J Supercomput (2017) 73:1691–1714
DOI 10.1007/s11227-016-1876-7

Topology mapping of irregular parallel applications on torus-connected supercomputers

Jingjin Wu[1] · **Xuanxing Xiong**[2] · **Eduardo Berrocal**[3] · **Jia Wang**[4] · **Zhiling Lan**[3]

Published online: 26 October 2016

Abstract Supercomputers with ever increasing computing power are being built for scientific applications. As the system size scales up, so does the size of interconnect network. As a result, communication in supercomputers becomes increasingly expensive due to the long distance between nodes and network contention. Topology mapping, which maps parallel application processes onto compute nodes by considering network topology and application communication pattern, is an essential technique for communication optimization. In this paper, we study the topology mapping problem for torus-connected supercomputers, and present an analytical topology mapping algorithm for parallel applications with irregular communication patterns. We consider our problem as a discrete optimization problem in the geometric domain of a

✉ Jingjin Wu
jwu45@uestc.edu.cn

Xuanxing Xiong
xxiong@synopsys.com

Eduardo Berrocal
eberroca@hawk.iit.edu

Jia Wang
jwang@ece.iit.edu

Zhiling Lan
lan@iit.edu

[1] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

[2] Design Group, Synopsys, Inc., Mountain View, CA 94043, USA

[3] Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA

[4] Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616, USA

torus topology, and design an analytical mapping algorithm, which uses numerical solvers to compute the mapping. Experimental results show that our algorithm provides high-quality mappings on 3-dimensional torus, which significantly reduce the communication time by up to 72%.

Keywords High-performance computing · Topology mapping · Communication optimization · Torus network · Analytical algorithm

1 Introduction

Large-scale scientific computing is vital for the advances in science and engineering, and typical applications range from quantum physics, material science, and geophysics, to astrophysics, cosmology, and many others. High-performance computing systems (i.e., supercomputers) with ever increasing computing power are being built to fulfill the demands of large-scale and complex scientific applications. As the system size scales up, better computing performance is achieved by using more compute nodes with multiple processing cores. Nevertheless, it is observed that the scaling of communication time usually degrades the overall scalability of many applications, thus becoming the performance bottleneck.

This phenomenon is basically due to the fact that the communication becomes increasingly expensive as the interconnection network becomes larger and more complex. The network diameter (i.e., the maximum distance between two nodes) keeps increasing, while the bisection bandwidth (i.e., the minimum total bandwidth between two equal parts of the supercomputer) relative to the number of nodes often decreases. As a result, the message latency is largely dependent on the distance between nodes and the congestion in the network.

The communication bottleneck problem has been well studied, and it is hard to scale dense communication patterns (e.g., all-to-all communications where each process communicates with all the other processes) beyond petascale systems. Fortunately, many scientific applications have sparse communication patterns with significant locality, e.g., the cells in a 3D Cartesian grid only communicate with adjacent cells. However, the actual traffic in the network may have no locality if an inappropriate mapping of processes onto nodes is used, resulting in high communication cost.

Given the network topology of the machine and the communication pattern of the application, finding a proper mapping of application processes onto physical nodes for reducing communication time is called topology-aware task mapping [3], or simply topology mapping [21]. Typically, the heavily communicating processes need to be mapped onto neighboring nodes so that the traffic is localized and the congestion in the network can be reduced.

According to the characteristics of inter-process communication, parallel applications can be divided into two categories: those with regular communication patterns, and those with irregular communication patterns. A regular pattern typically means that each process has the same number of communicating neighbors except for the boundary processes, e.g., 2D/3D mesh patterns are regular. Patterns without such property are considered to be irregular.

Torus-based networks [2,35] are commonly used in high-end supercomputers because of their linear scaling on per-node cost and their competitive communication performance [34]. For supercomputers with torus network topology, the research has been focused on mapping applications with regular 2D/3D mesh communication patterns [9,10,18,42], while the mapping of irregular communication patterns is less studied. In this paper, we explore the analytical placement technique [36,37] used by VLSI (very large-scale integration) physical design for topology mapping, and present an analytical topology mapping algorithm for parallel applications with irregular communication patterns on torus-connected supercomputers.

Our analytical topology mapping algorithm applies a two-stage strategy, namely global mapping and legalization, to determine an appropriate mapping of application processes onto physical nodes with minimal network traffic. In the global mapping stage, we approximate the topology mapping problem, which is discrete, by a continuous optimization problem that can be solved via analytical methods. This continuous optimization problem minimizes a measure of the total amount of traffic in the network, while at the same time establishing an almost-legal mapping of processes onto nodes. In the legalization stage, we derive a legal mapping from the almost-legal one via a novel migration algorithm. This algorithm minimizes the perturbation to the outcome of the global mapping stage in order to reduce the impact to network traffic. Both stages heavily depend on unconstrained quadratic programming whose optimal solution can be obtained by solving linear systems.

We evaluate our mapping algorithm on the 40-rack Blue Gene/P system at Argonne National Laboratory by using real-world applications with irregular communication patterns. Results show that our algorithm provides high-quality mappings on 3D torus, which significantly improves the communication performance. In particular, it significantly reduces the network traffic by up to 83%, achieving communication time reduction by up to 72%.

The rest of this paper is organized as follows. Section 2 introduces the background of topology mapping. Section 3 proposes the analytical mapping algorithm. After experimental results are shown in Sect. 4, we present concluding remarks in Sect. 5.

2 Background

2.1 Torus network topologies

An n-dimensional mesh network connects compute nodes into a mesh structure, where each node is directly connected with two other nodes in each physical dimension. An n-dimensional torus network is a mesh with additional links to connect the pair of nodes at the end of each physical dimension, so that the network diameter can be reduced by half. It is often designed to be reconfigurable to offer more scalability [5,6,8].

In practice, n-dimensional ($n \geq 3$) torus (see Fig. 1) is commonly used in supercomputers, including the IBM Blue Gene family and the Cray XT family. The network diameter can be a few dozens of hops. As studied in [9], for small and medium-sized messages, the message latency is dependent on the distance in hops. More importantly,

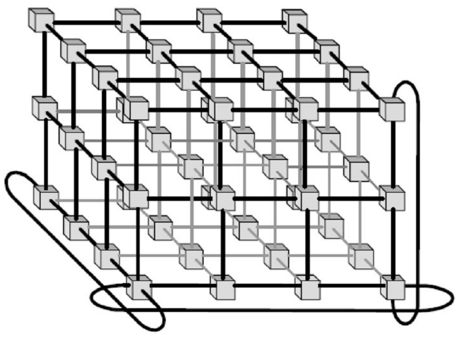
Fig. 1 A 3D torus network. Note that only *three wrap-around links* are shown for clarity of presentation. This figure is obtained from [35]

[9] also shows that the congestion in the network can increase message latencies significantly. For better communication performance, it is a must to explore efficient topology-aware mapping techniques to find proper mapping of processes onto nodes.

2.2 The topology mapping problem

For topology-aware mapping, the communication pattern of the application is represented as a communication graph. The network topology of the target computing platform is represented as a topology graph. In this work, we use an undirected communication graph $G_c(V_c, E_c)$ to model the communication pattern of the application. Each vertex in V_c represents a process, and each edge (i, j) in E_c has a weight $c_{i,j}$, which denotes the total amount of communication (in bytes) between processes i and j. In the topology graph $G_t(V_t, E_t)$, a vertex in V_t often denotes a compute node, and each edge in E_t denotes a direct link.

The mapping of processes onto nodes is specified by a labeling $\phi : V_c \rightarrow V_t$. The quality of mapping can be measured by hop-bytes [9], which is the total amount of inter-node communication (in bytes) weighted by the distance (in hops) between nodes. Let $d(\phi_i, \phi_j)$ be the distance, which is usually measured by the length of the shortest path (in hops) between ϕ_i and ϕ_j in the topology graph $G_t(V_t, E_t)$. Then the hop-bytes of a mapping ϕ can be computed as

$$\text{hop-bytes}(\phi) \triangleq \sum_{\forall (i,j) \in E_c} c_{i,j} d(\phi_i, \phi_j). \tag{1}$$

It represents the total amount of traffic in the interconnection network.

Congestion [21] is another important metric of mapping quality. The congestion on a link is defined as the total amount of messages (in bytes) transmitted on that link divided by the link capacity (i.e., bandwidth). The overall congestion of a mapping is the maximum congestion among all links, and it is a lower bound of the communication time. As there may be multiple shortest paths between two nodes, and supercomputers often use dynamic routing, it is often difficult to compute the congestion deterministically. Hence, the congestion metric is mainly of theoretical interest.

The mapping problem is often defined as finding a proper mapping that minimizes the communication cost. As studied in [21], finding the optimal mapping has been proven to be NP-hard.

2.3 Related works

A variety of algorithms have been proposed to map parallel application processes onto different topologies. These algorithms typically fall under the categories of physical optimization techniques and heuristic approaches. Specifically, physical optimization techniques include simulated annealing [26], genetic algorithms [15], graph contraction [7], and particle swarm optimization [31]. Although these optimization algorithms could provide good mapping quality, they take a long runtime to derive the mapping, and hence are not suitable for online topology mapping. In contrast, many heuristic approaches can better exploit the structure of the network topology and the communication pattern, thus being more efficient for practical use.

Graph embedding schemes were exploited to map processes onto nodes for 3D mesh and torus topologies in [42], and have been employed for scalable support of MPI topology functions in an IBM Blue Gene/L MPI library. Several mapping algorithms for regular 2D and 3D topologies were studied in [9]. MPI benchmarks were developed to evaluate the message latency in order to quantify the effects of network congestion, and geometric approaches were exploited to map regular 2D and 3D communication graphs effectively.

A recursive bipartitioning algorithm was proposed in [19] to map processes onto a hypercube topology. Each process's node assignment is gradually determined through partitioning the communication graph at each level. This technique was extended to handle general topologies in [29], where both the communication graph and the topology graph are bipartitioned recursively to derive the mapping of processes onto nodes.

In [11], the topology mapping problem was reduced to the graph isomorphism problem, and a heuristic algorithm based on sequences of pairwise interchanges and probabilistic jumps was proposed to solve practical mapping problems on a specific array processor. Another greedy heuristic for topology mapping was proposed in [3]. It uses estimation functions to evaluate the effects of mapping decisions. Several generic mapping techniques were studied in [21], including a graph similarity approach using the reverse Cuthill–McKee (RCM) ordering, a greedy heuristic, and a recursive bipartitioning algorithm. These have been implemented in the topology mapping library LibTopoMap [27], which was successfully tested on supercomputers with torus, fat-tree, and PERCS networks.

The TreeMatch algorithm [24] was proposed for topology mapping on multicore clusters, where the hardware topology of the computing platform (including the cache hierarchy) is modeled by a tree. For systems with a hierarchical communication architecture, e.g., clusters of SMP nodes, graph partitioning was exploited for finding an appropriate mapping in [33]. A hierarchical approach, which performs both inter-node mapping and intra-node mapping, was introduced in [40], and further improved and generalized in [41].

A topology mapping tool Rubik was developed for generating mappings of structured communication patterns onto torus topologies in [10], and it enabled the evaluation of different mappings, but it cannot optimize the mapping. Geometric partitioning was utilized for topology mapping on supercomputers with torus network topology in [18]. Both the tasks and compute nodes are partitioned by a geometric partitioning algorithm, so it can only be used for structured communication patterns.

In summary, the mapping of regular communication patterns onto regular topologies can be solved by graph embedding techniques and geometric approaches; the mapping of irregular communication patterns can be tackled by generic mapping technique. For mapping irregular communication patterns onto regular mesh/torus topologies [35], generic mapping strategies (e.g., [21,27]) are popular choices. However, they do not exploit the regular structure of mesh/torus, and the resulting mapping quality may not be satisfactory as shown by the results in Sect. 4. In this work, we present a new mapping algorithm which exploits the special feature of the mesh/torus topology for the mapping of parallel applications with irregular communication patterns.

3 Analytical mapping

This section presents the proposed analytical topology mapping algorithm for mapping irregular communication patterns onto 3D mesh/torus network topologies.

3.1 Preliminary

The topology mapping problem in the HPC domain is very similar to the VLSI placement problem in the electronic design automation (EDA) domain as illustrated in Table 1. The latter places VLSI circuits consisting of up to millions of logic gates onto a rectangular layout region, so as to achieve high performance with low cost. Total wire length (similar to hop-bytes) is the typical evaluation metric, and recent studies also consider timing-driven placement and routability-driven placement.

The research on both problems has resulted in similar solutions. In particular, the analytical placement technique [36,37] based on quadratic programming has been

Table 1 The relations between topology mapping and VLSI placement [36,37]

	Topology mapping	VLSI placement
Guest object	Parallel application processes modeled by a communication graph	Rectangular circuit blocks connected by wires
Host object	Computing system with specific topology modeled by a topology graph, e.g., 3D torus	Rectangular placement region
Optimization objective	The total amount of traffic in the network, i.e., hop-bytes	The total wirelength, i.e., half-perimeter wirelength
Other metrics	Average hops, dilation, congestion	Average wirelength, timing, routability

developed. It outperforms previous approaches in both runtime and solution quality, thus being widely used in commercial placement tools.

Inspired by the analytical placement technique [36,37] for VLSI design, we propose an analytical mapping algorithm for solving the topology mapping problem. Different from the VLSI placement problem, which maps circuit blocks onto a 2D placement region, we consider mapping processes onto compute nodes in 3D mesh/torus topologies. It is to be noted that the compute nodes are located at discrete positions, and each node is typically assigned equal number of processes. For ease of presentation, we consider mapping one application process per compute node in the following subsections. In order to map multiple processes onto a multicore compute node, the proposed analytical topology mapping algorithm can be applied after processes are properly partitioned into groups.

3.2 Algorithm overview

Since the solution space of the possible mappings is discrete and has a tremendous size when there are large numbers of processes and nodes, it is very challenging to obtain a proper ϕ to minimize hop-bytes as defined in Eq. (1). We propose to overcome such difficulty by a two-stage analytical topology mapping algorithm.

In the first stage named global mapping, we formulate and solve a continuous optimization problem that approximates the topology mapping problem. The decision variables are denoted by a labeling $\Phi : V_c \rightarrow \mathcal{R}^3$ that maps processes to points in the space $\mathcal{R}^3$. Clearly, the solution space of Φ is continuous. For each compute node in the 3D mesh/torus, we map it to an integral point in the same space representing its position in the network. Then, if we assume that a process will be mapped to a node if the distance between them in $\mathcal{R}^3$ is small, we can approximate the hops between a pair of processes i and j by a function of the distance between them. While different choices of the function and the distance may lead to different objectives to be minimized, we choose to use the square of the $\mathcal{L}_2$ norm distance, i.e., $(\Phi_i - \Phi_j)^T(\Phi_i - \Phi_j)$, such that the objective function of the continuous optimization problem is quadratic, and it can be solved by efficient quadratic programming techniques [12]. Moreover, using the square of the $\mathcal{L}_2$ norm distance as the objective function can help to better distribute the communication traffic among all the dimensions, thus reducing congestion. For example, we prefer to place a pair of communicating processes at nodal coordinates $(0, 0, 0)$ and $(1, 1, 1)$, rather than $(0, 0, 0)$ and $(3, 0, 0)$, because the former has smaller $\mathcal{L}_2$ norm distance than the latter, despite their identical hop distances.

In the second stage named legalization, ϕ is obtained from Φ by "moving" processes to nodes. Intuitively, the movements should only happen between nearby processes and nodes in order to minimize the perturbation to the outcome of global mapping—otherwise, hop-bytes as approximated by its objective function may be degraded. However, such movements are not always possible when multiple processes are close to a single node. To avoid such situation, we impose a constraint on the continuous optimization problem in the global mapping stage requiring Φ to be almost-legal. In other words, we require the processes to be distributed evenly with respect to the nodes

Fig. 2 The analytical topology mapping algorithm

```
Analytical Topology Mapping Algorithm
Stage 1: Global Mapping
1  Repeat
2      Perform global optimization to obtain $\Phi$
3      Terminate if $\Phi$ is almost-legal
4      Compute process shifting
5      Compute spreading forces
Stage 2: Legalization
6  Perform diffusion-like migration to obtain $\phi$
```

in $\mathcal{R}^3$. Then, in the legalization stage, a novel migration algorithm designed by us will be able to derive ϕ from the almost-legal Φ without degrading hop-bytes considerably.

The outline of our analytical topology mapping algorithm is shown in Fig. 2. The continuous optimization problem formulated in the global mapping stage is solved as a sequence of unconstrained quadratic optimization problems that gradually make Φ almost-legal by adding penalty terms to the objective function that approximates hop-bytes. In each iteration, we perform global optimization to solve for Φ, and then terminate the stage if it is almost-legal. Otherwise, we compute process shifting as the gradient of the uneven distribution of Φ and then compute spreading forces as the penalty terms to the objective function that lead to more even distribution in the next iteration. A diffusion-like migration algorithm is employed in the legalization stage to minimize the movements of processes. More details of each step are discussed in the following subsections.

3.3 Global mapping

As mentioned above, in the global mapping stage, we solve a continuous optimization problem with an objective function that approximates *hop-bytes* as follows.[1]

$$W(\Phi) \triangleq \sum_{\forall (i,j)\in E_c} c_{i,j}(\Phi_i - \Phi_j)^T(\Phi_i - \Phi_j). \tag{2}$$

We propose to heuristically determine the proper mapping of some processes to the eight corner nodes of 3D torus topology (detailed in Sect. 3.6). These processes will be fixed during the global mapping stage, and their coordinates will not be considered as decision variables.

To define the constraint that requires Φ to be almost-legal, we divide the portion of $\mathcal{R}^3$ that contains the nodes in the 3D mesh/torus into cubic bins, each holding a node at its center. For an $X \times Y \times Z$ topology, we first map the nodes to the integral points (i, j, k) for $i = 0, 1, \ldots, X-1$, $j = 0, 1, \ldots, Y-1$, and $k = 0, 1, \ldots, Z-1$. Then, the cubic bin for the node (i, j, k) is chosen as

[1] The physical meaning of Eq. (2) is introduced below. The communication graph of the application is modeled as a spring system, where each edge $(i, j) \in E_c$ is represented as a spring with corresponding spring constant being $c(i, j)$. The total energy of the springs is a quadratic function of their lengths. A mapping solution is obtained by minimizing the total energy to find a force equilibrium state.

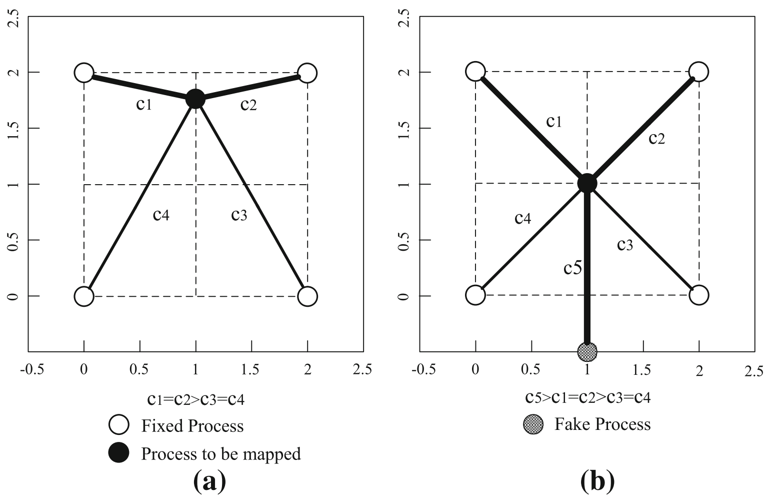

Fig. 3 An example of analytical mapping onto a 3 × 3 mesh/torus topology. $c1$ to $c5$ are the weights associated with the communications with fixed or "fake" processes that move the process to be mapped

$$\left\{(x,y,z)|x \in \left[i-\frac{1}{2}, i+\frac{1}{2}\right], y \in \left[j-\frac{1}{2}, j+\frac{1}{2}\right], z \in \left[k-\frac{1}{2}, k+\frac{1}{2}\right]\right\},$$

and the boundary for Φ is chosen as

$$\left\{(x,y,z)|x \in \left[-\frac{1}{2}, X-\frac{1}{2}\right], y \in \left[-\frac{1}{2}, Y-\frac{1}{2}\right], z \in \left[-\frac{1}{2}, Z-\frac{1}{2}\right]\right\}.$$

We define Φ to be almost-legal iff all processes are within the aforementioned boundary and each bin contains at most γ processes.

However, it remains difficult to minimize $W(\Phi)$ while requiring Φ to be almost-legal. With the observation that $W(\Phi)$ is a quadratic function of the decision variables, it is proposed in [36,37] that such continuous optimization problem can be approximated using unconstrained quadratic programming by employing penalty terms in addition to the original objective function. These penalty terms can be modeled as a set of "fake" processes V_f on the boundary that utilize "fake" communications $E_f \subseteq V_c \times V_f$ with associated weights to ensure Φ to be almost-legal. An example demonstrating how such "fake" processes may be utilized to move another process in order to modify Φ is shown in Fig. 3.

Nevertheless, since these penalty terms may interfere with the minimization of W, they should be introduced gradually. Starting with $V_f = \emptyset$, we will solve a sequence of unconstrained quadratic programming problems where in each iteration V_f and E_f will be updated based on the current solution so that an almost-legal Φ can eventually be obtained. Details follow.

3.3.1 Global optimization

Global optimization is the step in one global mapping iteration that utilizes unconstrained quadratic programming techniques to compute Φ with respect to the original objective function and the penalty terms. It is similar to the global optimization step of the analytical placement technique [36,37].

For ease of presentation, we extend Φ to $V_c \cup V_f$ such that for any $i \in V_c \cup V_f$, $\Phi_i = (x_i, y_i, z_i)^T$ is the coordinate of process or "fake" process i. Then, global optimization (i.e., Eq. 2) can be transformed into the following problem:

$$\begin{aligned} \text{Minimize } W_f(\Phi) &\triangleq \sum_{\forall (i,j) \in E_c \cup E_f} c_{i,j} (\Phi_i - \Phi_j)^T (\Phi_i - \Phi_j) \\ &= \sum_{\forall (i,j) \in E_c \cup E_f} c_{i,j} \left[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \right]. \end{aligned} \tag{3}$$

Let n be the number of processes in V_c whose coordinates are not fixed and denote them by $1, 2, \ldots, n$ without loss of generality. We may reorganize the decision variables as three vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$ and $\mathbf{z} = (z_1, z_2, \ldots, z_n)^T$. Then, Eq. (3) can be written in matrix form as

$$\begin{aligned} W_f(\Phi) = &\left(\mathbf{x}^T Q \mathbf{x} + 2\mathbf{d}_x^T \mathbf{x}\right) + \left(\mathbf{y}^T Q \mathbf{y} + 2\mathbf{d}_y^T \mathbf{y}\right) \\ &+ \left(\mathbf{z}^T Q \mathbf{z} + 2\mathbf{d}_z^T \mathbf{z}\right) + \text{constant}, \end{aligned} \tag{4}$$

where Q is an $n \times n$ symmetric positive definite matrix; $\mathbf{d}_x$, $\mathbf{d}_y$, and $\mathbf{d}_z$ are $n \times 1$ vectors. Specifically, the diagonal elements of Q represent the total amount of communication for the corresponding processes, i.e., $q_{ii} = \sum c_{i,j}, \forall (i, j) \in E_c \cup E_f$; each off-diagonal element of Q represents the communication between the respective process pair, i.e., $q_{i,j} = -c_{i,j}, \forall i \neq j$; $\mathbf{d}_x$, $\mathbf{d}_y$, and $\mathbf{d}_z$ are introduced by the communication with fixed/fake process, e.g., $\mathbf{d}_{x,i} = -\sum c_{i,f} x_f$, $\forall$ fixed/fake process f.

Since the objective function is separable into three independent parts as shown in Eq. (4), the optimal solution can be obtained by solving the following three systems of linear equations, each represents the sub-gradient of the objective function at one dimension.

$$Q\mathbf{x} + \mathbf{d}_x = 0,\ Q\mathbf{y} + \mathbf{d}_y = 0,\ Q\mathbf{z} + \mathbf{d}_z = 0. \tag{5}$$

As Q is symmetric positive definite, Eq. (5) can be solved very efficiently by direct solvers like Cholesky factorization [20], or iterative solvers like incomplete Cholesky-conjugate gradient (ICCG) [20]. Furthermore, if all fixed processes are within the boundary and all "fake" processes are on the boundary, the solution is guaranteed to be within the boundary. Therefore, the only remaining concern for Φ to be almost-legal is that some bins may contain more than γ processes, which will be addressed in the following iterations by updating "fake" processes and communications via process shifting and spreading forces in order to move processes out of heavily occupied bins.

3.3.2 Process shifting

Process shifting is the step in one global mapping iteration that decides where the processes in a heavily occupied bins should be moved to so that a more even process distribution of Φ can be obtained in the next iteration. However, since such movements will definitely lead to an increase to $W(\Phi)$, it is desirable that certain properties of the current global mapping solution Φ are preserved. Experiences in [36,37] suggest that the relative ordering of processes in each dimension should be kept in order to preserve the "optimality" of the current solution.

We first calculate the "workload" of each cubic bin as the number of the processes inside it. If all workloads are at most γ, we terminate global mapping. Otherwise, we decide the new coordinate of the processes for each dimension separately. Without loss of generality, considering the x-dimension, we group the cubic bins into $Y \times Z$ sets such that the bins in the same set are centered at the nodes with the same y- and z-coordinate. For each bin set, we form a regular one-dimensional structure on the interval $\left[-\frac{1}{2}, X - \frac{1}{2}\right]$ with X segments, each having a length of 1 and representing a bin in the set. Obviously, after we map the processes inside the bin set to the interval by their x-coordinates, the number of the processes on each segment equals its workload. Then, we resize the segments into unequal lengths according to their workloads such that the segments with high workloads expand, and those with low workloads shrink. The new x-coordinates of the processes are computed as if they are shifted proportionally as the segments expand or shrink, which clearly keeps the relative ordering of the processes. For details of the heuristic, the reader is referred to Sect. V of [37].

3.3.3 Spreading forces

While process shifting decides where the processes should be moved to, the penalty terms that actually move the processes to achieve a more even distribution in the next iteration are computed as spreading forces.

For each process i, we introduce a "fake" process f at (x_f, y_f, z_f) on the boundary and a communication link between i and f with the weight β to model the spreading force. Let the new position of a process i after shifting be (x_i', y_i', z_i'). If we assume all other processes are not shifted, then both $W(\Phi)$ and $W_f(\Phi)$ can be treated as functions of the position of i, denoted by $W^i(x_i, y_i, z_i)$ and $W_f^i(x_i, y_i, z_i)$. It is desirable that (x_i', y_i', z_i') minimizes $W_f^i(x_i, y_i, z_i)$, and the gradient of W_f^i at (x_i', y_i', z_i') should be 0, i.e., (x_f, y_f, z_f) and β should satisfy

$$\begin{aligned}
0 &= \left.\frac{\partial W_f^i}{\partial x_i}\right|_{x_i=x_i'} = \left.\frac{\partial W^i}{\partial x_i}\right|_{x_i=x_i'} + \beta(x_i' - x_f),\\
0 &= \left.\frac{\partial W_f^i}{\partial y_i}\right|_{y_i=y_i'} = \left.\frac{\partial W^i}{\partial y_i}\right|_{y_i=y_i'} + \beta(y_i' - y_f),\\
0 &= \left.\frac{\partial W_f^i}{\partial z_i}\right|_{z_i=z_i'} = \left.\frac{\partial W^i}{\partial z_i}\right|_{z_i=z_i'} + \beta(z_i' - z_f). \qquad (6)
\end{aligned}$$

If we define the resultant force of i to be the gradient of W^i at (x_i', y_i', z_i'), i.e.,

$$(F_x, F_y, F_z) \triangleq \left(\frac{\partial W^i}{\partial x_i}\bigg|_{x_i=x_i'}, \frac{\partial W^i}{\partial y_i}\bigg|_{y_i=y_i'}, \frac{\partial W^i}{\partial z_i}\bigg|_{z_i=z_i'} \right), \tag{7}$$

then Eqs. (6) can be rearranged as

$$(F_x, F_y, F_z)^T + \beta\big((x_i' - x_f), (y_i' - y_f), (z_i' - z_f)\big)^T = 0, \tag{8}$$

which indicates that the spreading force on i due to the penalty terms should have the same magnitude as the resultant force but opposite directions.

To actually solve Eq. (8), we first compute the resultant force from Eq. (7) and decide the direction of the spreading force. Then, the position (x_f, y_f, z_f) of the "fake" process f is obtained at the intersection of the spreading force direction and the boundary of the 3D domain. Finally, β is computed as

$$\beta = \frac{\sqrt{F_x^2 + F_y^2 + F_z^2}}{\sqrt{(x_i' - x_f)^2 + (y_i' - y_f)^2 + (z_i' - z_f)^2}}. \tag{9}$$

It is worth noting that the spreading forces do not accumulate over iterations. In each iteration, new forces are added, while the previous forces are discarded. Therefore, according to Sect. 3.3.1, the diagonal of matrix Q and the vectors $\mathbf{d}_x, \mathbf{d}_y, \mathbf{d}_z$ need to be updated to reflect the changes of the penalty terms for the next global optimization iteration.

3.4 Legalization

The legalization stage derives a feasible mapping based on the global mapping solution. Processes need to be migrated from the overloaded compute nodes to the empty nodes. However, a direct movement will break the relative ordering between nodes, leading to degraded mapping quality. In order to maintain the relative distribution of processes in the global mapping solution, we employ a diffusion-like migration algorithm proposed in [22] to minimize the process migration between nodes.

Let w_i be the workload of compute node i, i.e., the number of processes on node i. Let $\mathbf{b}$ be an n-dimensional vector, and its ith element $b_i \triangleq w_i - 1$ denotes the amount of processes that need to be moved out of node i (since each node should be assigned one process). If $b_i = -1$, then one process should be migrated to node i instead. Let L be the $n \times n$ Laplacian matrix of the topology graph defined as

$$L(i, j) \triangleq \begin{cases} \text{degree(node } i), & \text{if } i = j; \\ -1, & \text{if } i \neq j \text{ and node } i \text{ is adjacent to node } j; \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Fig. 4 The process migration algorithm

```
Process Migration Algorithm
1  Repeat
2      Solve $L\lambda = \mathbf{b}$ to obtain $\lambda$
3      Sort compute nodes such that their corresponding $\lambda_i$ is in non-increasing order
4      Foreach compute node $i$, greedily migrate processes to its neighboring node(s) $j$ if $b_i > 0$ and $\lambda_i > \lambda_j$, and update $b_i$, $b_j$ accordingly
5  Until each compute node is assigned one process
```

Then the optimal migration solution can be obtained by solving the linear equations

$$L\lambda = \mathbf{b}, \tag{11}$$

where λ is an n-dimensional vector of unknown variables. The amount of workload that needs to be moved from node i to its neighboring node j is given by $\lambda_i - \lambda_j$. However, the computed variables λ are real numbers, and the resultant migration is not feasible for moving processes between nodes.

To overcome this issue, we design an iterative migration algorithm as shown in Fig. 4 to perform process migration. Each iteration essentially contains three steps. We first compute the variables λ, then sort the compute nodes such that their corresponding variable λ_i is in non-increasing order, and finally migrate processes for each node by using a greedy heuristic. Specifically, for each compute node i, if $b_i > 0$ and there exists some neighboring node(s) j such that $\lambda_i > \lambda_j$, then we migrate $\min(b_i, \lfloor\lambda_i - \lambda_j\rfloor)$ process(es) to the neighboring node(s) j, and update b_i, b_j accordingly to reflect the change of workload. If b_i is still positive, we sort the neighboring nodes j such that their corresponding values $(\lambda_i - \lambda_j) - \lfloor\lambda_i - \lambda_j\rfloor$ is in non-increasing order (and of course, $\lambda_i > \lambda_j$), and migrate one process to each neighboring node until $b_i = 0$ or all the candidate neighbors have been processed. Moreover, in order to minimize hop-bytes, we always choose to migrate the process, which results in the minimum increase of hop-bytes. In each iteration of the process migration algorithm, at least one process will be moved out of the compute node with the maximum workload. So this process migration algorithm is guaranteed to terminate, and the number of iterations is bounded by the maximum workload.

3.5 Complexity analysis

Suppose the cost of solving Eq. (5) is $f_1(n)$, and the cost of solving Eq. (11) is $f_2(n)$. As process shifting and adding spreading forces all take $O(n)$ time, the running time of each iteration of global mapping is $f_1(n) + O(n)$. During each iteration of process migration, sorting the compute nodes takes $O(n \log n)$ time, and moving processes take $O(n)$ time. So the running time of each iteration of process migration is $f_2(n) + O(n \log n) + O(n)$.

Let k_1 and k_2 be the number of iterations for global mapping and legalization, respectively. Then the total mapping overhead of the proposed analytical mapping algorithm is

$$k_1 \times \big(f_1(n) + O(n)\big) + k_2 \times \big(f_2(n) + O(n \log n) + O(n)\big).$$

The overall time complexity is given by

$$O(f_1(n)) + O(f_2(n)) + O(n \log n).$$

As matrix Q in Eq. (5) is symmetric positive definite, and matrix L in Eq. (11) is symmetric positive semi-definite, both Eqs. (5) and (11) can be solved by efficient linear system solver in time $O(M^{1.31})$ [32], where M is the number of non-zeros in the left-hand-side matrix. For matrix Q, $M = \Theta(|E_c|)$, so we have $O(f_1(n)) = O(|E_c|^{1.31})$. For matrix L corresponding to 3D torus topology, $M = 7n$, and we have $O(f_2(n)) = O(n^{1.31})$. Hence, the time complexity of the proposed analytical mapping algorithm can also be represented as

$$O(|E_c|^{1.31}) + O(n^{1.31}) + O(n \log n).$$

3.6 Implementation

We have implemented the proposed analytical mapping algorithm as a topology mapping tool [4], which is publicly available for community use. In order to support the mapping of multiple processes onto each multicore node, we employ the graph partitioning tool METIS [28] to partition the processes into fixed groups before applying the proposed algorithm. As METIS does not guarantee equal partition of a graph, a greedy heuristic is used to balance the partitions if necessary, and it always moves the process which minimizes edgecuts. During the global mapping stage, we keep track of the maximum “workload” of compute nodes, and terminate global mapping when each compute node is assigned at most four processes (or four groups of processes for multicore mapping), i.e., $\gamma = 4$. The global mapping will also be terminated if it does not reduce the maximum “workload” of compute nodes in ten consecutive iterations. All the linear systems are solved by the Cholesky factorization package CHOLMOD [14].

Before applying the proposed analytical mapping algorithm, we need to determine the fixed processes, which are mapped onto the eight corner nodes of the 3D torus topology. Two heuristics are employed in order to support generic communication patterns. The first one is breadth first traversal, which was explored for topology mapping of general communication patterns in [9]. We start from a random node/process (typically node/process 0), traverse the topology/communication graph by using breadth first search, and arrange the nodes/processes into a list. The processes which correspond to the eight corner nodes are identified as fixed processes. The second heuristic is RCM ordering [21]. The nodes and processes are ordered by using the RCM algorithm, and the fixed processes are identified accordingly. In fact, the RCM algorithm is a variant of breadth first traversal, it orders the nodes traversed at each level according to their degrees.

4 Performance evaluation

In this section, we evaluate the proposed analytical mapping algorithm with several scientific applications on a production supercomputer.

4.1 Experimental setup

Experiments are carried out on the production IBM Blue Gene/P (BG/P) machine [23] named Intrepid at Argonne National Laboratory. Intrepid comprises 40 racks, arranged in five rows and interconnected by a highly scalable 3D torus network. Each compute node has one 850 MHz quad-core processor and 2GB memory. The entire system contains 164 K cores, offering a peak performance of 557 tera-flops. Each job is allocated a dedicated partition of the machine, so there is no intervention between jobs. Table 2 lists the network topology of the BG/P supercomputer for different partition sizes. It is to be noted that the 3D torus network reduces to a 3D mesh for small partitions with less than 512 compute nodes.

We use parallel sparse matrix multiplication with realistic input matrices from the University of Florida Sparse Matrix Collection [17] for performance tests. These sparse matrices are extracted from real-world scientific applications, and they represent the structures of realistic computing problems in different domains. Specifically, we select three sparse matrices for experiments, including F1, audikw_1 and nlpkkt120. Their properties are listed in Table 3, where "NNZ" denotes the number of non-zeros. The first two matrices are symmetric stiffness matrices, which model the elasticities of automotive crankshafts. The third one is a symmetric indefinite KKT matrix, which represents a nonlinear programming problem for a 3D PDE-constrained optimization. In order to perform matrix computation efficiently, scientific codes often use a graph partitioner to decompose the matrix into sub-matrices, and then use multi-

Table 2 Topology of the Blue Gene/P supercomputer

# Nodes	# Cores	Topology	Mesh/torus
64	256	$4 \times 4 \times 4$	Mesh
128	512	$4 \times 4 \times 8$	Mesh
256	1024	$8 \times 4 \times 8$	Mesh
512	2048	$8 \times 8 \times 8$	Torus
1024	4096	$8 \times 8 \times 16$	Torus
2048	8192	$8 \times 8 \times 32$	Torus

Table 3 Properties of sparse matrices

Name	Rows and columns	NNZ	NNZ per row
F1	343,791	13,590,452	39.53
audikw_1	943,695	39,297,771	41.64
nlpkkt120	3,542,400	50,194,096	14.17

ple processes/threads to compute in parallel. The communication between processes depends on the decomposition and the structure of the matrix. In our experiments, we use METIS [28] to partition the graphs represented by these sparse matrices, and analyze the edgecuts between partitions, which indicates the inter-process communication pattern. The communication test consists of 1000 iterations of inter-process communication. Specifically, the inter-process communication is implemented by posting non-blocking receives MPI_Irecv() and non-blocking sends MPI_Isend(), followed by a single MPI_Waitall() for all sends and receives.

Moreover, we also use a cosmology simulation code called Adaptive Refinement Tree (ART) [25,39,40,43] to test our mapping algorithm. ART involves the simulation of a cubic computational domain which represents the universe, and adopts adaptive mesh refinement (AMR) [30] to effectively model the universe with different densities. A domain decomposition scheme based on Hilbert space-filling curve (SFC) [13] is employed to determine the computational domain of each process. The resultant communication graph is highly sparse due to the spatial locality preserved by the SFC. As ART simulations consume a large amount of computing resources, the communication part of a production ART simulation is extracted for efficient performance tests. We choose this application for experiments, because its communication pattern is representative of many scientific applications based on finite element methods.

Figure 5 shows the communication pattern of the chosen test cases with 1024 processes in matrix form. Each blue dot at (i, j) represents the communication between processes i and j, and the total number of blue dots is given by "nz". As shown by the distribution of blue dots, the sparse matrix tests have different communication pattern, but all being sparse and highly irregular. The communication of ART is even more sparse and irregular, and most blue dots are near the diagonal. This is due to the fact that the ART communication is mainly nearest neighbor exchanges in the adaptively refined 3D computational domain. For all the test cases, the irregularity of communication is also shown in its message sizes, which may vary significantly between different process pairs.

For performance comparison, we also experiment with the topology mapping library—LibTopoMap [21,27], which supports three generic mapping strategies, including a graph similarity approach using the RCM ordering, a greedy heuristic, and a recursive bipartitioning algorithm.

Five different mapping mechanisms are evaluated. They are: (1) the system default "TXYZ" mapping on BG/P which is used as the baseline for performance comparison; (2) the proposed analytical mapping algorithm; (3) the graph similarity approach of LibTopoMap; (4) the greedy heuristic of LibTopoMap; (5) the recursive bipartitioning algorithm of LibTopoMap.

The programs are compiled by IBM XL compiler on Blue Gene/P, and all the experiments are run in virtual node mode, i.e., one process on each core. As the Cholesky factorization package CHOLMOD [14] used by our analytical mapper is not available on BG/P, we run the analytical mapper on a Linux server with Intel Xeon X5650 processor. The derived mapping is then employed for experiments on BG/P. In order for a fair comparison of different mapping mechanisms, we run the tests with different mappings in a single batch script.

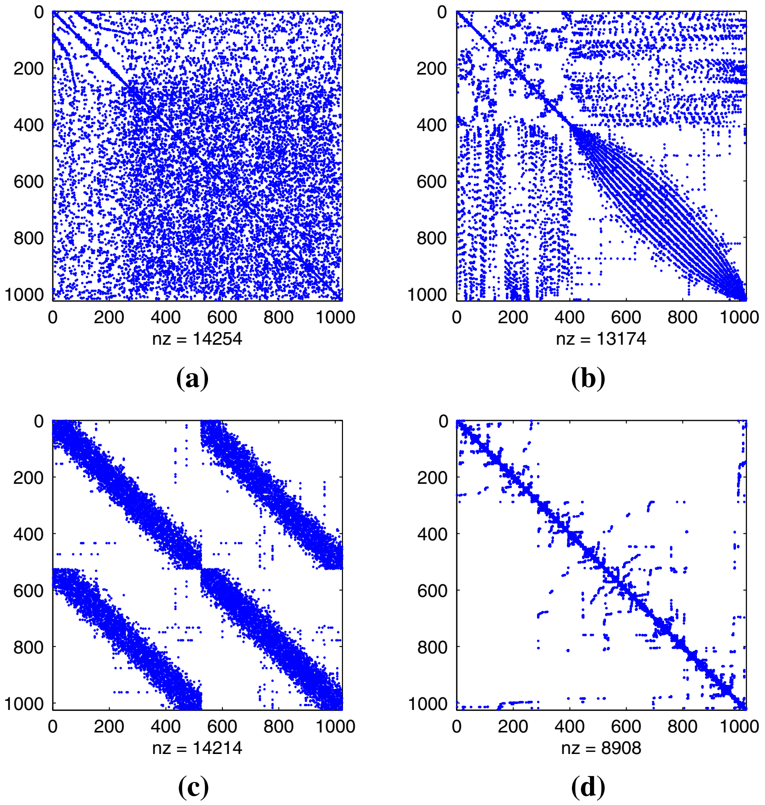

Fig. 5 The communication pattern of sparse matrix tests and the cosmology application named ART (1024 processes)

4.2 Fixed processes for global mapping

Before running the performance tests, we first run a set of experiments to evaluate the two heuristics (i.e., breadth first traversal and RCM ordering) for determining the fixed processes for global mapping (see Sects. 3.3, 3.6). A proper set of fixed processes can lead to a good mapping and vice versa. As the proposed analytical mapping algorithm aims to minimize hop-bytes, we compare the hop-bytes of the resulting mapping with respect to the system default mapping. As shown in Table 4, breadth first traversal and RCM ordering lead to similar hop-bytes reduction in most cases. On average, RCM ordering often has slightly larger hop-bytes reduction. Hence, we use RCM ordering to determine the fixed processes in production runs.

4.3 Results

We use three metrics to evaluate different mapping algorithms:

Table 4 Hop-bytes reduction of the proposed analytical mapping algorithm (relative to the system default mapping) by using breadth first traversal (BFT) and RCM ordering to determine the fixed processes

# Proc.	F1		audikw_1		nlpkkt120		ART	
	BFT (%)	RCM (%)	BFT (%)	RCM (%)	BFT (%)	RCM (%)	BFT (%)	RCM (%)
256	70.66	71.11	67.80	65.94	53.76	53.45	42.96	42.63
512	75.29	74.59	71.26	70.95	63.38	64.76	39.18	43.04
1024	76.80	78.61	73.79	74.30	69.39	70.48	49.07	48.49
2048	72.60	73.69	67.71	65.27	66.67	69.39	28.17	26.37
4096	75.68	77.02	71.16	71.07	75.15	74.58	27.10	31.53
8192	77.35	82.72	68.14	70.95	77.73	79.49	22.84	25.76

- Hop-bytes: the total amount of inter-node communication (in bytes) weighted by the distance (in hops) between nodes.
- Congestion: the maximum amount of messages (in bytes) transmitted on a single link (assume that all links have identical bandwidth).
- Communication time: the actual communication time measure by MPI_Wtime().

Figures 6, 7 and 8 present the relative hop-bytes, congestion, and communication time reductions of different mapping mechanisms compared to the system default mapping, respectively. Table 5 shows the corresponding communication time statistics for the system default mapping and the proposed analytical mapping algorithm. “Analytical” represents our analytical mapping algorithm, while “RCM”, “Greedy” and “Recursive” represent the three mapping strategies of LibTopoMap. Note that LibTopoMap aims to minimize the congestion in the network. It estimates the congestion of a mapping by routing each message on a shortest path (discovered by a shortest path algorithm). LibTopoMap returns an improved mapping with reduced congestion if such a mapping is found, otherwise, it returns the original mapping, i.e., the system default mapping in our setting. For example, LibTopoMap’s recursive mapper returns the system default mapping for “audikw_1”, so there is no hop-bytes and communication time reduction. Besides, for all the test cases, LibTopoMap failed to derive optimized mappings when the number of processes is more than 2K.

For sparse matrix tests, our analytical mapper significantly reduces hop-bytes and congestion by up to 83 and 81%, respectively, and achieves communication time reduction by up to 72%. LibTopoMap increases hop-bytes despite significant congestion reduction, and the resulting communication time is often increased. For ART, the analytical mapper improves hop-bytes and congestion by up to 48 and 66%, respectively, and reduces communication time by up to 59%. LibTopoMap significantly reduces the congestion of ART, but it largely increases the hop-bytes, leading to additional communication cost.

The analytical mapper is capable of reducing both hop-bytes and congestion for 3D torus, since the heavily communicating processes are properly mapped onto neighboring nodes during the global mapping stage, and the communication traffics are distributed among all the dimensions to reduce congestion. This is achieved by using the square of the $\mathcal{L}_2$ norm distance in the objective for global optimization (discussed

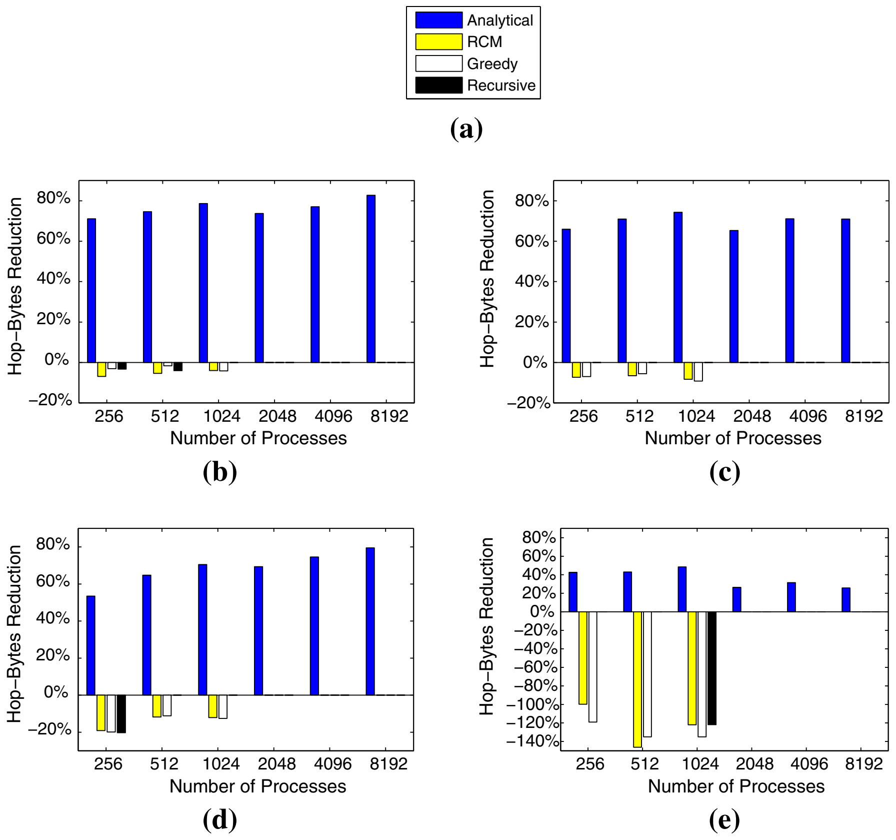

Fig. 6 Hop-bytes reductions of different topology mapping algorithms, relative to the system default mapping on BG/P. **a** Legend. **b** F1. **c** audikw_1. **d** nlpkkt120. **e** ART

in Sect. 3.2), and the global mapping solution is well preserved during the legalization stage. Generally, the hop-bytes reduction and the communication time reduction have a weak correlation for 3D torus. The hop-bytes represent the total amount of traffic in the network, while the communication time is dependent on the maximum message passing time among communicating process pairs. It is expected that communication performance can be improved by using a mapping with reduced hop-bytes, since the network is likely to be less congested.

Our study also shows that the maximum congestion on a single link may not be a proper measurement of the mapping quality, because it overlooks the overall traffic in the network. As a result, minimizing congestion alone may lead to increased hop-bytes and higher communication cost. After all, the congestion on a single link is just a lower bound of the communication time. From the optimization perspective, minimizing a lower bound of the objective does not necessarily reduce the objective. Another concern about congestion is that it can be very difficult to evaluate due to dynamical routing. Instead, hop-bytes is easy to compute, and it correlates better with the communication performance on 3D torus.

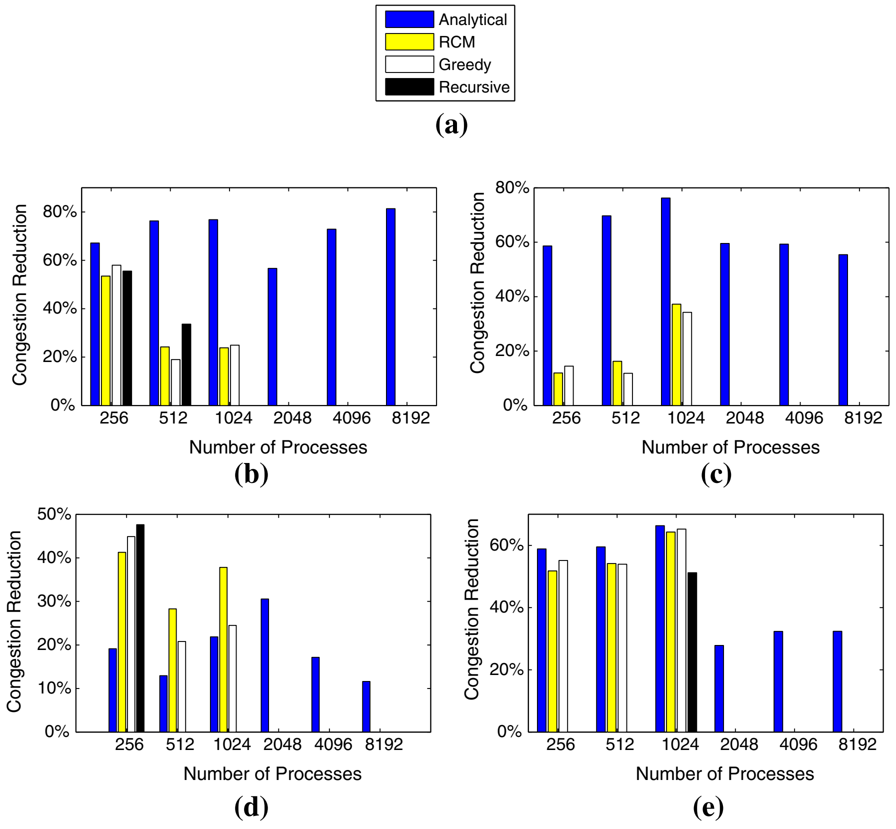

Fig. 7 Congestion reductions of different topology mapping algorithms, relative to the system default mapping on BG/P. **a** Legend. **b** F1. **c** audikw_1. **d** nlpkkt120. **e** ART

In practice, the effect of topology mapping on application execution time depends on the portion of communication to the total execution time. The amount of execution time saving typically comes directly from the amount of communication time reduction. For the ART application with our test input data, the total execution time by using 1024 processes is about 15 min, and the communication time for data transmission is about 20% of the total execution runtime. By using our analytical mapping algorithm, its communication time is reduced by 59%, which indicates about 12% improvement of the overall execution time.

4.4 Mapping overhead

Table 6 gives the overhead of the proposed analytical mapping algorithm. As this algorithm has two stages—global mapping and legalization, each of which involves solving linear systems iteratively until convergence, the runtime is dependent on the number of iterations required. The global mapping stage typically requires several dozens of iterations to finish as shown in Table 6, while the legalization stage always

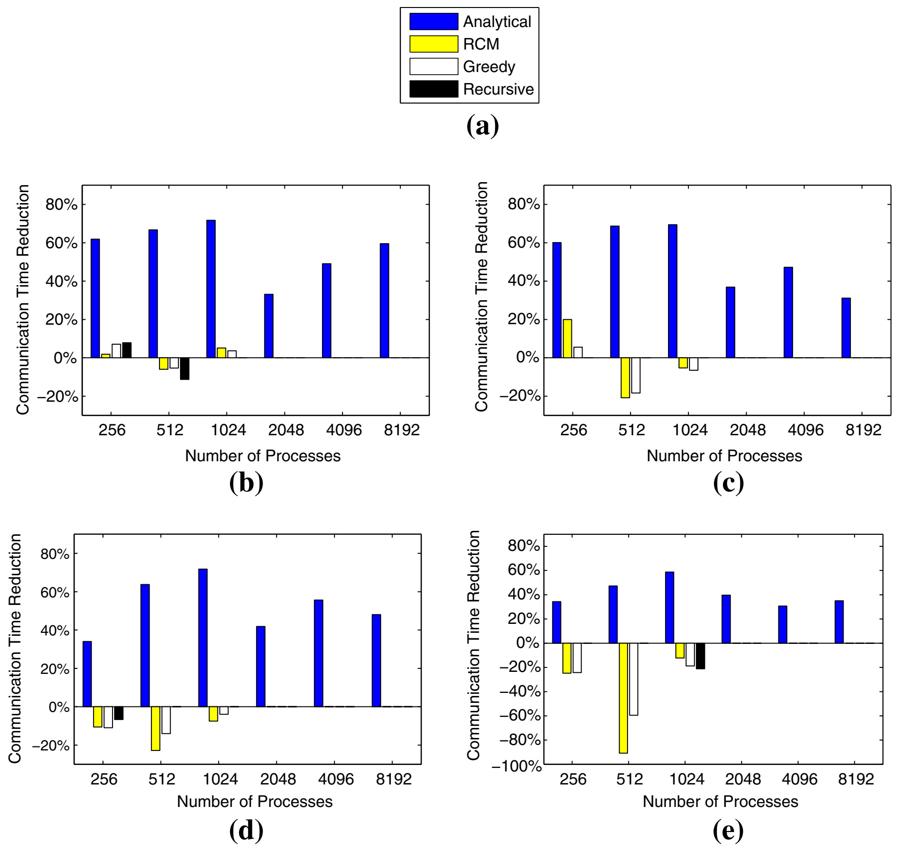

Fig. 8 Communication time reductions of different topology mapping algorithms, relative to the system default mapping on BG/P. **a** Legend. **b** F1. **c** audikw_1. **d** nlpkkt120. **e** ART

Table 5 Communication time statistics for the system default mapping and the proposed analytical mapping algorithm

Proc.	F1			audikw_1			nlpkkt120			ART		
	Def.	Ana.	Imp. (%)	Def.	Ana.	Imp. (%)	Def.	Ana.	Imp. (%)	Def.	Ana.	Imp. (%)
256	3.09	1.18	62	8.30	3.31	60	4.63	3.06	34	180.65	118.72	34
512	3.41	1.13	67	7.38	2.31	69	5.25	1.90	64	151.60	80.03	47
1024	2.86	0.81	72	5.99	1.83	69	4.18	1.18	72	174.54	71.99	59
2048	0.86	0.57	33	1.96	1.24	37	1.29	0.75	42	95.82	57.87	40
4096	1.30	0.66	49	2.18	1.15	47	2.10	0.93	56	93.81	65.01	31
8192	1.74	0.70	60	2.87	1.98	31	2.97	1.54	48	137.86	89.58	35

Proc. processes, *Def.* default mapping, *Ana.* analytical mapping, *Imp.* the improvement of communication time by using our analytical mapping over the default mapping; time (s)

Table 6 Overhead of our analytical mapping algorithm

# Proc.	F1		audikw_1		nlpkkt120		ART	
	Iter.	Time	Iter.	Time	Iter.	Time	Iter.	Time
256	14	0.010	22	0.007	33	0.005	27	0.011
512	32	0.015	34	0.010	32	0.013	33	0.014
1024	29	0.035	38	0.026	34	0.040	51	0.040
2048	33	0.093	41	0.075	41	0.115	44	0.088
4096	42	0.347	56	0.328	44	0.345	55	0.236
8192	49	1.086	49	0.783	56	1.265	49	0.579

Proc. processes, *Iter.* the number of iterations in the global mapping stage, *Time* the total runtime (s)

terminate within four iterations for all the test cases. The running time is mainly due to solving the linear systems (5) for global mapping, and it is often within 1 s. Consider that production runs of HPC applications typically take hours, the overhead of the proposed analytical mapping algorithm is negligible. As the number of processes increases, the mapping overhead increases superlinearly due to the superlinear complexity of solving sparse linear systems. To handle even more processes efficiently, the analytical mapping algorithm can be extended to perform hierarchical mapping [16].

5 Conclusion

In this paper, we have presented an analytical algorithm for mapping irregular parallel applications onto torus-connected supercomputers with 3D torus network topologies. The mapping is derived in two stages, namely global mapping and legalization. The global mapping stage obtains a rough mapping by minimizing the communication traffic through quadratic programming; the legalization stage derives a proper mapping by performing the minimum movements of processes between nodes. We have compared our design with several mapping algorithms by means of a number of representative scientific applications and benchmarks on production supercomputers. Experimental results with up to thousands of nodes show that the proposed analytical mapping method finds high-quality mappings for 3D torus, which significantly improves the communication performance. The experiments also demonstrate that the analytical mapping algorithm has small mapping overhead, thus being suitable for finding optimized mappings at runtime.

Future work is to study the optimization of congestion for topology mapping by considering the routing algorithm of the network [1]. Moreover, it is also of great interest to evaluate the power/energy savings [38] of the optimized mappings.

Acknowledgements This work is supported in part by US National Science Foundation Grants OCI-0904670 and CNS-1320125. This work is also supported in part by the National Natural Science Foundation of China Grant 61402083. The authors thank the Argonne Leadership Computing Facility for the use of their supercomputers.

References

1. Abdel-Gawad AH, Thottethodi M, Bhatele A (2014) RAHTM: routing algorithm aware hierarchical task mapping. In: Proc. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), p 325–335
2. Abts D (2011) The Cray XT4 and Seastar 3-D Torus interconnect. Encyclopedia of Parallel Computing, p 470–477
3. Agarwal T, Sharma A, Laxmikant A, Kale LV (2006) Topology-aware task mapping for reducing communication contention on large parallel machines. In: Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS)
4. Analytical Mapping Tool (2014) http://bluesky.cs.iit.edu/topomap/. Accessed 30 July 2014
5. Arabnia HR, Bhandarkar SM (1996) Parallel stereocorrelation on a reconfigurable multi-ring network. J Supercomput 10(3):243–269
6. Arabnia HR, Smith JW (1993) A reconfigurable interconnection network for imaging operations and its implementation using a multi-stage switching box. In: Proc. the 7th Annual International High Performance Computing Conference. The 1993 High Performance Computing: New Horizons Supercomputing Symposium, p 349–357
7. Berman F, Snyder L (1987) On mapping parallel algorithms into parallel architectures. J Parallel Distrib Comput 4(5):439–458
8. Bhandarkar SM, Arabnia HR (1995) The hough transform on a reconfigurable multi-ring network. J Parallel Distrib Comput 24(1):107–114
9. Bhatele A (2010) Automating topology aware mapping for supercomputers. Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana
10. Bhatele A, Gamblin T, Langer SH, Bremer PT, Draeger EW, Hamann B, Isaacs KE, Landge AG, Levine JA, Pascucci V, Schulz M, Still CH (2012) Mapping applications with collectives over sub-communicators on torus networks. In: Proc. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), p 97:1–97:11
11. Bokhari SH (1981) On the mapping problem. IEEE Trans Comput 30(3):207–214
12. Boyd S, Vandenberghe L (2009) Convex optimization. Cambridge University Press, Cambridge
13. Butz AR (1971) Alternative algorithm for Hilbert's space-filling curve. IEEE Trans Comput C–20(4):424–426
14. Chen Y, Davis TA, Hager WW, Rajamanickam S (2008) Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. ACM Trans Math Softw 35(3):22:1–22:14
15. Chockalingam T, Arunkumar S (1992) A randomized heuristics for the mapping problem: the genetic approach. Parallel Comput 18(10):1157–1165
16. Chung IH, Lee CR, Zhou J, Chung YC (2011) Hierarchical mapping for HPC applications. In: Proc. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), p 1815–1823
17. Davis TA, Hu Y (2011) The university of Florida sparse matrix collection. ACM Trans Math Softw 38(1):1–25
18. Deveci M, Rajamanickam S, Leung VJ, Pedretti K, Olivier SL, Bunde DP, Çatalyürek UV, Devine K (2014) Exploiting geometric partitioning in task mapping for parallel computers. In: Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS), p 27–36
19. Ercal F, Ramanujam J, Sadayappan P (1988) Task allocation onto a hypercube by recursive mincut bipartitioning. In: Proc. the Third Conference on Hypercube Concurrent Computers and Applications: Architecture, Software, Computer Systems, and General Issues, vol 1, C3P, p 210–221
20. Golub GH, Loan CFV (1996) Matrix computations, 3rd edn. The Johns Hopkins University Press, Baltimore, London
21. Hoefler T, Snir M (2011) Generic topology mapping strategies for large-scale parallel architectures. In: Proc. the International Conference on Supercomputing (ICS), p 75–84
22. Hu YF, Blake RJ, Emerson DR (1998) An optimal migration algorithm for dynamic load balancing. Concurr Pract Exp 10(6):467–483
23. IBM References for BG/P (2013) https://www.alcf.anl.gov/user-guides/bgp-references. Accessed 1 May 2013
24. Jeannot E, Mercier G, Tessier F (2014) Process placement in multicore clusters: algorithmic issues and practical techniques. IEEE Trans Parallel Distrib Syst 25(4):993–1002

25. Kravtsov AV, Klypin AA, Khokhlov AM (1997) Adaptive refinement tree: a new high-resolution N-body code for cosmological simulations. Astrophys J Suppl Ser 111:73–94
26. Lee C, Bic L (1989) On the mapping problem using simulated annealing. In: Proc. International Phoenix Conference on Computers and Communications, p 40–44. doi:10.1109/PCCC.1989.37357
27. LibTopoMap (2010) A generic topology mapping library. http://www.unixer.de/research/mpitopo/libtopomap/. Accessed 8 May 2013
28. METIS (2013) Graph partitioning tool. http://glaros.dtc.umn.edu/gkhome/views/metis. Accessed 6 May 2013
29. Pellegrini F (1994) Static mapping by dual recursive bipartitioning of process architecture graphs. In: Proc. the Scalable High-Performance Computing Conference, p 486–493
30. Plewa T, Linde T, Weirs VG (2005) Adaptive mesh refinement-theory and applications. Springer, Berlin
31. Salman A, Ahmad I, Al-Madani S (2002) Particle swarm optimization for task assignment problem. Microprocess Microsyst 26(8):363–371
32. Spielman D, Teng SH (2003) Solving sparse, symmetric, diagonally-dominant linear systems in time o(m1.31). In: Proc. IEEE Symposium on Foundations of Computer Science, p 416–427
33. Träff JL (2002) Implementing the MPI process topology mechanism. In: Proc. ACM/IEEE Conference on Supercomputing, p 28:1–28:14
34. Top 500 Supercomputer Sites (2015) http://www.top500.org/. Accessed 30 Nov 2015
35. The Gemini Network (2010) http://wiki.ci.uchicago.edu/pub/Beagle/SystemSpecs/Gemini_whitepaper.pdf. Accessed 1 May 2013
36. Viswanathan N, Chu CCN (2004) FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In: Proc. International Symposium on Physical Design, p 26–33
37. Viswanathan N, Chu CCN (2005) FastPlace: efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. IEEE Trans Comput Aided Design 24(5):722–733
38. Wallace S, Vishwanath V, Coghlan S, Tramm J, Lan Z, Papkay M (2013) Application power profiling on IBM Blue Gene/Q. In: Proc. IEEE International Conference on Cluster Computing (CLUSTER), p 1–8
39. Wu J, Gonzalez RE, Lan Z, Gnedin NY, Kravtsov AV, Rudd DH, Yu Y (2011) Performance emulation of cell-based AMR cosmology simulations. In: Proc. IEEE International Conference on Cluster Computing (CLUSTER), p 8–16
40. Wu J, Lan Z, Xiong X, Gnedin NY, Kravtsov AV (2012) Hierarchical task mapping of cell-based AMR cosmology simulations. In: Proc. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), SC ’12, p 75:1–75:10
41. Wu J, Xiong X, Lan Z (2015) Hierarchical task mapping for parallel applications on supercomputers. J Supercomput 71(5):1776–1802
42. Yu H, Chung IH, Moreira J (2006) Topology mapping for Blue Gene/L supercomputer. In: Proc. ACM/IEEE Conference on Supercomputing, p 52. doi:10.1109/SC.2006.63
43. Yu Y, Rudd DH, Lan Z, Gnedin NY, Kravtsov AV, Wu J (2012) Improving parallel IO performance of cell-based AMR cosmology applications. In: Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS), p 933–944