Generic Topology Mapping Strategies for Large-scale Parallel Architectures

Torsten Hoefler
University of Illinois at Urbana-Champaign
Urbana, IL, USA
htor@illinois.edu

Marc Snir
University of Illinois at Urbana-Champaign
Urbana, IL, USA
snir@illinois.edu

ABSTRACT

The steadily increasing number of nodes in high-performance computing systems and the technology and power constraints lead to sparse network topologies. Efficient mapping of application communication patterns to the network topology gains importance as systems grow to petascale and beyond. Such mapping is supported in parallel programming frameworks such as MPI, but is often not well implemented. We show that the topology mapping problem is NP-complete and analyze and compare different practical topology mapping heuristics. We demonstrate an efficient and fast new heuristic which is based on graph similarity and show its utility with application communication patterns on real topologies. Our mapping strategies support heterogeneous networks and show significant reduction of congestion on torus, fat-tree, and the PERCS network topologies, for irregular communication patterns. We also demonstrate that the benefit of topology mapping grows with the network size and show how our algorithms can be used in a practical setting to optimize communication performance. Our efficient topology mapping strategies are shown to reduce network congestion by up to 80%, reduce average dilation by up to 50%, and improve benchmarked communication performance by 18%.

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: Parallel Programming—*Topology Mapping*

General Terms

Performance

Keywords

Topology Mapping, MPI Graph Topologies

1. MOTIVATION

The number of nodes in the largest computing systems, and, hence, the size of their interconnection networks, is increasing rapidly: The Jaguar system at ORNL has over 18,000 nodes and larger systems are expected in the near future. These networks are built by interconnecting *nodes* (*switches* and *processors*) with *links*. Pin count, power and gate count constraints restrict the number of links per switch; typical sizes are: 24 (InfiniBand), 36 (Myrinet, InfiniBand), or 6 (Sea Star or BlueGene/P). Different topologies are used to construct large-scale networks from crossbars; e.g., k-ary n-cubes (hypercube, torus), k-ary n-trees (fat-trees), or folded Clos networks. Networks also differ in their routing protocols.

As the number of nodes grows larger, the *diameter* of the network (i.e., the maximum distance between two processors) increases; for many topologies, the *bisection bandwidth* (i.e., the minimum total bandwidth of links that need to be cut in order to divide the processors into two equal sets) decreases relative to the number of nodes.

This effect is well understood and it is generally accepted that dense communication patterns (such as an all-to-all communication where each node communicates to each other) are hard to scale beyond petascale systems. Luckily, the communication patterns of many applications are relatively sparse (each node communicate with a few others), and dense communications can be replaced by repeated sparse communications (e.g., the all-to-all communication used for the transpose in a parallel Fast Fourier Transform can be replaced by two phases of group transposes, each involving only $\Theta(\sqrt{P})$ processors [17]). Furthermore, the communication pattern often has significant *locality*, e.g., when most communication occurs between adjacent cells in a 3D domain. However, an inappropriate mapping of processes to the nodes of the interconnection network can map a logical communication pattern that is sparse and local into traffic that has no locality.

Finding an allocation of processes to nodes such that the sparse application communication topology efficiently utilizes the physical links in the network is called *topology mapping*. The problem has been much studied for regular communication graph and regular interconnection network topologies. In practice, both graphs are likely to be irregular: The communication pattern may be data-dependent (e.g., for finite-element on irregular meshes); it may consist of a superposition of multiple regular graphs (e.g., for computations that combine nearest-neighbor communications with global communication). The interconnection network may have a complex topology, with different links having different bandwidths (e.g., copper vs. optics), and with some links being disabled. The general problem has been much less studied.

ICS'11, May 31–June 4, 2011, Tuscon, Arizona, USA.

Our previous argument suggests that mapping regular and irregular applications to the network topology is becoming more and more important at large scale. MPI offers support for topology mapping. A user can specify the (regular or irregular) communication topology of the application and request the library to provide a good mapping to the physical topology [16, 7]. An MPI implementation then re-numbers the processes in the communicator so as to improve the mapping.The scalability and usability of the topology interface was recently improved in MPI-2.2 [12] to allow a scalable specification and edge weights that represent communication characteristics. Finding a good mapping is non trivial and MPI implementations tend to use the trivial identify mapping.

Our work supports the optimization of arbitrary process topologies for arbitrary network topologies and thus an efficient implementation of the MPI process topology interface. This enables transparent and portable topology mapping for all network topologies. Our work also addresses heterogeneous networks such as PERCS, where different physical links may have different bandwidths.

Our implementation is intended for renumbering processes as suggested by MPI, however, the developed techniques and our open-source library can also be applied to other parallel programming frameworks such as UPC or CAF.

1.1 Related Work

The mapping of regular Cartesian structures to different target architectures is well understood. Yu, Chung, and Moreira present different topology mapping strategies of torus process topologies into the torus network of BlueGene/L [23]. Bhatelé, Kalé and Kumar discuss topology-aware load-balancing strategies for molecular dynamic CHARM++ applications [2]. Their analysis enables mapping from mesh and torus process topologies to other mesh and torus network topologies and provides performance gains of up to 10%.

Several researchers investigated techniques to optimize process mappings with arbitrary topologies on parallel computers. Bokhari [3] reduces the mapping problem to graph isomorphism. However, his strategy ignores edges that are not mapped. It was shown later that such edges can have a detrimental effect on the congestion and dilation of the mapping. Lee and Aggarwal [15] improve those results and define a more accurate model which includes all edges of the communication graph and propose a two-stage optimization function consisting of initial greedy assignment and later pairwise swaps. Bollinger and Midkiff [4] use a similar model and simulated annealing to optimize process mappings.

Träff proposes an implementation strategy for strictly hierarchical networks such as clusters of SMPs [22]. He defines different optimization criteria and shows the potential of MPI topology mapping for several artificial graphs.

2. TOPOLOGY MAPPING

2.1 Terms and Conventions

We use a notation that extends that used for graph embeddings [19]. The formulation is similar to the fluid flow approximation used to study Internet traffic [13]. We represent the (logical) communication pattern using a weighted, directed graph $\mathcal{G} = (V_\mathcal{G}, \omega_\mathcal{G})$ $V_\mathcal{G}$ is the set of processes; the *weight* $\omega(uv)$ of the edge connecting $u \in V_\mathcal{G}$ to $v \in V_\mathcal{G}$ represents the volume of communication from process u to process v; the weight is zero if no such communication occurs. The graph $\mathcal{G}$ might be disconnected and isolated vertices can exist – representing the concurrent execution of multiple unrelated jobs.

Likewise, the (physical) interconnection network is represented by a weighted, directed graph $\mathcal{H} = (V_\mathcal{H}, C_\mathcal{H}, c_\mathcal{H}, \mathcal{R}_\mathcal{H})$. $V_\mathcal{H}$ is the set of physical nodes (processors and switches). If $u \in V_\mathcal{H}$ then $C_\mathcal{H}(u)$ is the number of processes that can be hosted at u (this represents multicore processors); $C_\mathcal{H}(u) = 0$ if u contains no processors (e.g., is a switch). $c_\mathcal{H}(uv)$ is the *capacity* (bandwidth) of the link connecting u to v (zero if there is no such link).

The function $\mathcal{R}_\mathcal{H}$ represents the routing algorithm. Let $\mathcal{P}(uv)$ be the set of simple paths (paths where each edge occurs at most once) connecting node $u \in V_\mathcal{H}$ to node $v \in V_\mathcal{H}$ For each pair of nodes uv, $\mathcal{R}_\mathcal{H}(uv)$ is a probability distribution on $\mathcal{P}(uv)$. Thus, if $p \in \mathcal{P}(uv)$ then $\mathcal{R}_\mathcal{H}(uv)(p)$ is the fraction of traffic from u to v that is routed through path p. In practice, routing algorithms tend to use a small fraction of the possible paths (e.g., only shortest paths), and the traffic is often distributed evenly across all used paths.

The topology mapping is specified by a function $\Gamma\colon V_\mathcal{G} \to V_\mathcal{H}$ which maps the vertices of $\mathcal{G}$ (processes) to vertices in $V_\mathcal{H}$ (nodes) such that no more than $C(v)$ vertices in $\mathcal{G}$ are mapped to each vertex $v \in V_\mathcal{H}$. We use the terms mapping and embedding interchangeably.

We now define two quality measures for a mapping: *Worst Case Congestion* (for short, *congestion*) and *Average Dilation* (for short, *dilation*). Let $|p|$ denote the length of path p. Then, the expected dilation of an edge uv of the communication graph is defined as

$$\mathit{Dilation}(uv) = \sum_{p \in \mathcal{P}(\Gamma(u)\Gamma(v))} \mathcal{R}_\mathcal{H}(\Gamma(u)\Gamma(v))(p) \cdot |p| \quad (1)$$

$\mathit{Dilation}(uv)$ is the average length of the path taken by a message sent from process u to process v. The average dilation is computed by weighting each inter-process communication by its frequency:

$$\mathit{Dilation}(\Gamma) = \sum_{u,v \in V_\mathcal{G}} \omega_\mathcal{G}(uv) \cdot \mathit{Dilation}(uv) \quad (2)$$

Dilation is the average number of edges traversed by packets – hence is a measure of the total "communication work" performed by the interconnection network; it is indicative of the total energy consumption of the interconnection network.

The congestion of a link uv of the interconnection network is the ratio between the amount of traffic on that link and the capacity of the link. The total traffic crossing an edge $e \in E_\mathcal{H}$ is

$$\mathit{Traffic}(e) = \sum_{u,v \in V_\mathcal{G}} \omega_\mathcal{G}(uv) \left(\sum_{p \in \mathcal{P}(\Gamma(u)\Gamma(v)), e \in p} \mathcal{R}_\mathcal{H}(\Gamma(u)\Gamma(v))(p) \right) \quad (3)$$

The congestion of edge e is defined as

$$\mathit{Congestion}(e) = \frac{\mathit{Traffic}(e)}{c_\mathcal{H}(e)}, \quad (4)$$

and the worst-case congestion is

$$\mathit{Congestion}(\Gamma) = \max_e \mathit{Congestion}(e) \quad (5)$$

Congestion(Γ) is a lower bound on the time needed for communication. Both congestion and dilation can be computed in polynomial time.

The chosen representation embodies certain assumptions that are satisfied in many cases:

We assume that bandwidth between processes hosted at the same processor node is practically unbounded. To do so formally, we add to each processor node a self-loop with infinite capacity.

We assume that switches are not a performance bottleneck: the traffic flowing through a switch is constrained by the bandwidth of the incoming and outgoing links, but not by the internal switch structure. If this is not the case, the internal switch structure needs to be represented, too.

We assume oblivious routing: the distribution of traffic between two nodes does not depend on other ongoing traffic.

We assume that the routing algorithm is fixed, and does not depend on the embedded communication graph: The knowledge of the application communication pattern is used to map processes to processors, but is not used to change the routing algorithm. The formalism can be adjusted to handle routing protocols that are application dependent, in which case, the routing function becomes part of the mapping.

The mapping problem is often expressed as a permutation of processes, following an initial assignment of processes to processors. The remapping is defined by a permutation π on the set $0 \ldots P-1$ of processes. For example, in MPI, the user can specify a logical communication graph for a communicator, and request that the processes in the communicator be physically mapped so as to improve the performance of this communication pattern; the permutation π is returned as a new rank order for the processes in the communicator if the `reorder` argument is set to true.

2.2 Practical Issues

The mapping framework presented in this paper assumes that a communication pattern is defined once for all processes running on the system, and the processes are mapped to processors once. In practice, it may be advantageous to periodically readjust the mapping; and remapping of the processes of a job may be restricted to the set of nodes allocated to the job. We discuss below how our framework can be extended to handle these concerns; a detailed performance analysis of these enhancements is beyond the scope of this paper.

The jobs running on the system can be periodically reconfigured, based on the observed communication pattern and the overall network topology. A remapping might be beneficial whenever the communication pattern of a job changes, or when jobs start or terminate. When we do so, we need to balance the overhead of remapping against the benefit of improved communication performance. We do not consider in this paper the problem of selecting an optimal remapping schedule, and focus only on the choice of an optimal mapping when (re)mapping is performed.

If each job is mapped independently, then the host graph for a job is taken to be the partition used for this job (we assume space partitioning): the processors allocated to the job and the switches that can be used to route between these nodes. If there is little interference between the traffic of different parallel jobs, then the capacity of each link in this host graph equals its physical capacity; if the interference is significant, then the traffic of other jobs can be represented as a reduction in the capacity of edges in the host graph seen by the job being mapped. A significant change in the background traffic may necessitate a remapping of processes to nodes.

2.3 An Example Mapping

Figure 1 shows a simple example. The host network topology $\mathcal{H}$ is a 2-ary 3-cube (3D cube) with $V_{\mathcal{H}} = \{0,1\}^3$ and $E_{\mathcal{H}} = \{(u,v) \in V_{\mathcal{H}} \times V_{\mathcal{H}} | \, u \text{ and } v \text{ differ in one bit}\}$; we assume dimension order routing in x,y,z-order. We use a four process job with the communication topology shown in Figure 1(b) as example. A possible initial mapping that maps $(0,1,2,3)$ to $(000, 111, 101, 100)$, in this order, is shown in Figure 1(c); the edges in $\mathcal{H}$ are annotated with the number of connections that are routed through the edge. The maximum congestion and dilation for this initial mapping are 2 and 3, respectively. A better mapping, shown in Figure 1(d), maps $(0,1,2,3)$ to $(101, 111, 000, 100)$, with both maximum congestion and dilation of 1.

2.4 The Mapping Problem

The mapping problem can be defined as finding a mapping Γ that minimizes some measure of the congestion or dilation. In this work, we focus on minimizing the maximum congestion (the algorithm runtime) and average dilation (the needed power to move the data). Our work is equally applicable to other optimization metrics.

We define the Topology Mapping Problem TMP as the problem of deciding if there exists a mapping Γ (or permutation π) that has congestion less or equal to x.

THEOREM 1. *TMP is NP-complete.*

PROOF. The congestion of a mapping can be computed in polynomial time, using Equations (3), (4) and (5). It follows that TMP is in NP: The NP algorithm guesses a mapping Γ, computes its congestion and returns TRUE is the congestion is less than x.

We now show a reduction to the "MINIMUM CUT INTO BOUNDED SETS" NP-complete problem' [10, ND17] to conclude our proof. The (reduced) min cut problem takes as input an undirected graph $G =< V, E >$, two specified vertices $s, t \in V$ and an integer L; it decides whether there exist a partition of the vertices into two disjoint sets V_1, V_2 such that $s \in V_1$, $t \in V_2$, $|V_1| = |V_2|$ and the number of edges between V_1 and V_2 is no more than L. an

Let $G =< V, E >, s, t, L$ be an instance of the min-cut problem. Let $P = |V|$. We construct a "dumbbell" host graph $\mathcal{H} = (V_{\mathcal{H}}, E_{\mathcal{H}}, C, c)$ that consists of two fully connected graphs $L_1 = L_2 = K_{P/2}$ and a single bidirectional edge $\bar{e}$ between arbitrary vertices $u_1 \in L_1$ and $u_2 \in L_2$. We set $C(v) = 1$ and $c(e) = 1$ for all edges, except edge u_1u_2; $c(u_1u_2) = c(u_2u_1) = P$. The construction is shown, for $P = 8$ in Figure 2. The routing function $\mathcal{R}_{\mathcal{H}}$ is defined to

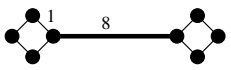
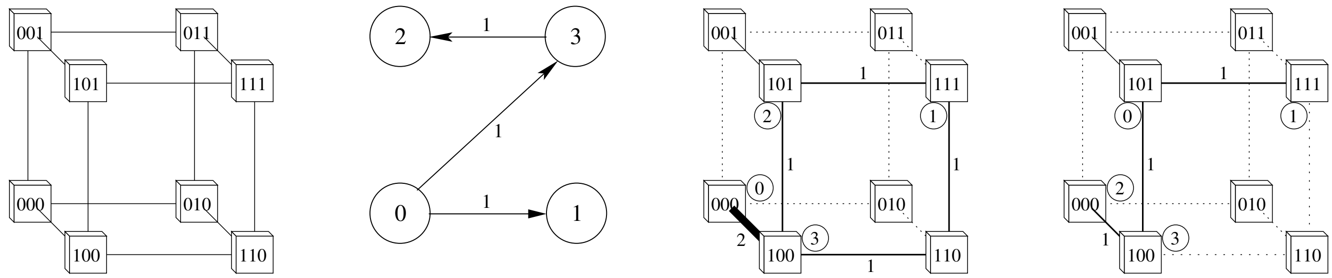


Figure 2: Dumbbell graph

route all traffic between two nodes in $\mathcal{H}$ through the unique shortest path connecting them.

We define the communication graph $\mathcal{G}$ to be the graph G with weight $\omega(e) = 1$ for each edge $e \in E$; the edge st is given weight P^4 if $st \notin E$, and weight $P^4 + 1$ if $st \in E$.



(a) Network Topology $\mathcal{H}$. (b) Process Topology $\mathcal{G}$. (c) Mapping Γ_1. (d) Mapping Γ_2.

Figure 1: A simple example for topology-aware mappings.

Any mapping of $\mathcal{G}$ to $\mathcal{H}$ defines a partition of $V_\mathcal{G}$ into two equal size sets $V_1 = \Gamma^{-1}(L_1)$ and $V_2 = \Gamma^{-1}(L_2)$. A mapping that minimizes congestion must map $\{s,t\}$ to $\{u_1,u_2\}$: This results in a congestion of $\leq P^3 + P$ whereas any other mapping will result in a congestion of $\geq P^4$ (since traffic from s to t will flow through an edge of capacity 1). In such a mapping, the most congested edge will be the edge u_1u_2; its congestion will be

$$P^3 + \frac{1}{P} \cdot |\{(v_1v_2) \in E \text{ s.t. } v_1 \in V_1 \text{ and } v_2 \in V_2\}|$$

Thus, any solution to the TMP instance, with $x = P^3 + L/P$, can be used to build a solution to the partition problem in polynomial time. It is easy to see that the converse is also true. □

2.5 Restricted Mapping Problem

We shall focus, from now on, on the simpler problem where routing only uses shortest paths and all shortest paths are used with equal probability.

The dumbbell graph $\mathcal{H}$ used in the proof of Theorem 1 has a unique shortest path between any two nodes; the routing function routes all traffic between two nodes on that shortest paths. Thus, the problem of finding an optimal mapping is still NP-hard if routing is restricted as above.

For the restricted routing problem, we do not need to specify explicitly the routing function; the host graph is defined as $(V_\mathcal{H}, C_\mathcal{H}, c_\mathcal{H})$, and the routing function is determined implicitly. The number of shortest paths between two nodes can be exponential in the number of vertices, so that the concise representation of the host graph can be exponentially smaller than an explicit one. Therefore, it is not obvious that computing the congestion or dilation of a mapping takes polynomial time, with this input representation. We show this is the case, below.

Determining the dilation of an edge $(u,v) \in \mathcal{G}$ is straightforward by computing the length of the shortest path from $\Gamma(u)$ to $\Gamma(v)$ in $\mathcal{H}$. This can be implemented with single-source-shortest-path (SSSP) from each vertex in time $\mathcal{O}(|V_\mathcal{G}| \cdot (|E_\mathcal{H}| + |V_\mathcal{H}| \cdot \log |V_\mathcal{H}|))$.

The congestion of an edge can be computed in polynomial time using an algorithm similar to the one used for computing betweenness centrality [5]. We present below a simple (nonoptimal) polynomial time algorithm.

Let $\sigma_i(s,t)$ be the number of paths of length i from s to t Then

$$\sigma_0(s,t) = \begin{cases} 1 & \text{if } s = t \\ 0 & \text{otherwise} \end{cases}$$

and

$$\sigma_i(s,t) = \sum_{u \text{ adjacent to } t} \sigma_{i-1}(s,u)$$

We compute $\sigma_i(s,t))$ for all pairs of nodes s,t and all $i \leq P$ in time $O(|V_\mathcal{H}|^2|E_\mathcal{H}|)$. The distance (the shortest path length) between any two nodes is equal to

$$d(s,t) = \min\{i : \sigma_i(s,t) > 0\}$$

and the number of shortest paths from s to t is equal to

$$\tau(s,t) = \sigma_{d(s,t)}(s,t)$$

Let $\tau(s,t,e)$ be the number of shortest paths from s to t going through edge e, Then, if $e = uv$, then

$$\tau(s,t,e) = \sum_{j+k=d(s,t)-1} \sigma_j(s,u) \cdot \sigma_k(v,t)$$

The traffic through edge e can now be computed as

$$\mathit{Traffic}(e) = \sum_{s,t} \omega(s,t) \frac{\tau(s,t,e)}{\tau(s,t)}$$

and the congestion equals to

$$\max_e \frac{\mathit{Traffic}(e)}{c(e)}$$

3. TOPOLOGY MAPPING ALGORITHMS

Previous work discussed different options for topology mapping. We start with an extension to a simple greedy algorithm which supports heterogeneous networks, discuss recursive bisection mapping and then discuss a new mapping strategy based on graph similarity. We also show how to support multicore nodes with established graph partitioning techniques.

3.1 Greedy Heuristic

Similar greedy algorithms have been proposed in previous work. Our greedy strategy, however, considers edge weights and thus enables mapping to heterogeneous network architectures.

Let the weight of a vertex $v \in V_\mathcal{G}$ be the sum of the weights of all edges $e = (v,u)$. The greedy mapping strategy starts at some vertex in $\mathcal{H}$, chooses the *heaviest* vertex in $\mathcal{G}$ and greedily maps its heaviest neighboring vertices in $\mathcal{G}$ to the neighboring vertices in $\mathcal{H}$ with the heaviest connections. The process is continued recursively. The detailed algorithm is presented in Algorithm 1. The greedy heuristic would find an optimal solution for the example in Figure 1 if it is started at vertex 100. This greedy approach is the most generic

Algorithm 1: Greedy Graph Embedding.

Input: Graphs $\mathcal{H}$ and $\mathcal{G}$, $C(v)$ for all $v \in V_\mathcal{H}$.
Output: Mapping $\Gamma : V_\mathcal{G} \to V_\mathcal{H}$, congestion $\rho(e)$ for all $e \in E_\mathcal{H}$.

1 $S \leftarrow V_\mathcal{G}$;
2 $Q \leftarrow$ empty priority queue;
3 $\hat{\omega} = \max_{e \in E_\mathcal{G}}\{\omega(e)\} \cdot |V_\mathcal{H}|^2$
4 initialize all $\rho(e)$ with $\hat{\omega}$; // forces minimal edge count
5 pick start vertex $s \in V_\mathcal{H}$;
6 **while** $S \neq \emptyset$ **do**
7 find vertex m with heaviest out-edges in S;
8 **if** $C(s) = 0$ **then**
9 pick new $s \in V_\mathcal{H}$ such that $C(s) \geq 1$;
10 $\Gamma(m) = s$; // map m to s
11 $S = S \backslash m$; // remove m from S
12 $C(s) = C(s) - 1$;
13 **foreach** $u | (m, u) \in E_\mathcal{G}$ and $u \in S$ **do**
14 $Q \leftarrow (m, u) | u \in S$; // add all neighbors
15 // ... of m that are still in S to Q
16 **while** $Q \neq \emptyset$ **do**
17 $(u, m) \leftarrow Q$; // heaviest edge in Q
18 **if** $C(s) = 0$ **then**
19 // find closest vertex $t \in V_\mathcal{H}$ to s with
20 // ... $C(t) \geq 1$ using a SSSP
21 // ... (e.g., Dijkstra's) algorithm
22 $s = t$;
23 $\Gamma(m) = s$; // map m to s
24 $S = S \backslash m$; // remove m from S
25 $C(s) = C(s) - 1$;
26 add $\omega((m, u))/c(f)$ to each $\rho(f)$ for all edges f on the shortest path $\Gamma(u) \rightsquigarrow \Gamma(m)$
27 **foreach** $u | (m, u) \in E_\mathcal{G}$ and $u \in S$ **do**
28 $Q \leftarrow (m, u) | u \in S$; // add all neighbors
29 // ... of m that are still in S to Q
30 subtract $\hat{\omega}$ from all $\rho(e)$; // correction from line 4

approach and works with all graphs and arbitrary values for $C(v)$.

THEOREM 2. *The runtime of the greedy mapping algorithm is* $\mathcal{O}(|V_\mathcal{G}| \cdot (|E_\mathcal{H}| + |V_\mathcal{H}| \log |V_\mathcal{H}| + |V_\mathcal{G}| \log |V_\mathcal{G}|))$.

PROOF. Each vertex in $\mathcal{G}$ will be removed exactly once from S. Picking a new vertex (lines 7/8) takes $\mathcal{O}(|V_\mathcal{G}|)$ with a linear scan. Checking if each of the neighbors of m should be added to Q (lines 13,24) can be done in $\mathcal{O}(|V_\mathcal{G}| \log |V_\mathcal{G}|)$. Line 16-19 issues an SSSP-run in $\mathcal{H}$ (e.g., Dijkstra's algorithm using a Fibonacci heap) for $\forall v \in V_\mathcal{G}$. Thus, the asymptotic run-time is $\mathcal{O}(|V_\mathcal{G}| \cdot (|E_\mathcal{H}| + |V_\mathcal{H}| \log |V_\mathcal{H}| + |V_\mathcal{G}| \log |V_\mathcal{G}|))$ □

3.2 Recursive Bisection Mapping

A second method to find a good topology mapping is recursive bisection. In this method, the weighted graphs $\mathcal{H}$ and $\mathcal{G}$ are recursively split with minimum weighted edge-cut into equal halves to determine the mapping. This technique proved successful to determine "static mappings" in the software package SCOTCH [18].

The minimal edge cut in the bisections maps "heavy" clusters in $\mathcal{G}$ to "strong" clusters in $\mathcal{H}$. Thus, this mechanism

Algorithm 2: Function *map_recursive*().

Input: Graphs $\mathcal{H}$ and $\mathcal{G}$, $C(v)$ for all $v \in V_\mathcal{H}$.
Output: Mapping $\Gamma : V_\mathcal{G} \to V_\mathcal{H}$.

1 // pre-condition: $\sum_{v \in V_\mathcal{H}} C(v) == |V_\mathcal{G}|$
2 **if** *more than one vertex* $v \in V_\mathcal{H}$ *with* $C(v) \neq 0$ **then**
3 $(C_1, C_2) = \text{bisect}(\mathcal{H}, C)$;
4 $(\mathcal{G}_1, \mathcal{G}_2) = \text{bisect}(\mathcal{G})$;
5 **if** $\sum_{c \in C_1} c == |V_{\mathcal{G}_1}|$ **then**
6 map_recursive$(\mathcal{H}, \mathcal{G}_1, C_1)$;
7 map_recursive$(\mathcal{H}, \mathcal{G}_2, C_2)$;
8 **else**
9 map_recursive$(\mathcal{H}, \mathcal{G}_1, C_2)$;
10 map_recursive$(\mathcal{H}, \mathcal{G}_2, C_1)$;
11 **else**
12 // map all n vertices in $\mathcal{G}$ to vertex with load n in $\mathcal{H}$

is expected to compute relatively good mappings. However, Simon and Teng show that in some cases, the recursive bisection approach might result in bad p-way partitions [21].

THEOREM 3. *The runtime of the recursive mapping algorithm is* $\mathcal{O}(|E_\mathcal{G}| \log(|V_\mathcal{G}|) + |E_\mathcal{H}| \cdot |V_\mathcal{G}|)$.

PROOF. The runtime of the multilevel k-way partitioning approach to bisect a graph $G = (V, E)$ is $\mathcal{O}(|E|)$ [20]. The depth of recursive calls to bisect $\mathcal{G}$ is $\lceil \log_2(|V_\mathcal{G}|) \rceil$ and the size of the graph G is halved in each step. Thus, the total runtime is $\sum_{k=0}^{\lceil \log_2(|V_\mathcal{G}|)-1 \rceil} 2^k \mathcal{O}(|E_\mathcal{G}|)/2^k = \log_2(|V_\mathcal{G}|)|E_\mathcal{G}| = \mathcal{O}(|E_\mathcal{G}| \log(|V_\mathcal{G}|))$. The depth of recursive calls to bisect $\mathcal{H}$ is the same as for $\mathcal{G}$ because the number of processors in $\mathcal{H}$ ($\sum_{v \in V_\mathcal{H}} C(v) == |V_\mathcal{G}|$) is equal to the $|V_\mathcal{G}|$. However, in $\mathcal{H}$, all vertices are considered at each recursion level of the bisection (only edges cut in previous recursions are removed). If we assume that no edges are cut (removed), then the runtime is $\sum_{k=0}^{\lceil \log_2(|V_\mathcal{G}|)-1 \rceil} 2^k \cdot \mathcal{O}(|E_\mathcal{H}|) = \mathcal{O}(|E_\mathcal{H}|) \cdot (|V_\mathcal{G}| - 1) = \mathcal{O}(|E_\mathcal{H}| \cdot |V_\mathcal{G}|)$. □

We used the METIS library [20] to compute a (2,1+ϵ)-balanced bisection. The bisection had to be balanced in some rare cases. Our library does this by moving the vertex with the lowest cumulative edge weight from the bigger to the smaller partition.

In the following, we discuss a new algorithm based on graph similarity. This algorithm has significantly lower time complexity and improves dilation and congestion.

3.3 Mapping based on Graph Similarity

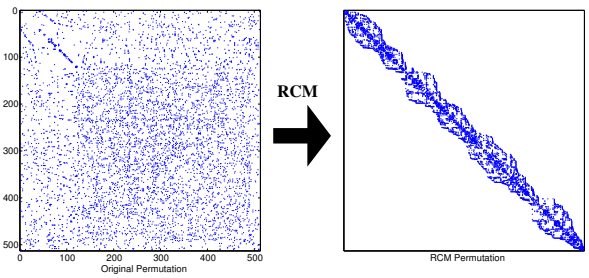
It is well known that there is a duality between graphs and sparse matrices and techniques from sparse linear algebra have been applied to solve graph problems [11]. The basic idea is that a graph's adjacency matrix can be modeled as a sparse matrix which enables the application of established techniques from sparse linear algebra.

A well-studied NP-hard problem is the reduction of the bandwidth of a sparse matrix which tries to eliminate non-zero elements that are far from the diagonal elements by re-numbering columns of the matrix. This can be used to bring the adjacency matrices two graphs $\mathcal{G}$ and $\mathcal{H}$ in a similar shape. This technique effectively transforms both graphs into a shape where edges are localized. The Reverse Cuthill

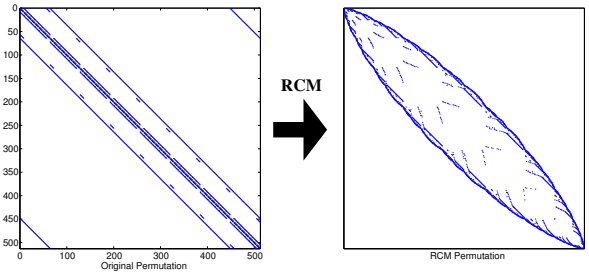
McKee (RCM) algorithm [6] is a successful heuristic for the bandwidth reduction problem.

RCM mapping applies the RCM algorithm to $\mathcal{G}$ and $\mathcal{H}$ to compute $\pi_\mathcal{G}$ and $\pi_\mathcal{H}$ and then computes the final process permutation $\pi(\pi_\mathcal{G}) = \pi_\mathcal{H}$, that is, $\pi = \pi_\mathcal{H} \circ \pi_\mathcal{G}^{-1}$. To handle mappings with $|\mathcal{G}| < |\mathcal{H}|$ correctly, all vertices v with $C(v) = 0$ are removed from $\mathcal{H}$. Despite potential disconnectivity on the sub-graph, RCM handles the proximity condition well and produces mappings with low dilation and congestion.

Figures 3(a) and 3(b) show the adjacency matrices for the problem graph $\mathcal{G}$ and the network graph $\mathcal{H}$, respectively.



(a) $\mathcal{G}$ adjacency map of the F1 matrix on 512 processes.



(b) $\mathcal{H}$ adjacency map for an 8x8x8 torus.

Figure 3: Example for RCM topology mapping of the F1 matrix to a torus network.

Figure 3(a) shows the adjacency matrix of the communication topology for a sparse matrix-vector product of the F1 matrix on 512 processes. This represents one of our application-use-cases and described in detail in Section 5.2. Figure 3(b) shows the physical topology of an 8x8x8 3-d torus network with 512 processes.

Both figures show the original permutation on the left and the RCM permutation on the right. RCM mapping is now based on the similarity between both RCM graphs. This effectively minimizes dilation and congestion.

THEOREM 4. *Let* $m = \max\{\textit{degree(v)}|v \textit{ in } V\}$. *RCM topology mapping computes a mapping in time* $\mathcal{O}(m_\mathcal{H} \log(m_\mathcal{H})|V_\mathcal{H}| + m_\mathcal{G} \log(m_\mathcal{G})|V_\mathcal{G}|)$.

PROOF. The complexity of RCM is $\mathcal{O}(m \log(m)|V|)$ [6] where $m = \max\{\text{degree(v)}|v \text{ in } V\}$. The algorithm applies RCM to $\mathcal{H}$ and $\mathcal{G}$ and the mapping can be computed from the results in $\mathcal{O}(|V_\mathcal{G}|)$. □

The discussions in the introduction suggests that $m_\mathcal{H} = \mathcal{O}(1)$ and scalable parallel algorithms often have $m_\mathcal{G} = \mathcal{O}(\log(|V_\mathcal{G}|))$. RCM is thus significantly faster than the greedy and the recursive mapping approaches and is a good candidate for large-scale systems.

3.4 Supporting Multicore Nodes

If compute nodes (vertices $v \in V_\mathcal{H}$) execute more than one process, then a graph partitioner can be used to divide $\mathcal{G}$ before other mapping strategies are applied. The common case where each allocated node executes the same number of processes $C(v) = p \; \forall v \in \Gamma(V_\mathcal{G})$ and the topology graph $\mathcal{G}$ needs to be partitioned into P/p equal pieces is supported by graph partitioners.

This technique benefits from the long experience in serial and parallel graph partitioning. Multiple heuristics for (k,1+ϵ)-balanced partitioning using geometric, combinatorial, spectral and multilevel schemes exist [8, §18].

Libraries, such as METIS [20] or SCOTCH [18] and their parallel versions offer optimized partitioning heuristics. However, most graph partitioners cannot guarantee perfectly (k,1)-balanced but (k,1+ϵ)-balanced partitions (for small ϵ). Thus, the partition might need to be corrected to be (k,1)-balanced. We use the ParMeTiS partitioner (Multilevel k-way Partitioning in $\mathcal{O}(|E_\mathcal{G}|)$ [20]) to compute (k,1+ϵ)-balanced partitions and balance the partitions if necessary.

3.5 Improving the Initial Solution

We now describe a heuristic that might further improve the found solution as was used in several previous works. Several heuristics exist for such problems. Threshold Accepting [9] is an improved algorithm for simulated annealing or hill climbing which takes an initial solution and tries to optimize it further by searching a local minimum. We use 20 iterations in the inner optimization loop and a time limit to determine the number of outer optimization iterations. Candidate solutions are modified by swapping two random positions in the mapping π. We will introduce a fast algorithm to estimate the congestion in Section 5.1 which is also used as weight function to minimize the optimization in our TA implementation. The asymptotic running time of each iteration of TA is equal to the running time of Algorithm 3 (cf. Theorem 5).

In the next section we describe how to effectively compose all strategies into a topology mapping framework and apply them to real-world network architectures.

4. A TOPOLOGY MAPPING LIBRARY

Several problems need to be solved in addition to the mapping problem in order to use topology mapping in practice. We show a mechanism that supports most interconnection networks and implement it in a portable library to perform automated topology mapping for parallel applications.

4.1 Determining the Network Topology

The first practical problem is to determine the network topology graph $\mathcal{H}$. This task can be handled manually based on the physical connections between compute nodes. However, many interconnection networks offer automated tools to query the topology and connectivity. Table 1 lists the tools that can be used to query the topology of different networks. All listed networks are supported by our implementation. The result of running those tools, the graph $\mathcal{H}$, is stored as an adjacency list in a configuration file on disk. When a parallel job starts up, each process loads $\mathcal{H}$ and identifies the vertex that it runs on. The identification is done with the hostname of the machine, that is, each vertex in the $\mathcal{H}$ file (representing a compute node) has its hostname

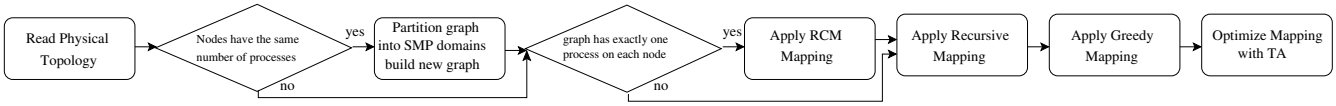


Figure 4: Optimization process flow. The mapping with the lowest congestion is chosen at the end.

Input: Graphs $\mathcal{H}$ and $\mathcal{G}$, mapping $\Gamma : V_\mathcal{G} \to V_\mathcal{H}$.
Output: congestion $\rho(e)$ for all $e \in E_\mathcal{H}$.
1 $\hat{\omega} = \max_{e \in E_\mathcal{G}} \{\omega(e)\} \cdot |V_\mathcal{H}|^2$
2 initialize all $\rho(e)$ with $\hat{\omega}$; // enforce paths with minimal number of edges in SSSP
3 **foreach** $e = (u, v) \in E_\mathcal{G}$ **do**
4 find shortest path $\Gamma(u) \rightsquigarrow \Gamma(v)$ in $\mathcal{H}$;
5 // implicitly minimizing congestion
6 increase edge weight $\rho(f)$ of each edge $f \in E_\mathcal{H}$ along path $\Gamma(u) \rightsquigarrow \Gamma(v)$ by $\omega(e)/c(f)$;
7 subtract $\hat{\omega}$ from all $\rho(e)$; // correct edge weights

Algorithm 3: Determine Congestion.

Interconnection Network (API)	Topology Query Tool(s)
Myrinet (MX)	`fm_db2wirelist`
InfiniBand (OFED)	`ibdiagnet & ibnet-discover`
SeaStar (Cray XT)	`xtprocadmin & xtdb2proc`
BlueGene/P (DCMF)	`DCMF API`

Table 1: Supported Topology Query Tools.

as attribute attached. Each process p has now access to the initial mapping $\Gamma(p)$ which is often not under the user's control (e.g., determined by the batch system). BlueGene/P is an exception where $\mathcal{H}$ is created on the fly after querying the Deep Computing Messaging Framework (DCMF) for all topology information.

4.2 Composing a Mapping Strategy

We now seek to permute processes so as to reduce congestion and dilation. We assume that interference with other jobs is negligible, so that congestion can be computed from the network topology, the location of the allocated processes and the communication graph.

If all $C(v) = p$ are all equal, then an optional graph partitioning phase as described in Section 3.4 is used to divide $\mathcal{G}$ into P/p partitions. The *topomapper* library uses ParMETIS [20] to perform partitioning and corrects the resulting (k,1+ϵ)-balanced partitioning by moving vertices from partitions with more then p vertices to partitions with less than p vertices. The correction step moves vertices with the least cumulative edge weight. After this optional partitioning step, a new graph $\mathcal{G}'$ that contains the partitions as vertices with $|V'_\mathcal{G}| = P/p$ is created. Only inter-partition edges from $\mathcal{G}$ remain in $\mathcal{G}'$ and vertices are numbered from 0 to $\frac{P}{p} - 1$.

A second step applies Greedy, Recursive, or RCM mapping as described in Sections 3.1, 3.2, and 3.3 respectively. The mappings can be optimized additionally by applying the threshold accepting algorithm discussed in Section 3.5.

The complete control flow of the optimization process is shown in Figure 4. All processes apply the optimization process, subsets of processes can perform different optimizations, for example, each process chooses a different starting vertex for the Greedy mapping. The permutation with the lowest congestion is chosen at the end of the optimization process and returned.

5. EXPERIMENTAL ANALYSIS

We analyze the efficiency and performance of mappings of irregular process topologies onto different multicore network topologies.

5.1 A Fast Algorithm to Assess Congestion

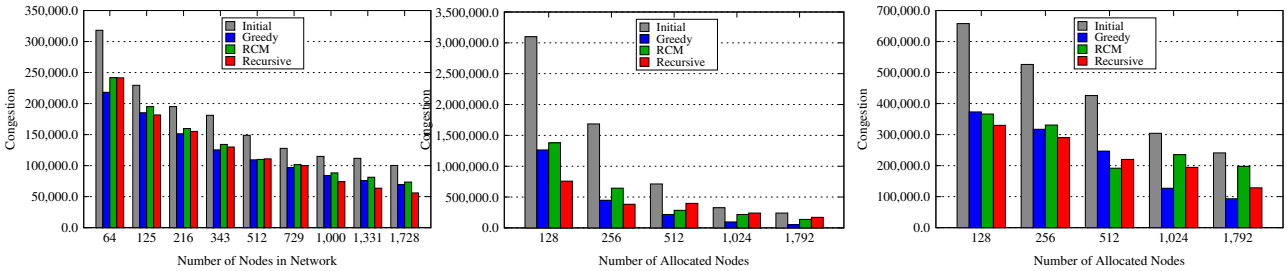
Assessing the congestion with the technique described in Section 2.5 is, due to the high time complexity ($\mathcal{O}(|V_\mathcal{H}|^2|E_\mathcal{H}|) = \mathcal{O}(|V_\mathcal{H}|^4)$), impractical at large scale. Thus, we propose a portable and fast heuristic for determining the approximate congestion of all edges in $\mathcal{H}$ in Algorithm 3. The congestion is computed by repeated shortest path calculations. To find the minimal congestion, the edge weights along used (shortest) paths are updated after each search to reflect the current load. This leads to an automatic balancing of edges along all paths. However, with this scheme, paths with more edges and less congestion on those edges might have shorter weighted distances. This is avoided by initializing the edges to a high weight $\hat{\omega} = \max_{e \in E_\mathcal{G}} \{\omega(e)\} \cdot |V_\mathcal{H}|^2$ so that a path with less edges always has a shorter weighted distance regardless of the congestion. Among all paths with the minimal number of edges, those with minimal congestion are then preferred.

THEOREM 5. *The runtime of Algorithm 3 is* $\mathcal{O}(|E_\mathcal{G}| \cdot (|E_\mathcal{H}| + |V_\mathcal{H}| \cdot \log |V_\mathcal{H}|))$.

PROOF. Exactly one SSSP-run on $\mathcal{H}$ (e.g., Dijkstra'a algorithm using a Fibonacci heap) is started for each edge in $\mathcal{G}$ (line 3–4). Thus, the asymptotic runtime of Algorithm 3 is $\mathcal{O}(|E_\mathcal{G}| \cdot (|E_\mathcal{H}| + |V_\mathcal{H}| \cdot \log |V_\mathcal{H}|))$. □

5.2 Real-world Irregular Process Topologies

Sparse matrix-vector multiplication is one of the most important kernels in large-scale scientific applications and can be used to solve a large class of scientific computing problems [8]. In order to capture the characteristics of real irregular applications, we use parallel sparse matrix-vector products with real-world input matrices from the University of Florida Sparse Matrix Collection [7]: F1, nlpkkt240, and audikw_1. All three matrices represent un-



(a) Congestion on 3d Torus (nlpkkt240). (b) Congestion in PERCS (nlpkkt240) (c) Congestion for Juropa (audikw_1)

Figure 5: Topology mapping results for different topologies.

structured matrices/grids. F1 and audikw_1 are symmetric stiffness matrices—approximating elasticities in structural mechanics—modeling automotive crankshafts. The nlpkkt240 matrix is the largest matrix in the collection and represents a nonlinear programming problem for a 3d PDE-constrained optimization. Table 2 lists the dimensions and number of non-zero (nnz) entries for each matrix.

Matrix Name	Rows and Columns	NNZ (sparsity)
F1	343,791	26,837,113 ($2.27 \cdot 10^{-4}$%)
audikw_1	943,695	39,297,771 ($4.4 \cdot 10^{-5}$%)
nlpkkt240	27,993,600	401,232,976 ($5 \cdot 10^{-7}$%)

Table 2: Properties of the test matrices.

The vector of a sparse matrix-vector product is initially distributed block-wise. Each element i of the vector requires all elements j where the matrix element $A_{i,j}$ is non-zero. Most matrix elements are zero and the pattern of non-zero elements depends on the structure of the input system. Thus, in order to minimize the communication, scientific codes usually partition the matrix with a graph partitioner and redistribute matrix and vector elements accordingly. We use ParMeTiS to find a decomposition of the matrix in order to minimize communication.[1] The domain-optimized decomposition is then used to derive the number of vector elements that need to be communicated from and to each process. We build a weighted MPI-2.2 process topology that reflects the communication requirements of the decomposition. The resulting distributed topology communicator [12] is used by our topology mapping library to optimize the process-to-node mapping.

All experiments presented below use the same input matrices which means that they simulate a strong scaling problem. We ran all experiments with up to 1,792 processes (or the maximum supported by the physical topology). All presented results used TA to refine the mapping until otherwise noted. Results without TA are omitted for brevity. TA improved the congestion between 2% and 9%.

5.3 Petascale Network Topologies

We investigate topologies that are used to build current and future petascale-class systems: A three-dimensional torus is used in the Cray XT-5 and IBM Blue Gene architectures. The IBM PERCS network [1] uses a heterogeneous hierarchical fully-connected topology to construct a 10 petaflop computer.

[1]This step should not be confused with graph partitioning for multicore topology mapping even though it uses the same tools!

We present only one representative matrix for each network topology due to space limitations. We also analyze only one process per node in Sections 5.3 and 5.4 because we assume that hybrid programming schemes will be used to exploit the full potential of those machines.

5.3.1 Three-Dimensional Torus

A k-dimensional torus of size $x_1 \times \cdots \times x_k$ has vertices $< m_1 \dots m_k >$, where $0 \leq m_i < x_i$ and edges connecting $< m_1 \dots m_k >$ to $< m_1 \dots m_i \pm 1(\mod x_i) \dots m_k >$, for $i = 1, \dots, k$.

We investigate 3-dimensional toruses with cube topologies ($x_1 = x_2 = x_3$) which maximize bisection bandwidth. Processes are mapped in lexicographical order, i.e., $< 0,0,0 >, < 0,0,1 >, \dots, < x_1 - 1, x_2 - 1, x_3 - 2 >, < x_1 - 1, x_2 - 1, x_3 - 1 >$ in the initial allocation.

Figure 5(a) shows the maximum congestion of mapping the communication topology that results from a domain-decomposition of the nlpkkt240 matrix to different 3d-Torus networks. The relative gain over the initial consecutive mapping increases with the network size. Greedy mapping reduces the maximum congestion by 27% for a 3^3 and up to 32% for a 12^3 torus network. RCM is slightly worse than greedy in all configurations, however, it reduces the dilation significantly. The recursive mapping algorithm delivers the best results at large scale where it outperforms greedy with a relative gain of 44% for a 12^3 network. The average dilation a 12^3 torus was 9.00, 9.03, 7.02, 4.50 for the initial, Greedy, RCM, and Recursive mappings, respectively. Recursive reduces the average dilation by 50% and might thus result in lowest power consumption.

The memory overhead to start the physical topology was between 0.63 kiB for 3^3 and 31.20 kiB for 12^3 respectively. It shows that RCM takes basically constant time (never more than 0.01 s) and Greedy and Recursive take up to 1s while TA can be infeasibly expensive with nearly 10 minutes.

5.3.2 PERCS Network

The PERCS topology [1] was designed by IBM to construct a multi-petaflop machine. The network consists of three different link types: LL, LR, and D with different speeds. Each endpoint connects to 7 neighbors via LL links with a rate of 24 GiB/s, 24 neighbors via LR links at a rate of 5 GiB/s, and up to 16 neighbors via D links with 10 GiB/s. Each stage (link-type) forms a fully-connected network. A set of nodes that is fully connected with LL links is called drawer and a set of nodes fully-connected with LL+LR links is called supernode; supernodes are fully connected by D links. Each drawer consists of 8 nodes and each supernode consists of 4 drawers. The size of the network is determined by the number of D links. The maximum distance between

any two nodes is three. A detailed description of the network and the topology can be found in [1]. We assume 9 D links per node which results in 9248 nodes total and we connect all D links randomly. The total topology occupies 1,445 kiB in main memory.

For the first simple example, we assume that processes are allocated and mapped consecutively to nodes in drawers and then drawers in supernodes. Figure 5(b) shows the result of topology mapping for this heterogeneous network architecture. Topology mapping can reduce the maximum congestion by up to 80% (P=1,792). The huge improvement comes from the effective exploitation of the different link speeds in the greedy strategy. RCM performs consistently slightly worse than greedy because it does not take the link capacities into account. Recursive achieves with 1.82 a lower average congestion than Greedy with 2.89 with 1,728 nodes. The benefits grow with the size of the allocation.

Again, RCM mapping consistently takes less than 0.01s while Greedy grows from 0.8s to 22s and Recursive from 4.51s to 7.51s. TA took 41 minutes at P=512 and was thus disabled for $P > 512$.

5.4 InfiniBand Network Topologies

We now investigate our topology mapping strategies on large-scale InfiniBand installations. We used the tools described in Section 4 to query the network topology of two large-scale systems, Juropa at the Jülich Supercomputing Center and Ranger at the Texas Advanced Computing Center. Both systems use InfiniBand topologies that are similar to fat-trees. The number of nodes in the systems were 3,292 for Juropa and 4,081 for Ranger. For our initial allocations, we use the order of hostnames like a batch-system does by default. As before, we assume one process per core in our analyses to investigate the quality of topology mapping separately from multicore mapping.

5.4.1 Juropa

Figure 5(c) shows the results for topology mapping of the communication patterns for the audikw_1 matrix on the Juropa cluster. The improvements are between 40% and 61% and grow with the number of mapped tasks. Greedy shows significantly better congestion results than RCM mapping. RCM provides a lower average dilation of 4.45 in comparison to Greedy with 5.8 and Recursive with 5.13 at P=1,792.

RCM is again fastest with less than 0.01s. Greedy takes between 0.16 s and 2.6s and Recursive between 0.63s and 1.21s, while TA is with up to 9 minutes only feasible at small scales. Juropa's complete topology occupied 87 kiB memory.

5.4.2 Ranger

Figure 6(a) shows the results of topology mapping on the Ranger cluster. The maximum congestion was improved by up to 50%, depending on the allocation size. Figure 6(b) shows the mapping times for the Ranger system. Again, Greedy performs significantly better than RCM at a much higher cost. RCM finished all mapping problems in less than 0.01s while Greedy used between 0.26s and 3.85s and Recursive between 0.76s and 1.5s. TA took up to 14 minutes for the largest problem and only improved it modestly. Ranger's complete topology occupied 134 kiB memory.

5.5 Benchmark Results

In our theoretical analysis and simulations, we made several assumptions on the (ideal) routing scheme and network behavior. The improvements reported by our mapping strategies are thus lower bounds and are hard to achieve in practice.

We now show benchmark results on Surveyor, an IBM BlueGene/P system at the Argonne National Lab, to demonstrate the utility of our topology mapping library and algorithms in practice.

As for the simulation, each process loads a part of the matrix, decomposes it with a graph partitioner, constructs an MPI-2.2 graph topology, and calls the topomapper library to optimize the mapping. The library exercises all options as described in Section 4 and returns an optimized mapping.

We measured the time to perform 100 communication phases in isolation and report the maximum time across all ranks before and after applying the mapping. We also compute a predicted time from the improvement in maximum congestion which is a lower bound to the actual improvement.

Figure 6(c) shows the time to perform the communication on the initial (consecutive) mapping, the time to perform the communication on an optimized (renumbered) mapping and the prediction of a run with 512 nodes. The mapping took 0.34s in all cases and the physical topology graph occupied 12 kiB memory. The measured performance gains lie between 10% and 18% depending on the matrix while the predictions were between 18% and 32%.

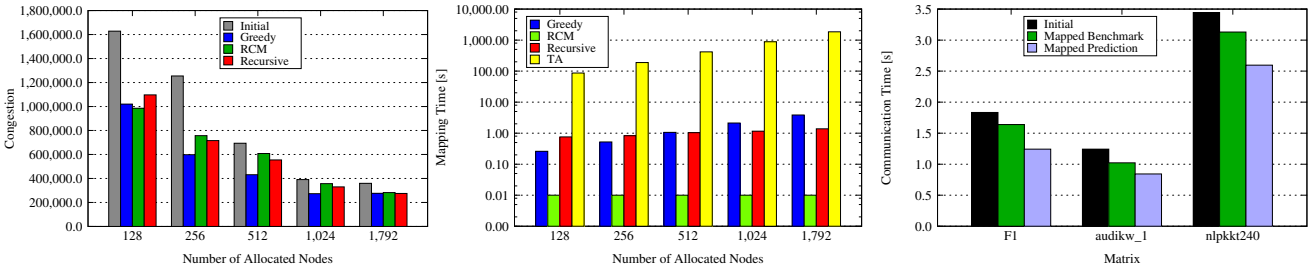
These experiments show that topology mapping leads to significant improvements in practical settings.

6. CONCLUSIONS AND FUTURE WORK

In this work, we defined the topology mapping problem and presented a proof that an finding an optimal solution to the problem is NP-hard. This opens the door to investigate the efficiency of different heuristics for topology mapping.

We propose different topology mapping algorithms that support arbitrary heterogeneous network and application topologies and showed their effective use in the context of sparse linear algebra computation. The proposed topology mapping algorithms have been implemented to support reordering in the intuitive distributed graph topology interface in MPI-2.2.

We showed improvements of the maximum congestion of up to 80% and our results indicate that the benefits of topology mapping grow with the system size. We analyzed the scalability of the different mapping approaches. Our theoretical and practical analysis shows that Greedy and Recursive are slower than RCM and that additional optimization with threshold accepting (TA) might be prohibitively expensive. Greedy scales approximately linearly with the system size for all our investigated application and network topologies which means that it might not be suitable for large mappings. Recursive mapping is faster but might result in worse congestion. However, RCM is fastest in theory and never took longer than 0.01s in our experiments. We also found that the Greedy performs well for minimizing congestion and Recursive and RCM for minimizing dilation. This creates interesting opportunities for further investigation. We conclude that TA can improve most mappings further but it is not scalable to large systems.



(a) Congestion on Ranger (nlpkkt240) (b) Times for Ranger (nlpkkt240) (c) Benchmark Results on BlueGene/P.

Figure 6: Topology mapping results for different networks.

Our proposed optimization framework utilizes the available parallelism in the system. It starts the Greedy algorithm at different source vertices on each node and simultaneously applies RCM and Recursive on one node each and selects the best solution found. We demonstrated speedups of up to 18% of the communication phase of a sparse matrix-vector multiplication on 512 BlueGene/P nodes.

We plan to investigate optimized strategies for initial process-to-node mappings on different architectures. The PERCS network topology presents multiple interesting challenges in this area.

Our implementation can immediately be used to optimize communication on petascale systems. However, the proposed mapping algorithms can scale to the size of exascale systems. The two metrics, maximum congestion and average dilation can be used to optimize and trade application runtime and power consumption on such systems. Exascale systems will need to exhibit substantially improved communication locality in order to achieve acceptable energy consumption [14]. The use of high quality mapping procedures will be essential to achieving this goal.

The topology mapper library is available at `http://www.unixer.de/research/libtopomap`.

Acknowledgments. We thank Peter Gottschling and Andrew Lumsdaine for many helpful discussions and comments. Thanks to Bernd Mohr for providing the Juropa topology and Len Wisniewski and the TACC for providing the Ranger topology. This work is supported by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois.

7. REFERENCES

[1] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony. The PERCS High-Performance Interconnect. In *Proc. of 18th Symposium on High-Performance Interconnects (HotI'10)*, Aug. 2010.

[2] A. Bhatelé, L. V. Kalé, and S. Kumar. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *ICS '09*, pages 110–116, New York, NY, USA, 2009. ACM.

[3] S. H. Bokhari. On the mapping problem. *IEEE Trans. Comput.*, 30(3):207–214, 1981.

[4] S. W. Bollinger and S. F. Midkiff. Heuristic technique for processor and link assignment in multicomputers. *IEEE Trans. Comput.*, 40(3):325–333, 1991.

[5] U. Brandes. A faster algorithm for betweenness centrality. *The Journal of Math. Sociology*, 25(2):163–177, 2001.

[6] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

[7] T. A. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 92, 1994.

[8] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors. *Sourcebook of parallel computing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[9] G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys.*, 90(1):161–175, 1990.

[10] M. Gary and D. Johnson. *Computers and Intractability: A Guide to NP-Completeness. New York: W H.* Freeman and Company, 1979.

[11] J. R. Gilbert, S. Reinhardt, and V. B. Shah. High-performance graph algorithms from parallel sparse matrices. In *PARA'06: Proceedings of the 8th international conference on Applied parallel computing*, pages 260–269, 2007.

[12] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, and J. L. Traeff. The Scalable Process Topology Interface of MPI 2.2. *Concurrency and Computation: Practice and Experience*, 23(4):293–310, Aug. 2010.

[13] R. Johari and D. Tan. End-to-end congestion control for the internet: delays and stability. *Networking, IEEE/ACM Transactions on*, 9(6):818 –832, Dec. 2001.

[14] P. Kogge et al. Exascale computing study: Technology challenges in achieving exascale systems. *DARPA Information Processing Techniques Office, Washington, DC*, 2008.

[15] S.-Y. Lee and J. K. Aggarwal. A mapping strategy for parallel processing. *IEEE Trans. Comput.*, 36(4):433–442, 1987.

[16] MPI Forum. *MPI: A Message-Passing Interface Standard. Version 2.2*, June 23rd 2009. `www.mpi-forum.org`.

[17] D. Pekurovsky. P3DFFT - Highly scalable parallel 3D Fast Fourier Transforms library. Technical report, 2010.

[18] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *HPCN Europe'96*, pages 493–498, 1996.

[19] A. L. Rosenberg. Issues in the study of graph embeddings. In *WG'80*, pages 150–176, London, UK, 1981.

[20] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, 2002.

[21] H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18:1436–1445, September 1997.

[22] J. L. Träff. Implementing the MPI process topology mechanism. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–14, 2002.

[23] H. Yu, I.-H. Chung, and J. Moreira. Topology mapping for Blue Gene/L supercomputer. In *SC'06*, page 116, New York, NY, USA, 2006. ACM.