PTRAM: A Parallel Topology- and Routing-Aware Mapping Framework for Large-Scale HPC Systems

Seyed H. Mirsadeghi and Ahmad Afsahi
ECE Department, Queen's University, Kingston, ON, Canada, K7L 3N6
Email: {s.mirsadeghi, ahmad.afsahi}@queensu.ca

***Abstract*—With the rapid increase in the size and scale of modern systems, topology-aware process mapping has become an important approach for improving system efficiency. A poor placement of processes across compute nodes could cause significant congestion within the interconnect. In this paper, we propose a new greedy mapping heuristic as well as a mapping refinement algorithm. The heuristic attempts to minimize a hybrid metric that we use for evaluating various mappings, whereas the refinement algorithm attempts to reduce maximum congestion directly. Moreover, we take advantage of parallelism in the design and implementation of our proposed algorithms to achieve scalability. We also use the underlying routing information in addition to the topology of the system to derive a better evaluation of congestion. Our experimental results with 4096 processes show that the proposed approach can provide more than 60% improvement in various mapping metrics compared to an initial in-order mapping of processes. Communication time is also improved by 50%. In addition, we also compare our proposed algorithms with 4 other heuristics from the LibTopoMap library, and show that we can achieve better mappings at a significantly lower cost.**

I. Introduction

As we move toward the Exascale era, both the number of nodes in a system and the number of cores within each node are rapidly increasing. HPC systems are becoming more and more complex, introducing increasing levels of heterogeneity in communication channels. In this regard, topology-aware process mapping has been shown to be a promising and necessary technique for efficient utilization of communication resources in modern large-scale systems [1]–[4]. Mapping processes based on the communication pattern of an application and the physical topology of the underlying system could prevent communication over slower channels, contention, and message transmission over long paths.

In particular, topology-aware mapping can help to reduce congestion within the interconnection network. Congestion is known to be an important factor that can highly affect communications performance. It has been shown that communications can suffer up to 6.5 times lower in bandwidth, and 5 times higher in latency in existing InfiniBand installations because of congestion [5]. Current trends show that as we move toward Exascale era, network congestion is expected to worsen in future. The rapid increase in the scale of modern systems highly increases the amount of traffic that is pushed into the network. Although network devices are persistently scaled up to meet the requirements of large scale systems, they still lag behind the rate at which the number of processing nodes is increased [6]. Therefore, we will see highly under-provisioned networks in Exascale systems, leading to higher amounts of congestion. Furthermore, as the systems scale up, network diameter tends to grow causing the messages to traverse more links to get to their destination. Consequently, each link will be shared among a higher number of messages, provoking more congestion [7].

In this paper, we propose PTRAM; a parallel topology- and rounting-aware mapping framework. Specifically, we propose two topology- and routing-aware process-to-node mapping algorithms in PTRAM. The proposed algorithms are parallel in nature, and take into account the routing information so as to derive a better evaluation of congestion across network links. For InfiniBand clusters in particular, we take into account the static assignment of routes across the links. Previous studies [5] show that most communication patterns cannot achieve full bisection bandwidth in practical fat-tree InfiniBand networks. The problem is not the number of available physical links; it is the static routing scheme which might oversubscribe some links while leaving others idle. To the best of our knowledge, this is the first work that takes into account the static routing scheme of InfiniBand to optimize the mapping.

In particular, we propose a core greedy mapping heuristic accompanied by a mapping refinement algorithm. The core heuristic attempts to minimize a hybrid metric that is used to evaluate candidate mappings from multiple aspects. The refinement algorithm on the other hand attempts to directly reduce maximum congestion by refining the mapping output from the greedy heuristic. We take advantage of parallelism in the design and implementation of our proposed algorithms. We believe parallelism is the key for having a truly scalable topology-aware mapping approach in current and future HPC systems. To the best of our knowledge, this is the first attempt in proposing parallel algorithms for topology-aware process mapping.

The rest of the paper is organized as follows. In Section II, we review the background material. Section III presents the related work. Section IV is devoted to our proposed mapping algorithms. Experimental results are provided in Section V, followed by conclusion and future directions in Section VI.

II. Background

In general, topology-aware mapping is an instance of the graph embedding problem [8] which is known to be NP-hard. Therefore, various heuristics are used to find suboptimal solutions with respect to some metric used to assess the quality of a particular mapping. *Hop-bytes* is the metric based on which the majority of mapping heuristics have been designed. For each individual message, hop-bytes represents the product of the message size and the number of hops traversed from

source to destination. The summation of such products over all communicated messages represents the hop-bytes value of a given mapping. Accordingly, mapping algorithms attempt to assign processes to processing elements so as to minimize the resulting hop-bytes value. Eliminating message sizes from hop-bytes will result in *dilation*; another metric for mapping that represents the total number of hops traversed by messages. Thus, hop-bytes can be considered as a *weighted* dilation, with the weights being the volume of messages.

Maximum congestion is another metric used in topology-aware mapping. In this context, congestion represents the accumulative traffic load that would pass through network links throughout the execution of an application. Therefore, it provides a static measure of congestion across the links. More specifically, the congestion value of each individual link is defined as the total volume of messages passing through that link, divided by the capacity of that link. Formally speaking, let $\tau : P \rightarrow N$ represent a mapping from the set of processes P to the set of nodes N. Moreover, let L denote the set of all links in the target system. For every pair of processes $(p, q) \in P$, we use $L(p, q, \tau) \subseteq L$ to denote the set of links used in the path from $\tau(p)$ to $\tau(q)$. Moreover, we denote by $SD(l, \tau)$ the set of all source-destination processes $(p, q) \in G_M$ for which the route from $\tau(p)$ to $\tau(q)$ involves link l. Note that $(p, q) \in SD(l, \tau)$ if and only if $l \in L(p, q, \tau)$. Now, the congestion value of a given link l is given by

$$\text{Congestion}(l) = \frac{\sum_{(p,q) \in SD(l,\tau)} w(p, q)}{c(l)}, \tag{1}$$

where $w(p, q)$ is the relative weight of communication between p and q, and $c(l)$ represents capacity of l. The maximum congestion metric of a mapping τ is then defined as:

$$\text{Congestion}(\tau) = \max_{l \in L} \text{Congestion}(l). \tag{2}$$

Topology-aware mapping can be considered in the context of any parallel programming paradigm. In particular, the Message Passing Interface (MPI) [9] is known to be the de facto standard for parallel programming. In MPI, communication is realized by explicit movement of data from the address space of one process to another. MPICH, MVAPICH and Open MPI are the most popular implementations of MPI. With respect to mapping, MPI provides various topology-related interfaces that *could* be used for topology-aware mapping. MPI provides a mechanism to describe the *virtual topology* of an application (also known as process topology). The virtual topology can be used to express the communication pattern of an application in terms of a graph. Each vertex of the graph represents a process while edges capture the communication relationships among the processes. Moreover, MPI topology functions can also be used to establish a desired mapping through a rank-reordering mechanism. Rank-reordering basically means changing the default ranks assigned to MPI processes. In this regard, it is assumed that the processes have initially been mapped onto a set of nodes/cores, and we want to change the mapping to improve communication performance. This can implicitly be done by changing the initial ranks assigned to processes.

III. Related Work

Several experiments performed on large-scale systems such as the IBM BG/P and Cray XT supercomputers verify the adverse effects of higher contention and hop-counts on message latencies in such systems [7]. Balaji et al. [1] have shown that different mappings of an application on large-scale IBM BG/P systems can significantly affect the overall performance. Accordingly, they propose a framework for finding the mapping with the least amount of contention among different mapping strings supported by the system. Bhatelé and Kalé [10] propose several heuristics based on the hop-bytes metric for mapping irregular communication graphs onto mesh topologies. The heuristics generally attempt to place highly communicating processes closer together. To this end, the authors present two techniques (spiraling and quadtree) for finding the closest processor to a given source in a 2D mesh.

Mercier and Clet-Ortega [11] use the Scotch library [12] to do the mapping. Communication pattern of the application is first modeled as a complete weighted graph where the volume of messages represent the edge weights. The physical topology also similarly modeled by another complete weighted graph with the edge weights coming from the memory hierarchy shared among cores. Scotch is then used to map the communication pattern graph onto the physical topology graph. Rodrigues et al. [13] propose a similar approach where they use Scotch for mapping the process topology graph onto the physical topology graph. However, they use the communication speed (measured by a ping-pong benchmark) between each pair of cores as the metric for assigning edge weights in the physical topology graph.

Rashti et al. [2] use the HWLOC library [14] and InfiniBand tools to extract the intra-node and network topologies respectively. The results are merged to build an undirected graph with edge weights representing the communication performance among the cores based on the distance among them. The mapping is then performed by the Scotch library. Ito et al. [15] propose a similar mapping approach except that they use the actual bandwidth among nodes measured at the execution time to assign the edge weights in the physical topology graph. The mapping itself is again done by the Scotch library. Jeannot and Mercier [16], [17] also use HWLOC to extract the physical topology of a single node, and represent it as a tree. The so-called TreeMatch algorithm is then used for mapping the process topology graph onto the physical topology tree. Most recently, Jeannot et al. [18] have extended this work by improving the complexity of the TreeMatch algorithm, and applying it in a partially distributed way.

Hoefler and Snir [3] consider the mapping problem in its general form for arbitrary process topologies and interconnection networks. They model the process topology by a directed weighted graph G and use the volume of communication to designate the edge weights. The target network is also modeled by a directed weighted graph H with edge weights representing the bandwidth of the channel connecting two nodes. However, intra-node topology is considered to be flat. Three heuristics have been proposed to map G onto H. In the greedy one, the vertices with higher individual outgoing edge weights are placed as close as possible to each other. The second heuristic uses a graph partitioning library to perform the mapping by recursive bi-partitioning of G and H into two equal-sized halves with minimum edge-cut weights. The third heuristic is based on graph similarity and models the mapping problem in terms of a bandwidth reduction problem

for sparse matrices. Considering adjacency matrices of the graphs representing G and H, matrix bandwidth reduction techniques are used to first bring the two matrices into a similar shape and then find the final mapping. Average dilation and maximum congestion are the two metrics used for assessing the quality of mappings. Sudheer and Srinivasan [19] propose three heuristics for mapping all of which use the hop-bytes as the metric. The first heuristic uses Greedy Randomized Adaptive Search Procedures (GRASP). This heuristic finds the local optima among several random initial solutions and chooses the one with lowest hop-bytes. The second heuristic is a modified version of the greedy heuristic proposed by Hoefler and Snir [3]. The third heuristic is a combination of GRASP and a graph partitioning technique.

In [20], the authors use the Multi-Jagged (MJ) geometric partitioning algorithm for task mapping with a focus on the mesh/torus systems that allow non-contiguous allocation of nodes. MJ partitions tasks and cores into a same number of parts. Tasks and cores with the same part numbers are then mapped onto each other. In [21], the authors propose PaCMap; an algorithm to peform allocation and mapping simultaneously. Pacmap first uses METIS [22] to partition the communication graph for multicore nodes. Then, nodes are chosen by an expansion algorithm which assigns a score value to the nodes with respect to the number of free nodes close to them. Tasks are mapped by iteratively finding the one with the highest communication volume, and mapping it onto the node that leads to the least hop-bytes value.

In [23], the authors propose RAHTM which uses a three-phased approach for mapping. The first phase clusters tasks for multicore nodes. The second phase uses mixed integer linear programming (MILP) to hierarcically map the clusters onto a series of 2-ary n-cubes. A greedy merge heuristic is used in the third phase to combine subproblem mappings. RAHTM uses maximum congestion as the metric, and also takes into account the underlying routing in the second and third phases. We exploit routing information in our proposed algorithms too. However, RAHTM is more suited for mesh/torus topologies, whereas we mainly target indirect fat-trees.

In [24], the authors propose a main greedy mapping heuristic plus two other refinement algorithms. The greedy algorithm as well as one of the refinement algorithms are designed to minimize hop-bytes, whereas the second refinement algorithm is designed to decrease congestion. The greedy heuristic iteratively chooses the process with the highest connectivity to the set of mapped processes, and assigns it to the node that results in the lowest value of hop-bytes. The corresponding node is found by performing a BFS search on the physical topology graph. The refinement algorithms attempt to decrease hop-bytes or congestion by task swapping. Unlike [24], we do not use any graph search algorithm in our greedy heuristic; we find the desired target node by explicit evaluation of each node. In addition, we use a hybrid metric rather than hop-bytes. Moreover, we exploit parallelism in the design and implementation of our algorithms.

IV. Parallel Topology- and Routing-Aware Mapping

We distinguish our proposed algorithms with the existing mapping algorithms from four main aspects. First, we use a hybrid metric in our heuristic which is more capable of distinguishing among various candidate mappings in terms of their quality. Individual metrics such as hop-bytes or maximum congestion could easily fail to differentiate two given mappings that actually have different qualities. A previous study [25] shows that hybrid metrics have a better correlation with application execution time. However, hybrid metrics have not yet been used in any mapping algorithms.

Second, we take into account the underlying routing algorithm of the target system and the actual distribution of routes across network links. This makes our mapping algorithms routing-aware, allowing them to capture the impacts of the underlying routing on the mapping. Exploiting routing information also enables us to have a more realistic measure of our mapping metric by keeping track of the load imposed over individual links.

Third, in most of the existing mapping heuristics, the quality metric is actually used for assessment purposes only. In other words, the metric is not directly optimized within the corresponding heuristic, and is rather used to measure the quality once the mapping is figured out. On the contrary, we attempt to optimize the metric directly at each step of our proposed algorithms. In other words, our proposed heuristic uses the actual values of the metric to decide where each given process shall be mapped.

Fourth, we exploit the parallelism capabilities of the target system in our algorithm designs. To the best of our knowledge, this is the first attempt in proposing parallel algorithms for topology-aware mapping. We believe parallelism is the key for having truly scalable mapping algorithms for current and future HPC systems.

A. Hybrid metric

Rather than using only hop-bytes or maximum congestion, we take advantage of a hybrid metric in our core mapping heuristic. The metric is a combination of four individual metrics, each of which evaluate a given mapping from a particular aspect. Specifically, it consists of the following four individual metrics: 1) hop-bytes (HB), 2) maximum congestion (MC), 3) non-zero congestion average (NZCA), and 4) non-zero congestion variance (NZCV). In this work, we consider a linear combination of these metrics to build up our hybrid metric as

$$\text{Hybrid Metric} = HB + MC + NZCA + NZCV. \quad (3)$$

The hop-bytes and maximum congestion components of our hybrid metric are same as those defined in II. We define *non-zero congestion average* as the average of traffic that passes through each non-idle link in the network. Formally speaking, non-zero congestion average for a given mapping is equal to the total sum of the traffic across all links, divided by the number of links that are actually utilized by the mapping. In other words, it is the average of traffic among links with a non-zero volume of traffic passing through them. Similarly, *non-zero congestion variance* is defined as the variance of traffic across the links with a non-zero load passing through them.

The main reason we use a hybrid metric is because hop-bytes or maximum congestion alone is limited in terms

of distinguishing among multiple candidate mappings. Hop-bytes will always give a lower priority to a mapping that causes traffic to go across multiple links. There are two main shortcomings with such a behavior. First, it does not take into account the fact that using more links (i.e., more hops) could potentially result in a better link utilization and traffic distribution. Better traffic distribution would in turn help to have a lower maximum congestion on links. Second, hop-bytes alone could not distinguish between two mappings that have the same hop-bytes value, but at the same time are different in terms of congestion.

On the other hand, using only maximum congestion will have its own shortcomings. The main issue here is that several mappings could all result in the same maximum congestion at a given step of the mapping algorithm, while providing essentially different potentials for mapping the rest of the processes. Such cases could specially happen when mapping a particular process on different nodes does not affect the load on the link with maximum congestion, but results in different traffic load on other links (or different hop-bytes values). Deciding solely based on maximum congestion does not allow us to distinguish among such mappings. Even with an equal maximum congestion, we would still like to choose the mapping that results in a relatively lower traffic load on the set of all links as it would provide higher chances to decrease maximum congestion in future steps of the mapping algorithm.

Accordingly, we use non-zero congestion average and variance as a measure for the *balancedness* of traffic across the links for a given mapping. There is a reason for why we do not use an ordinary average across *all* links in the network. An ordinary average will be equal to the total sum of the traffic divided by the total number of links in the target system. As the total number of links in the system is a fixed number, an ordinary average would be only reflective of the sum of the network traffic. Such a measure cannot tell us about the link utilization and traffic distribution characteristics of a given mapping. Non-zero congestion average on the other hand provides an implicit measure of both the traffic sum, and the degree of balancedness of the total traffic across the links. Similarly, non-zero congestion variance would provide us with additional information regarding the balancedness of traffic.

It is worth noting that one could use other combinations of the above metrics to build the hybrid metric. The linear combination used in this work is actually the simplest form in which all the individual metrics have the same weight. Figuring out the optimal combination of metrics is an interesting topic for further research. For instance, a better approach would be to assign a different weight to each individual metric with respect to the corresponding impact of that metric on performance.

B. Impact of routing awareness

In a parallel system, the congestion imposed on the set of network links is a function of both the topology and the underlying routing algorithm. This is specially true for InfiniBand where the underlying static routing can highly affect congestion [5]. Therefore, in our mapping heuristic, we take into account the routing information when we want to calculate congestion on each network link. This way, we would have a more realistic congestion model throughout all steps of our mapping algorithm. In the following, we use a simple example to clarify how routing awareness can make difference in process mapping.

Consider a target system with the topology shown in Fig. 1(a). The system consists of 4 nodes n_1, n_2, n_3, n_4, each having 2 cores. We would like to map 8 processes with the topology shown in Fig. 1(b) onto this system. The edge weights in Fig. 1(b) represent the communication volumes. For the sake of simplicity, we consider equal weights for all edges. The underlying routing of the system has been partially shown by the labels (n_1 to n_4) assigned to the four top links in Fig. 1(a). We only represent the routes in one way (i.e., bottom-up) as it suffices for the purpose of this example. The link labels represent the destination nodes associated to each link.

We start the mapping by assigning rank 0 to one of the cores (could be any of the cores). Having mapped four other processes, the mapping will be as shown in Fig. 2(a). The link labels in this figure represent the corresponding congestion values. For the sake of brevity, we represent the congestion values of all links in one direction only (i.e., bottom-up direction in the tree). Up to this point, routing awareness does not play any role and hence, the result would be the same for a routing-ignorant approach as well. However, when it comes to the next process (i.e., rank 5), we can see the impact of routing awareness.

Rank 5 can be assigned to a core on either n_3 or n_4. Fig. 2(b) and 2(c) show the final mapping and the corresponding link congestions with respect to each of such assignments. It can be seen that mapping rank 5 to a core on n_4 (Fig. 2(c)) will lead to a lower maximum congestion across the links. By exploiting the routing information, a routing-aware approach will choose the mapping shown in Fig. 2(c), whereas a routing-ignorant approach cannot distinguish between the two mappings.

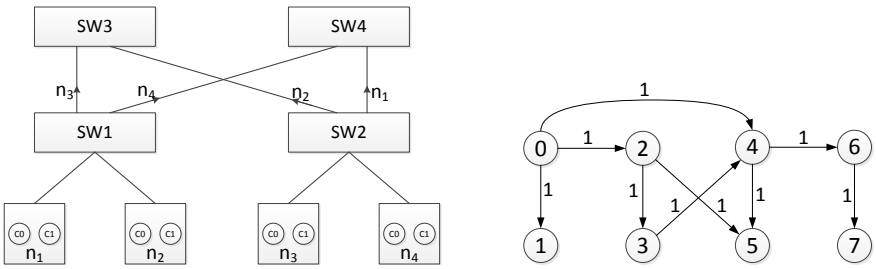
The above scenario is an example for how process mapping can potentially benefit from routing awareness. Of course this is just a very simple example on a very small system. As the system scales and the number of nodes and switches increase, the topology becomes more complex, and the impacts of routing awareness on congestion will also increase.

C. Mapping procedure

Fig. 3 shows a high-level abstraction of the framework that we use to perform the mapping. It should be noted that a similar framework has been used in other topology-aware mapping studies as well [3], and that we are not claiming it as our contribution in this work; we use it as the base for our mapping approach. Our contribution is the particular greedy heuristic and refinement algorithm, as well as the addition of routing information into it.

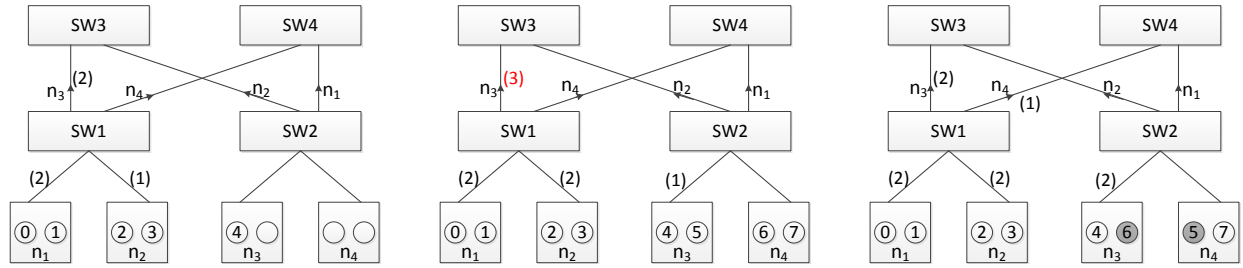
There exist three main steps in the framework shown in Fig. 3: 1) an initial partitioning of the application communication pattern, 2) an intermediate mapping of processes using a parallel mapping heuristic, and 3) a final mapping refinement. In the following, we explain the details of each step.

1) Initial partitioning: The mapping procedure starts with first converting the application communication pattern to a nodal communication pattern. The application communication



(a) Sample target system (b) Sample communication pattern graph

Fig. 1: A small sample target system topology and process communication pattern graph.



(a) The mapping layout for the first four processes.

(b) The mapping outcome from a routing-ignorant approach. Maximum congestion is equal to 3.

(c) The mapping outcome from a routing-aware approach. Maximum congestion is equal to 2.

Fig. 2: Different mappings of processes leading to different values of maximum congestion across the links.

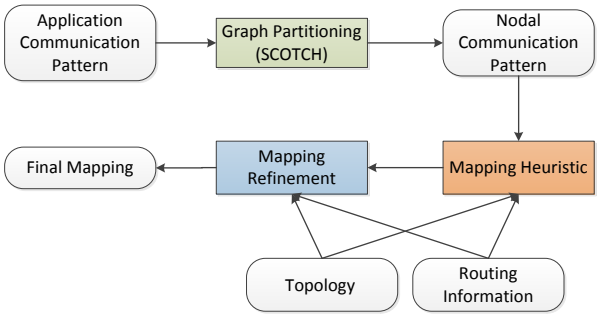


Fig. 3: Mapping framework: Various steps and components.

pattern represents the total volume of messages transferred between each of the individual processes in terms of a directed weighted graph. We assume that the communication pattern has already been extracted in a profiling stage with the corresponding results saved into a file. We have actually developed a profiler that captures the communication pattern of an application at the device layer of an MPI library. Such a profiling is done *once* for each application. Alternatively, the programmer can also directly provide the communication pattern through MPI process topologies.

Using a graph partitioning algorithm, we first partition the process topology graph into a number of partitions equal to the number of compute nodes in the target system Each node in the resulting graph will encapsulate a number of processes equal to the number of cores of each compute node. Thus, the resulting graph will be representative of the communication pattern among a set of *super processes*, each of which should be mapped onto a single compute node.

The partitioning can be done by any graph partitioning algorithm. There exist a number of well-known libraries such as SCOTCH [12] and METIS [22] for graph partitioning. In this work, we will use the SCOTCH library to perform the initial partitioning of the application communication pattern graph. It is worth noting that hereinafter, we use the term 'process' (or process topology graph) to refer to a super-process in the nodal communication pattern graph.

We use the initial partitioning stage to address multicore nodes, as in this work we are mainly concerned about mapping at the network layer. Initial partitioning will potentially put heavily communicating processes in one partition. By collectively mapping all the processes in a partition onto a same node, we take advantage of shared-memory channels for communications among such processes. However, our proposed algorithms can also be used directly across all cores.

2) Mapping heuristic: A parallel mapping heuristic lies at the core of our mapping procedure. Alg. 1 shows the various steps involved in our proposed heuristic. The heuristic takes the communication pattern matrix, the routing information, and the network topology as input, and provides a mapping from the set of processes to the set of nodes as the output.

Alg. 1 is parallel in nature, and is run by one *leader* process on each node. Let P_M and P_M^c denote the set of

Algorithm 1: PTRAM: Parallel topology- and routing-aware mapping: the core heuristic

Input : Set of all processes P, set of all nodes N, communication pattern matrix CP, routing information, links layout
Output: The mapping $\tau : P \rightarrow N$

1 $P_M \leftarrow \Phi$; // The set of mapped processes
2 **while** $P_M^c \neq \Phi$ **do**
  ; // There are more processes to map
3 $\alpha = \frac{1}{|P_M|+1}$;
4 **for** $q \in P_M^c$ **do**
5 $\delta = \sum_{r \in P_M} (CP_{qr} + CP_{rq}) + \alpha \sum_{s \in P_M^c} (CP_{qs} + CP_{sq})$;
6 **end**
7 $p_{next} \leftarrow q_{max}$; // Choose the process with maximum value of δ as the next process to map
8 Temporarily assign p onto self node ;
9 Calculate link congestions using routing information;
10 Find the hybrid metric value according to eq. (3) ;
11 Gather the metric value from all other node-leaders;
12 $n =$ node with the lowest value of metric;
13 $\tau(p) = n$; // Map p onto n
14 Update link congestions accordingly;
15 $P_M \leftarrow p$; // Add p to the set of mapped processes
16 **end**

mapped and unmapped processes respectively. The main loop in line 2 is run until all processes are mapped onto a target node. Within each iteration, first a new process is chosen as the next process (i.e., p_{next}) to map. A smart choice of such a process is very important in terms of the quality of the resulting mapping, especially as the heuristic is greedy in nature and does not involve any backtracking mechanism; When a process is mapped onto a node at a given iteration of the heuristic, its mapping will remain fixed in the following iterations of the heuristic. Accordingly, we choose p_{next} with respect to a metric denoted by δ in line 5 of Alg. 1. For each process such as q, the first summation in line 5 gives the total communication volume of q with all its already-mapped neighbors. The second summation represents the total communication volume with the unmapped neighbors. The process with the maximum value of δ is chosen as the next process for mapping at each iteration.

The parameter $0 < \alpha \leq 1$ is used to assign a relative (lower) weight to communications with unmapped neighbors in comparison to communications with mapped neighbors. The value of α is updated at every iteration (line 3) with respect to the number of mapped processes. Initially, α starts at 1 and decreases at each iteration as more processes are mapped. We use such an α value because the amount of communication with the set of unmapped neighbors is relatively more important at the initial steps of the algorithm as most of the neighbors of a given process have not yet been mapped. However, as more processes are mapped, we want to give a relatively higher weight to the communications with the set of mapped neighbors when choosing the next process for mapping.

The next major step in Alg. 1 is finding a target node for p_{next}. Target nodes are chosen based on the hybrid metric defined in eq. (3). At each iteration, we seek to map p_{next} onto the node that results in the lowest value of the hybrid metric. Thus, we explicitly measure the value of the metric at each iteration of the algorithm, and choose the target node accordingly. This is where we take advantage of parallelism in our heuristic. The leader process on each node[1] such as n is responsible for calculating the metric value resulting from mapping p_{next} onto n. This is done in lines 8 to 10 of Alg. 1. Next, all leader processes communicate with each other to gather the metric values corresponding to all nodes. Specifically, we use MPI_Allgather to accomplish this step. Having gathered the metric values from all nodes, the target node for p_{next} is set to be the node with the lowest value of the hybrid metric. In case of having multiple such nodes, we simply choose the one with the lowest leader ID. Finally, before going to the next iteration, each leader process updates its own local copy of link congestions with respect to the newly mapped process.

Choosing the target node as explained above has two main advantages. First, it allows us to explicitly evaluate each node when it comes to choosing a target node. Second, by engaging all nodes, we increase the quality of target node choices, and at the same time keep the corresponding costs bounded. Such a parallel approach would also significantly improve the scalability of our heuristic. Using one leader process on each node enables our heuristic to scale its search capability in accordance to increase in system size.

It is also worth noting that the routing information is fed to the algorithm from a file. The routing file consists of N^2 lines where N denotes the number of end nodes in the system. Each line corresponds to the route that connects a pair of nodes. This file is created only once for each system and is saved on the disk for future references. For InfiniBand, we have developed a simple script that uses the `ibtracert` tool to extract the routes in terms of the switches and the ports connecting each pair of nodes. Initially, each leader process loads the routing information to its memory. Since each leader process is only responsible for evaluating its own node n, it will only need to load a portion of the routing file into the memory; that is, the route from n to every other node, and the route from every other node to n. The advantage is a significant reduction in memory requirement (i.e., from an unscalable order of N^2 to a linear order of N).

3) Refinement algorithm: A shortcoming of the greedy heuristic is its lack of a backtracking mechanism; the heuristic is greedy in nature, and thus optimizing the metric at each stage does not necessarily guarantee optimality at the end. Therefore, we might still be able to tune the mapping further in a final refinement step. Accordingly, we propose a refinement algorithm as outlined by Alg. 2. At the refinement stage, our main goal is to specifically decrease maximum congestion on links by swapping particular processes.

Alg. 2 is run in parallel by one leader process on each node. At the beginning of the algorithm, all the leader processes have a copy of the mapping output from Alg. 1 in terms of an array. For a given node such as n, let p denote the process assigned to n based on the mapping array. In Alg. 2, the leader process on node n checks to see whether swapping p with other processes can decrease maximum congestion. To this end, we first find link l with maximum congestion (line 4), and see whether p communicates with any other process (q) across a route involving l (line 5). If so, we check whether swapping p

[1] In fact, this is done only by non-occupied nodes.

Algorithm 2: PTRAM: The refinement algorithm

Input : Set of all nodes N, communication pattern matrix CP, routing information, links layout
Output: The refined mapping $\tau : P \rightarrow N$

n =self node;
p = process assigned to n;
$maxCong$ = `findMaxCong()` ; `// Find the current value of maximum congestion`
l = `findMaxLink()` ; `// Find link with maximum congestion`
for q where $CP[p][q] \neq 0$ and $l \in L(p, q, \tau)$ **do**
    M = list of k closest nodes to n;
    for $m \in M$ **do**
        r =process mapped to m;
        Temporarily swap p with r;
        $newMaxCong$ = `findMaxCong()` ; `// Find the new value of maximum congestion`
        if $newMaxCong < maxCong$ **then**
            $maxCong = newMaxCong$;
            Save p and r;
        end
    end
end
Gather $maxCong$ and the associated swappings from all leaders;
Find the lowest value of $maxCong$;
Enforce the corresponding swapping;
Update link congestions accordingly;

with any of the processes on a node close to n will result in a lower maximum congestion. Among all the nodes, only the k nearest ones to n are considered[2]. Thus, each individual leader performs only a local search that is limited to a few number of its neighbor nodes. This will help to significantly reduce the high costs that are typically associated with refinement algorithms. At the same, since every leader process performs such a local search simultaneously, the algorithm will still explore a vast portion of all possible swappings.

Among all local swappings, each leader process saves (temporarily) the swapping that leads to the highest reduction in maximum congestion. Next, all the leader processes communicate with each other to collectively find the swapping that results in a globally (across all leaders) lowest maximum congestion. The corresponding swapping is enforced by all the leaders, and the congestion information is updated accordingly. Note that Alg. 2 can be repeated multiple times to decrease congestion iteratively. In our experiments, the algorithm runs until there is no congestion reduction between two consecutive iterations, or it reaches 10 iterations (whichever happens earlier). This is used as a mechanism to avoid unbounded number of refinement iterations.

V. Experimental Results

We carry out all the experiments on the GPC cluster at the SciNet HPC Consortium [26]. GPC consists of 3780 nodes with a total of 30240 cores. Each node has two quad-core Intel Xeon sockets operating at 2.53GHz, with a total of 16GB memory. Approximately one quarter of the cluster is connected with non-blocking DDR InfiniBand while the rest of the nodes are connected with 5:1 blocked QDR InfiniBand. For our experiments, we only use the nodes with QDR InfiniBand. These nodes are interconnected via Mellanox ConnectX 40Gb/s InfiniBand adapters. The network topology is a fat-tree consisting of two 324-port core switches and 103 36-port leaf switches. Fig. 4 shows the corresponding details. The numbers on the links represent the number of links that connect two switches. In particular, each leaf switch is connected to 30 compute nodes, and has 3 uplinks to each of the two core switches. Each core switch itself is in fact a 2-layer fat-tree consisting of 18 *line* and 9 *spine* switches (all 36-port). Each line switch is connected to 6 leaf switches, and also has 2 uplinks to each of the 9 spine switches. Finally, the nodes run Centos 6.4 along with Mellanox OFED-1.5.3-3.0.0, and we use MVAPICH2-2.0, and Scotch-6.0.0.

We use 3 micro-benchmarks as well as a real application to evaluate the performance of our proposed mapping approach. In addition, we also compare our proposed heuristics against those proposed and implemented by Hoefler and Snir in LibTopoMap [3]. In particular, LibTopoMap provides 4 mapping algorithms: 1) a general greedy heuristic, 2) an algorithm based on recursive graph bi-partitioning that uses METIS (Rec), 3) an algorithm based on matrix bandwidth reduction that uses the Reverse Cuthill McKee algorithm (RCM), and 4) the graph mapping functionality provided by the Scotch library. In all cases, LibTopoMap first uses ParMETIS [27] graph partitioner to handle multicore nodes.

We have integrated all the heuristics into MPI distributed graph topology function to enforce particular mappings through rank reordering. The results report the improvements achieved over a block in-order mapping of processes, where adjacent ranks are mapped onto the same node as far as possible before moving to any other node. Note that in all the figures, PTRAM refers to our proposal mapping package.

A. Micro-benchmark results

For our micro-benchmark evaluations, we use three micro-benchmarks that mimic the communication patterns seen in many parallel applications. The first micro-benchmark models a 2D 5-point halo exchange. In this micro-benchmark, the MPI processes are organized into a virtual 2-dimensional grid, and each process communicates with its two immediate neighbors along each dimension. The second micro-benchmark models a 3D 15-point halo exchange in which MPI processes are arranged into a virtual 3-dimensional grid. Each process communicates with its two immediate neighbors along each dimension (6 neighbors), as well as 8 corner processes. The third micro-benchmark models an all-to-all communication over sub-communicators. The processes are arranged into a logical 2-dimensional grid with each column representing a separate sub-communicator. We then perform an MPI_Alltoall within each sub-communicator.

1) Metric values: Fig. 5 shows the resulting metric values for 4096 processes. The numbers show the normalized values over the default mapping. As shown, for all three benchmarks, PTRAM can successfully decrease the value of all four metrics. Moreover, it outperforms the heuristics provided by LibTopoMap. For the 2D 5-point micro-benchmark in particular, we can see more than 60% reduction in the value of all four metrics in Fig. 5(a). The highest reduction is seen for congestion average which is about 73%. Maximum congestion is also decreased by 68%. At the same time, we see that Libtopomap has led to an increase in the metric values. The

[2]we use a value of $k = 7$ in our experiments

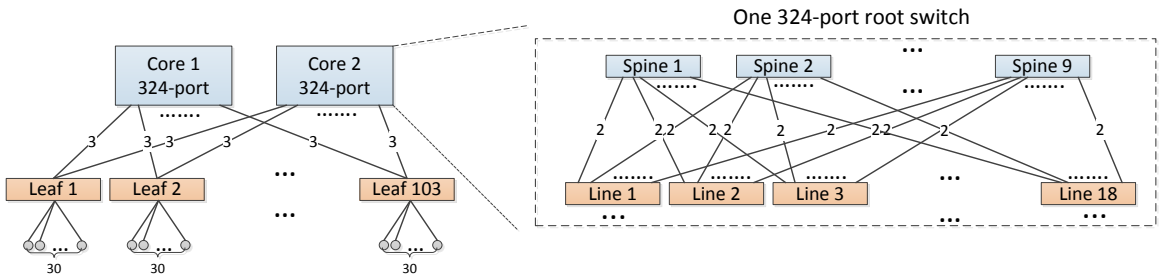
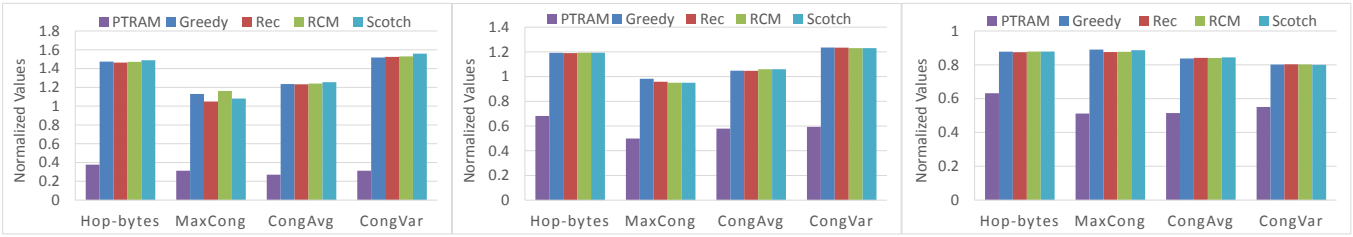
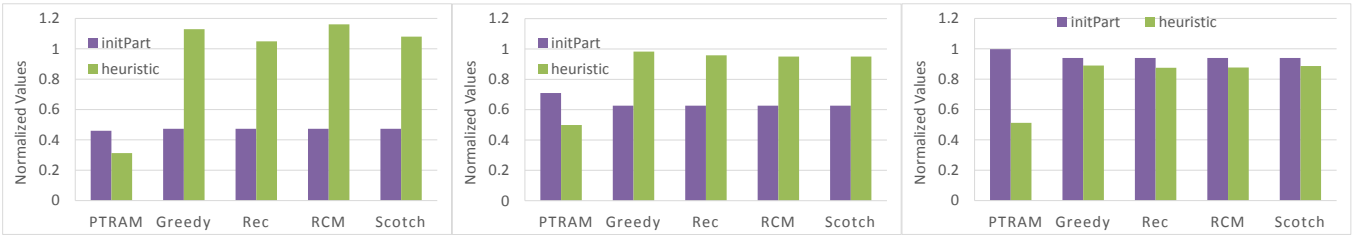


Fig. 4: Network topology of the GPC cluster at SciNet.



(a) 2D 5-point (b) 3D 15-point (c) sub-communicator all-to-all

Fig. 5: The resulting metric values for 3 micro-benchmarks and various mapping algorithms with 4096 processes.



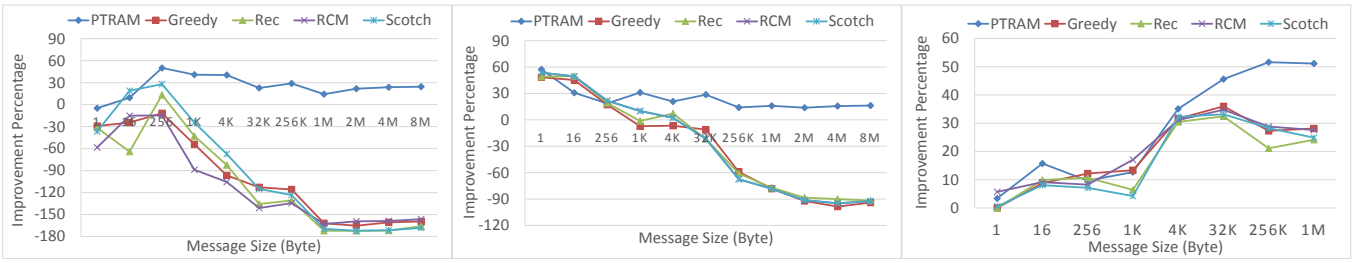
(a) 2D 5-point (b) 3D 15-point (c) sub-communicator all-to-all

Fig. 6: Maximum congestion improvement details with respect to the initial graph partitioning stage for 4096 processes.

highest increase is seen for the hop-bytes and congestion variance metrics. PTRAM outperforms LibTopoMap for multiple reasons. First, PTRAM explicitly evaluates all potential nodes for mapping a process at each step. Second, it considers a hybrid metric and takes multiple criteria into account. Third, it takes into account the routing information as well as the precise layout of network links which would allow the algorithm to have a more realistic evaluation of the metrics. The greedy algorithm in LibTopoMap considers congestion in terms of a weighted shortest path routing algorithm which does not necessarily match the system's actual routing. The three other algorithms are all distance-based only. For the 3D 15-point micro-benchmark (Fig. 5(b)), the improvements are relatively lower. However, we can still see 50% reduction in maximum congestion for PTRAM. We can also see that PTRAM would still outperform LibTopoMap by a factor of 2 (at least).

We are also interested to evaluate the performance of each heuristic with respect to the initial partitioning stage. Recall that in both PTRAM and LibTopoMap, there are two major stages. First, a graph partitioner library is used to cluster multiple individual processes into a single group (we use Scotch for PTRAM, and LibTopoMap uses ParMetis). Second, the heuristics are used to collectively map each group of processes onto a particular node in the system. Thus, in Fig. 6, we show how the improvements reported in Fig. 5 are broken down across these two stages. 'initPart' shows the results corresponding to a simple in-order node-mapping of process clusters returned by the initial graph partitioning stage. 'heuristic' represents the results for each particular heuristic. Due to lack of space, we only show these results for the maximum congestion metric. The trend is almost the same for other metrics as well.

Fig. 6(a) shows that for the 2D 5-point micro-benchmark and PTRAM, the initial partitioning reduces maximum congestion to about 0.45. PTRAM then decreases maximum congestion further to about 0.3; i.e., we can achieve an additional 14% reduction in the initial maximum congestion by using PTRAM. We can also see that the initial partitioning provides a similar



(a) 2D 5-point (b) 3D 15-point (c) sub-communicator all-to-all

Fig. 7: Communication time improvements over the default mapping for 3 micro-benchmarks with 4096 processes.

performance in both PTRAM and LibTopoMap. However, we see that the mapping result from LibTopoMap heuristics cause an increase in the final maximum congestion. In other words, an in-order node-mapping of process partitions returned by ParMetis in LibTopomap would result in a lower maximum congestion than those returned by the heuristics. A similar trend is seen for the 3D 15-point micro-benchmark in Fig. 6(b). One difference is the lower maximum congestion achieved by 'initPart' in LibTopoMap than PTRAM. This implies that ParMetis has done a better job than Scotch for the initial partitioning. Despite this, PTRAM has still outperformed the other 4 heuristics by providing 21% improvement on top of the initial partitioning stage. Thus, we see a higher share of improvement coming from PTRAM (21%) compared to the case for 2D 5-point micro-benchmark (14%).

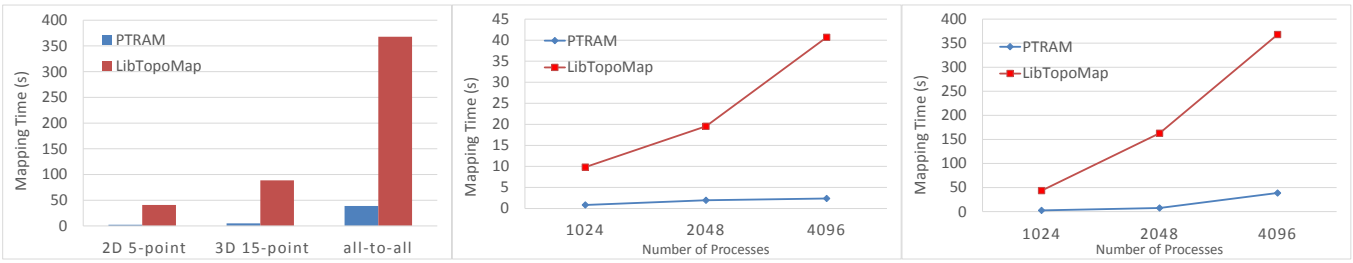
For sub-communicator all-to-all shown in Fig. 6(c), the most notable difference is that almost all the improvement comes directly from PTRAM. We can see that 'initPart' has almost no impact on maximum congestion, whereas PTRAM successfully provides 50% improvement. This is an important observation as it shows that an appropriate node-mapping of processes can still significantly decrease maximum congestion on top of an initial muticore partitioning. In addition, we see that LibTopoMap heuristics have also reduced maximum congestion on top of the initial partitioning stage. Again, we see better 'initPart' results for LibTopoMap than PTRAM.

2) Communication time: Fig. 7 presents the communication time results for each micro-benchmark. The results show the average of 500 communication iterations. We report the improvement percentage over the default mapping. As shown in Fig. 7(a), PTRAM can decrease the communication time of the 2D 5-point micro-benchmark by up to 50% for messages below 4KB. For message above 4KB, we see an almost steady improvement of about 25%. However, we see that LibTopoMap mappings have led to an increase in communication time. This is expected and is in accordance with the results we saw in Fig. 5 for the metric values. As the mapping result from LibTopoMap increases both hop-bytes and maximum congestion, the communication time is highly expected to increase. On the other hand, we see that PTRAM's successful attempt in decreasing the metric values has actually been reflected in terms of lower communication times. A similar trend is seen for the 3D 15-point micro-benchmark above 1KB message sizes (Fig. 7(b)). However, despite its negative impact on the metrics, LibTopoMap has improved performance by up to 50% for messages below 1KB. We are not sure the root cause of this, and need to investigate it further.

For the sub-communicator all-to-all micro-benchmark shown in Fig. 7(c), we see improvement with both PTRAM and LibTopoMap across all message sizes. For messages below 4KB, the improvements are lower and almost the same for all heuristics. However, PTRAM starts to outperform LibTopoMap for larger messages, and provides up to 50% improvement versus about 30% provided by LibTopoMap. This is again in accordance with the metric results shown in Fig. 5(c), where we see improvement in all metric values for all heuristics. However, higher improvements achieved by PTRAM result in higher improvements in communication time for messages above 4KB. In general, we see a good correlation between the results for communication time improvements in Fig. 7 and those shown for the metrics in Fig. 5.

3) Mapping time: Fig. 8(a) shows the total mapping time for each of the micro-benchmarks with 4096 processes. The mapping time is almost the same for all four heuristics provided by LibTopoMap. Therefore, here we only report the time corresponding to one of them (i.e., the greedy heuristic). As shown by the figure, PTRAM imposes a much lower overhead in comparison to LibTopoMap for all three micro-benchmarks. This is mainly owed to the underlying parallel design of our heuristics that allows for fast (and yet high-quality) mappings. In particular, PTRAM spends 2.3 and 5 seconds to derive the mapping for the 2D 5-point and 3D 15-point micro-benchmarks respectively. This increases to 38 seconds for the sub-communicator all-to-all micro-benchmark. The reason is that in the all-to-all micro-benchmark, each process communicates with a relatively higher number of other processes. Therefore, the corresponding communication pattern matrix will be denser which in turn adds to the amount of computation performed by each node-leader. The same behavior is seen for LibTopoMap as well. It is worth noting that in practice, mapping costs are incurred only once for each application. Mapping results are saved, and reused later on for subsequent runs of an application.

Finally, in Fig. 8(b) and 8(c) we report the mapping time with three different number of processes for the 2D 5-point and all-to-all micro-benchmarks. 3D 15-point has similar result to 2D 5-point, and hence we do not repeat it here. We see that for both micro-benchmarks, PTRAM achieves a lower mapping time than LibTopoMap with 1024, 2048, and 4096 processes. Also, by comparing the two figures, we see a better



(a) 4096 processes, 3 benchmarks (b) 2D 5-point (c) sub-communicator all-to-all

Fig. 8: Mapping time overheads and scalability for PTRAM and LibTopoMap with various micro-benchmarks.

scalability for PTRAM and the 2D 5-point micro-benchmark than the all-to-all one. This is again rooted in the difference between the two micro-benchmarks in terms of the sparsity of their corresponding communication patterns. With the all-to-all pattern, the increase in the number of processes also increases the number of neighbors for each process, adding extra computation load to node-leaders. With the 2D 5-point pattern however, the number of neighbors do not increase and remains fixed (4 neighbors).

B. Application results

In this section, we evaluate our proposed mapping approach in the context of a real parallel application. To this end, we use Gadget [28] which is an applications for various cosmological N-body simulations, ranging from colliding and merging galaxies, to the formation of large-scale structure in the Universe. In particular, we use the version 3 of the application that can be attained from [29]. We run Gadget with 4096 processes, and report the corresponding results.

Fig. 9(a) shows the normalized metric values for varius heuristics. It is shown that PTRAM can decrease the metric values to about 0.3 of the default mapping. LibTopoMap heuristics do not have much of an impact on hop-bytes, congestion average, and congestion variance. However, 3 of the heuristics lead to an increase in maximum congestion, and RCM leaves it almost intact.

Fig. 9(b) compares the communication time of the application for vaiour mappings. We can see that PTRAM provides 10 seconds reduction in communication time of the application compared to the default mapping. It also outperforms the other heuristics. With the greedy heuristic, the communication time is almost the same as the default mapping, whereas recursive (Rec), RCM, and Scotch have slightly increased the communication time. Finally, Fig. 9(c) shows the overhead corresponding to each mapping approach. Again, we see that PTRAM provides a significantly lower overhead. PTRAM spends 317 seconds to find the desired mapping, whereas this is close to 4,000 seconds for LibTopoMap.

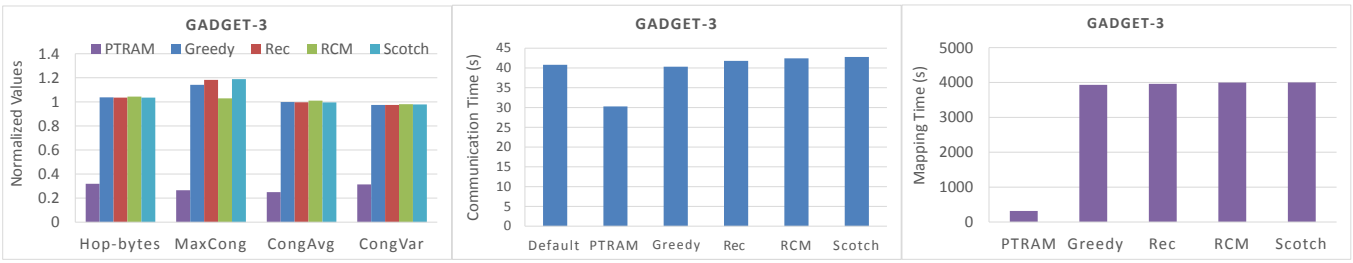
VI. Conclusion and Future Work

In this work we proposed two algorithms for topology-aware process mapping in HPC systems. The algorithms take advantage of a parallel design to expand the mapping search space, and at the same time attain scalability. In addition, the algorithms exploit the underlying routing information of the system for a more realistic congestion model. Also, we use a hybrid metric that allows for a more comprehensive evaluation of candidate mappings. The experimental results suggest that our proposed approach can successfully decrease various metric values, as well as the communication time itself.

As for future directions, we first intend to study more sophisticated combinations of the individual metrics for building the hybrid metric. Further analytical and experimental studies could help to find an optimal combination for the hybrid metric. Moreover, we intend to improve the scalability of communications among node-leaders by making such communications hierarchical. Node-leaders could be clustered into multiple disjoint groups. All members within a group communicate with each other, whereas inter-group communications are limited to group-leaders only. Next, we would like to improve our heuristics further by parallelizing the computations performed by each node-leader across all the cores within each node. We also seek to extend our approach to cover the mapping at the intra-node layer too. Extending our approach towards systems with adaptive routing is another direction for future research. We could use an approximation of an adaptive routing to represent the routing information. For instance, we could consider the default shortest paths used by an adaptive routing algorithm. In this case, PTRAM would attempt to map the processes so that congestion on the default shortest paths is minimized. This could potentially decrease the need for using alternative paths (that might not be optimal) by the adaptive routing. Finally, we are also interested in conducting experiments at larger scales.

Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada Grant #RGPIN/238964-2011, Canada Foundation for Innovation and Ontario Innovation Trust Grant #7154. Computations were performed on the GPC supercomputer at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto. We would like to especially thank Scott Northrup for his technical support regarding our experiments on GPC. We also thank Mellanox Technologies and the HPC Advisory Council for the resources to conduct the early evaluation of this research.



(a) Metric values (b) Communication time (c) Mapping time

Fig. 9: Gadget-3: Results for metric values, communication time, and mapping time of the application with 4096 processes.

REFERENCES

[1] P. Balaji, R. Gupta, A. Vishnu, and P. Beckman, "Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems," *Computer Science-Research and Development*, vol. 26, no. 3-4, pp. 247–256, 2011.

[2] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, "Multi-core and network aware MPI topology functions," in *Proceedings of the European MPI Users' Group conference on Recent advances in the message passing interface*, 2011, pp. 50–60.

[3] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the International Conference on Supercomputing*, 2011, pp. 75–84.

[4] A. Bhatelé, G. R. Gupta, L. V. Kalé, and I.-H. Chung, "Automated mapping of regular communication graphs on mesh interconnects," in *Proceedings of the International Conference on High Performance Computing*, 2010, pp. 1–10.

[5] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *Cluster Computing, 2008 IEEE International Conference on*. IEEE, 2008, pp. 116–125.

[6] H. Subramoni, P. Lai, S. Sur, and D. K. Panda, "Improving application performance and predictability using multiple virtual lanes in modern multi-core InfiniBand clusters," in *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 2010, pp. 462–471.

[7] A. Bhatelé and L. V. Kalé, "An evaluative study on the effect of contention on message latencies in large supercomputers," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[8] A. L. Rosenberg, "Issues in the study of graph embeddings," in *Graphtheoretic Concepts in Computer Science*, 1981, pp. 150–176.

[9] "Message Passing Interface Forum, http://www.mpi-forum.org/, last accessed 2015/12/21."

[10] A. Bhatelé and L. V. Kalé, "Heuristic-based techniques for mapping irregular communication graphs to mesh topologies," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*. IEEE, 2011, pp. 765–771.

[11] G. Mercier and J. Clet-Ortega, "Towards an efficient process placement policy for MPI applications in multicore environments," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2009, pp. 104–115.

[12] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *Proceedings of the International Conference on High-Performance Computing and Networking*, 1996, pp. 493–498.

[13] E. R. Rodrigues, F. L. Madruga, P. O. A. Navaux, and J. Panetta, "Multi-core aware process mapping and its impact on communication overhead of parallel applications," in *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*. IEEE, 2009, pp. 811–817.

[14] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in HPC applications," in *Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, 2010, pp. 180–186.

[15] S. Ito, K. Goto, and K. Ono, "Automatically optimized core mapping to subdomains of domain decomposition method on multicore parallel environments," *Computers & Fluids*, vol. 80, no. 0, pp. 88 – 93, 2013.

[16] E. Jeannot and G. Mercier, "Near-optimal placement of MPI processes on hierarchical NUMA architectures," in *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part II*, ser. Euro-Par'10. Springer-Verlag, 2010.

[17] G. Mercier and E. Jeannot, "Improving MPI applications performance on multicore clusters with rank reordering," in *Proc. EuroMPI*, 2011, pp. 39–49.

[18] E. Jeannot, G. Mercier, and F. Tessier, "Process placement in multicore clusters: Algorithmic issues and practical techniques," *IEEE Tran. Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2013.

[19] C. Sudheer and A. Srinivasan, "Optimization of the hop-byte metric for effective topology aware mapping," in *High Performance Computing (HiPC), 2012 19th International Conference on*. IEEE, 2012, pp. 1–9.

[20] M. Deveci, S. Rajamanickam, V. J. Leung, K. Pedretti, S. L. Olivier, D. P. Bunde, U. V. Catalyurek, and K. Devine, "Exploiting geometric partitioning in task mapping for parallel computers," in *Proc. Parallel and Distributed Processing Symposium*, 2014, pp. 27–36.

[21] O. Tuncer, V. J. Leung, and A. K. Coskun, "Pacmap: Topology mapping of unstructured communication patterns onto non-contiguous allocations," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 37–46.

[22] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[23] A. H. Abdel-Gawad, M. Thottethodi, and A. Bhatele, "Rahtm: routing algorithm aware hierarchical task mapping," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 325–335.

[24] M. Deveci, K. Kaya, B. Ucar, and U. V. Catalyurek, "Fast and high quality topology-aware task mapping," in *Proc. Int. Symp. Parallel and Distributed Processing*, 2015, pp. 197–206.

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

[26] C. L. et al., "Scinet: Lessons learned from building a power-efficient top-20 system and data centre," *Journal of Physics: Conference Series*, vol. 256, no. 1, 2010.

[27] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 71–95, 1998.

[28] V. Springel, "The cosmological simulation code gadget-2," *Monthly Notices of the Royal Astronomical Society*, vol. 364, no. 4, pp. 1105–1134, 2005.

[29] "Gadget-3 application code, http://http://www.prace-ri.eu/ueabs/#gadget, last accessed 2015/12/21."