques for synthesis of network-on-chip architectures," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, 2004, pp. 422–429.

[10] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proceedings of the 2005 international symposium on Low power electronics and design*, ser. ISLPED '05, 2005, pp. 387–392.

[11] J. Cong, C. Liu, and G. Reinman, "ACES: application-specific cycle elimination and splitting for deadlock-free routing on irregular network-on-chip," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10, 2010, pp. 443–448.

[12] U. Ogras, J. Hu, and R. Marculescu, "Communication-centric soc design for nanoscale domain," in *Application-Specific Systems, Architecture Processors, 2005. ASAP 2005. 16th IEEE International Conference on*, 2005, pp. 73–78.

[13] S. Le Beux, G. Nicolescu, G. Bois, Y. Bouchebaba, M. Langevin, and P. Paulin, "Optimizing configuration and application mapping for MPSoC architectures," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, 2009, pp. 474–481.

[14] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, and J. L. Traeff, "The Scalable Process Topology Interface of MPI 2.2," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 4, pp. 293–310, Aug. 2010.

[15] T. Hoefler and M. Snir, "Generic Topology Mapping Strategies for Large-scale Parallel Architectures," in *Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11)*. ACM, Jun. 2011, pp. 75–85.

[16] A. Bhatele, G. R. Gupta, L. V. Kale, and I.-H. Chung, "Automated mapping of regular communication graphs on mesh interconnects," in *Proceedings of IEEE International Conference on High Performance Computing*, ser. HiPC '10, Dec. 2010. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6152722

[17] A. Bhatele, "Automating topology aware mapping for supercomputers," Ph.D. dissertation, Dept. of Computer Science, University of Illinois, August 2010. [Online]. Available: http://hdl.handle.net/2142/16578

[18] 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.

[19] B. Towles, W. J. Dally, and S. Boyd, "Throughput-centric routing algorithm design," in *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, 2003, pp. 200–209.

[20] B. Towles and W. J. Dally, "Worst-case traffic for oblivious routing functions," in *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '02, 2002, pp. 1–8.

[21] G. Benson, C.-W. Chu, Q. Huang, and S. Caglar, "A comparison of mpich allgather algorithms on switched networks," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Lecture Notes in Computer Science, J. Dongarra, D. Laforenza, and S. Orlando, Eds. Springer Berlin Heidelberg, 2003, vol. 2840, pp. 335–343.