The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Hierarchical Task Mapping for Parallel Applications on Supercomputers

Jingjin Wu · Xuanxing Xiong · Zhiling Lan

Received: date / Accepted: date

Abstract As the scale of supercomputers grows, so does the size of the interconnect network. Topology-aware task mapping, which maps parallel application processes onto processors to reduce communication cost, becomes increasingly important. Previous works mainly focus on the task mapping between compute nodes (i.e., inter-node mapping), while ignoring the mapping within a node (i.e., intra-node mapping). In this paper, we propose a hierarchical task mapping strategy, which performs both inter-node and intra-node mapping. We consider supercomputers with popular fat-tree and torus network topologies, and introduce two mapping algorithms: (1) a generic recursive tree mapping algorithm, which can handle both inter-node mapping and intra-node mapping; (2) a recursive bipartitioning mapping algorithm for torus topology, which efficiently partitions the compute nodes according to their coordinates. Moreover, a hierarchical task mapping library is developed.

The majority of this work was done when Jingjin and Xuanxing were Ph.D. students at Illinois Institute of Technology.

J. Wu
School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, China, 611731
Tel.: +86-28-61831656
E-mail: jwu45@uestc.edu.cn

X. Xiong
Design Group
Synopsys Inc., Mountain View, CA 94043
E-mail: xxiong@synopsys.com

Z. Lan
Department of Computer Science
Illinois Institute of Technology, Chicago, IL 60616
E-mail: lan@iit.edu

Experimental results show that the proposed approach significantly improves the communication performance by up to 77% with low runtime overhead.

Keywords Topology-aware task mapping · hierarchical task mapping · parallel applications · communication optimization · fat-tree network · torus network

1 Introduction

In order to meet the ever increasing needs of scientific applications, high performance computing (HPC) systems keep scaling up toward exascale by integrating more compute nodes, each of which include increasing number of processing cores. As the number of compute nodes increases, so does the size of the interconnect network. Typically, closer processors can communicate more efficiently. As many applications have sparse communication pattern, it is therefore important to carefully map the application processes onto the HPC system for the optimal communication performance. To achieve this goal, we need to find a proper process-to-processor mapping by considering both the physical topology of the HPC system and the communication pattern of the application. This is commonly called *topology-aware task mapping*, or simply *topology mapping*[1].

Fat-trees and tori are two widely used network topologies in modern supercomputers, where fat-trees are often used to build large-scale clusters and tori are typically deployed in massively parallel computers [Top(2014)]. A fat-tree network uses switches to connect compute nodes into a tree structure [Leiserson(1985)]. The compute nodes are at the leaves, while intermediate nodes represent switches. From the leaves to the root, the available bandwidth of links increases, i.e., the links become "fatter". Infiniband networks are representative examples of fat-tree topology [Subramoni et al(2012)Subramoni, Potluri, Kandalla, Barth, Vienne, Keasler, Tomko, Schulz, Moody, and Panda]. An n-dimensional torus network is a mesh with additional links to connect the pair of nodes at the end of each physical dimension [Abts(2011)], so that the network diameter can be reduced by half. For instance, 3-dimensional (3D) torus is used in IBM Blue Gene/P machines and Cray XT5 systems.

From the application perspective, the communication pattern of an application can be *regular* or *irregular*. Typically, a regular pattern means that each process has the same number of communicating neighbors (e.g., the 2D/3D mesh/torus patterns are regular). Otherwise, the communication pattern is considered to be irregular.

The topology-aware task mapping problem has been extensively studied and proven to be NP-hard [Bokhari(1981), Hoefler and Snir(2011)]. Although several physical optimization algorithms [Lee and Bic(1989), Chockalingam and Arunkumar(1992), Berman and Snyder(1987), Salman et al(2002)Salman,

[1] We use "task mapping" and "topology mapping" interchangeably.

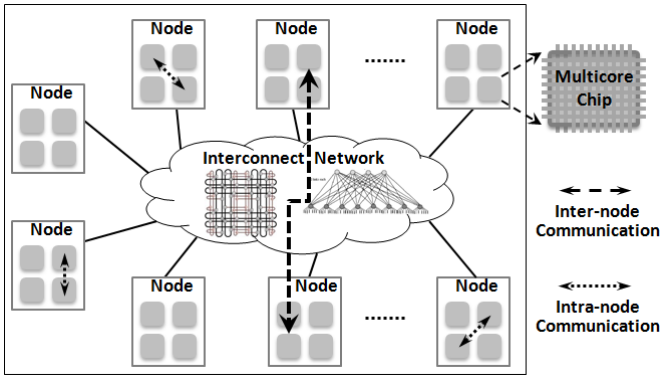


Fig. 1 Inter-node and intra-node communication in supercomputers.

Ahmad, and Al-Madani] have been explored, the research is limited to theoretical studies due to the long running time of these optimization algorithms. Most practical studies target a specific network topology or even a particular application communication pattern. Representative works include the graph partitioning mapping scheme for hierarchical topology [Träff(2002)], the topology detection method for fat-tree topology [Subramoni et al(2012)Subramoni, Potluri, Kandalla, Barth, Vienne, Keasler, Tomko, Schulz, Moody, and Panda], graph embedding schemes for mapping regular communication pattern onto mesh/torus [Yu et al(2006)Yu, Chung, and Moreira], the mapping heuristics introduced in [Bhatele(2010)] for mesh/torus, and so on. There are also application-specific studies, such as [Bhatele and Kale(2008)] and [Wu et al(2012)Wu, Lan, Xiong, Gnedin, and Kravtsov]. Unlike the above studies, Hoefler et al. explored several generic topology-aware task mapping strategies for different network topologies [Hoefler and Snir(2011)].

Modern machines commonly utilize a hierarchical design, where nodes are linked via interconnect networks and each node contains several multi-core sockets (see Fig. 1). The intra-node communication also show different performance levels as shown by our experimental results in Section 5. However, existing studies mainly focus on inter-node mapping with intra-node mapping less studied. To the best of our knowledge, only the TreeMatch algorithm [Jeannot and Mercier(2010)] is developed for intra-node mapping. Moreover, there is few work on concurrent support of both inter-node mapping and intra-node mapping.

In this paper, we propose to perform hierarchical task mapping for parallel applications on supercomputers. Specifically, we consider HPC systems with fat-tree and torus network topologies, and propose mapping strategies to perform inter-node mapping and intra-node mapping in a hierarchical manner. The major contributions are as follows:

1. A generic methodology to perform hierarchical task mapping in two steps: (1) optimize inter-node mapping by considering the network topology and the inter-node communication pattern; (2) optimize intra-node mapping

according to the physical topology of each compute node and the corresponding intra-node communication pattern.
2. A generic recursive tree mapping algorithm is presented for both inter-node and intra-node mapping. It uses a topology tree to represent the hierarchy of the physical topology or the proximity of processors, and recursively partitions the processes according to the tree in order to find a proper mapping. More importantly, the algorithm is applicable for various physical topologies, including fat-tree and torus which are two widely-used interconnect network topologies in modern machines.
3. A recursive bipartitioning mapping algorithm is designed for task mapping onto torus topology. It efficiently partitions the nodes according to their coordinates in the torus network. As demonstrated by the experimental results, its mapping quality can be comparable to that of the recursive tree mapping algorithm for many cases, while it has much smaller mapping overhead. More importantly, it can handle torus topology with non-contiguous allocations, where the allocated nodes for a job are discrete in the torus network.
4. A hierarchical task mapping library HierTopoMap [HTM(2014)] has been developed for supercomputers with fat-tree and torus topologies. It integrates the proposed algorithms to perform inter-node mapping and intra-node mapping properly. This library has been tested on production supercomputers (including TACC Stampede [Sta(2014)], NICS Kraken [Kra(2013)] and ALCF Intrepid [Int(2013)]) with a set of benchmarks and applications. Results show that performing intra-node mapping after inter-node mapping can further improve the communication performance, and the overhead for intra-node mapping is negligible. Overall, the proposed mapping strategies can find high quality mappings with low runtime overhead, which significantly improves the communication performance by up to 77%.

The rest of this paper is organized as follows. Section 2 introduces the task mapping problem and related works. Section 3 presents the proposed hierarchical task mapping strategy. Section 4 describes the design of the hierarchical task mapping library. After experimental results are shown in Section 5, we conclude the paper in Section 6.

2 Background

2.1 Problem Statement

The problem of mapping parallel application processes onto physical processors of HPC systems can be considered as a graph embedding problem [Aleliunas and Rosenberg(1982)]. Typically, the communication pattern of the application is represented as a directed graph $G = (V_G, E_G)$ (denoted as *guest graph*), where V_G is the set of processes. Each edge $(i, j) \in E_G$ has a weight $c(i, j)$, which denotes the amount of communication (in bytes) from process i to process j. Similarly, the physical topology of the computing platform is

represented as an undirected graph $H = (V_H, E_H)$ (denoted as *host graph*). Each vertex in V_H often denotes a processor or a switch, and each edge in E_H often represents a direct link (or the cost of communication). A mapping ϕ from processes V_G to processors in V_H is an embedding of the guest graph G into the host graph H.

The quality of a mapping can be measured by several metrics, such as dilation, congestion [Hoefler and Snir(2011)], and hop-bytes [Bhatele(2010)]. In practice, *hop-bytes* is often used as the evaluation metric, since it is relatively easy to compute and typically correlated with the actual communication performance. Specifically, hop-bytes represents the total traffic load of the interconnect network. It is the total amount of communication (in bytes) weighted by the shortest path distance (in hops) between processors, i.e.,

$$\textit{hop-bytes}(\phi) \triangleq \sum_{\forall (i,j) \in E_G} c(i,j) d\big(\phi(i), \phi(j)\big), \tag{1}$$

where $d\big(\phi(i), \phi(j)\big)$ is the distance. Moreover, hop-bytes also indicates the amount of energy required for data transmission.

The mapping problem is often defined as finding a mapping that minimizes some evaluation metric in order to reduce the communication cost. In general, finding the optimal mapping is NP-hard [Bokhari(1981), Hoefler and Snir(2011)], and much work has been done to explore various solution techniques.

2.2 Related Works

Previous works on topology mapping typically fall under two categories: (1) physical optimization techniques, including simulated annealing [Lee and Bic(1989)], genetic algorithm [Chockalingam and Arunkumar(1992)], graph contraction [Berman and Snyder(1987)] and particle swarm optimization [Salman et al(2002)Salman, Ahmad, and Al-Madani]; (2) heuristic approaches. Although physical optimization algorithms can find high quality mappings, their computation cost is often too high to enable efficient mapping at runtime. As a result, they are limited to theoretical studies and rarely used in practice. In contrast, well-designed heuristics can better exploit the structure of the network topology and the characteristics of the communication pattern, and are capable of deriving good mappings cheaply, thus being more efficient for practical use.

The recursive bipartitioning algorithm is one of the representative heuristics for topology mapping. It was initially proposed by Ercal et al. in [Ercal et al(1988)Ercal, Ramanujam, and Sadayappan] for mapping processes onto a hypercube topology, and extended to handle general topologies in [Pellegrini(1994)]. It has been implemented in the software package SCOTCH [pellegrini and Roman(1996)], and proven to be an effective approach in deriving static mappings.

For systems with a hierarchical communication architecture, e.g., clusters of SMP nodes, Träff [Träff(2002)] proposed to use graph partitioning for finding an appropriate mapping. For efficient topology mapping on InfiniBand networks, Subramoni et al. [Subramoni et al(2012)Subramoni, Potluri, Kandalla, Barth, Vienne, Keasler, Tomko, Schulz, Moody, and Panda] proposed to detect the InfiniBand network topology by using the neighbor joining algorithm.

Yu et al. [Yu et al(2006)Yu, Chung, and Moreira] exploited graph embedding schemes to map regular 2D/3D communication pattern onto 2D/3D mesh/torus topology, and developed a topology mapping library, which was used on BG/L supercomputers to support MPI topology functions. Bhatelé [Bhatele(2010)] developed MPI benchmarks to evaluate the effect of network congestion on message latency, and proposed a set of heuristics for mapping parallel applications onto regular mesh/torus.

Agarwal et al. [Agarwal et al(2006)Agarwal, Sharma, Laxmikant, and Kale] proposed a greedy heuristic by using estimation functions which are used to evaluate the effects of mapping decisions. Hoefler et al. developed a topology mapping library LibTopoMap [Hoefler and Snir(2011), Lib(2011)], which included a greedy heuristic, a recursive bipartitioning approach, and a mapping strategy based on RCM ordering [Cuthill and McKee(1969)]. In order to solve large-scale mapping problems efficiently, Chung et al. [Chung et al(2011)Chung, Lee, Zhou, and Chung] proposed a hierarchical mapping algorithm to perform mapping in a divide-and-conquer manner.

In addition to the aforementioned works on inter-node mapping, Jeannot et al. [Jeannot and Mercier(2010)] proposed the TreeMatch algorithm for mapping processes onto cores within a multicore compute node, and this algorithm was used in [Mercier and Jeannot(2011)] to support the MPI_Dist_graph_create function.

Rashti et al. [Rashti et al(2011)Rashti, Green, Balaji, Afsahi, and Gropp] proposed to use a weighted graph to model the whole physical topology of the computing system, including both the inter-node topology and the intra-node topology, and used the recursive bipartitioning algorithm in SCOTCH [pellegrini and Roman(1996)] to find a proper mapping. While this seems an effective approach to tackle inter-node and intra-node mapping for some physical topologies, a weighted graph is not a proper representation of several physical topologies in modern machines. For example, the InfiniBand network topology and the intra-node topology can be better represented by topological trees (without edge weights), and the topology of non-contiguous nodes allocated to a user application on Cray XT5 machines cannot be effectively modeled by a sparse weighted graph.

In summary, existing studies focus on inter-node mapping, and overlook the mapping within a compute node. Generic mapping algorithms like recursive bipartitioning [Pellegrini(1994),Hoefler and Snir(2011)] and greedy heuristics [Agarwal et al(2006)Agarwal, Sharma, Laxmikant, and Kale, Hoefler and Snir(2011)] can find good mappings, but they are likely to be outperformed by other heuristics like graph partitioning [Subramoni et al(2012)Subramoni,

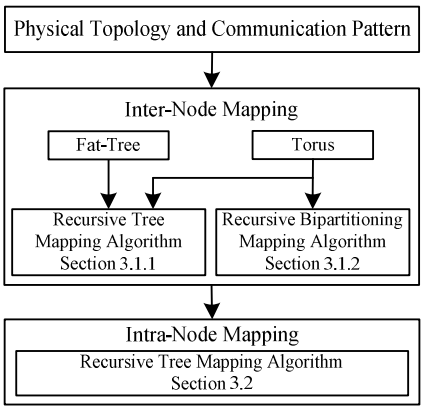


Fig. 2 Flow of the proposed hierarchical task mapping strategy.

Potluri, Kandalla, Barth, Vienne, Keasler, Tomko, Schulz, Moody, and Panda] (according to the fat-tree topology) and graph embedding [Yu et al(2006)Yu, Chung, and Moreira] (on to regular mesh/torus), which are well-designed to exploit the structure of specific mapping problems. Typically, a proper inter-node mapping reduces the communication between nodes by placing heavily communicating processes on the same compute node, leading to a large amount of intra-node communication. Hence, it is important to optimize the intra-node mapping, in addition to inter-node mapping. Although the TreeMatch algorithm [Jeannot and Mercier(2010)] has been proposed as an effective heuristic for intra-node mapping, to the best of our knowledge, there is little work on concurrent support of both inter-node mapping and intra-node mapping.

3 Hierarchical Task Mapping Strategy

We consider mapping parallel application processes onto HPC systems with multicore compute nodes (see Fig. 1), where each compute node is assigned multiple MPI processes. The physical topology of the computing system consists of two parts: *network topology* (i.e. inter-node topology) and *intra-node topology*. As they have different characteristics, which can be exploited by different mapping techniques, we propose to perform inter-node mapping and intra-node mapping in two steps, so that the mapping can be better optimized. At first, we perform inter-node mapping to reduce the amount of traffic in the interconnect network. Second, we perform intra-node mapping to determine the proper mapping of processes to processors for each compute node.

Specifically, for inter-node mapping, we target both *fat-tree* and *torus* topologies. Note that torus topologies can be further divided into two categories: (1) those with contiguous allocation, e.g., IBM Blue Gene machines, where a dedicated continuous partition of the system is allocated to each job [Tang et al(2011)Tang, Lan, Desai, Buettner, and Yu]; (2) those with non-contiguous allocation, e.g., Cray XT series, where discrete compute nodes are

allocated to each job. For intra-node mapping, the topology can be represented as a tree, which models the intra-node hierarchy with leaves being logic processors. We introduce *a generic recursive tree mapping algorithm* for both inter-node and intra-node mapping. In addition, we also design an efficient recursive bipartitioning mapping algorithm specifically for mesh/torus topology. Fig. 2 summarizes the flow of the proposed hierarchical mapping strategy, which naturally exploits the hierarchy of the computing system. The following subsections present the details.

3.1 Inter-Node Mapping

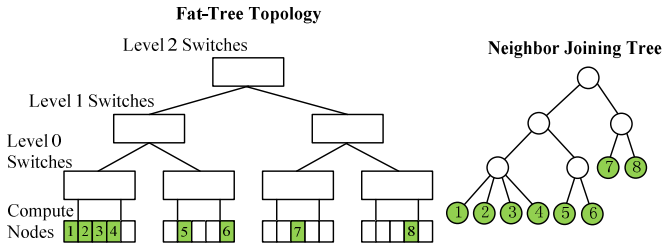
The inter-node mapping problem can be formally stated as follows. Given the network topology and the application communication pattern, which is represented by the directed graph $G = (V_G, E_G)$, find a proper mapping of processes onto compute nodes to reduce communication cost. Here, we assume that the compute nodes are identical, and they are assigned the same number of processes. This is common in HPC.

In order to achieve better mapping efficiency, we use a *preprocessing step* before applying specific mapping strategies. The graph $G = (V_G, E_G)$ is first converted into an undirected graph, whose edge weight represents the total communication volume between respective processes. Then the processes are divided into equal-sized groups by partitioning the undirected communication graph. The number of process groups is equal to the number of compute nodes, and each group of processes will be mapped onto a compute node. The communication relation between these process groups can be represented as an undirected graph $\widehat{G} = (\widehat{V}, \widehat{E})$, which will be used by mapping algorithms discussed in the following subsections. As the multilevel k-way partitioning scheme [Karypis and Kumar(1998)] computes the partition of a graph with $|E|$ edges in $O(|E|)$ time, the time complexity of this preprocessing step is $O(|E_G|)$.

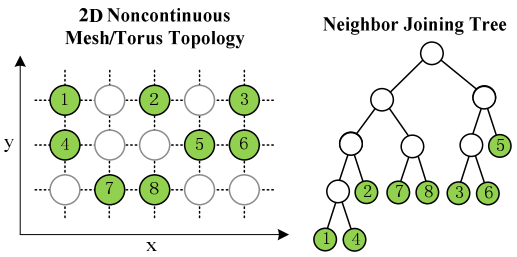
3.1.1 Recursive Tree Mapping Algorithm

We adopt the neighbor joining algorithm [Subramoni et al(2012)Subramoni, Potluri, Kandalla, Barth, Vienne, Keasler, Tomko, Schulz, Moody, and Panda] to detect the fat-tree topology, and then recursively partition the communication graph $\widehat{G} = (\widehat{V}, \widehat{E})$ according to the hierarchy of the topology to derive a proper mapping. More importantly, we extend this approach for task mapping onto torus topology.

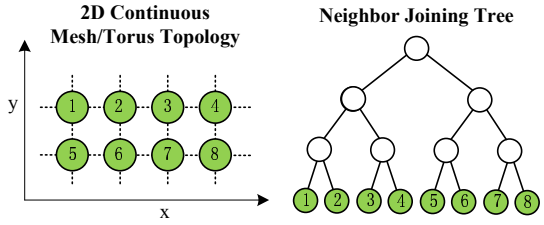
The neighbor joining algorithm constructs a tree topology based on the hop distances between nodes by using an iterative procedure. In each iteration, it identifies a group of nodes, which are closest to each other; joins this group of nodes to create a new node; and computes the distances between the new node and the remaining nodes. The joined nodes are connected with the new node to form a tree structure (also called "neighbor joining tree"), and the



(a) Fat-Tree



(b) Mesh/Torus – non-contiguous allocation



(c) Mesh/Torus – contiguous allocation

Fig. 3 Neighbor joining for fat-tree and mesh/torus. The nodes allocated to the parallel application are light green.

iteration terminates when all the nodes are "joined". This algorithm has a time complexity of $O(n^2)$, where n is the number of nodes.

As neighbor joining only relies on the distances between nodes, it can be used as a generic approach for topology detection. In particular, it is well suited for detecting logic tree topologies on non-contiguous allocation machines, such as the InfiniBand based machine illustrated in Fig. 3 (a). It can also represent the proximity relations of the compute nodes in torus networks as topology trees (see Fig. 3 (b) and (c)). Each leaf of the neighbor joining tree represents a compute node, and each non-leaf node denotes a group of compute nodes that are leaves of the subtree rooted at itself.

As summarized by the recursive tree mapping algorithm in Fig. 4, the mapping can be obtained by partitioning the communication graph recursively to match the compute node groups at each level from root to leaves, and the processes are mapped onto the corresponding compute nodes. For each level, graph

Algorithm 1 Recursive Tree Mapping

Input: communication graph $\widehat{G} = (\widehat{V}, \widehat{E})$,
topology tree T.
Output: mapping $\phi : \widehat{V} \rightarrow$ leaves of T.

```
1  tree_mapping(Ĝ, T)
2  {
3    if (root(T) is a leaf) {
4      φ(i) = root(T), ∀ process i ∈ V̂;
5      return;
6    }
7    k = the number of children of root(T);
8    T_i be the subtree rooted at the ith child
     of root(T);
9    // partition Ĝ into subgraphs Ĝ_i = (V̂_i, Ê_i),
10   // 1 ≤ i ≤ k, where |V̂_i| = the number of
11   // leaves of T_i.
12   (Ĝ_1, Ĝ_2, ··· , Ĝ_k) ← graph_partition(Ĝ);
13   for i = 1 to k {
14     tree_mapping(Ĝ_i, T_i);
15   }
16 }
```

Fig. 4 The recursive tree mapping algorithm.

partitioning minimizes the communication between compute node groups. As a result, most communication would be between neighboring compute nodes of the same group, leading to better communication performance.

Lemma 1 *Given the inter-node communication graph $\widehat{G} = (\widehat{V}, \widehat{E})$, where $|\widehat{V}|$ is equal to the number of compute nodes, it takes $O(|\widehat{V}|^2)$ time to build the neighbor joining tree, and the time complexity of the recursive tree mapping algorithm is $O(|\widehat{E}| \log |\widehat{V}|)$.*

Proof The expected hight of the neighboring joining tree is $O(\log |\widehat{V}|)$, and the graph partitioning at each level takes $O(|\widehat{E}|)$ time, thus Lemma 1 follows.

3.1.2 Recursive Bipartitioning Mapping Algorithm for Torus Topology

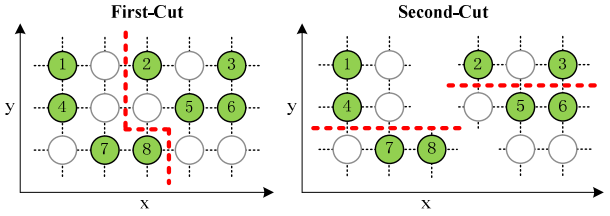
Although the recursive tree mapping algorithm can be applied for torus topology, building the neighbor joining tree can be expensive as shown by the results in Section 5.2.2. In order to take advantage of the geometric structure of torus for efficient mapping, we propose to partition the set of nodes allocated to the user application according to their coordinates, and design a recursive bipartitioning algorithm (see Fig. 5) for finding a proper mapping. Both the communication graph and the set of compute nodes are bipartitioned recursively until the mapping is derived. We partition the set of compute nodes

Algorithm 2 Recursive Bipartitioning Mapping

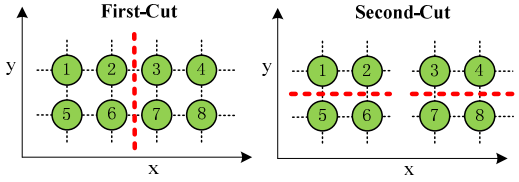
Input: communication graph $\widehat{G} = (\widehat{V}, \widehat{E})$,
the set of compute nodes $\mathcal{N}$.
Output: mapping $\phi : \widehat{V} \rightarrow \mathcal{N}$.

```
1  bipartitioning_mapping(Ĝ, N)
2  {
3    if (|N| == 1) {
4      φ(i) = N, ∀ process i ∈ V̂;
5      return;
6    }
7    // bipartition Ĝ into subgraphs Ĝ_i = (V̂_i, Ê_i),
8    // i = 1, 2; bipartition N into subsets N_1, N_2,
9    // such that |V̂_1| = |N_1| and |V̂_2| = |N_2|.
10   (Ĝ_1, Ĝ_2) ← graph_bipartition(Ĝ);
11   (N_1, N_2) ← node_bipartition(N);
12   bipartitioning_mapping(Ĝ_1, N_1);
13   bipartitioning_mapping(Ĝ_2, N_2);
14 }
```

Fig. 5 The recursive bipartitioning mapping algorithm for inter-node mapping on torus topology.



(a) Mesh/Torus – noncontiguous allocation



(b) Mesh/Torus – contiguous allocation

Fig. 6 Recursive bipartitioning of mesh/torus topologies. The nodes allocated to the parallel application are light green.

along their largest dimension, so that the compute nodes within the partitioned subsets are close to each other. A bipartitioning example is shown in Fig. 6. Note that for a set of compute nodes $\mathcal{S}$, the sizes of their dimensions depend on the coordinates of nodes, e.g., the size of the x-dimension is computed as

$$\max_{i \in \mathcal{S}} x_i - \min_{i \in \mathcal{S}} x_i + 1,$$

where x_i is the coordinate of node i.

Lemma 2 *Given the inter-node communication graph $\widehat{G} = (\widehat{V}, \widehat{E})$ and the coordinates of allocated compute nodes, it takes $O\big((|\widehat{V}| + |\widehat{E}|) \log |\widehat{V}|\big)$ time to find a proper mapping for torus topology by using the recursive bipartitioning mapping algorithm.*

Proof There are $O(\log |\widehat{V}|)$ levels of recursion calls, where $|\widehat{V}| = |\mathcal{N}|$ is the number of compute nodes. The graph partitioning at each level can be done in $O(|\widehat{E}|)$ time. Bipartitioning a set of n compute nodes according to their coordinates is computationally equivalent to finding the median of n random numbers. It can be done in expected $O(n)$ time. Hence, the time complexity of bipartitioning the sets of nodes at each level is $O(|\widehat{V}|)$. It follows that the time complexity of the whole recursive bipartitioning algorithm is $O\big((|\widehat{V}| + |\widehat{E}|) \log |\widehat{V}|\big)$.

In fact, the recursive bipartitioning mapping algorithm can be viewed as a special case of the recursive tree mapping algorithm, where the topology tree is a binary tree obtained by recursively bipartitioning the compute nodes in a top-down manner (the neighbor joining algorithm adaptively builds the topology tree in a bottom-up manner). According to the proof of Lemma 2, for torus topology, it takes $O\big(|\widehat{V}| \log |\widehat{V}|\big)$ time to build such a binary topology tree by partitioning the nodes according to their coordinates. This is more efficient than using the neighbor joining algorithm to build the topology tree, which takes $O\big(|\widehat{V}|^2\big)$ time. Hence, the overall mapping overhead of the recursive bipartitioning mapping algorithm would be smaller than that of the recursive tree mapping algorithm. However, as the recursive bipartitioning mapping algorithm only uses the coordinates for partitioning, it does not consider the wrap-around links of torus topology, while the recursive tree mapping algorithm can handle wrap-around links since it constructs the topology tree from hop distances. Despite these differences, both algorithms tend to derive similar mappings for torus topology with continuous allocation as shown by the example in Fig. 3 (c) and Fig. 6 (b). These two mapping algorithms are compared in Section 5.

3.2 Intra-Node Mapping

After the inter-node mapping procedure is done, the processes to be mapped onto each compute node are determined. For each compute node, an undirected

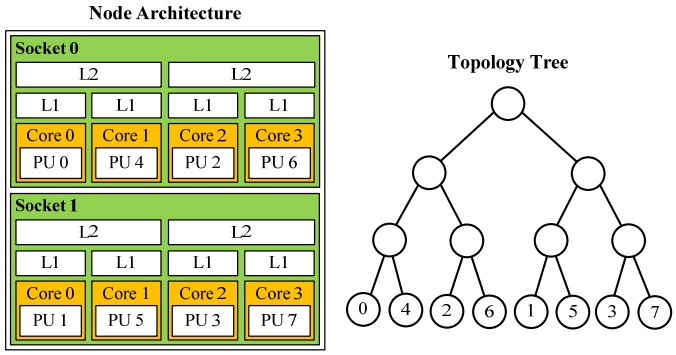


Fig. 7 A node architecture and its corresponding topology tree.

graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ can be constructed to represent the communication pattern between the processes (to be mapped onto it), and then used for intra-node mapping.

The Portable Hardware Locality (hwloc) library [Broquedis et al(2010)Broquedis, Clet-Ortega, Moreaud, Furmento, Goglin, Mercier, Thibault, and Namyst] detects the architectural components of compute nodes, including NUMA memory nodes, processor sockets, cores, processing units (PU, i.e. logical processors or "threads"), etc. The hierarchical topology of the architecture is represented as a tree, whose leaves are processing units as shown in Fig. 7. Each non-leaf node represents a group of processing units, which are leaves of the subtree rooted at the non-leaf node itself.

In order to take advantage of the hierarchical architecture, we propose to apply the recursive tree mapping algorithm (see Fig. 4) to find a proper mapping by using the communication graph $\widetilde{G}$ and the intra-node topology tree. The heavily communicating processes are mapped onto neighboring processing units for efficient communication. In particular, the mapping is derived in a top-down manner, which is different from the bottom-up TreeMatch algorithm proposed in [Jeannot and Mercier(2010)]. More importantly, this top-down strategy enables us to better optimize the mapping globally. To the best of our knowledge, this is the first time that the recursive tree mapping algorithm is utilized for intra-node mapping.

Lemma 3 *Given the intra-node communication graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ and the intra-node topology tree, it takes $O\big(|\widetilde{E}| \log |\widetilde{V}|\big)$ time to find a proper intra-node mapping by using the recursive tree mapping algorithm.*

Proof Lemma 3 follows from Lemma 1.

4 HierTopoMap: Hierarchical Task Mapping Library

We have implemented the proposed mapping algorithms into a user-level library called HierTopoMap. It is written in C++, and the source is available

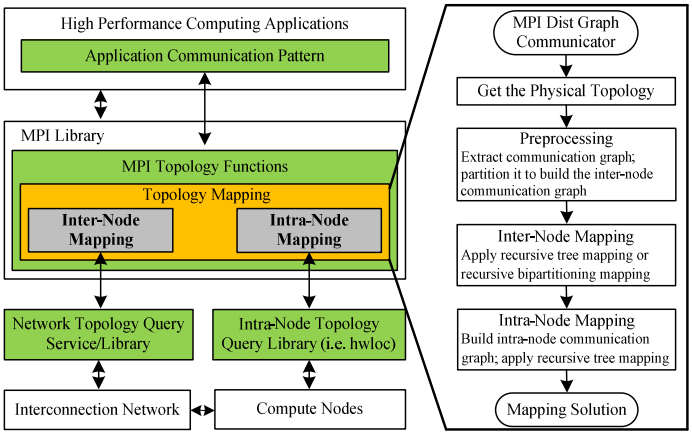


Fig. 8 Overview of our HierTopoMap library for hierarchical task mapping of parallel applications.

at [HTM(2014)]. It has about 4K lines of code excluding external libraries. Currently, HierTopoMap supports InfiniBand-connected supercomputers (i.e., fat-tree topology), Cray XT5 machines (i.e., torus topology with non-contiguous allocation), and IBM Blue Gene/P systems (i.e. torus topology with contiguous allocation). For InfiniBand network, it utilizes the topology query service to get the distance between compute nodes. On Cray XT5 and IBM Blue Gene/P machines, it uses the machine specific topology query libraries to get the network topology. For intra-node mapping, our library utilizes the Portable Hardware Locality (hwloc) library [Broquedis et al(2010)Broquedis, Clet-Ortega, Moreaud, Furmento, Goglin, Mercier, Thibault, and Namyst] to detect the intra-node tree topology.

Fig. 8 presents the overall topology mapping framework and the flow of our hierarchical task mapping library. Our library accepts an MPI Dist graph communicator [MPI(2012)] as input, and computes an optimized mapping, i.e., a rank reordering. It can be used to support MPI topology functions. The major APIs are

- `int HTM_Topomap(MPI_Comm distgr, int *newrank);`
  The generic API for hierarchical task mapping. By default, it uses the recursive tree mapping algorithm for inter-node mapping on fat-tree topology, and the recursive bipartitioning mapping algorithm for inter-node mapping on torus topology.
- `int HTM_Topomap_tree(MPI_Comm distgr, int *newrank);`
  This API performs hierarchical task mapping with the recursive tree mapping algorithm for inter-node mapping.
- `int HTM_Topomap_bipart(MPI_Comm distgr, int *newrank);` This API performs hierarchical task mapping with the recursive bipartitioning mapping algorithm for inter-node mapping. It is only applicable for torus topology.

Table 1 Experimental Platforms

Production Supercompuer	TACC Stampede [Sta(2014)]	NICS Kraken [Kra(2013)]	ALCF Intrepid [Int(2013)]
System	Dell Linux Cluster	Cray XT5	IBM Blue Gene/P
Number of Cores	462,462	112,896	163,840
Network Topology	2-level Fat-Tree	3D Torus (non-contiguous allocation)	3D Torus (contiguous allocation)
Processor(s) per Node	2 eight-core Xeon E5	2 six-core AMD Opteron	1 quad-core PowerPC
Processor Frequency	2.7 GHz	2.6 GHz	850 MHz
Memory per Node	32 GB	16 GB	2 GB
Ranking in Top500 list (June 2013)	6th	30th	58th

Specifically, our library extracts the communication graph from the MPI Dist graph communicator, and uses a graph partitioner to build the inter-node communication graph. Then inter-node mapping and intra-node mapping are performed sequentially. Note that each compute node performs intra-node mapping for itself, respectively. The graph partitioning tool METIS [MET(2013)] is employed to partition the communication graphs. As the partitioning returned by METIS may be imbalanced, we correct the imbalanced partitioning by greedily moving vertices from overloaded partitions to underloaded partitions. The correction step moves the node, which results in the minimum overall edgecut after movement.

5 Experiments

In this section, we evaluate the performance of the proposed hierarchical mapping strategy on production supercomputers by using a set of benchmarks and applications.

5.1 Experimental Setup

5.1.1 Experimental Platforms

Experiments are carried out on three production machines: Stampede at TACC [Sta(2014)], Kraken at NICS [Kra(2013)], and Intrepid at Argonne Leadership Computing Facility [Int(2013)]. Stampede is a Dell Linux cluster interconnected by Mellanox FDR InfiniBand network in a two-level fat-tree topology. It has $6,400+$ Dell PowerEdge server nodes, each of which has two eight-core Xeon E5 processors. Kraken is a Cray XT5 system with a 3D torus network topology. It is comprised of $9,408$ compute nodes and each node contains two six-core AMD Opteron processors. Intrepid is an IBM Blue Gene/P (BG/P) system with a 3D torus network for point-to-point message passing. It consists of 40 racks with 1024 nodes per rack. Each node is equipped with a quad-core PowerPC processor. The peak performance of these chosen supercomputers ranked 6th, 30th, and 58th in the topo500 list (June 2013) [Top(2014)], respectively. Table 1 lists the major system specifications of these machines. It

Table 2 Properties of the Test Suite

Test Case	Comm. Pattern	Application Problem
3D Stencil	regular	Computation on a regular 3D mesh
F1	irregular	Matrix computation for solving the elasticities of automotive crankshafts
audikw_1	irregular	
nlpkkt120	irregular	Matrix computation for a 3D PDE-constrained optimization problem
ART	irregular	Cosmology simulation on a 3D structured adaptive mesh

is to be noted that Kraken and Intrepid have different node allocation strategies. The nodes allocated to a parallel application on Cray XT5 systems may be discrete in the 3D torus network, while IBM Blue Gene machines allocate compute nodes interconnected by a regular 3D mesh/torus network to each job[2].

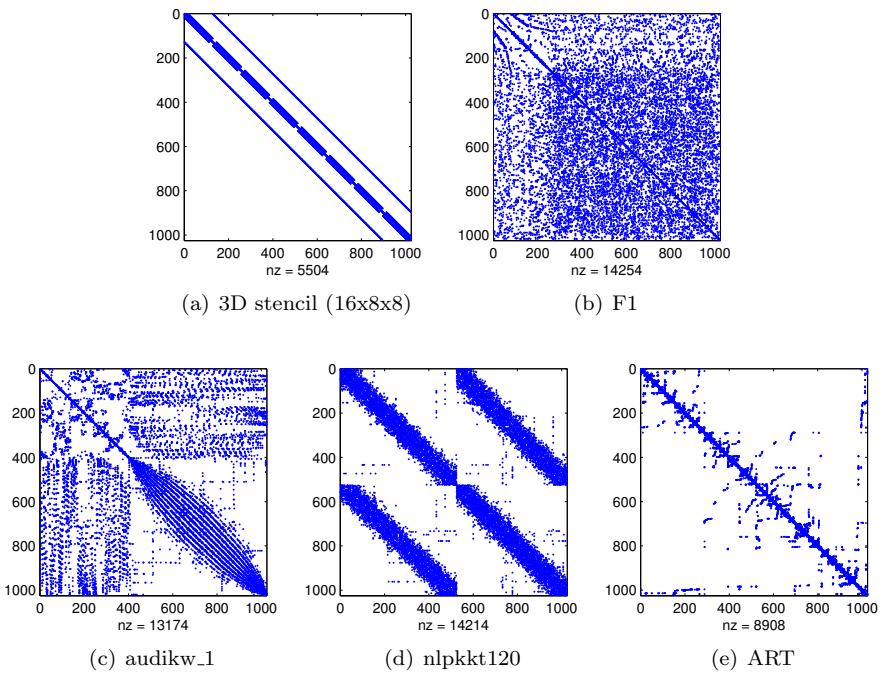
5.1.2 Test Suite

Our test suite consists of five benchmarks and/or applications, each representing different application communication patterns. Our first test case is a 3D stencil benchmark. The processes are mapped onto a 3D grid, and each process communicates with its immediate neighbors in each dimension. The inter-process communication is implemented by posting non-blocking receives MPI_Irecv() and non-blocking sends MPI_Isend(), followed by a single MPI_Waitall() for all sends and receives.

We also use three sparse matrices from the University of Florida Sparse Matrix Collection [Davis and Hu(2011)] as our test cases. They are F1, audikw_1 and nlpkkt120. The first two matrices are symmetric stiffness matrices, which model the elasticities of automotive crankshafts. The third one is a symmetric indefinite KKT matrix, which represents a nonlinear programming problem for a 3D PDE-constrained optimization. In order to perform matrix computation efficiently, scientific codes often use a graph partitioner to decompose the matrix into sub-matrices, and then use multiple processes/threads to compute in parallel. The communication between processes depends on the decomposition and the structure of the matrix. In our experiments, we use METIS [MET(2013)] to partition the sparse matrices, and analyze the inter-process communication pattern, which is then utilized for communication tests.

In addition, we experimented with a real cosmology simulation code called Adaptive Refinement Tree (ART) [Kravtsov et al(1997)Kravtsov, Klypin, and Khokhlov, Wu et al(2011)Wu, Gonzalez, Lan, Gnedin, Kravtsov, Rudd, and Yu, Yu et al(2012)Yu, Rudd, Lan, Gnedin, Kravtsov, and Wu]. The application uses a cubic computational domain to model the universe, and employs adaptive mesh refinement (AMR) [Plewa et al(2005)Plewa, Linde, and Weirs] for

[2] On Blue Gene/P systems, if the number of allocated nodes is less than 512, then their network topology is a mesh.



(a) 3D stencil (16x8x8) (b) F1

(c) audikw_1 (d) nlpkkt120 (e) ART

Fig. 9 The communication pattern of the test suite (1024 processes).

efficient computation. The computational domain is divided into many cells, which are refined and de-refined dynamically during simulation. Each process performs simulation for a local computational domain, and communicates with other processes to exchange updated boundary information. The communication is mainly nearest-neighbor exchanges, which are representative of many scientific applications. As ART simulations consume a large amount of computing resources, we extract the communication part of a production ART simulation for performance tests.

Table 2 summarizes the properties of our test suite, and Fig. 9 shows the communication pattern of these test cases on 1024 processes. Each blue dot at (i, j) denotes the communication between processes i and j, and "nz" is the total number of blue dots. The 3D stencil benchmark has a regular communication pattern, while the three matrix tests show highly irregular communication. Most communication of ART is between neighboring processes as shown by its sparse and diagonally dominant pattern.

5.1.3 Mapping Mechanisms

The following mapping mechanisms are evaluated:

1. The system default mapping, which is topology-agnostic;
2. The pure inter-node mapping by using the recursive tree (RT) mapping algorithm shown in Fig. 4, denoted by "RT:inter";

Table 3 Mapping Mechanisms Tested on Each Machine

Machine	Stampede	Kraken	Intrepid
system default	Yes	Yes	Yes
RT:inter	Yes	Yes	Yes
RT:inter+intra	Yes	Yes	No
RB:inter	No	Yes	Yes
RB:inter+intra	No	Yes	No
LibTopoMap [Lib(2011)]	No	No	Yes

3. "RT:inter" + intra-node mapping, denoted by "RT:inter+intra";
4. The pure inter-node mapping by using the recursive bipartitioning (RB) mapping algorithm shown in Fig. 5, denoted by "RB:inter";
5. "RB:inter" + intra-node mapping, denoted by "RB:inter+intra".

As the proposed recursive bipartitioning (RB) mapping algorithm is not applicable for fat-tree topology, we only evaluate the first three mechanisms on Stampede, while all of these mechanisms are tested on Kraken. Intrepid does not have a hierarchical intra-node topology, so we do not perform intra-node mapping on it.

Furthermore, we compare the above mapping mechanisms with the generic topology mapping library libTopoMap [Hoefler and Snir(2011),Lib(2011)]. Unfortunately, we do not have the root permission to obtain the physical interconnect topology of Stampede, which is required by LibTopoMap for topology mapping, and LibTopoMap cannot handle the topology mapping on Kraken due to the non-contiguous node allocation. Hence, we only experimented with LibTopoMap on Intrepid. Table 3 summarizes the experiments conducted on these production machines. *Hop-bytes* and *communication time* are employed as the evaluation metrics.

5.1.4 Execution Setups

In all the experiments, we assign an MPI process on each core, and run jobs in production mode without dedicated nodes, i.e. there are other users sharing the interconnection network (on Stampede and Kraken). It is to be noted that different runs often get different compute nodes, and the interference of other running applications may also be different (especially for Stampede and Kraken). To fairly compare the mapping mechanisms, for each set of experiments with a particular number of processes, we run all the tests with different mapping mechanisms in a single batch script.

5.2 Results

Before we show the experimental results to evaluate our hierarchical task mapping schemes, we first present the average intra-socket and inter-socket PingPing communication times on Stampede and Kraken (see Table 4). The table clearly shows that the intra-socket communication is faster than the

Table 4 Average Intra-Socket and Inter-Socket Communication Time (PingPing)

#bytes	#repetitions	Stampede (us)			Kraken (us)		
		Intra.	Inter.	difference	Intra.	Inter.	difference
0	1000	0.51	0.69	35.33%	0.56	0.81	43.38%
1	1000	0.53	0.73	38.40%	0.58	0.74	27.57%
2	1000	0.52	0.73	38.96%	0.59	0.74	26.96%
4	1000	0.52	0.73	39.14%	0.60	0.75	25.26%
8	1000	0.52	0.71	37.44%	0.62	0.74	20.89%
16	1000	0.52	0.75	42.35%	0.61	0.75	21.90%
32	1000	0.53	0.77	44.84%	0.64	0.83	28.95%
64	1000	0.54	0.79	47.54%	0.65	0.84	30.43%
128	1000	0.55	0.86	55.91%	1.09	1.31	20.06%
256	1000	0.59	1.03	73.17%	1.16	1.39	19.82%
512	1000	0.64	1.15	79.36%	1.37	1.64	19.84%
1024	1000	0.73	1.34	82.42%	1.75	2.08	19.19%
2048	1000	0.92	1.68	83.29%	2.57	3.02	17.77%
4096	1000	1.31	2.27	73.08%	4.39	5.17	17.76%
8192	1000	3.13	3.87	23.74%	7.30	8.87	21.45%
16384	1000	3.88	4.61	18.89%	14.64	16.95	15.76%
32768	1000	5.10	5.89	15.34%	27.86	32.63	17.13%
65536	640	7.67	8.50	10.78%	53.92	64.14	18.94%
131072	320	13.68	14.62	6.85%	36.74	38.11	3.74%
262144	160	28.47	29.27	2.83%	71.09	71.76	0.93%

inter-socket communication. For small- and medium-sized messages, as the communication time is relatively small, the performance gap is significant. In other words, the table demonstrates the necessity to consider the performance gap between intra-socket and inter-socket and perform intra-node mapping for communication optimization on production systems.

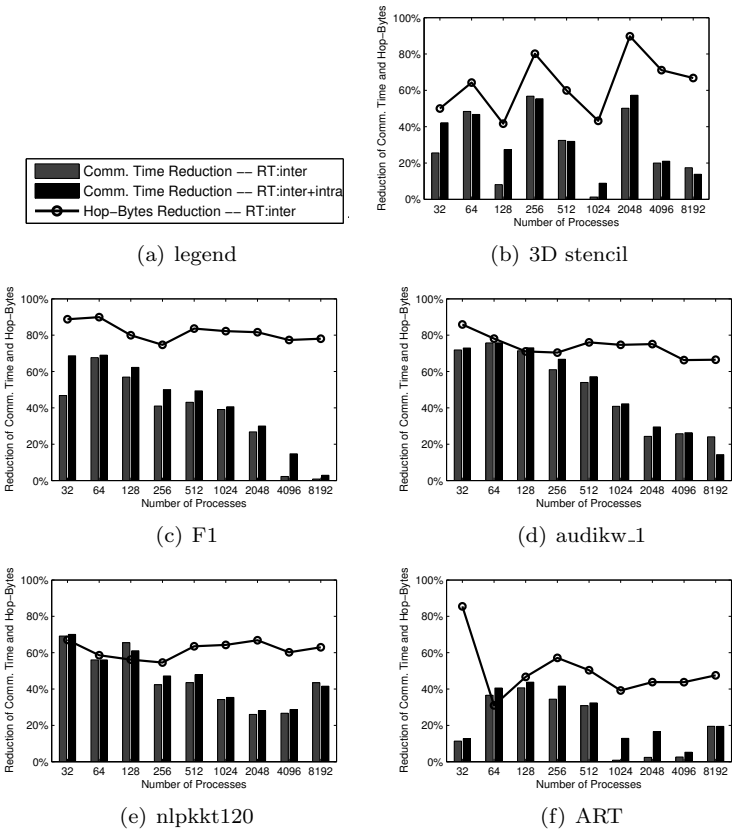
5.2.1 Comparison of Mapping Mechanisms

To demonstrate the performance of mapping mechanisms, our results are structured to answer the following questions:

(1): For fat-tree topology, how much performance gain can we achieve by using the recursive tree mapping algorithm for inter-node mapping? How much additional performance improvement can we get by performing both inter-node mapping and intra-node mapping?

As shown in Fig. 10, on Stampede, the inter-node mapping with the recursive tree mapping algorithm significantly reduces hop-bytes by up to 90%, and reduces the communication time by up to 76%. By performing inter-node mapping and intra-node mapping in a hierarchical manner, we can further improve the communication performance for many cases. In the best case ("F1" test with 32 processes), the additional performance gain of intra-node mapping is up to 22%. For ART and the sparse matrix tests, as the number of processes increases, the performance gain of task mapping tends to decrease. That's because we fixed the original application problem sizes. When more processes are used, the inter-process message sizes would become smaller, and the fat-tree network tends to be less congested.

(2): For torus topology with non-contiguous allocation, we can use the recursive tree mapping algorithm or the recursive bipartitioning mapping algorithm for inter-node mapping. Which algorithm provides better performance?



(a) legend (b) 3D stencil

(c) F1 (d) audikw_1

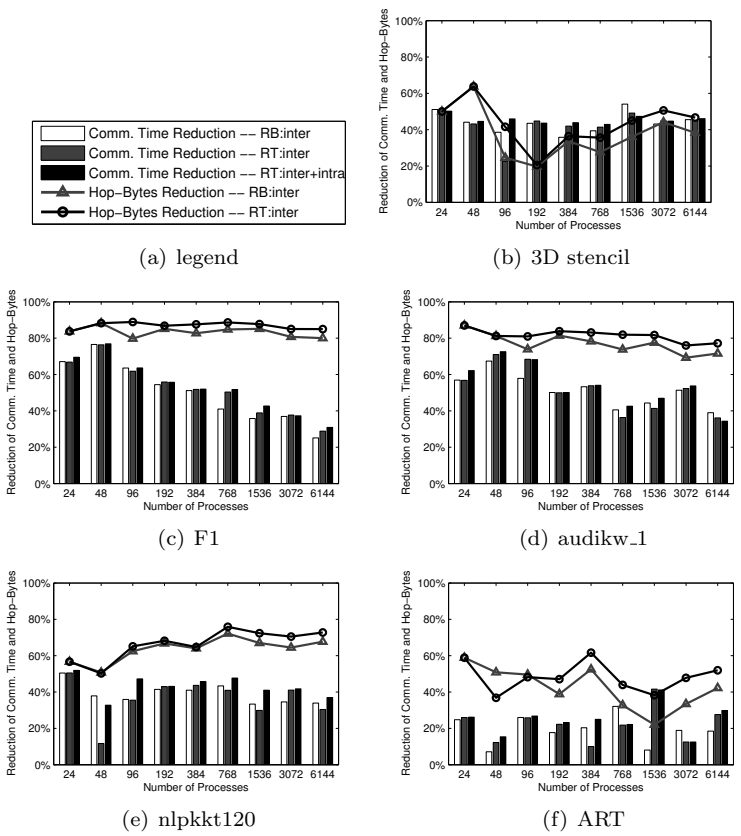
(e) nlpkkt120 (f) ART

Fig. 10 Performance gain of topology mapping (relative to the system default mapping) on Stampede – fat-tree topology.

How much performance gain can we achieve? How much additional performance improvement can we get by performing both inter-node mapping and intra-node mapping?

Fig. 11 presents the performance test results on Kraken. As the performance of "RB:inter+intra" is similar to that of "RT:inter+intra", it is omitted in the figure for succinct presentation. The recursive tree mapping algorithm typically achieves more hop-bytes reduction than the recursive bipartitioning mapping algorithm, since it considers the wrap-around links by using the hop distances to construct the topology tree, while the recursive bipartitioning mapping algorithm ignores the wrap-around links. However, these two algorithms lead to similar communication time reduction on average. Overall, the hop-bytes reduction is up to 89%, and the communication time reduction is up to 77%. The intra-node mapping step achieves up to 23% additional communication time reduction (the best case is "3D stencil" test with 96 processes).

(3): For torus topology with contiguous allocation, we can also use the recursive tree mapping algorithm or the recursive bipartitioning mapping algo-



(a) legend (b) 3D stencil

(c) F1 (d) audikw_1

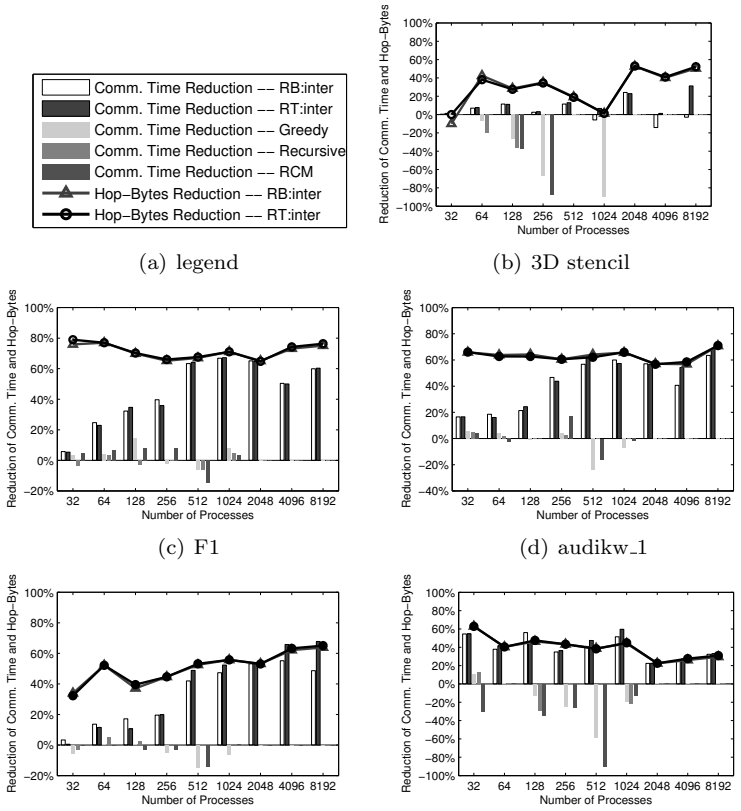
(e) nlpkkt120 (f) ART

Fig. 11 Performance gain of topology mapping (relative to the system default mapping) on Kraken – 3D torus topology with non-contiguous allocation.

rithm for inter-node mapping. Which algorithm provides better performance? How much performance gain can we achieve?

Fig. 12 shows the performance results on Intrepid. "Greedy", "Recursive" and "RCM" represent the three mapping mechanisms of LibTopMap [Lib(2011)]: a greedy heuristic, a recursive bipartitioning algorithm, and an RCM ordering-based approach. Note that the recursive bipartitioning algorithm of LibTopMap partitions the topology graph with a graph partitioner, while our recursive bipartitioning mapping algorithm partitions the compute nodes according to their coordinates. Moreover, LibTopMap aims to minimize the network *congestion*, which represents the worst-case contention among all links in the network. It uses a heuristic to compute the congestion in order to evaluate the quality of a mapping, and returns an optimized mapping with reduced congestion if such a mapping is found. In our experiments, LibTopMap failed to derive optimized mappings for some test cases, so the corresponding performance gain is 0 (i.e., no bar is shown).

Clearly, both the recursive tree mapping algorithm and the recursive bipartitioning mapping algorithm achieve similar hop-bytes reduction, and they also



(a) legend (b) 3D stencil

(c) F1 (d) audikw_1

(e) nlpkkt120 (f) ART

Fig. 12 Performance gain of topology mapping (relative to the system default mapping) on Intrepid – 3D torus topology with contiguous allocation.

achieve similar communication time reduction for most test cases. This confirms the observation (from Fig. 3 (c) and Fig. 6 (b)) that these two algorithms provide similar mappings for torus topology with contiguous allocation. More importantly, both mapping algorithms significantly outperform LibTopMap, achieving hop-bytes reduction by up to 79% and communication time reduction by up to 69%. As the number of processes increases, the performance gain of our mapping algorithms tends to increase for sparse matrix tests, indicating the importance of task mapping at large-scale.

Although LibTopMap can reduce the congestion for many test cases, it may increase the communication cost, and can only achieve minor performance gain in the best case. This is due to the fact that congestion is an indirect measure of the communication time. It represents the data transmission time of the most congested link. However, the actual communication time is more likely to be dependent on the aggregate congestion of multiple links, which is equivalent to hop-bytes if we sum up the congestion of all links (assume that the links have identical bandwidth).

Table 5 Mapping Overhead on Stampede – Fat-Tree Topology (Time in Seconds)

Number of Processes		256	512	1024	2048	4096	8192
RT:inter+intra	Build Topology Tree	0.000	0.001	0.001	0.007	0.008	0.032
	Preprocessing	0.003	0.005	0.024	0.030	0.120	0.261
	Inter-Node Mapping	0.000	0.001	0.001	0.002	0.005	0.029
	Intra-Node Mapping	0.000	0.000	0.000	0.001	0.002	0.002
	Total	0.003	0.007	0.027	0.040	0.135	0.324

Table 6 Mapping Overhead on Kraken – 3D Torus Topology with Non-Contiguous Allocation (Time in Seconds)

Number of Processes		192	384	768	1536	3072	6144
RT:inter+intra	Build Topology Tree	0.000	0.001	0.003	0.011	0.042	0.180
	Preprocessing	0.006	0.012	0.022	0.053	0.088	0.192
	Inter-Node Mapping	0.001	0.002	0.012	0.012	0.021	0.045
	Intra-Node Mapping	0.001	0.001	0.007	0.005	0.001	0.007
	Total	0.008	0.016	0.044	0.081	0.152	0.425
RB:inter+intra	Preprocessing	0.006	0.013	0.026	0.046	0.102	0.198
	Inter-Node Mapping	0.001	0.002	0.005	0.021	0.043	0.054
	Intra-Node Mapping	0.001	0.001	0.002	0.003	0.004	0.007
	Total	0.008	0.016	0.033	0.070	0.149	0.259

Table 7 Mapping Overhead on Intrepid – 3D Torus Topology with Contiguous Allocation (Time in Seconds)

Number of Processes		256	512	1024	2048	4096	8192
RT:inter	Build Topology Tree	0.040	0.155	0.608	2.438	9.724	38.738
	Preprocessing	0.079	0.167	0.351	0.732	1.536	3.229
	Inter-Node Mapping	0.051	0.120	0.277	0.634	1.424	3.175
	Total	0.170	0.442	1.236	3.804	12.683	45.143
RB:inter	Preprocessing	0.082	0.174	0.366	0.763	1.596	3.354
	Inter-Node Mapping	0.071	0.162	0.364	0.824	1.828	4.014
	Total	0.153	0.336	0.730	1.586	3.424	7.368
LibTopoMap	greedy	0.096	0.380	1.525	NA	NA	NA
	recursive	12.324	36.736	133.052	NA	NA	NA
	rcm	0.005	0.012	0.028	NA	NA	NA

5.2.2 Mapping Overhead

Mapping overhead is another important part of performance comparison. Tables 5 to 7 list the mapping overhead for the sparse matrix test "nlpkkt120" on Stampede, Kraken and Intrepid, respectively. The mapping overhead for other test cases is similar. We report the cost for each step of the hierarchical task mapping strategy, and explicitly list the execution time to build the inter-node topology tree by using the neighbor joining algorithm.

On Stampede, the neighbor joining algorithm efficiently builds the inter-node topology tree with low cost, and most of the mapping overhead is due to the preprocessing stage, which partitions the original communication graph into number of nodes parts, and builds the inter-node communication graph. The inter-node mapping stage is fairly efficient, and the cost for intra-node mapping is negligible.

On Kraken, the neighbor joining algorithm takes more execution time to build the inter-node topology tree, since it needs more iterations to construct the binary tree representation of torus topology than to detect fat-tree topology. The cost for preprocessing, inter-node mapping and intra-node mapping is similar to that of Stampede (consider the difference in their CPU performance). Most mapping overhead is due to preprocessing and building the

inter-node topology tree. Specifically, the recursive tree mapping algorithm has larger overhead than the recursive bipartitioning mapping algorithm. This is attributable to the fact that building the inter-node topology tree is more expensive than recursively bipartitioning the compute nodes according to their coordinates. As the number of processes increases, it is observed that the execution time for intra-node mapping may increase because of the cost for building the intra-node communication graph.

On Intrepid, the mapping overhead is much larger than that of Stampede and Kraken due to the relatively low processing power of its CPU and the small amount memory per core. As the cost for building the inter-node topology tree dominates the mapping overhead, the recursive tree mapping algorithm is more costly than the recursive bipartitioning mapping algorithm. Table 7 also lists the execution time of LibTopMap for comparison [3]. The mapping overhead of our algorithms is comparable to that of the greedy heuristic of LibTopMap.

5.2.3 Result Summary

In general, for compute nodes with a hierarchical architecture, performing intra-node mapping after the inter-node mapping stage can achieve better communication performance. As the number of processes per node is small, the overhead for intra-node mapping is negligible. Consider that more and more CPU cores are put on a compute node, the intra-node topology mapping will become increasingly important.

The recursive tree mapping algorithm is highly efficient for inter-node mapping onto fat-tree topology and intra-node mapping, and it can also be applied for inter-node mapping onto torus topology. Our recursive bipartitioning mapping algorithm efficiently partitions the compute nodes in torus topology according to their coordinates. Its mapping overhead is smaller than that of the recursive bipartitioning algorithm while achieving comparable communication performance. Consider that communication-intensive applications often have substantial amount of communication time, e.g., the ART code spends up to several hundreds of seconds on inter-process communication for simulating a single time step, the mapping overhead of our mapping mechanisms is relatively small compared to the reduction in communication time, and thus being suitable for practical use.

6 Conclusion

In this paper, we have presented a hierarchical task mapping strategy for modern supercomputers. It finds a proper mapping of processes onto processors by performing inter-node mapping and intra-node mapping in a hierarchical manner. Moreover, we have introduced a generic recursive tree mapping algorithm for inter-node mapping and intra-node mapping, and designed a

[3] LibTopMap failed to derive mapping solutions when the number of processes is larger than 1024.

recursive bipartitioning mapping algorithm for torus topology. The proposed mapping algorithms have been implemented in a hierarchical task mapping library, which supports fat-tree and torus network topologies. Our experiments with a set of benchmarks/applications on three production systems show that our design can effectively reduce communication cost with very low runtime overhead. For inter-node mapping, the recursive tree mapping is suitable for fat-tree topology, while the recursive bipartitioning mapping algorithm is more practical for torus topology due to its low overhead. In addition, performing intra-node mapping can further improve the communication performance.

In future, we plan to extend to the proposed mapping algorithms to handle more complex networks such as 5D Torus, and to extend the hierarchical task mapping library to support more platforms.

Acknowledgements This work is supported in part by National Science Foundation grant OCI-0904670.

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper. URL: `http://www.tacc.utexas.edu`

This material is based upon work supported by the National Science Foundation under Grant numbers 0711134, 0933959, 1041709, and 1041710 and the University of Tennessee through the use of the Kraken computing resource at the National Institute for Computational Sciences (`http://www.nics.tennessee.edu`).

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.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

References

[Lib(2011)] (2011) LibTopoMap: A Generic Topology Mapping Library. `http://www.unixer.de/research/mpitopo/libtopomap/`

[MPI(2012)] (2012) MPI: A message-passing interface standard. version 3.0. `http://www.mpi-forum.org/`

[Int(2013)] (2013) ALCF Intrepid. `https://www.alcf.anl.gov/intrepid`

[MET(2013)] (2013) METIS: Graph Partitioning Tool. `http://glaros.dtc.umn.edu/gkhome/views/metis`

[Kra(2013)] (2013) NICS Kraken User Guide. `https://www.xsede.org/web/guest/nics-kraken`

[Sta(2014)] (2014) TACC Stampede User Guide. `http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide`

[Top(2014)] (2014) Top 500 Supercomputer Sites. `http://www.top500.org/`

[HTM(2014)] (2014) TOPOMap. `http://bluesky.cs.iit.edu/topomap/`

[Abts(2011)] Abts D (2011) The Cray XT4 and Seastar 3-D Torus interconnect. Encyclopedia of Parallel Computing pp 470–477

[Agarwal et al(2006)Agarwal, Sharma, Laxmikant, and Kale] Agarwal T, Sharma A, Laxmikant A, Kale LV (2006) Topology-aware task mapping for reducing communication contention on large parallel machines. In: Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS)

[Aleliunas and Rosenberg(1982)] Aleliunas R, Rosenberg AL (1982) On embedding rectangular grids in square grids. Computers, IEEE Transactions on C-31(9):907–913, DOI 10.1109/TC.1982.1676109

[Berman and Snyder(1987)] Berman F, Snyder L (1987) On mapping parallel algorithms into parallel architectures. Journal of Parallel and Distributed Computing 4(5):439–458

[Bhatele(2010)] Bhatele A (2010) Automating topology aware mapping for supercomputers. PhD thesis, University of Illinois at Urbana-Champaign, Urbana

[Bhatele and Kale(2008)] Bhatele A, Kale LV (2008) Application-specific topology-aware mapping for three dimensional topologies. In: Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp 1–8

[Bokhari(1981)] Bokhari SH (1981) On the mapping problem. IEEE Transactions on Computers 30(3):207–214

[Broquedis et al(2010)Broquedis, Clet-Ortega, Moreaud, Furmento, Goglin, Mercier, Thibault, and Namyst] Broquedis F, Clet-Ortega J, Moreaud S, Furmento N, Goglin B, Mercier G, Thibault S, Namyst R (2010) hwloc: A generic framework for managing hardware affinities in hpc applications. In: Proc. the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp 180–186, DOI 10.1109/PDP.2010.67

[Chockalingam and Arunkumar(1992)] Chockalingam T, Arunkumar S (1992) A randomized heuristics for the mapping problem: The genetic approach. Parallel Computing 18(10):1157–1165

[Chung et al(2011)Chung, Lee, Zhou, and Chung] Chung IH, Lee CR, Zhou J, Chung YC (2011) Hierarchical mapping for HPC applications. In: Proc. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp 1815–1823

[Cuthill and McKee(1969)] Cuthill E, McKee J (1969) Reducing the bandwidth of sparse symmetric matrices. In: Proc. the 24th National Conference, pp 157–172

[Davis and Hu(2011)] Davis TA, Hu Y (2011) The university of florida sparse matrix collection. ACM Trans Math Softw 38(1):1–25

[Ercal et al(1988)Ercal, Ramanujam, and Sadayappan] Ercal F, Ramanujam J, Sadayappan P (1988) Task allocation onto a hypercube by recursive mincut bipartitioning. In: Proc. the third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues - Volume 1, C3P, pp 210–221

[Hoefler and Snir(2011)] Hoefler T, Snir M (2011) Generic topology mapping strategies for large-scale parallel architectures. In: Proc. the international conference on Supercomputing (ICS), pp 75–84

[Jeannot and Mercier(2010)] Jeannot E, Mercier G (2010) Near-optimal placement of MPI processes on hierarchical NUMA architectures. In: Proc. the 16th International Euro-Par Conference on Parallel Processing: Part II, pp 199–210

[Karypis and Kumar(1998)] Karypis G, Kumar V (1998) Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing 48(1):96–129

[Kravtsov et al(1997)Kravtsov, Klypin, and Khokhlov] Kravtsov AV, Klypin AA, Khokhlov AM (1997) Adaptive refinement tree: A new high-resolution N-body code for cosmological simulations. The Astrophysical Journal Supplement Series 111:73–94

[Lee and Bic(1989)] Lee C, Bic L (1989) On the mapping problem using simulated annealing. In: Proc. International Phoenix Conference on Computers and Communications, pp 40–44, DOI 10.1109/PCCC.1989.37357

[Leiserson(1985)] Leiserson CE (1985) Fat-trees: Universal networks for hardware-efficient supercomputing. IEEE Transactions on Computers 34(10):892–901

[Mercier and Jeannot(2011)] Mercier G, Jeannot E (2011) Improving MPI applications performance on multicore clusters with rank reordering. In: Proc. the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface, pp 39–49

[Pellegrini(1994)] Pellegrini F (1994) Static mapping by dual recursive bipartitioning of process architecture graphs. In: Proc. the Scalable High-Performance Computing Conference, pp 486–493

[pellegrini and Roman(1996)] pellegrini F, Roman J (1996) Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In: High-Performance Computing and Networking, Lecture Notes in Computer Science, vol 1067, pp 493–498

[Plewa et al(2005)Plewa, Linde, and Weirs] Plewa T, Linde T, Weirs VG (2005) Adaptive Mesh Refinement–Theory and Applications. Springer, Berlin

[Rashti et al(2011)Rashti, Green, Balaji, Afsahi, and Gropp] Rashti MJ, Green J, Balaji P, Afsahi A, Gropp W (2011) Multi-core and network aware MPI topology functions. In: Proc. the 18th European MPI Users' Group conference on Recent Advances in the Message Passing Interface, pp 50–60

[Salman et al(2002)Salman, Ahmad, and Al-Madani] Salman A, Ahmad I, Al-Madani S (2002) Particle swarm optimization for task assignment problem. Microprocessors and Microsystems 26(8):363–371

[Subramoni et al(2012)Subramoni, Potluri, Kandalla, Barth, Vienne, Keasler, Tomko, Schulz, Moody, and Panda] Subramoni H, Potluri S, Kandalla K, Barth B, Vienne J, Keasler J, Tomko K, Schulz K, Moody A, Panda D (2012) Design of a scalable infiniband topology service to enable network-topology-aware placement of processes. In: Proc. International Conference on High Performance Computing, Networking, Storage and Analysis, pp 1–12, DOI 10.1109/SC.2012.47

[Tang et al(2011)Tang, Lan, Desai, Buettner, and Yu] Tang W, Lan Z, Desai N, Buettner D, Yu Y (2011) Reducing fragmentation on torus-connected supercomputers. In: Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp 828–839

[Träff(2002)] Träff JL (2002) Implementing the MPI process topology mechanism. In: Proc. ACM/IEEE conference on Supercomputing, pp 28:1–28:14

[Wu et al(2011)Wu, Gonzalez, Lan, Gnedin, Kravtsov, Rudd, and Yu] Wu J, Gonzalez RE, Lan Z, Gnedin NY, Kravtsov AV, Rudd DH, Yu Y (2011) Performance emulation of cell-based AMR cosmology simulations. In: Proc. IEEE International Conference on Cluster Computing (CLUSTER), pp 8–16

[Wu et al(2012)Wu, Lan, Xiong, Gnedin, and Kravtsov] Wu J, Lan Z, Xiong X, Gnedin NY, Kravtsov AV (2012) Hierarchical task mapping of cell-based AMR cosmology simulations. In: Proc. International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pp 75:1–75:10

[Yu et al(2006)Yu, Chung, and Moreira] Yu H, Chung IH, Moreira J (2006) Topology mapping for Blue Gene/L supercomputer. In: Proc. ACM/IEEE Conference on Supercomputing, p 52, DOI 10.1109/SC.2006.63

[Yu et al(2012)Yu, Rudd, Lan, Gnedin, Kravtsov, and Wu] Yu Y, Rudd DH, Lan Z, Gnedin NY, Kravtsov AV, Wu J (2012) Improving parallel IO performance of cell-based AMR cosmology applications. In: Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp 933–944