Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

OHTMA: an optimized heuristic topology-aware mapping algorithm on the Tianhe-3 exascale supercomputer prototype*

Yi-shui LI†, Xin-hai CHEN†, Jie LIU†‡, Bo YANG,
Chun-ye GONG, Xin-biao GAN, Sheng-guo LI, Han XU

Science and Technology on Parallel and Distributed Processing Laboratory,
National University of Defense Technology, Changsha 410073, China

†E-mail: liyishui_lys@163.com; chenxinhai1995@aliyun.com; liujie@nudt.edu.cn

Received Feb. 9, 2019; Revision accepted July 12, 2019; Crosschecked May 9, 2020

Abstract: With the rapid increase of the size of applications and the complexity of the supercomputer architecture, topology-aware process mapping becomes increasingly important. High communication cost has become a dominant constraint of the performance of applications running on the supercomputer. To avoid a bad mapping strategy which can lead to terrible communication performance, we propose an optimized heuristic topology-aware mapping algorithm (OHTMA). The algorithm attempts to minimize the hop-byte metric that we use to measure the mapping results. OHTMA incorporates a new greedy heuristic method and pair-exchange-based optimization. It reduces the number of long-distance communications and effectively enhances the locality of the communication. Experimental results on the Tianhe-3 exascale supercomputer prototype indicate that OHTMA can significantly reduce the communication costs.

Key words: High-performance computing; Topology mapping; Heuristic algorithm
https://doi.org/10.1631/FITEE.1900075 **CLC number:** TP319

1 Introduction

With the advent of the multicore architecture, the scale of high-performance computing (HPC) is growing up rapidly. In addition to the number of nodes in a system, the number of processing cores available within each node has increased dramatically (Tuncer et al., 2015). The number of processing cores in HPC systems has increased from 65 536 for Blue Gene in 2005 to 2 397 824 for Summit in 2018 (https://www.top500.org/). This increase has not only led to a leap in the development of computing power of HPC systems, but also created a more complex memory hierarchy and network infrastructure. The constraints of communication performance become evident when large-scale applications run on HPC systems because of the increased traffic, but the restricted development of network bandwidth cannot catch up with the current requirement of large-scale systems. Consequently, mapping of processes and physical topologies plays a significant role in boosting performance (Hoefler et al., 2014). Congestion is a dominant factor that can significantly affect communication performance, and a refined mapping of application tasks can effectively reduce this congestion. Beacuse the underlying topology is not typically taken into consideration in default mapping

‡ Corresponding author
* Project supported by the National Key Research and Development Program of China (No. 2017YFB0202104)
ORCID: Yi-shui LI, https://orcid.org/0000-0001-7826-7504; Jie LIU, https://orcid.org/0000-0003-3745-7541

strategies and many links are needed in each traversal as a result of the large network diameter, the interconnection network could be congested heavily (Bhatele and Laxmikant, 2009).

In this regard, topology-aware mapping has been proved to be a practical method that can help reduce communication costs within the interconnection network of large-scale systems and improve the communication performance. An optimized mapping of processors on the idle computing nodes can reduce network congestion significantly. For a large-scale application, a lot of execution time can be saved if most of the communication tasks are physically placed closer to each other, especially when the major communication takes place within a compute node or a compute frame (Chen et al., 2018).

The topology-aware mapping problem which is aimed to find an optimized mapping between tasks and topologies can be formalized as a quadratic assignment problem (QAP) (Sudheer and Srinivasan, 2012). It has been proved that solving this problem is non-deterministic polynomial-time hard (NP-hard) (Sahni and Gonzalez, 1976), but many researchers continue to propose various heuristics to find sub-optimal solutions, that is, to minimize the evaluation metric. Therefore, higher heuristic efficiency and precision are required to optimize the topology-aware mapping. The practical results suggest that reducing the evaluation metric can effectively lead to a decrease in communication time (Jeannot et al., 2014).

In this work, we study the topology-aware mapping problem on the Tianhe-3 exascale supercomputer prototype. The main contributions can be summarized as follows:

1. We propose an optimized heuristic topology-aware mapping algorithm (OHTMA). The main idea of this algorithm is to generate a greedy primary mapping strategy, use numerous pair-exchange operations to minimize the evaluation metric within finite iterations, and backtrack to the best result.

2. We optimize this new algorithm to minimize its runtime, which can improve its practicability.

3. We evaluate its performance on the Tianhe-3 exascale supercomputer prototype. Four NAS parallel benchmark (NPB, https://www.nas.nasa.gov/publications/npb.html) suites and two scientific applications are used to evaluate the effectiveness of our algorithm.

2 Basic definitions

In the literature, the application information and network topology are generally represented as the following two graphs:

1. Task (process) graph

Parallel processes are represented as a weighted directed graph $G_{\mathrm{p}} = (V_{\mathrm{p}}, E_{\mathrm{p}})$, where vertices V_{p} represent the processes and edges E_{p} represent the direct communication between processes. The weight w_{ab} of edge $e_{ab} = (v_a, v_b) \in E_{\mathrm{p}}$ denotes the amount of communications in bytes between vertices a and b, where $v_a, v_b \in V_{\mathrm{p}}$. The out-degree of vertex v_a denotes the number of processes to which process a sends information.

2. Topology (processor) graph

The network topology is represented as a directed graph $G_{\mathrm{n}} = (V_{\mathrm{n}}, E_{\mathrm{n}})$. Each vertex in V_{n} represents a processor and an edge in E_{n} represents a direct link in the topology (Agarwal et al., 2006).

In our work, we use the adjacent matrices of these two graphs: process matrix $\boldsymbol{A}$ and physical topology matrix $\boldsymbol{B}$.

2.1 Quadratic assignment problem model

The existing research has formalized topology-aware mapping as a QAP model (Brandfass et al., 2013), that is, find a one-to-one mapping f between processes and available processors to minimize the communication cost. For an application with n processes that need to be mapped to n processors, we can define a process matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and a physical topology matrix $\boldsymbol{B} \in \mathbb{R}^{n\times n}$. The mapping problem can be transformed into minimizing the total communication cost. Each mapping f, always presented as an array such as $\{0, 1, 2, 3n-1\}$, is a solution to QAP. So, the definitions of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and the strategy of finding the best mapping attract many researchers. Although the mapping problem is formalized into this simple model, this problem has been proven to be an NP-hard problem (Sahni and Gonzalez, 1976).

2.2 Hop-byte metric

The hop-byte metric is one of the widely used metrics for judging the communication performance produced by a mapping algorithm (Sudheer and Srinivasan, 2012). The hop-byte metric is the total size of inter-processor communication in bytes

weighted by the distance between the respective end processors. For processes i and j, the number of hops between them is represented as $b_{f(i)f(j)} \in \boldsymbol{B}$, and the size of the communication message is represented as $a_{ij} \in \boldsymbol{A}$. We define Hopbyte($f$) to represent the cost metric when the p processes are mapped on the topology under mapping f. The sum of these products over all messages indicates the hop-byte value of a given mapping used to assess it, expressed as

$$\text{Hopbyte}(f) = \sum_{i=0}^{p-1} \sum_{j=0}^{p-1} a_{ij} b_{f(i)f(j)}, \tag{1}$$

where $a_{ij} \in \boldsymbol{A}$ and $b_{f(i)f(j)} \in \boldsymbol{B}$.

Our work is to find an optimized mapping method to minimize Hopbyte(f) in a reasonable time.

3 Related works

Topology-aware mapping is a process to find an optimized allocation of processes to compute nodes. Many researchers have conducted much research to find an optimal approach for efficiently reducing the communication overhead. Hoefler and Snir (2011) presented three mapping algorithms: greedy heuristic, recursive bisection mapping, and graph-similarity-based mapping. The greedy heuristic algorithm chooses one of the processes with the heaviest out-degree and greedily maps its heaviest neighboring processes to the neighboring processors with the heaviest connections. This algorithm is the most generic approach. In the recursive bisection mapping algorithm, the minimal edge cut method maps the "heavy" clusters in the weighted process graph to the "strong" clusters in the weighted processor graph. The METIS library (Karypis and Kumar, 1998) is used to compute recursive bi-partitioning of the process topology graph and the physical topology graph into two equally sized halves with minimum edge-cut weights. However, this algorithm cannot obtain good performance on the supercomputer architecture. The third algorithm is based on graph similarity. The basic idea of this algorithm is that the graph adjacency matrix can be modeled as a sparse matrix that can apply the techniques from sparse linear algebra on topology mapping. Hoefler and Snir (2011) selected the reverse Cuthill McKee (RCM) algorithm, which is used to solve the bandwidth reduction problem by reordering the matrices. The RCM algorithm handles the proximity condition well and produces mappings with low dilation and congestion.

Generally, serval leading basic algorithms are used. The main approaches of topology mapping include the graph partitioning method and the heuristic algorithm, which is based on experiments and available information about the problem.

The heuristic algorithm is one of the most effective methods for generating an approximate exact solution in a short time when solving an NP-hard problem. Jeannot and Mercier (2010) proposed an algorithm called "TreeMatch," which maps processes to resources to reduce the communication cost of the whole application. The TreeMatch algorithm uses a heuristic method to find the subset of processes with minimum weights. Bhatele (2010) proposed several heuristics methods to handle irregular communication graphs and presented two techniques to find the closest processor to a given source in a two-dimensional (2D) mesh. The main idea of these heuristic methods is to reduce the average number of hops traveled per byte on mesh topologies. Deveci et al. (2015) proposed a greedy heuristic algorithm that combines two refinement algorithms. The heuristic part aims to minimize the value of hop-bytes by mapping the processes to the processors with high connectivity, and a breadth first search (BFS) based task-selection algorithm is designed to find the corresponding node. Mirsadeghi and Afsahi (2016) proposed a hybrid metric that is used to evaluate the candidate mapping from two aspects: hop-bytes and congestions. The refinement algorithm they used attempts to minimize congestion. This mapping framework can be applied only to a system that can provide specific congestion information.

The topology mapping research based on open-source graph partitioning software is another research method. Many open-source graph partitioning libraries like METIS, Scotch (Pellegrini and Roman, 1996), and Jostle (Walshaw and Cross, 2007) are used to implement mapping optimization. Mercier and Clet-Ortega (2009) used the Scotch library to map the communication pattern graph onto the physical topology graph. The Scotch library implements dual recursive bipartitioning algorithms to compute static mappings for graphs. Rodrigues et al. (2009) used a similar approach where they use the Scotch library to map the process topology graph

onto the physical topology graph. Tuncer et al. (2015) used the recursive graph bisection (RGrB) algorithm to recursively split both communication and physical topology graphs into equal halves using minimum weighted edge-cuts and to map the remaining task(s) to the remaining node at the end of the recursion. The implementation of the RGrB algorithm is based on LibTopoMap, and the METIS library is used for bisectioning. Although this technique demonstrates efficient mappings, it is also shown that it may result in poor p-way partitions. Wang et al. (2015) first applied clustering analysis to topology mapping, and proposed a process mapping optimization method based on clustering analysis guided by an aggregated QAP model. In this method, they used a spectral clustering algorithm to analyze process communication pattering and then mapped the process clusters to the physical topology.

4 New topology-aware mapping method

We propose an optimized heuristic topology-aware mapping algorithm based on the greedy method, pair-exchange, and backtrace. The whole framework of this algorithm includes making a primary mapping and optimization.

4.1 Input matrices

To assess the communication information of the submitted works, we define the communication pattern matrix $\boldsymbol{A}$. Message size is a major factor affecting the communication overhead. The communication performance bottleneck can be different with different message sizes; for instance, small messages are more sensitive to communication latency than large messages. Thus, the communication volume between processes is the measure for modeling this matrix. A_{ij} indicates the communication volume from processes i to j.

Define the hop matrix $\boldsymbol{B}$. The hop matrix indicates the number of hops required when each unit message communicates among the inter-processors. We need to obtain only the processor ID and the required number of hops from the underlying system. B_{ij} is set as the number of route hops traversed from processes i to j. Therefore, the product of the communication message size and the number of hops is the total communication cost. Essentially, this matrix should show the disparity of the communication bandwidth between the inter-processors. Based on the typical supercomputer topology, we categorize the route hops into three types, which will be discussed in detail in Section 4.2.

4.2 Topology of the Tianhe-3 exascale supercomputer prototype

Different from the three-level fat-tree topology of the Tianhe Express-2 network (Liao et al., 2015), the interconnect network of Tianhe-3 exascale supercomputer prototype adopts a 2D-mesh-like topology. Different from the typical 2D-mesh topology which is a regular 2D network structure, the new topology of Tianhe-3 is shown in Fig. 1a.

The switches are distributed as an $n \times m$ mesh. Each chip connects 96 compute nodes, which are divided into two equal parts (left and right parts). Chips in the same row or column are connected to each other. Thus, there are three hop situations among compute nodes: two compute nodes existing on one chip, two compute nodes located on two separate chips in the same row or column, and two compute nodes located on two separate chips in different rows and columns. These three situations are shown in Figs. 1b–1d.

Table 1 indicates the specific values of these three situations. As Table 1 indicates, we suppose that two nodes are on the same side (left or right) in their chips. When the two nodes are within one chip, the hop value is one. When the two nodes are located on two chips that are in the same row or column, the hop value is three (Fig. 1c). When the two chips are in different rows and columns, the hop value is five. If the two nodes are on different sides (one on the right side and the other on the left side), the communication requires an additional hop in each situation.

Table 1 Values of hops in different situaitons

Location relationship between compute nodes	Side of processors	Hop value
Within a chip	On the same side	1
	On different sides	2
Within a same row or column	On the same side	3
	On different sides	4
In different rows and different columns	On the same side	5
	On different sides	6

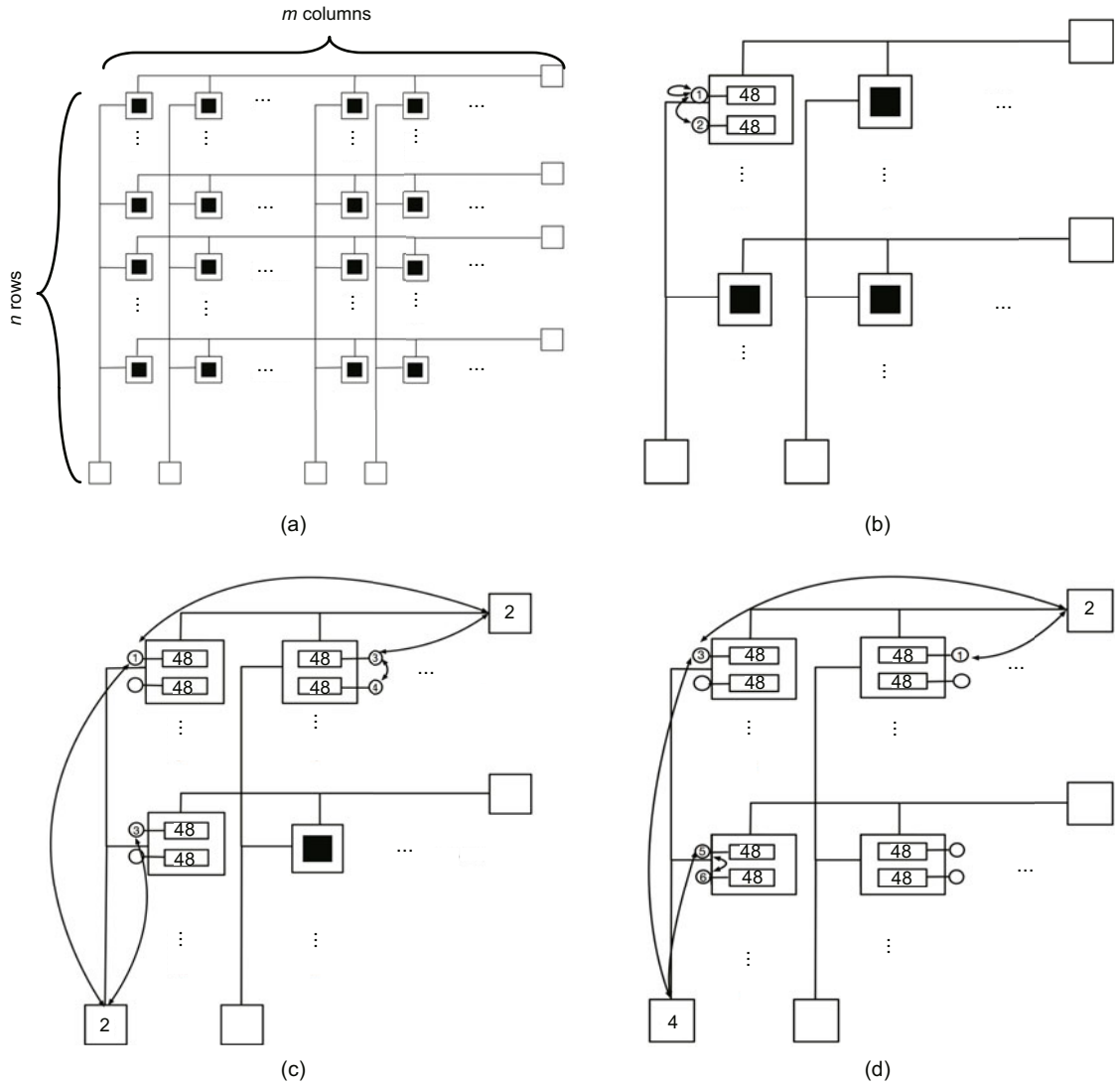

Fig. 1 Topology of the Tianhe-3 exascale supercomputer prototype (a) and three hop situations: (b) two compute nodes within a chip; (c) two compute nodes within a same row or column; (d) two compute nodes in different rows and columns

4.3 Optimized mapping strategy

The proposed OHTMA algorithm is based on the hop-byte metric and designed to optimize the communication performance. Algorithm 1 shows the pseudocode of the main procedures.

We tend to make a preliminary mapping that assigns the processes with large communication volumes to idle processors with small communication distances. Algorithm 1 first defines four sets, P_{selected}, $P_{\text{unselected}}$, N_{selected}, and $N_{\text{unselected}}$, indicating the sets of allocated processes, unallocated processes, selected processors, and unselected processors, respectively. First, we use a loop to give each process a mapping schedule. Within each initial iteration, we define an indicator comm_p of the total communication volume of process p and an indicator hops_n of the required communication hop summation of idle processors n in the first place. In the calculation of these two indicators, a weight factor is introduced to strengthen the effect of the selected process-processor pairs on the next selection, and this weight will become higher as the iterations continue. Then, we map the process with the maximum comm to the processor with the minimum number of hops, and move this process from $P_{\text{unselected}}$ to P_{selected} and processor from $N_{\text{unselected}}$ to N_{selected}. The iterations will be repeated until all the processes

Algorithm 1 Optimized heuristic topology-aware mapping algorithm (OHTMA) on Tianhe-3

Require: process P, idle processor N, communication pattern matrix $\boldsymbol{A}$, and hop matrix $\boldsymbol{B}$
Ensure: final mapping result f

// Begin the initial part
// At the beginning, all the processes and processors are unselected
$P_{\text{selected}} \leftarrow \varnothing$, $N_{\text{selected}} \leftarrow \varnothing$
$P_{\text{unselected}} \leftarrow P$, $N_{\text{unselected}} \leftarrow N$
$k \leftarrow 0$
while $k < |P|$ **do**
  #pragma omp parallel for
  for each process p in $P_{\text{unselected}}$ **do**
    $\text{comm}_p \leftarrow \sum_{i \in P_{\text{selected}}} A_{pi} + \dfrac{\sum_{j \in P_{\text{unselected}}} A_{pj}}{1 + |P_{\text{selected}}|}$
  end for
  #pragma omp parallel for
  for each processor n in $N_{\text{unselected}}$ **do**
    $\text{hops}_n \leftarrow \sum_{i \in N_{\text{selected}}} B_{ni} + \dfrac{\sum_{j \in N_{\text{unselected}}} B_{nj}}{1 + |N_{\text{selected}}|}$
  end for
  // Select the processes with the maximum comm_p and the processors with the minimum number of hops
  $p_{\max} \leftarrow \max\{\text{comm}_p\},\ p \in P_{\text{unselected}}$
  $n_{\min} \leftarrow \min\{\text{hops}_n\},\ n \in N_{\text{unselected}}$
  // Update mapping f and the subsets
  $f(p_{\max}) \leftarrow n_{\min}$
  $P_{\text{selected}} \leftarrow P_{\text{selected}} + \{p_{\max}\}$, $N_{\text{selected}} \leftarrow N_{\text{selected}} + \{n_{\min}\}$
  $P_{\text{unselected}} \leftarrow P_{\text{unselected}} - \{p_{\max}\}$, $N_{\text{unselected}} \leftarrow N_{\text{unselected}} - \{n_{\min}\}$
  $k \leftarrow k + 1$
end while
// Begin mapping optimization
Status$[|P|] \leftarrow 0, k \leftarrow 0$
// Loop is a user-defined number of iterations
while $k <$ loop **do**
  #pragma omp parallel for
  for each $i, j \in P$, where status$[i] \neq 1$ && status$[j] \neq 1$ **do**
    // Calculate the hop-byte changes after exchanging these two processes
    $\text{Cost}_{ij} = \sum_{l \in P} A_{il} B_{f(i)f(l)} + A_{jl} B_{f(j)f(l)} - \sum_{r \in P} A_{ir} B_{f(j)f(r)} + A_{jr} B_{f(i)f(r)}$
  end for
  // Find the maximum element in the **Cost** matrix
  $\text{Cost}_{mn} \leftarrow \max\{\textbf{Cost}\}$
  // Pair[ ] is used to store the exchanged pair and Result[ ] is used to store the change of hop-byte
  Pair $\leftarrow$ Pair $+ \{(m, n)\}$, Result $\leftarrow$ Result $+ \{\text{Cost}_{mn}\}$
  Status$[|P|] \leftarrow 1, k \leftarrow 1$
  Exchange $f(m)$ and $f(n)$
end while
// Calculate the sum of the first n terms (n from 0 to |Result|) in Result[ ]
Sum[ ] $\leftarrow$ sum(Result[ ])
// Find index t of the maximum value in Sum[ ]
$t \leftarrow$ max(Sum[ ])
Backtrack to the first t exchanges and obtain the final mapping f
Return f

are allocated to the processors primarily; that is, $P_{\text{unselected}}$ and $N_{\text{unselected}}$ are empty sets. A relatively reasonable subset of all the compute nodes will be extracted by steps 4–19 in Algorithm 1. Then we need to optimize this preliminary mapping.

After that, a pair-exchange method is applied to the optimization of the preliminary mapping. The main idea is to find any two entities within the subset of processes that should be exchanged and to update the mapping strategy. First, according to Eq. (1), we calculate an exchange difference value matrix **Cost**. Then, we select the pair of processes with the maximum **Cost** entry that represents the greatest saving if the allocation of these two processes is exchanged. Meanwhile, mark and store the processes that have been exchanged to avoid repetitive computation, and save the new mapping after updating. The iteration number of loops is set by users. At the end of all the iterations, we backtrack all the results after each exchange and obtain the maximum **Cost** summation of the first n exchanges. This backtracking mechanism overcomes the redundant update in the local pair-exchange-based method, and can return the status with the smallest communication cost generated by an intermediate operation.

5 Experimental results

In this section, we compare our OHTMA with other typical methods on Tianhe-3 and analyze the experimental results.

5.1 Experiment benchmarks and applications

We chose the following benchmarks and applications to evaluate the performance of our algorithm:

1. NPB suite

NPB suite is a set of programs designed to evaluate the performance of parallel supercomputers (Bailey et al., 1991). The implementations of NPB are available in commonly used programming models like MPI and OpenMP. In our experiments, we chose NPB version 3.3.1 and selected four programs shown in Table 2 with different communication patterns.

Problem sizes in NPB are predefined and indicated as different classes. We chose the class D problem (the large test problem, 16x size increase from each of the previous classes).

2. Two scientific applications

The two scientific applications selected were Sweep3D and Snap. Sweep3D is a benchmark code, solves a 1-group time-independent discrete ordinates neutron transport problem, and calculates the flux of neutrons through each cell of a 3D grid along several directions of travel (Wylie et al., 2010). It uses a pipelined wave-front method and a 2D process mesh. Snap is a memory consumption scientific application that mimics the 3D deterministic Sn transport equations (Zerr and Baker, 2013). It is designed to gauge system performance with problems typically encountered in the discrete ordinates transport community, such as multi-thread communication.

Table 2 Four NPB programs used in this study

Name	Problem
CG	Conjugate gradient, irregular memory access, and communication
BT	Block tri-diagonal solver
SP	Scalar penta-diagonal solver
LU	Lower-upper Gauss-Seidel solver

5.2 Experiment platform and setting

We performed the experiments on the Tianhe-3 exascale supercomputer prototype. The topology of Tianhe-3 has been introduced in Section 4.2. In the experiments, we ran the benchmarks and applications in 256, 512, 1024, 2048, and 4096 processes, in which BT and SP in Table 2 require the square number of processes; therefore, we carried out BT and SP with only 256, 1024, and 4096 processes. While the number of processes was increasing, we doubled the Snap cells in the z direction and the grid points of Sweep3D in the y direction. The variable loop is defined as half of the number of uploaded processes.

We evaluated our mapping method compared with two default mapping algorithms on Tianhe-3 (in-order and round-robin) and two typical topology mapping algorithms (greedy heuristic algorithm and RCM mapping algorithm) proposed by Hoefler and Snir (2011). To obtain accurate experimental results, we ensured that there were no other applications running on those nodes, tested each algorithm 10 times, and calculated the average value.

5.3 Results and analysis

Fig. 2 shows the resulting normalized metric values for different mapping methods with different numbers of processes. Fig. 3 shows the corresponding communication time of each benchmark program.

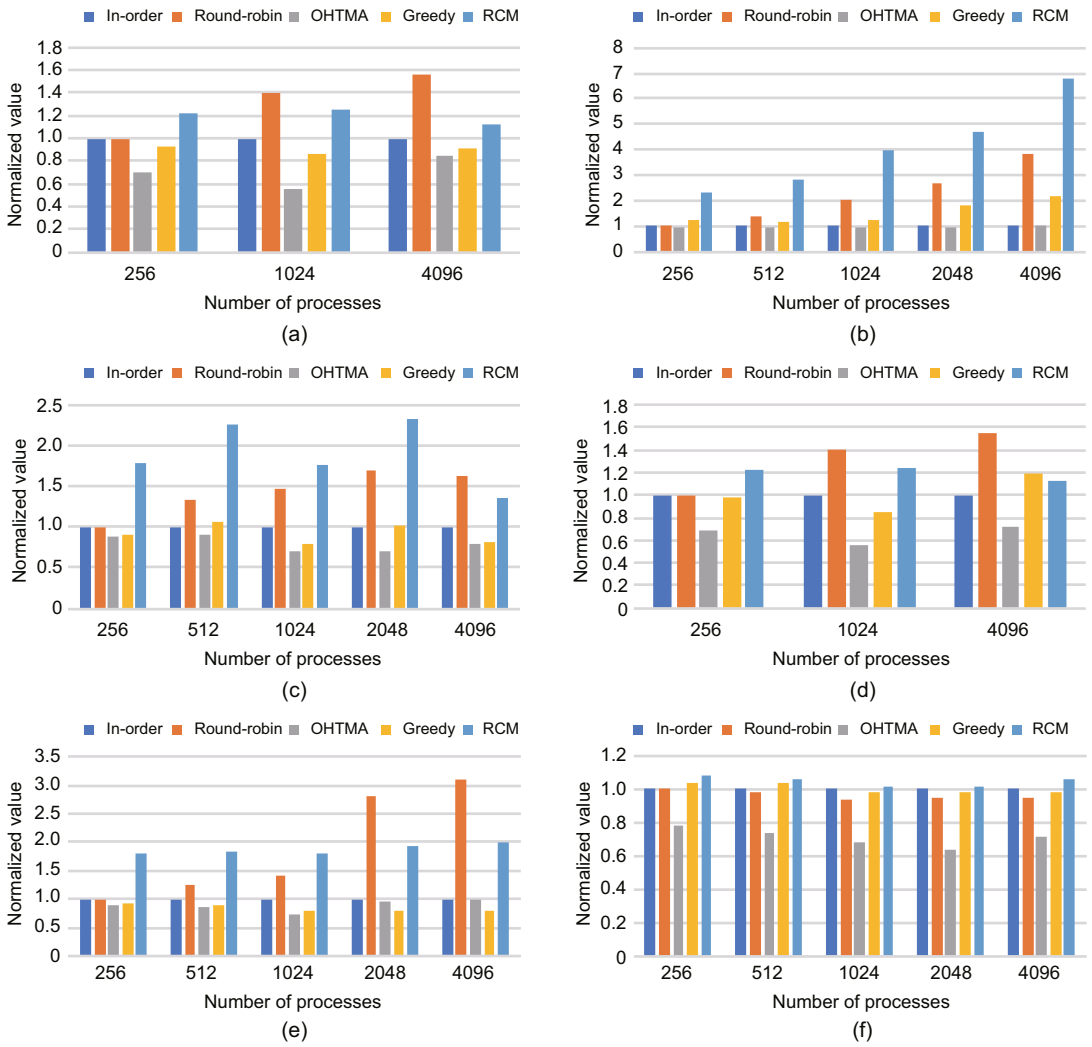

Fig. 2 Normalized cost metric for various mapping algorithms on Tianhe-3: (a) BT; (b) CG; (c) LU; (d) SP; (e) Sweep3D; (f) Snap

BT: block tri-diagonal solver; CG: conjugate gradient, irregular memory access, and communication; LU: lower-upper Gauss-Seidel solver; SP: scalar penta-diagonal solver

All results were normalized values over the default in-order mapping algorithm. As shown in Figs. 2 and 3, it is obvious that our algorithm had better performance than other strategies. As for hop-bytes, OHTMA obtained a reduction up to 43.9%, and led to as much as 37.3% saving in communication time.

As for applications composed mostly of point-to-point communications, such as BT and SP, their communication granularity is large and the number of messages is small. Figs. 2a and 2d show that OHTMA provided a large reduction in hop-bytes compared with other methods. It is obvious that RCM and round-robin were the most costly algorithms. As for BT, our strategy achieved a 29.7% reduction and a 45.3% reduction on average in hop-bytes compared with the in-order and round-robin strategies, respectively. At the same time, it obtained good performance on the communication time (Fig. 3a), and saved 13.5% of the communication time compared with the in-order strategy. We can see that the heuristic greedy algorithm obtained a little improvement compared with the RCM and default strategies. In contrast, our method saved an additional 11.6% communication time. Our method delivered a noticeable performance improvement on SP. This benchmark obtained improvements of 34.3%, 48%, 34.2%, and 44.7% compared with the four algorithms, in-order, round-robin, greedy, and RCM,

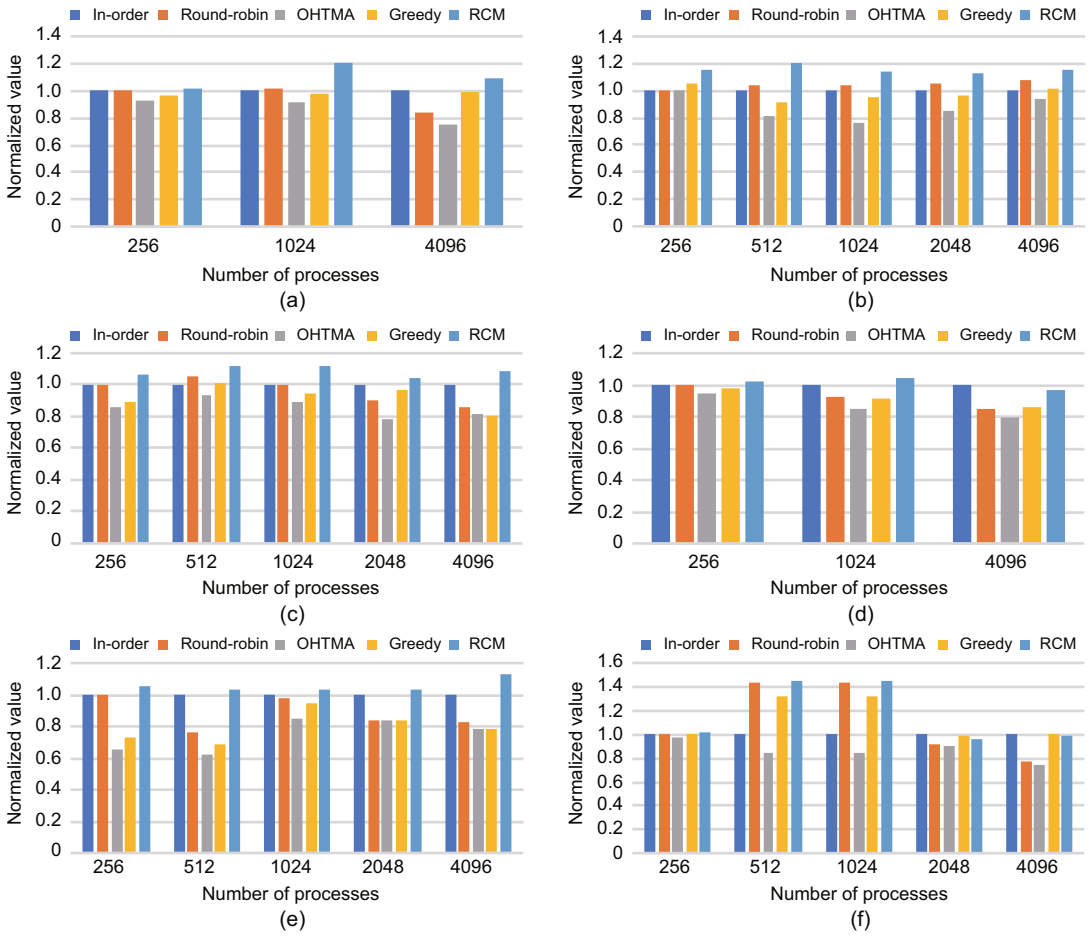

Fig. 3 Normalized communication time for various mapping algorithms on Tianhe-3: (a) BT; (b) CG; (c) LU; (d) SP; (e) Sweep3D; (f) Snap

BT: block tri-diagonal solver; CG: conjugate gradient, irregular memory access, and communication; LU: lower-upper Gauss-Seidel solver; SP: scalar penta-diagonal solver

respectively. Our method decreased the communication time by up to 20%.

Different from the former two benchmarks, CG includes a lot of irregular long-distance communications. The results indicated that OHTMA obtained slightly larger improvement in the hop-byte metric. The cost metric fell by only 4% based on the in-order strategy; however, as shown in Fig. 3b, the improvement in communication time was significant. The practical operation results show that, compared with other methods, our method can reduce the communication cost.

LU is a benchmark composed mainly of communications with small-sized messages (no more than 40 bytes). Figs. 2c and 3c show that RCM was the least effective method of topology mapping and that the increments of round-robin and RCM hop-bytes were proportional to the number of processes. By contrast, OHTMA obtained steady performance on optimization, especially concerning the cost metric. The cost metric of LU after remapping by our method decreased by 20.36% and 12.3% compared with the in-order method and greedy heuristic algorithm, respectively.

With regard to two scientific applications, our method obtained a better performance on Snap than on Sweep3D. Fig. 2f indicates that the cost metrics of the four strategies were similar and that our method obviously minimized this value compared with other methods. It achieved average improvements of 29.6%, 26.7%, 29.5%, and 32.4%, compared with the in-order, round-robin, greedy, and RCM strategies, respectively. However, Fig. 2e shows that Sweep3D obtained a large improvement compared

with two default strategies and the RCM strategy, but performed the worst compared with the greedy heuristic algorithm. Figs. 3e and 3f show the same results about communication time: OHTMA saved 16.7% (up to 19.5% and down to 13.2%) communication time on Snap but obtained only a 5.9% reduction on Sweep3D compared with the greedy heuristic algorithm.

Globally, the optimized method shows that it outperforms the two default and the two typical mapping algorithms with these six benchmark programs. It is obvious that for applications with infrequent and short-distance communications, like BT, SP, and Sweep3D, the heuristic algorithms (optimized algorithm and greedy algorithm) produce better performance than other algorithms. Moreover, when the message size is small (like Sweep3D), our algorithm provides performance that is approximately comparable to that of the greedy algorithm, but when the message size is large (like BT and SP), our optimized algorithm can exceed the greedy algorithm. It is precisely because our method tends to map the tasks with large-size messages on the compute nodes within a chip or within a same row and column, which saves significant communication time. As for the applications with variable distance communications (like LU and Snap), the round-robin and RCM algorithms may miss the features of the communication pattern. So, these algorithms obtain the worse performance. OHTMA improves communication performance significantly according to the communication time of these applications. Meanwhile, for the applications with long-distance communications like CG, the advantage of remapping through the optimized algorithm can be more evident when the scale of the problem is large.

The mapping time is another important indicator. We optimized the algorithm implementation. Because variables (like comm, hops, and cost) in an iteration can be calculated in parallel, we used OpenMP to reduce the runtime of the mapping algorithm. After applying the optimized algorithm implementation, the mapping time decreased from 10 to 4.7 s when the number of application processes was 256, and decreased from 195 to 98 s when the number of processes was 512. The mapping time is a one-time cost, so it has practical meaning. In the future work, we will reduce the mapping time by simplifying the computation of symmetric communications.

6 Conclusions

In this study, we proposed a new optimized heuristic topology-aware mapping algorithm (OHTMA). This algorithm provides a greedy mapping strategy and imports the pair-exchange method in the mapping optimization operation. We took both the communication pattern and actual topology of the compute platform into consideration. We first proposed a new topology-aware mapping algorithm and then attempted to apply it on the Tianhe-3 exascale supercomputer prototype. Two default mapping algorithms and two typical mapping algorithms were compared, and OHTMA obtained satisfactory experimental improvements. OHTMA can effectively reduce both the hop-byte value and communication time. In the future, we will intend to detail the communication pattern and reduce the runtime of the remapping process.

Contributors

Yi-shui LI designed the research. Jie LIU guided the research. Xin-hai CHEN helped perform experiments. Yi-shui LI drafted the manuscript. Bo YANG, Chun-ye GONG, and Xin-biao GAN helped organize the manuscript. Sheng-guo LI and Han XU helped modify the manuscript. Yi-shui LI revised and finalized the paper.

Compliance with ethics guidelines

Yi-shui LI, Xin-hai CHEN, Jie LIU, Bo YANG, Chun-ye GONG, Xin-biao GAN, Sheng-guo LI, and Han XU declare that they have no conflict of interest.

References

Agarwal T, Sharma A, Laxmikant A, et al., 2006. Topology-aware task mapping for reducing communication contention on large parallel machines. Proc 20th IEEE Int Parallel & Distributed Processing Symp, p.1-10. https://doi.org/10.1109/IPDPS.2006.1639379

Bailey DH, Barszcz E, Barton JT, et al., 1991. The NAS parallel benchmarks—summary and preliminary results. Proc ACM/IEEE Conf on Supercomputing, p.158-165. https://doi.org/10.1145/125826.125925

Bhatele A, 2010. Automating Topology Aware Mapping for Supercomputers. PhD Thesis, University of Illinois at Urbana-Champaign, Urbana, USA.

Bhatele A, Laxmikant V, 2009. An evaluative study on the effect of contention on message latencies in large supercomputers. Proc IEEE Int Symp on Parallel & Distributed Processing, p.1-8. https://doi.org/10.1109/IPDPS.2009.5161094

Brandfass B, Alrutz T, Gerhold T, 2013. Rank reordering for MPI communication optimization. *Comput Fluid*,

80:372-380.
https://doi.org/10.1016/j.compfluid.2012.01.019

Chen X, Liu J, Li S, et al., 2018. TAMM: a new topology-aware mapping method for parallel applications on the Tianhe-2A supercomputer. Proc 18th Int Conf on Algorithms and Architectures for Parallel Processing, p.242-256. https://doi.org/10.1007/978-3-030-05051-1_17

Deveci M, Kaya K, Uçar B, et al., 2015. Fast and high quality topology-aware task mapping. Proc IEEE Int Parallel and Distributed Processing Symp, p.197-206. https://doi.org/10.1109/IPDPS.2015.93

Hoefler T, Snir M, 2011. Generic topology mapping strategies for large-scale parallel architectures. Proc Int Conf on Supercomputing, p.75-84. https://doi.org/10.1145/1995896.1995909

Hoefler T, Jeannot E, Mercier G, 2014. An overview of topology mapping algorithms and techniques in high-performance computing. In: Jeannot E, Žilinskas J (Eds.), High-Performance Computing on Complex Environments. Wiley, Hoboken, New Jersey, USA. https://doi.org/10.1002/9781118711897.ch5

Jeannot E, Mercier G, 2010. Near-optimal placement of MPI processes on hierarchical NUMA architectures. In: D'Ambra P, Guarracino M, Talia D (Eds.), Euro-Par 2010 Parallel Processing. Springer Berlin Heidelberg, Germany, p.199-210. https://doi.org/10.1007/978-3-642-15291-7_20

Jeannot E, Mercier G, Tessier F, 2014. Process placement in multicore clusters: algorithmic issues and practical techniques. *IEEE Trans Parall Distrib Syst*, 25(4):993-1002. https://doi.org/10.1109/TPDS.2013.104

Karypis G, Kumar V, 1998. METIS—A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Ordering of Sparse Matrices. Technical Report, University of Minnesota, Minneapolis, USA.

Liao X, Pang Z, Wang K, et al., 2015. High performance interconnect network for Tianhe system. *J Comput Sci Technol*, 30(2):259-272. https://doi.org/10.1007/s11390-015-1520-7

Mercier G, Clet-Ortega J, 2009. Towards an efficient process placement policy for MPI applications in multicore environments. In: Ropo M, Westerholm J, Dongarra J (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer Berlin Heidelberg, Germany, p.104-115. https://doi.org/10.1007/978-3-642-03770-2_17

Mirsadeghi SH, Afsahi A, 2016. PTRAM: a parallel topology-and routing-aware mapping framework for large-scale HPC systems. Proc IEEE Int Parallel and Distributed Processing Symp Workshops, p.386-396. https://doi.org/10.1109/IPDPSW.2016.146

Pellegrini F, Roman J, 1996. SCOTCH: a software package for static mapping by dual recursive bipartitioning of process and architecture graphs. Proc Int Conf and Exhibition on High-Performance Computing and Networking, p.493-498. https://doi.org/10.1007/3-540-61142-8_588

Rodrigues E, Madruga F, Navaux P, et al., 2009. Multi-core aware process mapping and its impact on communication overhead of parallel applications. Int Symp on Computers and Communications, p.811-817. https://doi.org/.10.1109/ISCC.2009.5202271

Sahni S, Gonzalez T, 1976. P-complete approximation problems. *J ACM*, 23(3):555-565. https://doi.org/10.1145/321958.321975

Sudheer CD, Srinivasan A, 2012. Optimization of the hop-byte metric for effective topology aware mapping. Proc 19th Int Conf on High Performance Computing, p.1-9. https://doi.org/10.1109/HiPC.2012.6507513

Tuncer O, Leung VJ, Coskun AK, 2015. PaCMap: topology mapping of unstructured communication patterns onto non-contiguous allocations. Proc 29th ACM on Int Conf on Supercomputing, p.37-46. https://doi.org/10.1145/2751205.2751225

Walshaw C, Cross M, 2007. JOSTLE—parallel multilevel graph-partitioning software: an overview. In: Magoulès F (Ed.), Mesh Partitioning Techniques and Domain Decomposition Methods. Saxe-Coburg Publications, Stirlingshire, UK, p.22-58. https://doi.org/10.4203/csets.17.2

Wang T, Qing P, Wei D, et al., 2015. Optimization of process-to-core mapping based on clustering analysis. *Chin J Comput*, 38(5):1044-1055 (in Chinese).

Wylie BJN, Böhme D, Mohr B, et al., 2010. Performance analysis of Sweep3D on Blue Gene/P with the Scalasca toolset. Proc IEEE Int Symp on Parallel & Distributed Processing, Workshops and PhD Forum, p.1-8. https://doi.org/10.1109/IPDPSW.2010.5470816

Zerr RJ, Baker RS, 2013. Snap: SN (Discrete Ordinates) Application Proxy-Proxy Description. Technical Report, LA-UR-13-21070, Los Alamos National Laboratory, Los Alamos, USA.