SUMMARY  Inter-node communication is essential in parallel computation. The performance of parallel processing depends on the efficiencies in both computation and communication, thus, the communication cost is not negligible. A parallel application program involves a logical communication structure that is determined by the interchange of data between computation nodes. Sometimes the logical communication structure mismatches to that in a real parallel machine. This mismatch results in large communication costs. This paper addresses the node-mapping problem that rearranges logical position of node so that the degree of mismatch is decreased. This paper assumes that parallel programs execute one or more collective communications that follow specific traffic patterns. An appropriate node-mapping achieves high communication performance. This paper proposes a strong heuristic method for solving the node-mapping problem and adapts the method to a genetic algorithm. Evaluation results reveal that the proposed method achieves considerably high performance; it achieves 8.9 (4.9) times speed-up on average in single-(two-)traffic-pattern cases in $32 \times 32$ torus networks. Specifically, for some traffic patterns in small-scale networks, the proposed method finds theoretically optimized solutions. Furthermore, this paper discusses in detail about various issues in the proposed method that employs genetic algorithm, such as population of genes, number of generations, and traffic patterns. This paper also discusses applicability to large-scale systems for future practical use.

key words: interconnection networks, node mapping, genetic algorithms

1. Introduction

Performance of parallel computation mainly consists of two issues of computation and communication, since inter-node communication is essential in the parallel processing. Thus, many studies address reduction of communication costs for high performance of parallel execution as well as reduction of computation time. One of the efforts is the interconnection network technology [1], [2], which includes topology as a system organization and routing algorithms of packets.

A parallel computer is constructed by a specific physical structure, which, in most cases, cannot be changed flexibly. On the other hand, logical structures that are given by actual parallel application programs vary in a topological view. If the logical structure of an application program matches to the physical one (of the real machine), the program runs efficiently on the machine. Otherwise, if the logical and physical structures largely mismatch, the parallel machine suffers from inefficient execution of the parallel program.

In a parallel program in a message-passing model, every computing node sends messages as the program specifies. By summing up the source-to-destination communications, the application program forms a directed graph that represents a traffic pattern. As we discussed above, in general, the logical directed graph of the application program differs from the physical structure of the real machine.

As the degree of the mismatch between the two graphs increases, communication overheads also increase and the performance of parallel execution is limited since communication latency increases. From the opposite viewpoint, if we can reduce the mismatch, the net performance of the machine increases. The node-mapping problem is widely discussed in decades of history of the parallel computation and many studies that intend to reduce the mismatch are reported.

So far, discussions and solutions in the past studies are about topological mapping of a single traffic pattern. However, application programs may change communication patterns in different phases of execution. Thus, even the mapping result is optimized for a specific communication pattern, the mapping does not guarantee efficient communication in other communication patterns in the different computing phases.

This paper addresses the node-mapping problem that rearranges logical positions of nodes so that the degree of mismatch is reduced. This paper assumes that a parallel application program executes one or more collective communications that follow specific traffic patterns. This paper furthermore assumes that each node sends a fixed amount of messages during each collective communication session.

This paper extends our prior work [3] in which fundamental ideas and methods are reported with preliminary results in a small-scale (8x8) system. Although the prior work reveals the fundamental efficiency, many issues remain open to discuss practical issues that include further improvements and in-depth considerations for practical applications. The contribution of this paper consists of two major parts; one is a strong heuristics that is applicable to large-scale systems, and the other one is comprehensive discussion for wide applications.

As the former contribution, this paper discusses strong heuristics that can solve the node-mapping problem at a practical level. The proposed method is adopted to genetic
algorithm and the resulting genetic method solves the problem and obtains satisfactory results that can accelerate parallel computation.

Additionally, this paper offers in-depth discussions about the proposed method as the second contribution. The discussion includes population of genes, number of generations in the genetic method, and characteristics of traffic patterns. Furthermore, this paper discusses reduction method of computation time for large-scale systems.

The rest of this paper is organized as follows. Section 2 summarizes the node-mapping problem and overviews the results in our prior work [3]. Section 3 proposes the strong heuristics method and is application to a genetic algorithm. This section includes evaluation results of the proposed methods. Following the evaluation results, Sect. 4 places in-depth discussions. Furthermore, Sect. 5 discusses application to large-scale systems. Section 6 summarizes related work of this paper. Finally, Sect. 7 concludes this paper.

2. Preliminaries

2.1 Problem Statement: Node Mapping

A parallel computer consists of a set of computing nodes that are connected by communication links (or busses). Thus, the organization of the parallel computer is modeled as a directed graph $G_r(E_r, V_r)$, called array graph [4], where a vertex represents a computing node and an edge represents a communication link. During the execution of a parallel application program, computing nodes communicate to each other according to the program. The whole communications in the parallel system form a communication pattern that can be represented as a directed graph $G_p(E_p, V_p)$, called problem graph.

In general, when a parallel program is represented as $G_p$, each vertex in $G_p$ should be mapped to an appropriate vertex in $G_r$. When the both directed graphs $G_p$ and $G_r$ differ and $G_p$ is not a subgraph of $G_r$, possible communication overheads may disturb efficient execution of the parallel program so that the performance of the parallel system is limited. The overheads include long latency and contention (congestion).

Node mapping problem is formalized as searching a mapping function that can map $G_p$ to $G_r$ appropriately. Efficient mapping methods have been discussed actively for decades of parallel computation history as we refer to Sect. 6. Many of studies consider a unique application geometry that is mapped to a physical one, i.e., $G_p \rightarrow G_r$. This paper extends the number of problem graphs so that the application program may change its communication pattern according to its execution phase, i.e., $G_{p1} \rightarrow G_r$, $G_{p2} \rightarrow G_r$, ..., $G_{pn} \rightarrow G_r$. We assume that communication patterns that correspond to $G_{p1}$, $G_{p2}$, ..., $G_{pn}$ appear sequentially (i.e., not simultaneously) in the execution of a parallel program.

Each vertex $v \in V_r$ has its unique address $x$. We assume that the physical topology of a parallel machine is two-dimensional torus for simplification of discussions. Furthermore, we assume that the address $x^p$ is represented also in a two-dimensional space. Thus, a computing node that is located at the physical address $x^p$ has a logical address $x$ that represents the mapping result.

The logical address $(x^r)$ is not changed during the parallel execution since dynamic re-arrangement of node-mapping is impractical due to large migration costs. Thus, we modify the formulation of the node-mapping problem as finding an optimal mapping function of $\Gamma(x)$ where

$$x^r = \Gamma(x^p).$$

Collective communication is carried out on the basis of logical addressing. Assume that a source node $x^p$ sends a packet to its destination $x^p$. In this case, the packet actually starts from the physical node $x^p = \Gamma^{-1}(x^p)$ and it destines the logical node $x^p$ at $x^p = \Gamma^{-1}(x^p)$.

2.2 Preliminary Results

2.2.1 Pairwise Exchange

In general, the node-mapping problem is NP-hard and it is natural to introduce meta-heuristic methods such as simulated annealing and genetic algorithms to obtain practical solutions for a given node-mapping problem. We have introduced a genetic algorithm with some levels of heuristics to solve the node-mapping problems that are defined in the previous section.

In a class of node-mapping problems, where this paper stands on, every vertex in the problem graph is mapped to an corresponding vertex in the array graph by one-to-one and vice versa. Thus, for example, duplicated assignment of multiple problem vertices to an array vertex is disallowed.

To maintain the strict constraint in the node-mapping problem, the pairwise exchange method is widely used in many studies. The method selects two vertices in the graph and swaps the mapped (logical) locations. The method is fundamental and it is a key in solving node-mapping problems, thus, we have started from the method and extended it for better solutions.

A naïve pairwise exchange method that selects vertices randomly can serve only poor performance in its solutions, thus, it is impractical. In our preceding work [3], we have presented two heuristic methods based on the pairwise exchange.

One of the two methods is the geometric swap. The method includes single-, line- and box-swap that intend to keep geometric features in the graph. Single-swap is equivalent to the random pairwise exchange. Line(box)-swap method selects two regions in the 2D-torus that form a specific geometry of line segment (box) and swaps every vertex

---

1In the prior work [3], we used individual methods. The single-, line-, and box-swap methods have inclusive relationship, i.e., single-swap ∈ line-swap ∈ box-swap, thus, we introduce the geometric swap as a containment class in this paper.
in the selected region on the one-by-one basis. The selected two regions allow 0-, 90-, 180-, and 270-degree rotation and symmetry in $x$- and/or $y$-axis.

The behavior of the pairwise exchange methods is basically evolutilonal since the methods continuously improve their solutions through many repetitive exchanges. During the evolutilional process of the method, we can expect that intermittent solutions have partially optimized. Thus, we intend that the geometric swap possibly maintains the partial optimized geometry in the intermittent results.

The second method, neighboring swap, has been introduced to aggressively enhance the pairwise exchange. It firstly selects a traffic pattern $Pa$ and further selects a source–destination node pair that has the longest distance in the selected traffic pattern. Then, it finds an appropriate vertex that potentially becomes a neighbor of the selected vertex in the mapped situation.

Figure 1 shows the operation of the neighboring swap method. The method randomly selects a node $As$ and it sends packet to $Ad$ according to the given traffic pattern $Pa$. Then, the method searches an appropriate node $Bs$ that satisfies the following conditions (Fig. 1 (a)).

\[
AsBd \leq AsAd \quad \text{and} \\
(AsBd + BsAd) \leq (AsAd + BsBd + r) \quad \text{and} \\
BsAd \leq r \quad \text{and} \\
AsBd \leq r.
\] (2)

where $Bs$ sends packets to $Bd$ in the given traffic pattern and $r$ is given as the radius of a search region. The method swaps the $Bs$ and $Ad$ nodes in terms of logical address so that the communication distances from $As$ to $Ad$ and from $Bs$ to $Bd$ are shortened to the radius $r$ (Fig. 1 (b)). The logical address of $Ad$ is changed to that of $Bd$ and the corresponding node is represented as $Bd'$ in Fig. 1 (b). Similarly, $Bd$ is changed to $Ad'$ in the figure.

2.2.2 Genetic Approach

In our genetic methods of node-mapping, we use a mapping matrix as the representation of a gene. A mapping matrix represents the mapping function $x' = \Gamma(x)$ that specifies a logical address $x'$ from the physical address $x$ for each node in the whole system. Since we assume 2D-torus topology as the physical configuration, both logical and physical addresses are represented in a 2-dimensional space. The mapping matrix is an $N\times N$ matrix whose member at $(x, y)$ shows the logical address $x' = (x, y)$.

A genetic method consists of a selection (survival) operation and a genetic operation. The method has a specified population of genes, each of them represents a candidate solution. The method then selects survivors in the next generation and generates children from the survivors by the genetic operation. The genetic operation, in general, includes mutation and crossover.

In the previous work [3], we use the geometric swap and neighboring swap as genetic (mutation) operations. The selection operation furthermore consists of the sorting order of evaluation metrics and the ratio of survivors. The evaluation metrics include the duration time of collective communication that is measured by a simulator and average number of hops (avg.hops)\(^1\) for every node in the system to its destination specified by a given traffic pattern. We use the following three variants in evaluation of metrics:

- **Duration first.** (sc) Genes are sorted by the duration-time order. If the duration is same, avg.hop is used.
- **Avg.hop first.** (ah) Genes are sorted by the avg.hop order. If the avg.hop is same, duration is used.
- **Multiplication of duration and avg.hops.** (ml) Genes are sorted by multiplied value of duration time and avg.hop.

3. Improved Heuristics and Genetic Operation

Our preliminary work has revealed preferable performance of the GA-based method, however, the results are insufficient for discussing applicability and practicality for larger-scale systems, and in-depth discussions are also required. Genetic methods, in general, can offer strong capability in searching (quasi-)optimal solutions in non-linear systems. For small-scale systems, the exploration method possibly finds exact solutions that are optimal for the given problem, since the search space is relatively small. However, in large-scale systems, the search space becomes extremely large according to the system size. The large search space results in non-optimal (but near-optimal) solutions. This leads us to explore more efficient methods.

This section discusses enhanced methods in the GA-based method according to the mutation and selection operations. The key-points in the discussion are capability of search. Strong heuristics and deep surviving strategy will increase search capability, although, they will suffer local-minima. Thus, we introduce crossover and local-best ideas to mitigate the local-minima phenomena. Furthermore, this section also offers evaluation results.

3.1 Strong Heuristics

One of the significant results in our prior work is the heuristctic level in the mutation operations. Section 2.2.1 offers two

---

\(^1\)For convenience, we use an abbreviation form ‘avg.hop’ for the average number of hops.
major mutation methods, i.e., geometric swap and neighboring swap, and the prior work compares the two methods with the random exchange. Lessons from the work suggest that random methods are weak in optimization performance and that strong heuristics are preferable for fast convergence to (quasi-)optimized solutions.

In single-traffic-pattern cases, the geometric swap method works well so that it finds (almost) optimized solutions. During the repetitive mutation operations, the method tries to rearrange the node-mapping suitable for a given traffic pattern. However, in multiple-traffic-pattern cases, the rearrangement possibly destroys the optimal mapping for the alternative traffic pattern(s).

We have reached a new mutation method though the above discussions. The new method is based on the neighboring swap but the key point is that the method supports two traffic patterns simultaneously. Figure 2 illustrates operations of the new method, named enhanced neighboring swap. At first, the method randomly selects a traffic pattern (Pa) and a node (Ao). Here we assume that the node Ao sends packets to As and also that As sends packets to Ad according to the selected traffic pattern Pa.

Then, the method selects a neighboring node Bs that is located within the distance r from Ao. The node Bs sends packets to Bd according to the traffic pattern Pb. Furthermore, it randomly selects an alternative traffic pattern (Pb). Bs sends packets to Bt in the traffic pattern Pb. If Bt is not located within the radius r from As, the method re-select another candidate node Bs until |AoBs| ≤ r and |AsBt| ≤ r. If the method cannot find appropriate Ao, As, Bs, and Bt nodes, the method re-starts from selecting another Ao.

Figure 2 (a) shows the situation of the first selection step. Note that the red and green arrows in Fig. 2 (a) show the source-destination pairs according to the selected traffic patterns Pa and Pb, respectively. In the second step, the enhanced neighboring swap method swaps logical addresses of following two node-pairs simultaneously: As ↔ Bs and Ad ↔ Bd, as Fig. 2 (b) shows the result. In the figure, in the same way in Fig. 1, the logical addresses of As and Bs nodes are exchanged and those notations are changed to Bs' and As', respectively. Also Ad and Bd exchange their own logical address to each other and notations are changed to Bd' and Ad', respectively.

3.2 Weighted Selection in Mutation

In the prior work, every surviving gene evenly generates its children by a given mutation method. For example, assuming that the population is 1000 and that one-fourth of the population are surviving, top 250 genes survive and each survivor produces three children.

Each survivor has its own ranking with respect to the evaluation metrics. We can expect that a strong survivor will produce strong children that possibly overcome the existing genes in the next (or future) generation. Thus, if higher-order survivors produce more children, the genetic operation will find preferable solutions rapidly.

We introduce weighted selection of survivors for a mutation operation. Assume that genes $g_0, g_1, \ldots, g_{P-1}$ are ordered in the evaluation metric, where the superscript $t$ shows the generation number. Figure 3 shows the pseudo code of the selection algorithm. $P$ is the number of genes (i.e., population). In the pseudo code in Fig. 3, $s$ and $d$ show the ranks of the parent and child genes in the next generation, respectively. In this method, genes $g_0, g_1, g_0^*, g_1^*, g_0^*, g_1^*$, $g_0^*, g_1^*, \ldots$ produce children in order.

We can expect a strong search operation by using this method. On the other hand, strong genes possibly cause strong local-minima phenomena that falls into unsatisfactory solutions. We will further discuss countermeasures against the local-minima in Sect. 3.4.

3.3 Crossover

In the prior work, we employ only mutation methods as genetic operations and omitted crossover operations for preliminary discussions. However, discussions on the crossover methods are essential for discussing efficient genetic operations. During the repetitive genetic operations, an intermittent solution possibly includes partially optimized mapping result, even when it cannot offer a fully optimized result. If another unoptimized gene inherits the partially optimized portion, the inheritor gains larger benefit than that of pairwise exchange.

In this paper, we introduce a simple crossover operation as Fig. 4 shows. The crossover operation firstly selects arbitrary two genes $g_i$ and $g_k$ as Figs. 4 (a) and (b) show. The method then selects nodes As in $g_i$ and As' in $g_k$. As and As' send packets to Ad and Ad', respectively. In this example in Fig. 4, the location of Ad' is closed to As compared to the original Ad. Then, the method selects an arbitrary node Ad" that is located between Ad and Ad' and the method let Ad" be the destination node of As by swapping the logical addresses of Ad" and Ad.
In our prior work, steep genetic operations achieve good performance in a short period of generations, however, they often suffer from the problem of local-minima. In a typical local-minima situation, for example, most of genes run within a quite narrow search space for generations. Steep genetic operation has difficulty in escaping from the pitfall of local-minima.

To avoid concentrating narrow search space but maintaining good search capability, we introduce the local-best idea from particle swarm optimization (PSO). In a typical PSO method, each particle searches solutions repeatedly by moving in the search space [5]. During the search operation, particles are grouped and the best particle (i.e., the local-best) in each group is selected. Furthermore, the global best particle is also selected. Each particle moves forward to the global and/or local best one. This maintains wide search space for avoiding narrow local-minima.

In this paper, all genes are grouped where each group has \( n_g \) members. Each group is numbered and neighboring two groups overlap by shifting \( n_t \) members. Figure 5 shows the configuration of groups. Each gene belongs to multiple groups according to the given parameters \( n_g \) and \( n_t \).

Genes are sorted within the group and top one-fourth (or one-eighth) genes survive so that each of them produces three (seven) children by the mutation operation. Thus, the search operation is carried out by the group basis. When every group is independent to others, the search operation is equivalent to a parallel operations in small population. Small population has fatal disadvantage in the quality of the solutions, thus, the global best in the total population is inevitable for strong search capability. Overlap between two adjacent groups propagates the local-best in a specific group to its adjacent groups. Thus, we can expect both steep search capability from the global best and, at the same time, wide search space from the local best.

### 3.5 Evaluation of Enhanced Methods

As we propose variations of mutation methods, selection methods and sorting order, an actual genetic operation is represented as a combination of the three kinds of methods. In this paper, we select the ah method for sorting order (Sect. 2.2.2) and detailed comparison with other sorting methods is discussed in Sect. 4.1. Thus, in this section, we evaluate major variants in the ah method.

We use a symbol to identify an actual genetic method. The symbol consists of two digits \( v \) and \( w \) with a leading character ‘s’: in the form of ‘suvw’. \( s00 \) and \( s05 \) employ the geometric swap (Sect. 2.2.1). \( s00 \) mutates genes by single-swap (37.5%), line-swap (25.0%), box-swap (12.5%), and random search (25.0%). \( s05 \) uses only single-swap. \( s2w \) employs the neighboring swap method (Sect. 2.2.1), and \( s5w \) employs the enhanced neighboring swap method (Sect. 3.1). The second digit \( w \) specifies the selection method as Table 1 shows. \( s99 \) is used specifically for the random search which is not a genetic method.

We implemented an evaluation platform for evaluating the node-mapping methods, which is extended from our interconnection network simulator based on the cellular automata principle [6], [7] as a simulation engine. As the node-mapping methods aim at reduction of the duration time in collective communication, the major evaluation function in the genetic operation is the duration time for each gene that represents the logical mapping result. The simulation engine is used for accurately measuring the duration times.

In the genetic method, we use 1,000 populations of genes and each genetic method runs for 1,000 generations, by default. In each genetic run, all genes are initialized randomly and their evaluation metrics (duration time and avg.hop by default) are measured. Then, the genes are sorted according to the specific order (avg.hop first by default) and specified number of top genes are selected to produce their children for the next generation. During the execution of the genetic method, the best duration time and the avg.hop are recorded. When the genes do not update the best scores for a specified period of generations\(^1\), the generic method regards the system state as a local-minima and it re-initializes all genes. Note that partial initialization of genes

\(^1\)The period is 100 generations for 8 network, 200 generations for 16 × 16 network, and 400 generations for 32 × 32 network.

### Table 1: Meaning of surviving digit.

<table>
<thead>
<tr>
<th>Selection Method</th>
<th>Surviving Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (s*0)</td>
<td>survive top 25.0%</td>
</tr>
<tr>
<td>1 (s*1)</td>
<td>survive top 12.5%</td>
</tr>
<tr>
<td>2 (s*2)</td>
<td>survive top 50.0%</td>
</tr>
<tr>
<td>3 (s*3)</td>
<td>weighted selection (Sect. 3.2)</td>
</tr>
<tr>
<td>4 (s*4)</td>
<td>crossover (Sect. 3.3)</td>
</tr>
<tr>
<td>5 (s*5)</td>
<td>survive top 25.0% in each local group (Sect. 3.4)</td>
</tr>
<tr>
<td>6 (s*6)</td>
<td>survive top 12.5% in each local group (Sect. 3.4)</td>
</tr>
</tbody>
</table>
does not work well since, in a local-minima situation, most of the genes show the same (or quite close) solution that behaves as a strong attractor.

We firstly measure the baseline duration time \( (t_{db}) \) in the mapping condition \( x^L = x^P \). Then, we run a genetic method to obtain its solution that has the shortest duration time \( (t_{dg}) \) in the genetic run. Speed-up ratio is calculated as \( r_g = t_{db} / t_{dg} \). Since the raw results have sometimes large variations, we use an average value of ten runs for each

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Traffic patterns used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbrev.</td>
<td>description</td>
</tr>
<tr>
<td>bcmp</td>
<td>bit-complement. ( w_{2n-1}w_{2n-2} \ldots w_0 \rightarrow w_{2n-1}w_{2n-2} \ldots w_0 )</td>
</tr>
<tr>
<td>brev</td>
<td>bit-reverse. ( w_{2n-1}w_{2n-2} \ldots w_0 \rightarrow w_0 \ldots w_{2n-2}w_{2n-1} )</td>
</tr>
<tr>
<td>brot</td>
<td>bit-rotation. ( w_{2n-1} \ldots w_1w_0 \rightarrow w_0w_{2n-1} \ldots w_1 )</td>
</tr>
<tr>
<td>shfl</td>
<td>perfect shuffle. ( w_{2n-1}w_{2n-2} \ldots w_0 \rightarrow w_{2n-2} \ldots w_0w_{2n-1} )</td>
</tr>
<tr>
<td>torn</td>
<td>tornado. ( W \rightarrow \text{mod}(W + N/2, N^2) )</td>
</tr>
<tr>
<td>trns</td>
<td>transpose. ( (X,Y) \rightarrow (Y,X) )</td>
</tr>
</tbody>
</table>

Table 3 | Abbreviation of traffic pattern. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abbrev.</td>
<td>compact form</td>
</tr>
<tr>
<td>bcmp</td>
<td>bc</td>
</tr>
<tr>
<td>brev</td>
<td>bv</td>
</tr>
<tr>
<td>brot</td>
<td>bt</td>
</tr>
<tr>
<td>shfl</td>
<td>sh</td>
</tr>
<tr>
<td>torn</td>
<td>to</td>
</tr>
<tr>
<td>trns</td>
<td>tr</td>
</tr>
</tbody>
</table>

Figure 6 shows the six traffic patterns used in this paper. Each traffic pattern is represented in the form of abbreviation as Table 3 shows. For example, brev shows the bit-reverse traffic and bv is the compact form of brev. The compact form is mainly used in multiple-traffic-pattern cases, where concatenation of the compact forms are used to represent traffic patterns. For example, bcbt is the concatenation of bc and bt, thus, the symbol shows the traffic patterns bcmp and brot. Similarly, bcvtb shows a three-traffic-pattern case bcmp–brev–brot.

Evaluation results of the speed-up ratios are given in Figs. 6, 7, and 8. In these figures, the horizontal axis shows traffic patterns, and the vertical axis shows speed-up ratio. ‘avg’ in the horizontal axis shows the geometric mean of the results. These figures illustrate that the enhanced neighboring swap (i.e., \( s5w \)) achieves considerable performance. Figure 9 summarizes average speed-up ratios of individual \( s5w \) methods, where we can find \( s53 \) is hopeful. In this figure, \( N=n \) means \( n \times n \) network and \( pat \) denotes \( p \)-traffic-pattern cases. Note that the maximal speed-up ratio in our evaluation is limited, since the minimal duration time is theoretically limited by the physical law.

Figure 10 shows visualization of traffic patterns in some solutions by the \( s53 \) method in 16 \( \times \) 16 network. In this figure, a circle represents a computing node and an arrow shows a communication path on which a packet is transferred. If multiple arrows are drawn in the same direction between two adjacent nodes, the multiplicity represents the degree of conflicts of packets at the corresponding link. On
the other hand, no packets are routed where no arrows are drawn. In this figure, bcmp pattern shows simple packet paths as Figs. 10 (a) and (b) show. Similarly, brev pattern also has simple paths (as Figs. 10 (a) shows). However, brot has complicated paths and many conflicts. These results reveal that quality of the solution depends on the traffic pattern. We will discuss the problem in Sect. 4.4.

Fig. 7 Speed-up ratio (16 × 16 network).

Fig. 8 Speed-up ratio (32 × 32 network).

Fig. 9 Speed-up ratio (s5*).

Fig. 10 Two-traffic-pattern solution examples.
4. Discussion

Previous section shows the fundamental advantages of the proposed method. For wide application of the proposed method to the practical problems, rich information, which includes underlying characteristics, detailed behaviors, and prospect for practical systems, is required. This and the next sections answer the requirements and offer wide knowledge through extensive discussions.

This section firstly discusses appropriate evaluation functions (Sect. 4.1), on which the prior work does not show distinct difference in the small-scale systems. The section then unveils underlying characteristics and detailed behaviors, which include required numbers of generations (Sect. 4.2) and populations (Sect. 4.3). Furthermore, this section discusses relationship between the nature of a traffic pattern and the degree of performance improvement by the proposed method (Sect. 4.4), aiming at versatile application to the third traffic patterns that are not evaluated in this paper. Prospects for practical (large-scale) configurations are discussed in the next section.

4.1 Sorting Order of Evaluation Metrics

In this paper, we use two evaluation metrics, i.e., duration time of collective communication and average number of hops (avg.hop). Every gene has its evaluation metrics and when determining survivors of genes, the genetic method sorts the genes by the evaluation metrics. As Sect. 2.2.2 describes, we use sc (duration first), ah (avg.hop first), and ml (multiplied value) methods.

In many other genetic applications, genes are sorted by the major evaluation function, i.e., duration time in this paper. Why we use the two evaluation metrics is that duration time is insufficient for estimating potentials of the genetic operation in future generations.

A long duration time comes from (a) long distance from the source node to its destination and (b) heavy congestion in the way to the destination. Thus, even if two source–destination node pairs show the same duration time, their distances may differ due to the difference of their congestion situations. Thus, we consider that the use of only one metric (duration time) is inappropriate and that the avg.hop is practical as the alternative metric.

Figure 11 shows performance (speed-up ratio) in terms of gene-selection methods. In this evaluation, we select the s00, s05, and s54 methods. In many cases, the ah method performs the best. This method selects genes that specifies every source-destination nodes within a short distance on average. Suppose that a path from a source node to its destination is short and packet latency is long, this means that the path is (partly) congested. Since the congestion arises from overlapping of other source-destination paths, if the overlapping of path is resolved, packet latency is drastically reduced.

We can understand that the preferable effect of the ah method is advantageous in the initial and intermittent steps in the repetitive genetic operations. When the genetic operation finds an appropriate solution, the solution has possibly the smallest avg.hop value as well as the smallest duration time.

4.2 Generation and Solution Quality

Repetitive operation is essential in the genetic methods. Figure 12 shows the history of the best duration time in some genetic operations. In the figure, the horizontal axis shows the generation number and the vertical axis shows the best
Fig. 13  Time-series of the best duration time for different sorting order (16 × 16 network, two-traffic-pattern, s54 method).

duration time as the result of the genetic operation.

Figure 13 also shows the time-series of the best duration time but in terms of the sorting order (Sect. 4.1). Intermittent steep curves show re-initialization of genes. Our method, except the random search, monitors the update history of the best metrics (i.e., duration time and avg.hop). When the best metrics are not updated for a given period of generations, the genetic method considers that the system falls in a local-minima situation and the method re-initializes the whole genes.

In Fig. 13 (a), the ml method finds the solution after re-initialization at about 450th generation. In Fig. 13 (b) shows a local-minima clearly. The sc method re-initializes genes at about 300th generation, but, it falls a local-minima situation in which the best duration time is not sufficiently small. By comparing the time-series curves, we can find that the ah method has preferable characteristic in rapid convergence to a good solution.

4.3 Population and Solution Quality

In this paper, we evaluate the genetic methods with 1,000 genes of populations, regardless of the system sizes. Roughly speaking, less genes result in poor solutions and, on the other hand, excessive number of genes waste computing power and time. Thus, we should clarify the relationship between population of genes and quality of solutions.

Figures 14 to 20 show our evaluation results. In these figures, the horizontal axis shows population of genes and the vertical axis shows speed-up ratio. Note that we use log scaling in x-axis. These figures reveal many things as follows.

The first point is that some genetic methods are sufficiently powerful for some specific traffic patterns regardless of population of genes. Concretely, Fig. 14 (a) shows that the curve of s53 is flat for population. This means that only a small number of genes are sufficient for the genetic method to solve brev traffic. Note that such small number of genes are not always sufficient for the method as Fig. 14 (b) shows.

As a natural estimation, the larger the system size be-
comes, the more genes are required for satisfactory solutions. This fact is confirmed by comparing corresponding combinations of traffic patterns depicted in Figs. 14, 17, and 20 for single-traffic-pattern cases, Figs. 15 and 18 for two-traffic-pattern, and Figs. 16 and 19 for three-traffic-pattern.

The s53 method requires only a small number of genes in small-scale systems, however, for large-scale systems, even the powerful genetic method requires as many genes as possible. Evaluation time length practically determines the population and generations. With respect to the time length, we will discuss further in Sect. 5.1.

During the repetitive genetic operations, scores of genes distribute and the distribution function forms a bell-like shape since the genes are randomly initialized and further modified with a certain level of randomness. Assume that the distribution function follows the normal distribution, by a rough approximation, with the average $\mu$ and standard deviation $\sigma$. According to the statistics theory, the ratio of genes whose scores are less than $\mu - 2\sigma$ is about $0.02275$ and $0.00135$ for $\mu - 3\sigma$. This means that $1/0.02275 \approx 43.95 = 44$ genes are required to find $2\sigma$ solution.
and \(1/0.00135=740.74\approx741\) genes for \(3r\) solution.

Figure 21 shows the distribution of 1,000 genes by the duration time score. The horizontal axis shows the duration time of genes and the vertical axis shows the number of genes that have the specific duration time. In this figure, \(s_{SS},g_{GG}\) indicates that the corresponding curve shows distribution of the \(s_{SS}\) method at \(GG\)-th generation. From this figure, we can find that the peak of the distribution moves leftward during the repetitive genetic operations. The moving speed of \(s_{54}\) is clearly faster than that of \(s_{05}\), as the \(s_{54}\) method achieves much better performance than \(s_{05}\).

In terms of population, the \(s_{54},g_{999}\) curve, i.e., the \(s_{54}\) method at \(999\)th generation, has some steep peaks and the highest peak appears at the smallest duration time. This fact means that many of genes in the the genetic method find the best solution at this time, however, the method does not fall in a local-minima situation as a whole. Thus, if the method employs more population, we can expect more chance to find better solutions. This consideration supports the gradient increase of speed-up shown in Fig. 18 (b).

Figure 21 suggests that some genetic methods find many different solutions that achieve the best performance (i.e., the shortest duration time). In the \(s_{05}\) method, which has not so strong capability, only a few genes have the best score even after 1,000 genetic operations are completed as Fig. 21 (a) shows as \(s_{05},g_{999}\). On the other hand, as Fig. 21 (b) shows, the \(s_{54}\) method has a steep peak at \(t = 167\) in duration time [cycles]. This peak illustrates that the method has many genes that mark the best score in the genetic system.

Figure 22 shows the number of different solutions in the \(8 \times 8\) network for various traffic patterns. During the repetitive genetic operations, two or more genes possibly have the same mapping information. To avoid duplicated counts of the best solutions, we introduced CRC-32 as a signature of mapping information. We regard that any two genes that have the same CRC-32 value have the identical mapping information. Blue bars in Fig. 22 show the net number of the best solutions and green bars (on top of the blue bars) show duplicated ones. Note that Fig. 22 includes the \(s_{05}\) and \(s_{99}\) results, which are too few to be clearly recognized in the figure.

### 4.4 Traffic Pattern and Solution Quality

As evaluation results shown in Figs. 6 to 8, achieved speed-up ratio by the genetic methods vary according to the traffic pattern. Concretely, in single-traffic-pattern cases, the method can find fully optimized mapping for \(bcmp,\ brev,\) and \(trns\) patterns. On the other hand, solutions for \(brot,\ shfl,\) and \(torn\) patterns are insufficient in performance.

These performance differences of the two groups of traffic patterns come from the degree of freedom in the fully optimized mapping. In the former group of traffic patterns, all of the packet communications are pairwise basis, i.e., a node \(A\) sends packets to another node \(B\) and \(B\) sends to \(A\). Thus, when the node \(A\) is mapped, \(B\) should be mapped in the neighborhood of \(A\), and the third node \(C\) has no constraint in its mapping location. This leads the genetic method to many synonyms of fully optimized solutions.

On the other hand, the latter group of traffic patterns (i.e., \(brot,\ shfl,\) and \(torn\)) include loop structures (cycles) that employ more than two members of nodes. For example, assume that the graph representation of a traffic pattern includes a directed cycle that is composed of nodes \(A, B, C,\ldots\) and so on. The node \(B\) should be a neighbor of \(A,\) furthermore, \(C\) should be placed near \(B\). This results in small degree of freedom in node mapping operations and the genetic operation has difficulty in finding fully optimized solutions for this class of traffic patterns.

To confirm our hypothesis that is discussed above, we introduce some artificial traffic patterns that intend different levels of degree of freedom in mapping operation. We assume that the directed graph of each traffic pattern consists of cycle(s) (ring: \(r\)) and pair(s) (\(p\)). Table 4 shows 15 traffic patterns, each of them is mixture of rings and
pairs. Two parameters, \( n_{\text{div}} \) and \( g_{\text{div}} \), specify the actual configuration of the traffic pattern, where the traffic pattern includes \( g_{\text{div}} \) rings whose size is \( N \times N/n_{\text{div}} \). Thus, remaining \( N \times N - N \times N/d_{\text{div}} \times g_{\text{div}} \) nodes are paired.

We further introduce ring(pair)-only configuration of traffic pattern, as Table 5 shows. In Tables 4 and 5, \( r \) and \( p \) represent ring and pair structures, respectively. For example, the ‘8*3, 2p*20’ configuration specifies that the traffic pattern employs three rings that consist of eight members and 20 pairs as its configuration. Members in each ring or pair are selected randomly in the following evaluations.

Figure 23 shows the speed-up ratios for the artificial traffic patterns in single-traffic-pattern condition. In these evaluations, we use the s53 method. In a small-scale network, as Fig. 23(a) shows the results in 8 \( \times \) 8 network, no significant difference is found. This result comes from high searching capability of our method.

On the other hand, in a larger-scale system, as Fig. 23(b) shows, significant difference in performance (speed-up ratio) is unveiled. By comparing \( r_{\text{par}}, r_{\text{rng}}, d_{\text{rng}}, q_{\text{rng}}, o_{\text{rng}}, x_{\text{rng}}, \) and \( z_{\text{rng}} \) (which are given in Table 5), we can recognize that higher degree of freedom marks higher performance (\( r_{\text{par}} \) and \( z_{\text{rng}} \)). Similar results are found in other combinations of traffic patterns: such as (\( cr_{01}, cr_{02}, cr_{03}, cr_{04}, cr_{05}, \) and \( cr_{06} \)), and (\( cr_{07}, cr_{08}, \ldots, cr_{14} \)).

Results that are drawn in Fig. 23 reveal rough relationship between the degree of freedom in mapping operation and performance speed-up ratio. For example, \( r_{\text{rng}}, d_{\text{rng}}, q_{\text{rng}}, o_{\text{rng}}, \) and \( x_{\text{rng}} \) do not show significant difference, and, similarly, combinations of \( cr_{07} \) to \( cr_{12} \) do not. Thus, at this time, precise estimation of performance improvement by the proposed method is limited for a new traffic pattern. The precise estimation is our future work.

### Table 4

Artificial traffic pattern: mixture of rings (\( r \)) and pairs (\( p \)).

<table>
<thead>
<tr>
<th>(Abb.-rev.)</th>
<th>( n_{\text{div}} )</th>
<th>( g_{\text{div}} )</th>
<th>8( \times )8 network</th>
<th>16( \times )16 network</th>
<th>32( \times )32 network</th>
</tr>
</thead>
<tbody>
<tr>
<td>cr00</td>
<td>2</td>
<td>1</td>
<td>32r<em>1, 2p</em>16</td>
<td>128r<em>1, 2p</em>64</td>
<td>512r<em>1, 2p</em>256</td>
</tr>
<tr>
<td>cr01</td>
<td>4</td>
<td>1</td>
<td>16r<em>1, 2p</em>24</td>
<td>64r<em>1, 2p</em>96</td>
<td>256r<em>1, 2p</em>384</td>
</tr>
<tr>
<td>cr02</td>
<td>4</td>
<td>3</td>
<td>16r<em>3, 2p</em>8</td>
<td>64r<em>3, 2p</em>32</td>
<td>256r<em>3, 2p</em>128</td>
</tr>
<tr>
<td>cr03</td>
<td>8</td>
<td>1</td>
<td>8r<em>1, 2p</em>28</td>
<td>32r<em>1, 2p</em>112</td>
<td>128r<em>1, 2p</em>448</td>
</tr>
<tr>
<td>cr04</td>
<td>8</td>
<td>3</td>
<td>8r<em>3, 2p</em>20</td>
<td>32r<em>3, 2p</em>90</td>
<td>128r<em>3, 2p</em>320</td>
</tr>
<tr>
<td>cr05</td>
<td>8</td>
<td>5</td>
<td>8r<em>5, 2p</em>12</td>
<td>32r<em>5, 2p</em>48</td>
<td>128r<em>5, 2p</em>192</td>
</tr>
<tr>
<td>cr06</td>
<td>8</td>
<td>7</td>
<td>8r<em>7, 2p</em>4</td>
<td>32r<em>7, 2p</em>16</td>
<td>128r<em>7, 2p</em>64</td>
</tr>
<tr>
<td>cr07</td>
<td>16</td>
<td>1</td>
<td>4r<em>1, 2p</em>30</td>
<td>16r<em>1, 2p</em>120</td>
<td>64r<em>1, 2p</em>480</td>
</tr>
<tr>
<td>cr08</td>
<td>16</td>
<td>3</td>
<td>4r<em>3, 2p</em>26</td>
<td>16r<em>3, 2p</em>104</td>
<td>64r<em>3, 2p</em>416</td>
</tr>
<tr>
<td>cr09</td>
<td>16</td>
<td>5</td>
<td>4r<em>5, 2p</em>22</td>
<td>16r<em>5, 2p</em>88</td>
<td>64r<em>5, 2p</em>352</td>
</tr>
<tr>
<td>cr10</td>
<td>16</td>
<td>7</td>
<td>4r<em>7, 2p</em>18</td>
<td>16r<em>7, 2p</em>72</td>
<td>64r<em>7, 2p</em>288</td>
</tr>
<tr>
<td>cr11</td>
<td>16</td>
<td>9</td>
<td>4r<em>9, 2p</em>14</td>
<td>16r<em>9, 2p</em>56</td>
<td>64r<em>9, 2p</em>224</td>
</tr>
<tr>
<td>cr12</td>
<td>16</td>
<td>11</td>
<td>4r<em>11, 2p</em>10</td>
<td>16r<em>11, 2p</em>40</td>
<td>64r<em>11, 2p</em>160</td>
</tr>
<tr>
<td>cr13</td>
<td>16</td>
<td>13</td>
<td>4r<em>13, 2p</em>6</td>
<td>16r<em>13, 2p</em>24</td>
<td>64r<em>13, 2p</em>96</td>
</tr>
<tr>
<td>cr14</td>
<td>16</td>
<td>15</td>
<td>4r<em>15, 2p</em>2</td>
<td>16r<em>15, 2p</em>8</td>
<td>64r<em>15, 2p</em>32</td>
</tr>
</tbody>
</table>

### Table 5

Ring- and pair-only artificial traffic patterns.

<table>
<thead>
<tr>
<th>(Abb.-rev.)</th>
<th>description</th>
<th>8( \times )8</th>
<th>16( \times )16</th>
<th>32( \times )32</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpar</td>
<td>random pair only</td>
<td>2p*32</td>
<td>2p*128</td>
<td>2p*512</td>
</tr>
<tr>
<td>rrng</td>
<td>one large ring</td>
<td>64r*1</td>
<td>256r*1</td>
<td>1024r*1</td>
</tr>
<tr>
<td>drng</td>
<td>two rings</td>
<td>32r*2</td>
<td>128r*2</td>
<td>512r*2</td>
</tr>
<tr>
<td>qrng</td>
<td>four rings</td>
<td>16r*4</td>
<td>64r*4</td>
<td>256r*4</td>
</tr>
<tr>
<td>orng</td>
<td>eight rings</td>
<td>8r*8</td>
<td>32r*8</td>
<td>128r*8</td>
</tr>
<tr>
<td>xrrng</td>
<td>16 rings</td>
<td>4r*16</td>
<td>16r*16</td>
<td>64r*16</td>
</tr>
<tr>
<td>zrrng</td>
<td>4-member rings</td>
<td>4r*16</td>
<td>4r*64</td>
<td>4r*256</td>
</tr>
</tbody>
</table>

Figure 24 shows the time-series curves of duration time in some artificial traffic patterns (16 \( \times \) 16 network).

As we clarified in Sect. 4.3, the larger the size of network grows, the larger population is required for good solutions. This shows that we should pay extremely high costs for large-scale networks.

As a preliminary evaluation, we measured execution times of the genetic operations. Figure 25 shows the results in single-pattern-traffic cases of the s50 genetic method. The evaluation environment is as follows: Intel Core i7-7700 CPU (3.60GHz), Ubuntu 16.04.4 LTS operating system, and gcc version 5.4.0.

This figure shows that the required elapsed time exceeds the linear relation by the network size. For example, the elapsed time of 16 \( \times \) 16 network (\( t_{\text{el}}(16\times16) \)) exceeds four times of that of 8 \( \times \) 8 (\( t_{\text{el}}(8\times8) >> 4 \times t_{\text{el}}(8\times8) \)).
methods of computational complexity in solving large-scale problems. This section discusses from two different approaches: lightweight evaluation function and reuse.

5.1 Reduction of Evaluation Cost

The fundamental objective of this research is to increase performance of collective communication. The performance is measured as a duration time, thus, we cannot ignore the duration time in evaluating individual gene. The largest problem in the evaluation cost is that the duration time is measured only by simulation. In this paper, we use our simulator that is based on the cellular-automata principle [7]. Although the simulator performs high-speed, the performance is not sufficient for the purpose of this paper, since, in large-scale networks, simulation time becomes dominant in our method. Furthermore, evaluations of genes are repeated millions of times (in case of 1,000 generations of 1,000 genes). Thus, we should discuss alternative methods to reduce simulation steps.

In Sect. 4.1, we discussed about the sorting order of the performance metrics (i.e., duration time and avg.hops), although, the relationship between the two metrics is not clarified at this time. As a simple estimation, when a node-mapping \( x^t \) performs short duration time, we can expect short distance between source–destination pair on average, and vice versa.

Figure 26 shows the relationship of the two metrics in \( \text{brev} \) and \( \text{shfl} \) traffic patterns in the \( 16 \times 16 \) network. In this evaluation, the simulator records all intermittent results whenever it finds the best duration time and avg.hops. This figure also shows linear lines for the two traffic patterns as \( \text{brev}(x) \) and \( \text{shfl}(x) \), which are determined by the least square method.

This figure illustrates that the avg.hop metric shows communication performance (measured by duration time) regardless of the actual genetic operations (s00, s05, s99, and s53). These preliminary results suggest that the avg.hop metric is possibly enough for the evaluation function in genetic operations, except that the relationship deviates in a low avg.hop region in which the avg.hop is about less than 1. Thus, this suggests that the avg.hop metric works well as the evaluation function of the genetic operation, but it does not guarantee the same levels of solutions from the full genetic operation (a\( h \)). Thus, we should evaluate the acceleration methods from the viewpoints of execution speed of the total genetic operations and quality of the results.

Recall that the avg.hop metric is calculated based on the mapping vector \( x^t \) and the given traffic pattern. On the other hand, the duration time metric requires a simulation run for every mapping vector \( x^t \). Although the simulation performance is considerably accelerated [7], computational cost of a simulation run is much higher than that of avg.hop. Thus, we can expect that the computational costs in the genetic operation, specifically in the evaluation functions, is reduced, if the simulation run is omitted or reduced.

Furthermore, we introduce the third metric, the path overlapping factor, that shows the average number of paths that go through individual physical links. We can expect that the path overlapping factor represents the degree of possible contentions on average. If this value is high, heavy packet contention is estimated in the traffic pattern under the mapping \( x^t \). On the other hand, when the value is low, we can expect smooth traffic.

We introduce further genetic operations as follows for an experimental purpose.

- Sort only by avg.hop with duration time evaluation in every 10 generations (ho).
- Sort only by avg.hop with duration time evaluation at the last generation (hl).
- Sort by avg.hop and path overlapping factor with duration time evaluation in every 100 generations (hw).

As a rough estimation, if the simulation run is the dominant factor in the evaluation function, the ho method will accelerate the execution time by 10 times, and also hl and hw methods will achieve 100 or more times faster.

Figure 27 shows the acceleration results in single- and two-traffic-pattern cases in \( 8 \times 8 \), \( 16 \times 16 \), and \( 32 \times 32 \) networks. In this figure, the speed-up ratio is calculated by dividing the elapsed time of the a\( h \) method by the corresponding elapsed time of the ho, hl, or hw. The number of generation in the genetic operation is 1,000.

As Fig. 27(a) shows, the newly introduced methods accelerate the genetic operation. In the \( 8 \times 8 \) network configuration, the ho method stably performs at about 10

---

**Fig. 25** Elapsed time for 1,000 generations of genetic operations.

**Fig. 26** Relation ship between best avg.hop and best duration time.
times speed-up. The hl method achieves about 100 times speed-up in some single-traffic-pattern cases (in brot, shfl, and torn patterns). Performance of the hw method is slightly lower than that of hl.

Figure 28 shows the quality of the results by the acceleration methods (ho, hl, and hw). The vertical axis shows the relative performance from the ah method. An error-bar on top of each bar shows the maximal and minimal scores in the measured samples. This figure shows satisfactory quality in the acceleration methods.

The acceleration methods vary their speed-up ratios in the traffic pattern and the network size. It is undesirable that speed-up in the large-scale network is small, as Fig. 27 (c) shows, since the speed-up ratios are obviously smaller than those of smaller networks. The possible reason of the curious result is the number of retries in the enhanced neighboring swap. As Sect. 3.1 describes, the enhanced neighboring swap method selects appropriate candidate of pairwise exchange. If the selection is inappropriate, the method re-selects another candidate until it finds satisfactory one. Some traffic patterns require large numbers of retries (re-selections) during the operation. Figure 29 shows the number of retries in the ho, hl, and hw methods. The figure shows some specific traffic patterns (brot, shfl, and torn) have much less retries than those of other traffic patterns. These results suggest the obstacles to speed-up, which are our future work.

5.2 Reuse of Other Execution Results

The other viewpoint for improving performance in large-scale networks is rapid convergence to a preferable result. Our idea is that the (quasi-)optimized results in single-traffic-pattern are effective as the initial genes in multiple-traffic-pattern cases. For example, we can expect that the proposed method finds considerably good solutions for the bcmp traffic even in a large-scale network. We can furthermore expect that some of the solutions work well for another traffic pattern, brot for example, as well as bcmp. Thus, when the results of the single-traffic-pattern are used as initial genes for multiple-traffic-pattern problems, bcmp–brot for example, they will support rapid convergence to good solutions. We call the method reuse method.

As a preliminary evaluation, we use the s50 method
and acquire as many as possible of single-traffic-pattern results as candidate genes for the two-traffic-pattern problem in advance. In the evaluation, at most 10,000 candidates are used for each traffic pattern. The method firstly measures the duration time for each candidate, then, it selects the top-1000 ones as the initial genes for the two-traffic-pattern problem. If a traffic pattern has less than 1,000 candidates, randomly generated genes are used instead. Figure 30 shows the speed-up ratios of the reuse method (denoted by \( \text{re} \)) and the \( \text{ah} \) method in two-traffic-pattern cases. This figure illustrates that the reuse method acquires better results than those of the \( \text{ah} \) method in many combinations of traffic patterns.

Figure 31 shows the time-series of the best duration time in \( \text{bcmp-brot} \) traffic pattern. As this figure shows, the reuse method (\( \text{re} \)) improves the initial performance and the method converges the result at an early stage of the generations. It sometimes achieves better performance than that of \( \text{ah} \), as Fig. 30 shows, however, its total performance improvement remains comparable. Discussions for further improvements are our future work.

6. Related Work

6.1 Topology Mapping

In the decades of history in parallel processing, efficient execution of parallel application programs on parallel computers has been a crucial problem. Inter-node communication is inevitable in parallel programs and it forms a certain topology that is represented by a (directed) graph. On the other hand, as a parallel computer is organized by computing nodes and links, it forms another (directed) graph. If both two graphs match to each other, the parallel application program runs efficiently on the parallel computer with no communication overheads, and otherwise, the net performance is degraded.

This motivates many studies to discuss efficient methods that can match the two graphs so that a parallel program runs at a certain level of effectiveness. Early studies have discussed the problem from mathematical viewpoints. Saad et al. discussed the topological characteristics of hypercubes [8]. Wu et al. discussed embedability of various application topologies (such as trees) onto the hypercube topology that corresponds to real parallel machines [9].

Bokhari formalized the mapping problem and proposed a mapping method based on the pairwise exchange [4]. This work formalizes the mapping problem as a one-to-one mapping of a logical node in a problem \( V_p \) onto the machine’s physical node \( V_a \) (the subscript \( a \) stands for ‘array’). The work assumes that every node in the \( V_p \) graph corresponds to an appropriate one in \( V_a \), and that any edge that connects two \( V_p \) nodes is non-directed one (i.e., bidirectional communication). As a metric of the mapping result, this work introduces cardinality that counts the number of edges in the \( G_p \) that is mapped between two adjacent vertices in \( G_a \).

Ranka et al. proposed algorithms that map n-dimensional meshes on a star graph [10]. In this work, they introduced expansion, dilation, and congestion metrics and showed that an n-degree star graph efficiently simulate an n-dimensional mesh. Lee et al. reported a practical method for evaluating objective functions by introducing the weight idea so that it can estimate communication overheads [11]. This work reports evaluation results on 8- and 16-node hypercubes. Ibarra et al. reported experimental results of existing mapping methods applied to representative systolic algorithms [12]. The experiments run on a 64-node NCUBE/7 machine. Work by Kavianpour et al. is based on the realistic assumption that a problem graph does not match to an array graph [13]. The work introduces dilation-bound and expansion ratio parameters to quantitatively evaluate the mapping results. Chaudhary et al. placed a wide survey on the mapping algorithms and proposed a generalized algorithm for mapping [14]. The work introduces extended host graph to represent a mapping solution of a problem graph. It also introduces modified pairwise exchange.

Many of the early studies placed formal discussions on topological mapping of non-directed graphs. However, a practical application of a mapping method requires realistic solutions that can resolve substantive problems that include communication overheads.

Hou et al. assumes a generalized communication pattern as LCC (Linear-Complement Communication) that is a superset of the traffic patterns used in this paper [15]. The work claims that the channel contention is one of the largest keys in the performance issue of LCC. To minimize the channel contention, the work introduces processor reordering (i.e., mapping) that is determined at the compile time. The proposed algorithm requires \( O(n^3) \) of computation com-
plexity and evaluation results on 8-dimensional hypercube are reported. The fundamental idea of this work is close to that of this paper, however, actual approach is different.

Hoefler et al. showed that closeness between the problem graph and array graph affects quality of the mapping results by applying many existing methods [16]. This result is intuitively understandable, however the work quantitatively reveals that the benefit is large in large-scale networks: the proposed method reduces congestion (that is calculated by the number of paths that goes through a link) by 80% and also reduce dilation by 50%.

This paper is unique in the underlying assumption of communication traffic. This paper assumes collective communication and it considers different types of collective communications in different traffic patterns in order as well as a single collective communication. As many of studies on the mapping problem are based on pairwise exchange, this paper is also based on the pairwise exchange in principle, although this paper proposes strong heuristics that is employed in genetic operations.

6.2 Node Mapping in NoC

After NoC (Network-on-Chip) has emerged, many studies are reported for solving the NoC-specific problems. One of the remarkable feature in the NoC architectures is application specific, while the traditional parallel computation studies are based on general parallel applications. Practical problems in the NoC architecture include limitation of link bandwidth in the NoC. As the remarkable difference from parallel computation, a typical NoC architecture is designed for a specific application. Sahu et al. placed a wide survey on the mapping problems on NoC [17].

Murali et al. proposed mapping methods (minimum-path routing and traffic splitting) for NoC architectures by considering bandwidth requirements from the application [18]. Five-node DSP applications are evaluated by cycle accurate simulation by SystemC. Hu et al. proposed an energy- and performance-aware mapping method under the assumption of the NoC architecture in which computing nodes are regularly placed (typically in mesh topology) [19].

Bambha et al. proposed a synthesis method to obtain the best topology that matches an NoC application [20]. The proposed method employs a genetic algorithm that includes mutation and crossover, based on the traditional pairwise exchange. This work reports evaluation results on some small-scale problems (from four to eight nodes) with small population (100 genes).

Chen et al. proposed a mapping method that maps a parallel application onto a specific NoC architecture (MPSoC) [21]. The proposed method runs (1) task scheduling, (2) processor mapping, (3) data mapping, and (4) packet routing in order.

Maqsood et al. focuses on the congestion in communication [22]. This work introduces improved betweenness cardinality. By considering the amount of the shared data between source and destination nodes of communication, the method increases the accuracy in estimating the degree of congestion. The shared data correspond to the volume of the inter-node communication. The method does not mention routing algorithms.

Wide diversity in NoC applications poses discussions on wide range of problems. Wang et al. discussed mapping methods of IP cores on NoC architectures [23]. Chen et al. discussed their RAIR (Region-Aware Interference Reduction) under the assumption that multiple applications run in the separated regions in a NoC architecture [24]. The proposed method reduces interference between adjacent regions. Das et al. discussed the memory access interferences between multiple different applications that run in a NoC architecture [25]. Different from the work of Chen et al., this work focuses on memory access behaviors. Namazi et al. discussed mapping methods for voting architectures at a fault-tolerance viewpoint [26].

Many of NoC approaches commonly assume embedded applications, which are typically media applications that include MPEG-4 and H.264. These applications have coarse-grained parallelism and bandwidth of inter-node communication is not balanced. The underlying assumptions are totally different from those in this paper.

6.3 Mapping in Virtual Network

Virtual networks also require to discuss on the node-mapping problem as Butt et al. placed [27]. Cheng et al. discussed the mapping problem under the assumption that multiple VNs are simultaneously in operation [28]. This assumption is close to that of this paper, however, our assumption is distinct in sequential use of traffic patterns.

6.4 Evolutional Methods

As the absolute performance of computers increases, recent studies employ evolvable methods such as simulated annealing and genetic algorithms (GA). Bambha et al. [20] extends their methods from traditional pairwise exchange to genetic algorithms. Nedjah et al. reports experimental results of mapping methods with some GA tools [29]. Jang et al. reported a MIQP (Mixed Integer Quadratic Programming) based method named A3MAP-GA (Architecture-Aware Analytic MAPping, [30]). The GA operation in the method is crossover based on mutation matrix of nodes (that corresponds to pairwise exchange). The work evaluates typical embedded applications such as MPEG4 VOPD. Radu et al. showed the EGA (Energy- and performance-aware Genetic Algorithm) method that is based on GGA (General GA, [31]). Tosun et al. proposed a mapping method based on simulated annealing and genetic algorithm [32]. The work reports evaluation results on MP4 and some artificial applications on 4 × 4 NoC.

It is natural to employ evolitional methods to solve complicated problems as the related work does. This paper also introduces genetic algorithm to solve the node-mapping problem. As described in Sect.6.1, this paper is unique in
strong heuristics and underlying communication conditions.

6.5 Reduction of Computing Time in Mapping

Evolutional methods require large resources in computation power and run time, in general. Thus, some efforts are reported, in which computation time is reduced. Lu et al. proposed a method that simplifies the structure of the (physical) parallel computer to that the mapping method reduces computational costs. The proposed method simply divides the system into corner-, edge-, and center-clusters [33] (‘cluster’ corresponds to ‘region’ here). This simplification can drastically reduce search space. Yang et al. posed a different viewpoint [34]. The work discusses the optimized temperatures in simulated annealing methods of node-mapping.

Acceleration of the genetic operation is one of the most crucial problems for the proposed method to be applied to large-scale systems. We discussed the problem in Sect. 5 and the results in the section suggest some hints on evaluation metrics and initialization of genes, however, the study is still underway. The results in Sect. 4.3 shows that large-scale systems require more population of genes for satisfactory results. This means that extremely large-scale systems require unrealistic (long) computation time to solve. We will discuss strong heuristics that work more efficiently than the enhanced neighboring swap method as our future work.

6.6 Rearrangement of Data Location

Some data distribution methods are discussed from the viewpoint of efficient communication. As High-Performance Fortran (HPF) supports a certain level of data distribution in parallel execution, literature [35] and [36] discusses data distribution methods that are based on an idea similar to that in this paper.

6.7 Different Approaches

Regularity in the interconnection architecture sometimes obstacles the communication performance. Some studies report improved performance by breaking the regularity at a certain level. Singh et al. proposed a packet routing method GOAL (Globally Oblivious Adaptive Locally) that is a derivation of oblivious routing [37]. The core idea is detour routing to break the regularity in the traffic pattern. Koibuchi and Fujiwara et al. reported topological approach in breaking the regularity for performance improvement in interconnection networks [38], [39]. Zhu et al. showed a different approach that balances the latencies in the packet delivery under the simultaneous execution situation of multiple applications [40].

7. Conclusion

This paper addresses the node-mapping problem for the parallel application programs that include one or more collective communications in different traffic patterns. This paper inherits the pairwise exchange to find an appropriate mapping results as a fundamental operation, however, this paper proposes strong heuristics based on the pairwise exchange. Furthermore, this paper introduces genetic algorithms as a fundamental search strategy.

The major contributions of this paper are not only proposing the new heuristic methods and their evaluation results but also discussing from wide range of the node-mapping methods. Furthermore, this paper discusses applicability and problems for large-scale systems. The results in this paper reveal that the proposed methods achieve considerable performance for small-scale systems, 8.9 (4.9) times speed-up on average in single-(two-)traffic-pattern cases in 32 × 32 networks, however, large-scale systems require long computation time for satisfactory results, which is our future work.

As this paper shows fundamental effectiveness in the 2D-torus networks and also shows essential characteristics and behaviors, the proposed method is hopeful to adopt to other classes of interconnection networks. Since the method is based on the distance of source–destination nodes of packets, we believe that the method is applicable to many types of networks on the condition that any packet is relayed by nodes (or routers) until it reaches at the destination node. However, this paper does not necessarily guarantee the absolute effectiveness in the wide applications. Thus, wide applicability of the proposed method is also our future work.

Acknowledgments

This work was partly supported by JSPS KAKENHI Grant Numbers 16K00068 and 17K00072.

References


Takashi Yokota received his B.E., M.E., and Ph.D. degrees from Keio University in 1983, 1985 and 1997, respectively. He joined Mitsubishi Electric Corp. in 1985, and was engaged in several research projects in special-purpose, massively parallel and industrial computers. He was engaged in research and development of a massively parallel computer RWC-1 at Real World Computing Partnership as a senior researcher from 1993 to 1997. From 2001 to 2009, he was an associate professor at Utsunomiya University. Since 2009, he has been a professor at Utsunomiya University. His research interests include computer architecture, parallel processing, network architecture and design automation. He is a member of IPSJ, IEICE, ACM, and the IEEE Computer Society.

Kanemitsu Ootsu received his B.S. and M.S. degrees from University of Tokyo in 1993 and 1995 respectively, and later he obtained his Ph.D. in Information Science and Technology from University of Tokyo in Japan. From 1997 to 2009, he is a research associate and then an assistant professor at Utsunomiya University. Since 2009, he is an associate professor at Utsunomiya University. His research interests include high-performance computer architecture, multi-core multithread processor architecture, mobile computing devices, binary translation, and dynamic optimization. He is a member of IPSJ, IEICE, and ISCIE.

Takeshi Ohkawa received his B.E., M.E. and Ph.D. degrees in Electronics Engineering from Tohoku University in 1998, 2000 and 2003, respectively. He was a postdoctoral fellow at Tohoku University, where he engaged in the reconfigurable LSI and software/hardware co-design in 2003. In 2004, he joined National Institute for Advanced Industrial Science and Technology (AIST), Japan as a researcher, where he started working on distributed object technology and its application to FPGA. From 2009, he started work in start-up company TOPS systems, where he engaged in R&D of a heterogeneous multi-core processor and its software platform. From 2011, he was an assistant professor at Utsunomiya University. Since 2019, he is an associate professor at Tokai University. His research interests include reconfigurable technology, software/hardware co-design, parallel and distributed architecture and component-based design technology. He is a member of ACM, IEEE, IEICE, and IPSJ.