IEICE TRANS. INF. & SYST., VOL.E106–D, NO.11 NOVEMBER 2023

PAPER

Enhancing Cup-Stacking Method for Collective Communication

Takashi YOKOTA[†a)], **Kanemitsu OOTSU**[†], ***and*** **Shun KOJIMA**[†*], ***Members***

SUMMARY An interconnection network is an inevitable component for constructing parallel computers. It connects computation nodes so that the nodes can communicate with each other. As a parallel computation essentially requires inter-node communication according to a parallel algorithm, the interconnection network plays an important role in terms of communication performance. This paper focuses on the collective communication that is frequently performed in parallel computation and this paper addresses the Cup-Stacking method that is proposed in our preceding work. The key issues of the method are splitting a large packet into slices, re-shaping the slice, and stacking the slices, in a genetic algorithm (GA) manner. This paper discusses extending the Cup-Stacking method by introducing additional items (genes) and proposes the extended Cup-Stacking method. Furthermore, this paper places comprehensive discussions on the drawbacks and further optimization of the method. Evaluation results reveal the effectiveness of the extended method, where the proposed method achieves at most seven percent improvement in duration time over the former Cup-Stacking method.

key words: *interconnection networks, parallel computers, collective communication, packet splitting, packet scheduling, genetic algorithms*

1. Introduction

A parallel computer consists of (many) computation nodes that are connected by an interconnection network [1], [2]. As parallel computing essentially requires communications between the computation nodes and the communication performance largely affects (or sometimes dominates) the total performance of the parallel computation, the interconnection network is an inevitable component.

A typical interconnection network of today's parallel computer forms as an aggregate of routers (or switches) that are connected by (physical) links to each other. In an interconnection network, a router receives a packet from the corresponding node, the packet is transferred in a router-to-router fashion according to the routing algorithm, and, finally the packet arrives at the destination node. Actually, many packets simultaneously move in the network and this raises many research issues.

In many interconnection networks, each router executes the packet-transfer operations individually and no central control is applied. Many parallel computers assume lossless packet switching, where no packets are dropped even when the network is severely congested. To support the assumption of the lossless network, the interconnection network employs buffers that keep up the incoming packets to absorb the temporal conflict of packets.

However, when the buffer becomes full, it blocks further incoming flow of packets for avoiding packet drops, and the corresponding packet transfer is suspended. This packet blocking sometimes causes further blocking like a chain-reaction and the blocked situation propagates over the network. We recognize the situation as congestion.

A congested situation spreads fast and it sustains long, in general. Once the congested situation covers the network, the communication performance is drastically degraded. Thus, as a countermeasure to the congestion, efficient congestion control is a crucial issue in discussing the interconnection network methods as well as high-speed and high-performance networks.

This paper focuses on the congestion control issue from a unique viewpoint of Cup-Stacking [3]. This paper assumes packet-switching communication in a regular network (2D-torus) that employs a deterministic routing algorithm. This paper specifically discusses collective communication performance and takes a static approach where the traffic pattern is determined beforehand. This paper comprehensively discusses possible extensions of the Cup-Stacking principle for obtaining better congestion control methods.

The rest of this paper is organized as follows. After Sect. 2 overviews related work, Sect. 3 states our previous work, the Cup-Stacking method. Then, this paper starts preliminary discussions to extend the method in Sect. 4, and proposes the extended Cup-Stacking method in Sect. 5. This paper furthermore discusses two different aspects of drawbacks and improvements. Section 6 deals with the computational complexity issue to reduce the evaluation costs. Section 7 investigates some ideas for optimal solutions. After that, overall discussions and future work are placed in Sect. 8. Then, Sect. 9 concludes this paper. Appendix A shows the common evaluation environment and conditions.

2. Related Work

The start-point of this work is to relax network congestion for accelerating communication performance. In a dense communication situation, a local conflict of packets causes a temporal block of the packet transfer, and, sometimes, the blocked packet fills up a packet buffer and causes another suspension of packet transfer. When the blocking and suspension spread in the network, as Pfister et al. stated as *tree*

Manuscript received October 4, 2022.
Manuscript revised June 15, 2023.
Manuscript publicized August 22, 2023.
[†]The authors are with Utsunomiya University, Utsunomiya-shi, 321–8585 Japan.
[*]Presently, the author is with the University of Tokyo.
a) E-mail: yokota@is.utsunomiya-u.ac.jp
DOI: 10.1587/transinf.2022EDP7179

saturation [4], heavy congestion appears and the communication capability of the network is drastically degraded.

With respect to congestion control in the interconnection network, some static/dynamic approaches are discussed toward smooth communication flows in the network. One of the typical approaches is *throttling* [5] that restricts new packet injection in a crowded situation. Another approach is *pacing* [6] that intends to suppress a crowded situation by inserting a sufficient interval of packet injection. Takano et al. also reported a pacing technique in an MPI communication in a Grid environment [7].

Timing of packet transfer is one of the crucial concerns for efficient network communication. Morie et al. discuss MPI rank optimization from a viewpoint of conflict-avoidance [8]. In optical and wireless systems, efficient communication methods are discussed with respect to low power consumption [9]–[12]. However, their objectives do not match our issue.

Agarwal has mentioned in literature [13]: *"We believe higher throughput might be achieved with splitting if some delay is introduced between sending the submessages at the source, or by randomizing the routes taken by the submessages."* As this message suggests, we can now find some studies in terms of packet splitting. Matsuda et al. [14] presented an MPI collective communication in a high-speed WAN environment. They split a message to some sub-messages so that the sub-messages can be transferred in different paths in the WAN. Since the communication distributes over the network resources, efficient packet transfer can be expected. Tu et al. also presented a method that split messages in MPI communication [15], however, their objective is to fit the (sub-)message to the L1 cache in a SMP multiprocessor system.

With respect to a *delay* idea, we have presented a kind of packet scheduling method by introducing a meta-heuristic method of particle swarm optimization (PSO) [16]. However, this study deals only with an optimal schedule of packet injection and no idea of packet splitting nor re-shaping is placed.

The core ideas of the Cup-Stacking are (1) splitting a large packet into multiple slices, (2) re-shaping the slice so that the effective thickness of the slice becomes sufficiently thin, and, (3) launching the successive slices with a short interval. No similar studies are reported so far, until our preceding paper [3], and this paper extends the fundamental idea of the original Cup-Stacking.

3. Cup-Stacking Method

Literature [3] presents our preliminary work on the Cup-Stacking method. The method initially comes from our observation results of the network behaviors in some collective communication in a two-dimensional torus network, where each node injects a packet that destines a certain node according to the given traffic pattern. When the payload of a packet is large, the network is heavily congested and the duration time becomes extremely long (Fig. 1 (a)).

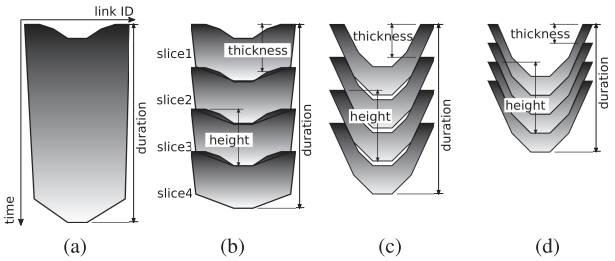


Fig. 1 Abstract model of Cup-Stacking concept.

Our previous work found that the duration time could be shortened when every packet is split (into slices) and the slices are injected with an appropriate interval (Fig. 1 (b)). This situation illustrates the network behavior as cup-stacking. If the cup is appropriately formed, cups can be stacked densely (Fig. 1 (c)).

Inspired by this characteristic, we have reached the Cup-Stacking method. The novel method forms the slice communication through Genetic Algorithm (GA) so that the cups are stacked densely and the total duration time is drastically reduced (Fig. 1 (d)).

As our previous work [3] reported, the proposed method considerably reduces the duration time of collective communication in some traffic patterns, however, discussions on the preliminary results suggest some room for further improvement. So, this paper comprehensively investigates and extends the original Cup-Stacking method.

4. Different Virtual Channel Arbitration

The previous paper concentrates on obtaining an ideal form of *cup* (i.e., slice traffic) by re-shaping, however, it does not touch internal functions of routers. This paper firstly thinks of an arbitration method of packets, since literature [17] shows that a different arbitration method conducts the network to a different behavior that results in different performance scores.

An arbitration method allows incoming packets to a router to go out in a different order. When packets A and B race for the same resource (e.g., an output port), an arbitration method selects A, for example, while an alternative one firstly selects B instead of A.

We assume a simple (non-pipelined) router whose organization is illustrated in Fig. 2 ([18]). In the router model, the arbitration task in a router consists of two steps: *queue selection* and *virtual channel (VC) selection*. In the queue selection, every input queue whose queue-top is a header flit sends a transfer request to an appropriate VC (in an output port) that is determined by the routing algorithm. Then, if the requested output VC is idle, the arbitration grants one of the transfer requests. Note that the queue selection is performed in a channel-by-channel fashion when multiple virtual channels have valid transfer requests. The queue selection corresponds to the central crossbar-switch in Fig. 2.

In the VC selection, when an output port is requested for multiple virtual channels simultaneously, the port selects

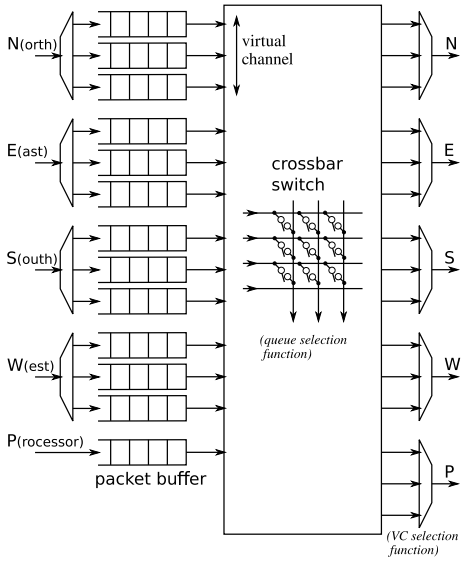


Fig. 2 Router model.

Table 1 Variants in channel selection arbitration.

symbol	category	start from	evaluation order
C00	round-robin	previous channel	ascending
C01	round-robin	next channel	ascending
C02	round-robin	previous channel	descending
C03	round-robin	next channel	descending
C04	fixed priority	channel 0	ascending
C05	fixed priority	channel 1	ascending
C06	fixed priority	channel 2	ascending
C07	fixed priority	channel 0	descending
C08	fixed priority	channel 1	descending
C09	fixed priority	channel 2	descending

one of the requested channels. The VC selection corresponds to the multiplexer in every output port that is depicted in Fig. 2.

As the first-step discussion in this paper, we introduce variance in arbitration in the VC selection phase. Actually, we prepare ten arbitration schemes that are categorized as round-robin and fixed priority. A round-robin scheme searches from the previously selected (or next) channel in the descending (or ascending) order. A fixed priority one starts from a certain channel (0, 1, or 2) in the descending (or ascending) order. Table 1 summarizes the schemes with associated symbols C00–C09.

4.1 Evaluation by Simple Stacking

We evaluated the effects of the arbitration schemes listed in Table 1 on the collective communication performance. For every arbitration scheme in Table 1, we measured duration times by varying the inter-slice interval. We call the evaluation *simple stacking*. In this evaluation, the injection timing of individual packets is not arranged, i.e., all the packets are injected in unison in every slice. Evaluation environment and conditions are summarized in Appendix A.

Figure 3 shows the results in the following condition: bit-rotation (brot) traffic pattern, two slices each of eight-flit

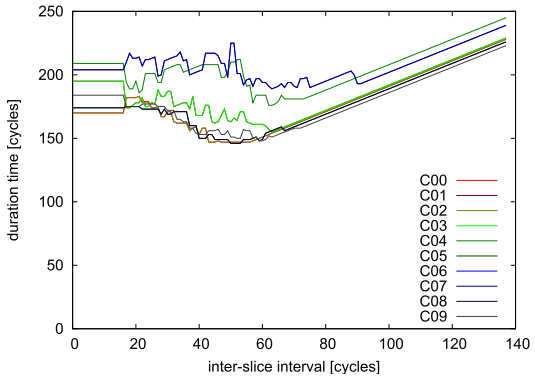


Fig. 3 Changes in duration times for inter-slice intervals. (16×16 2D-torus, brot traffic, two slices, 8-flit packets)

Table 2 Shortest duration times in simple stacking. (16×16 2D-torus, two slices, 8-flit packets)

	C00	C01	C02	C03	C04	C05	C06	C07	C08	C09
bcmp	104	102	104	102	104	91	101	101	91	91
brev	179	194	179	194	172	208	173	173	208	208
brot	147	155	147	155	176	146	189	189	146	148
shfl	152	154	152	154	161	147	159	159	147	150
torn	131	131	131	131	131	138	131	131	138	138
trns	138	138	138	138	138	137	138	138	137	137

[cycles]

packet size, and a 16×16 two-dimensional torus network. Although some of the arbitration schemes show an identical curve, we can find considerable variants in arbitration schemes.

Table 2 summarizes the shortest duration times for every arbitration scheme (C00–C09) in every traffic pattern. Other evaluation conditions are the same with those in Fig. 3. Underlined values show the shortest duration times in each traffic pattern. The underlined results are also summarized in the third column in Table 4[†]. We can find that the selection of an arbitration scheme is an alternative and important key to improve the cup-stacking method. The first and second columns in Table 4 show the duration times in non-split cases and results in our previous paper [3], respectively. Parenthesized values are shown for comparison purposes.

4.2 Evaluation with Timing Adjustment

Evaluations in Fig. 3 and Table 2 do not include timing adjustment of packet injection in every slice. We further evaluated the shortest duration times for every combination of arbitration scheme and traffic pattern by adjusting injection timing in every slice in a GA manner. Appendix A describes the evaluation environment and conditions.

Table 3 summarizes the results[††]. Underlined values

[†]Evaluation results are integrated in Table 4 in an at-a-glance form. The bottom row refers to the corresponding sections and literature.

[††]Literature [3] uses the C00 scheme for evaluation. Some results in Table 3 differ from those in the literature, since all the arbitration schemes C00–C09 are newly evaluated and some values of C00 differ from constitutive indeterminacy in the GA method.

Table 3 Shortest duration times in simple stacking with timing adjustment. (16×16 2D-torus, two slices, 8-flit packets)

	C00	C01	C02	C03	C04	C05	C06	C07	C08	C09
bcmp	90	94	90	93	97	88	93	92	87	86
brev	153	157	153	155	156	162	157	156	162	163
brot	140	146	140	146	150	135	154	153	136	136
shfl	131	130	132	130	136	134	136	135	134	135
torn	131	131	131	131	131	138	131	131	138	138
trns	138	138	138	137	138	136	138	138	136	136

[cycles]

show the shortest duration times, which are also summarized in the fourth column in Table 4.

We can find considerable improvement in duration time by adjusting injection timing, except in tornado (torn) and transpose (trns) traffic patterns. We can also recognize that selection of the arbitration scheme still affects the shortest duration time.

5. Extended Cup-Stacking Method

The previous section shows that the arbitration schemes affect the performance of the Cup-Stacking method. This result suggests that the Cup-Stacking method has enough room for further improvement of performance by extending parameters in exploiting optimal solutions. This section firstly argues an ideal solution and its feasibility, and then, it leads to a practical solution that is the core part of this paper.

5.1 Toward an Ideal Solution

Here, we discuss an ideal situation for the Cup-Stacking method. Literature [3] has left the following fact and open consideration:

- after forming a slice shape, slices can be stacked with an inter-slice interval that is less than the thickness of the slice, and
- the phenomena suggests that a slice includes *bubbles* that can tolerate inter-slice interference.

As an extreme solution, if we can obtain an ideal slice, the stacked slices perform ideally, i.e., they can achieve the theoretically best performance.

The problem here is how we can obtain the ideal slice. As an ultimate solution, we discuss precise control of packet flow. Ideally, if a router can precisely control the timing (and order) of packets that pass across the router, the slice shape can be fully optimized for the theoretical minimum. For example, suppose a router receives four packets A, B, C, and D from one or more input ports and channels during a slice communication, the timing (and order) of the transferred packet is strictly scheduled in advance, such as the packet A goes out at T_A-th cycle and B goes out at T_B-th, and so on.

Although the ideal method has the potential for fully optimized solutions, it is not practical due to the vast exploitation space. As the size of the network grows, the number of passing packets increases largely. In this method, as each output port has its own packet schedule, the total search space of the schedule becomes extraordinarily large. We draw a conclusion that the ideal solution is not practical due to the extremely large search space.

5.2 Toward the Practical Solution

As we discussed in the previous subsection, we abandon the perfectly precise (and ideal) solution and leave it for our future work. Then, we discuss the secondary but practical one. The results in the previous section suggest that arbitration schemes could be a hopeful option for obtaining meaningful solutions.

As a simple discussion, key issues in the ideal method (in Sect. 5.1) are timing and order of the transferred packets via each output port. In the Cup-Stacking method, injection scheduling can control the arrival time of incoming packets to a router. Our solution in the extended Cup-Stacking is to employ arbitration methods. An arbitration method possibly changes the order of the transferred packet when the packets conflict. Although the arbitration methods do not offer precise control of the order of output packets, if we can adjust the arbitration method in every output port, we can expect that a meta-heuristic method can find preferable set of arbitration methods in every router so that the duration time can be satisfactorily reduced.

As described in Sect. 4, an arbitration task in the router consists of two steps and one of these, queue selection, has been left untouched. The queue selection step arbitrates the requests from input queues (in input ports) to a certain virtual channel (in an output port). In our router model [18], the central crossbar-switch connects between every input queue (in input ports) and every output virtual channel (in output ports).

The second step, VC selection, arbitrates simultaneous requests from the virtual channels in the same output port. In the queue selection, practical action is not so complicated, since substantive choices are limited by the routing algorithm and virtual channel allocation scheme. Actually, in our router model, possible choices are directions of input ports. For example, channel-0 in the east output port may have only three choices for selection as CPU port (channel 0), north input port (channel 0), and west port (channel 0). The maximum number of selection candidates is four.

Our next idea is to represent the evaluation order in the queue selection choices. The evaluation order is determined at every virtual channel in every output port. The queue selection step firstly refers to the evaluation order and arbitrates the incoming requests according to the order. This paper operates the evaluation in a round-robin fashion according to the order information, i.e., when the selected request is completed, its corresponding entry stands at the tail of the line.

5.3 Extending Chromosome Expression in GA

Discussions in this section have introduced several parame-

ters to optimize the Cup-Stacking method. We summarize the parameters (genes) that formulate a chromosome in the GA meta-heuristic method as follows.

Inter-slice interval (V): each slice begins to send a split packet with this interval. This parameter is applied to the whole nodes[†].

Injection timing (I_i): after a new slice begins, each node pauses for a period of cycles until it injects a new slice packet. This parameter is given for every node individually.

Channel selection scheme (C_{id}): as Sect. 4 mentioned, this parameter formulates the second step of arbitration in our router model. This parameter is given for every output port in every router.

Queue selection order (Q_{idc}): this parameter specifies the evaluation order in the first arbitration step in our router model. This parameter is given for every virtual channel in every output port in every router.

Note that i indicates a node as i-th node, d specifies the input/output port that includes the CPU port, i.e.,

$$d \in \{\mathrm{N(orth)}, \mathrm{E(ast)}, \mathrm{S(outh)}, \mathrm{W(est)}, \mathrm{P(rocessor)}\}, \tag{1}$$

and c represents the virtual channel number:

$$c \in \{0, 1, 2\}. \tag{2}$$

The i-th node injects its s-th slice packet at the $(sV + I_i)$-th cycle from the beginning ($s = 0, 1, 2, \ldots, S-1$, where S is the total number of slices). Each node (referred to as i-th node) arbitrates the requests of the incoming packets according to the two parameters Q_{idc} and C_{id} for queue selection and VC selection, respectively.

Then, we refer to the geometric issues from our previous work [3], before we discuss the definition ranges of the parameters. A slice is conceptually projected as a cup that has a certain height and thickness. Given the height t_d and the effective thickness T_e, duration time of S slices is described as

$$T_d \le T_e(S-1) + t_d. \tag{3}$$

t_d is given as the duration time of a slice and effective thickness is given by the maximum *occupation time*, which is defined as the time length between the first and last flits that are transferred through a physical link. Given that the occupation time of k-th link is O_k, the effective thickness of a slice is defined as

$$T_e = \max(O_k). \tag{4}$$

The minimum value of the inter-slice interval V equals to the length of packets, l_c, and the maximum value is $T_e - l_c$. Thus, the range of V is given as

$$l_c \le V \le T_e - l_c. \tag{5}$$

Injection timing I_i may have the smallest value 0 and it does not exceed $t_d - l_c$:

$$0 \le I_i \le t_d - l_c. \tag{6}$$

Each queue selection order Q_{idc} parameter is represented as an array of five entries, where *five* corresponds to the number of input/output ports. Each element $Q_{idc}[*]$ represents a possible input direction. The router checks whether a valid request comes from the direction $Q_{idc}[j]$ from $j = 0, 1, \ldots, 4$, and it accepts the first request. If a request is accepted, the entries of Q are rotated so that it forms the round-robin fashion. Unused entry is marked as N/A. Each channel selection parameter has one of ten symbols listed in Table 1:

$$C_{id} \in \{\mathrm{C00}, \mathrm{C01}, \ldots, \mathrm{C09}\}. \tag{7}$$

5.4 Evaluation of Extended Method

(1) Evaluation by Full GA

We evaluated the extended Cup-Stacking method that employs the chromosome expressions shown in Sect. 5.3. The evaluation environment and conditions are given in Appendix A. The fifth column in Table 4 shows the results, where marked as *full GA*.

We can find that the extended Cup-Stacking method outperforms the original Cup-Stacking method with at most seven percent improvement in duration time (by comparing the second (prev. work/C00) and fifth (proposal/full GA) columns in Table 4). The proposed method achieves at most 1.91 times improvement in shfl traffic performance from the non-split case (as listed in the first column (prev. work/S=1) in Table 4).

(2) Evaluation without Timing Adjustment

We further evaluated the extended Cup-Stacking with a simplified version of chromosome expression, where timing adjustment is omitted. In this evaluation condition, every node injects its packet just at the beginning of the slice ($I_i = 0$). The evaluation results are shown in the sixth column in Table 4 where marked as *w/* T_{adj}.

This evaluation insists to show the potential of the timing adjustment by comparing the full-GA results in the fifth column in the table. In the traffic patterns bcmp, brot and shfl, the proposed method (full-GA) outperforms the *w/o* T_{adj} cases. This result suggests that the timing adjustment is one of the key issues for re-shaping the slice. On the other hand, from the brev traffic results, the arbitration scheme is the alternative key. Furthermore, results in this section place another issue in resolving ideal slice configuration. Sections 7.1 and 8 discuss the issue.

6. Reduction of Complexity in Cup-Stacking

The extended Cup-Stacking method, proposed in the previous section, achieves considerable performance enhancement from the original method. The good performance

[†]We have tried to change the interval on the per-node basis. The experiment was failed due to the vast search space in the GA method.

comes from the strong capability of the meta-heuristic method, genetic algorithm (GA). In general, GA methods essentially require large computational complexity. They sometimes take a long time for obtaining satisfactory results. Our proposed methods, including the original Cup-Stacking and the extended one, are not the exceptions. Actually, we need large computational resources and long execution times until we can obtain the results given in the previous section. Thus, we should discuss how we can reduce the complexity in the Cup-Stacking method.

6.1 Slice Extrapolation

Our first idea is *extrapolation*. A typical GA method consists of the following two steps: (1) evaluation of chromosomes and (2) genetic operation. A genetic operation, which includes mutation and crossover, generates the chromosomes that are alive in the next generation.

The dominant part of the Cup-Stacking method, in terms of execution time, is the former step, i.e., chromosome evaluation. To obtain an evaluation value of a chromosome, we should run a simulator that can precisely simulate the interconnection network in which the parameters embedded in the chromosome are applied. The simulation process requires a large cost in time, although we use a high-speed simulator [18].

The execution time of the simulator depends on the system size and the number of slices. If we can obtain satisfactory results by extrapolating the results in fewer slices, it is worth discussing. We call the method *Slice Extrapolation*. For example, if a four-slice Cup-Stacking is resolved by using single-slice results, the total evaluation time can be reduced to 1/4.

Here, we discuss the actual *extrapolation method*. In this paper, we use chromosome information that is recorded during GA runs. We extended our GA-based evaluation system so that it can record whole chromosome information that marks the best score of the evaluation value.

We further modified the evaluation system so that it can reload a set of recorded chromosome information to accurately re-configure the network states and parameters and find an optimal inter-slice interval time for the configuration that corresponds to the recorded chromosome.

The system firstly runs a simulation of a single-slice communication after it reads the corresponding chromosome and it measures the occupation time, i.e., effective thickness of a slice. Then, as the second step, the system runs simulations of the targeted number of slices repeatedly until it finds the minimum duration time. During each evaluation, it varies the inter-slice intervals from the packet length to the occupation time.

The evaluation results are listed in the seventh column in Table 4, labeled as *slice extr./S1–S2* where two-slice performance is conducted from single-slice results. The results show that the extrapolation method does not exceed the extended Cup-Stacking method given in the previous section, however, these performance scores are close.

Furthermore, for comparison purposes, we evaluated other combinations of the extrapolation method. The eighth and ninth columns in Table 4, marked as *S2–S2* and *S4–S2*, respectively, show the results. S2–S2 means that chromosome information in the two-slice evaluation is used to *extrapolate* the two-slice cup-stacking. These results may sound meaningless at first glance, however, they confirm the appropriateness of the extended Cup-Stacking method with respect to the inter-slice interval.

S4–S2 similarly means that results of four-slice evaluations are used for a two-slice situation. The method is meaningless in terms of reducing evaluation costs, although, we can learn that the results in a certain condition in terms of the number of slices are not always applicable to different slice conditions, i.e., in this case, four-slice results do not override those of the two-slice condition.

6.2 Rough Estimation Method

The extrapolation method, given in the previous subsection, succeeds in short evaluation time with satisfactory solutions that are close to the extended Cup-Stacking method. However, the extrapolation method is not optimized to fit the Cup-Stacking concept: cups (i.e., slices) are compactly stacked with short thickness (i.e., inter-slice interval). That is, the extrapolation method does not evaluate the thickness. Thus, as the second step in this section, we discuss an alternative and practical method that corresponds to the fundamental idea of Cup-Stacking.

Our idea here is simple. We extended the evaluation function in the GA operation. The preceding methods, including Cup-Stacking and extrapolation methods, evaluate no other factors than the duration time of the collective communication. As a *rough estimation* method, we modified the evaluation function to estimate the targeted duration time T_d^{est} by using the single-slice performance metrics:

$$T_d^{est} = T_e(S - 1) + t_d \tag{8}$$

as a simple application of Eq. (3), where S represents the number of slices, T_e is the effective thickness, and t_d is the duration time of a slice as described in Sect. 5.3.

The rough estimation method firstly runs GA operations of single-slice cases with the modified evaluation function (Eq. (8)), and records the whole information of the final chromosome that marks the best estimation score. Then, the method reloads the recorded chromosome information and investigates the optimal inter-slice interval in the same way with Sect. 6.1.

Evaluation results of the rough estimation method are shown in the tenth and eleventh columns in Table 4, marked as *E2–S2* and *E4–S2*, respectively. E2–S2 means that superior chromosomes are selected by the estimation result of Eq. (8) with $S = 2$ and the best chromosome is recorded in every GA run.

E4–S2 results are conducted from the estimation results with $S = 4$ instead of $S = 2$ in the E2 configuration. The E4–S2 configuration is experimental for compar-

ison purposes. Note that this configuration does not raise evaluation costs in GA operations, unlike the experimental configuration of S4–S2 in the extrapolation method (in Sect. 6.1). Furthermore, we can find that this configuration achieves slightly better performance than those of E2–S2 in the bcmp and brot traffic patterns.

The major difference between E2–S2 and E4–S2 is the weight of the effective thickness (occupation time) in the evaluation function. As stated in the previous work [3] and given in Eq. (3), the actual minimum duration time is much smaller than the estimated values from Eq. (8), except in torn and trns traffic patterns. This phenomenon suggests that a slice has a level of flexibility that can absorb a strong pressure in the stacked situation. The shorter thickness does not guarantee a better performance in the Cup-Stacking, however, we can find some levels of correlation between the effective thickness and Cup-Stacking performance, from the results in Table 4.

6.3 Post-Stacking Method

As we described in Sects. 6.1 and 6.2, we further extended our GA-based evaluation system to record the whole information of chromosome that marks a high-score at that time. Thus, each GA run records the chromosome information of the final solution and its synonyms, which can perform the same score to the final (and best) solution.

The number of the recorded synonyms of the best-score has a wide variety in each GA run, according to our observations. Many of GA runs record hundreds or often thousands of synonyms. Characteristics of the synonyms, in terms of behavior, are far from uniform. The recorded chromosomes have a potential to perform high-scores.

This consideration led us to the further novel idea of *Post-Stacking method*. Basically, this method mines and unveils valued solutions from the synonyms, however, we should discuss practical issues, i.e., possible extra-large number of synonyms. For example, when we run 1,000 GA sessions and each of them produces 1,000 synonyms, the method has to examine one million candidates.

To fit the practicality problem, we further introduce the following metrics that can represent the internal behavior of the network so that we can suppress the number of synonyms.

the number of in-order arbitrations (*ir*):
the number of out-of-order arbitrations (*or*):
these metrics reflect the first half of the arbitration task, queue selection (described in Sect. 4). The queue selection arbitrates the requests from the input queues and grants only one of them. In a multi-slice situation in the Cup-Stacking method, slices are started with a certain interval, however, sometimes packets that are launched in the preceding slice co-exist with those from the subsequent slice. Thus, the arbitration process possibly blocks a preceding packet and passes succeeding one and vice versa. *or* counts the number of blocks in the out-of-order cases, and *ir* counts the in-order cases.

the number of in-order VC blocking (*ib*):
the number of out-of-order VC blocking (*ob*):
similar to *ir* and *or*, these metrics count the number of blocked cases in-order and out-of-order, respectively, in the second half of the arbitration task, virtual channel selection.

the number of physical line blocking (*lb*):
this metric counts the total number of physical lines where output packets are blocked due to unavailable (busy) states of the corresponding input buffers.

We extended our GA-based evaluation system so that the best-score information is associated with the introduced metrics (*ir*, *or*, *ib*, *ob*, and *lb*) as well as the effective thickness (T_e, which equals to the maximum occupation time (*oc*)). When the evaluation system finds a new synonym, it compares the properties of the synonym to the current best-score record. If one or more properties of the synonym exceed that of the recorded best-score, the system records the whole chromosome information and updates the best-score record.

Evaluation results of the post-stacking method are shown in the twelfth to fifteenth columns in Table 4. Results in the twelfth column, marked as C00(S1), are conducted by applying the Post-Stacking method on the recorded synonyms during the fixed VC arbitration (C00) with timing adjustment in single-slice condition (S1). These results corresponds to those in the first column in Table 3.

Results in the thirteenth to fifteenth columns show the Post-Stacking results for the extended Cup-Stacking on one-, two-, and four-slice situations that are marked as S1, S2, and S4, respectively. Results in the S1 column, which corresponds to the fifth column (i.e., the extended Cup-Stacking with full GA), show the effectiveness of the Post-Stacking in terms of performance. The results in the column are very close to those of the slice extrapolation, which are shown in the seventh column (marked as S1–S2). This means that the mining effort in the Post-Stacking hardly improves the performance, however, it can achieve very close performance to the slice extrapolation.

Results in the S2 and S4 columns are for comparison purposes. Results in the S2 column are conducted from the extended Cup-Stacking method in the two-slice situation. Each result in the column (S2) marks the same value with its corresponding one. This proves that the results of the extended Cup-Stacking are sufficiently optimal in terms of inter-slice interval[†].

On the other hand, by considering that the S4 evaluation requires four and two times longer time than that of S1 and S2, respectively, the S4 column does not show satisfactory results. This result comes from the difference in inter-slice behavior in S2 and S4 situations. In the optimal

[†]The Post-Stacking method searches the shortest duration by adjusting the inter-slice interval. Thus, if the recorded chromosome (in S2 evaluation) is not fully optimized in inter-slice interval, the method will find the optimal one and improve performance.

solution, the succeeding slice is launched before the preceding one is not completed, thus, contiguous slices interfere to each other. In our observations, in multiple slice conditions, slice behaviors are not the same due to the inter-slice interference. Thus, it is natural that the solutions (in the form of recorded chromosome) are different in S2 and S4 cases.

7. Improving Solutions

The previous section discussed to reduce computational complexity, with respect to practical issues, in the Cup-Stacking principle. This section argues further improvements in performance in terms of short duration time.

7.1 Optimal Inter-Slice Interval

Inter-slice interval is the only parameter that is commonly used in all routers. As Fig. 3 illustrates, a duration time curve shows a local minima in a macroscopic view, however, the curve does not monotonically decrease (nor increase) toward the local-minima point.

We have arrived at a *perturbation* idea, where the inter-slice interval is optimized for the duration time for every chromosome in every GA generation. In the modified method, called *perturbation method*, at every chromosome evaluation process, the inter-slice interval is varied with the range of ±five cycles and the best one survives.

Evaluation results of the method are shown in the sixteenth and seventeenth columns in Table 4. Results in the sixteenth column, marked as full GA, correspond to those of the fifth column (proposal/full GA), and the seventeenth column results correspond to the sixth column.

As the table shows, the perturbation method can mark the best scores, which exceed the extended Cup-Stacking in many traffic patterns, however, we should recognize the fatal drawback. The perturbation method requires additional simulation runs according to the variation of inter-slice intervals. Since simulation time is dominant in the Cup-Sacking methods, the additional runs result in about ten times long execution time. We measured 5540.1 seconds and 563.2 seconds elapsed times with and without perturbation, respectively, in a common condition of 16x16 network, brev traffic and 2-slice communication on a Ubuntu 20.04.5LTS platform with Intel Core i5-10500T and 16GiB memory.

It stays in the opposite position from those in the methods in Sect. 6, however, the method marks the best-performance and suggests further improvements in the Cup-Stacking approach.

As a possible extension of the perturbation method, we can discuss extending the injection delay parameters in each node, since similar characteristics of non-linearity will be observed in the parameter. However, we will leave the discussion for future work.

7.2 Additional Sort Keys and Functions

Section 6.3 introduced some network metrics for representing internal behaviors numerically and the section uses the metrics so that it can distinguish synonyms of chromosomes effectively. However, we did not discuss the meaning of the metrics in that section.

For example, out-of-order arbitration (*or*) and out-of-order VC blocking (*ob*) metrics represent the degree of interference between the successive slices. Other metrics, such as T_e (effective thickness) and *lb* (physical line blocking), also characterize the network behaviors at a certain level. For example, a large *lb* suggests that the network is heavily congested by frequent collisions of packets.

This subsection introduces some metrics as the secondary sort keys in the GA operations (i.e., crossover function). Chromosomes are sorted by the first key (i.e., duration time) and, if the key value is the same, the secondary key is used for sorting. Alternatively, we tried to change the first sort-key to T_e.

Furthermore, we introduce two new functions in the GA-based evaluation system: *slice re-initialization* and *slice interlocking*. The former re-initializes the router functions in a node when the node receives a packet that destines the node itself. This idea comes from the original cup-stacking concept where the cups form exactly the same shape to ease stacking. Some of the arbitration methods employ a round-robin fashion (as described in Sects. 4 and 5), thus, succeeding slices possibly have different *shapes*. We can expect that re-initialization fixes up the broken slice and achieves shorter duration times[†].

The latter one, slice interlocking, is another idea to reshape the slice geometry. In the sections before, each node injects a packet according to the inter-slice interval and injection delay. Slice interlocking temporarily suspends the injection of a new packet until the node receives a packet that destines the node. This function intends to reduce inter-slice interference.

We experimentally evaluated the additional items individually. Table 5 lists the additional items with their associated symbols (T01–T10)[††]. Table 6 shows the evaluation results of T01–T10 in the 16×16 2D-torus network.

Results in the category of additional sort keys in Table 6 are similar to each other except T07 (physical line blocking, *lb*). These results suggest that (1) out-of-order behaviors (T02 (*or*) and T03 (*ob*)) and VC blocking (T06 ($tb = ib + ob$)) do not significantly affect the communication performance with some worse situations, (2) effective thickness (T04 and T05 (T_e)) also does not show significant difference in the performance results, which is against our expectation, and (3) physical line blocking (T07 (*lb*)) is a hopeful candidate for future improvement in the Cup-Stacking method.

Catastrophic results of T09 (sort only by the effec-

[†]Note that the re-initialization does not guarantee strictly the same shape of slices.

[††]T01 employs no additional items and it is identical to the extended Cup-Stacking method that is presented in Sect. 5. Thus, the first column in Table 6 is a simple copy from the fifth column in Table 4.

Table 4 Evaluation results.

	prev. work		VC arb. (C0*)		proposal		slice extr.			rough est.		post-stacking				optimal int.	
	S=1	C00	simple	w/ T_{adj}	full GA	w/o T_{adj}	S1 –S2	S2 –S2	S4 –S2	E2 –S2	E4 –S2	C00 (S1)	S1	S2	S4	full GA	w/o T_{adj}
bcmp	109	91	91	86	85	91	87	(85)	(86)	88	(87)	89	87	(85)	(86)	83	90
brev	199	151	172	153	145	138	148	(145)	(144)	149	(149)	151	148	(145)	(144)	141	137
brot	178	143	146	135	134	139	137	(134)	(136)	137	(135)	142	137	(134)	(136)	130	137
shfl	243	133	147	130	127	136	130	(127)	(127)	126	(128)	133	130	(127)	(127)	125	135
torn	123	131	131	131	131	131	131	(131)	(131)	131	(131)	131	131	(131)	(131)	131	131
trns	130	138	137	136	136	137	136	(136)	(136)	136	(136)	138	136	(136)	(136)	135	136
(refer to)	lit. [3]		§4.1	§4.2	§5.4		§6.1			§6.2		§6.3				§7.1	

Table 5 List other items on optimization.

symbol	category	description
T01	(no additional issues)	
T02	additional sort key	out-of-order arbitration (*or*)
T03	additional sort key	out-of-order VC blocking (*ob*)
T04	additional sort key	effective thickness (T_e)
T05	additional sort key	T_e in reverse order
T06	additional sort key	total VC blocking ($tb = ib + ob$)
T07	additional sort key	physical line blocking (*lb*)
T08	additional function	re-initialize every slice
T09	different sort key	sort by T_e
T10	additional function	slice interlocking

Table 6 Evaluation results of additional items.

	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10
bcmp	85	85	85	85	86	85	84	84	94	86
brev	145	145	147	146	147	145	144	145	160	148
brot	134	134	134	135	136	135	134	134	148	135
shfl	127	126	127	125	127	126	126	126	137	129
torn	131	131	131	131	131	131	131	131	133	131
trns	136	136	136	136	136	136	136	135	136	136

[cycles]

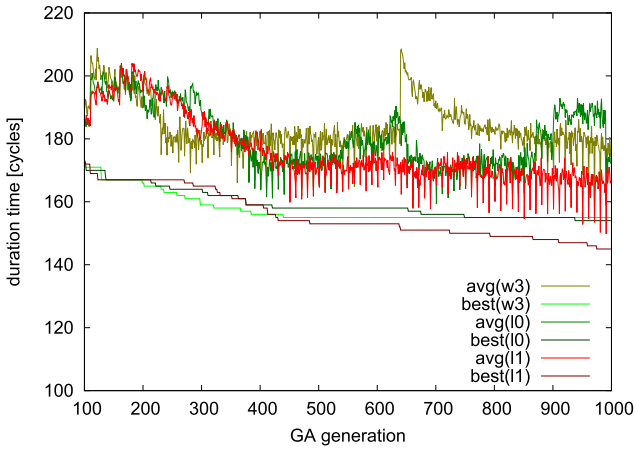


Fig. 4 Time series variation examples in GA operation.

tive thickness (T_e)) are explained that the method does not consider the targeted duration time. Re-initialization (T08) marks preferable results in some traffic patterns and it can also be a hopeful candidate. Slice interlocking (T10) fails to achieve close results to the extended Cup-Stacking, however, it does not largely contribute to the performance.

Although Table 6 shows good results, no one in T01–T10 can mark extraordinary results. We can have high expectations of additional sort keys (*lb* and T_e) and slice re-initialization. The clear advantage of the additional keys is cost-effective since the dominant part of the GA approach consumes time in the chromosome evaluation in which the communication is precisely simulated. Discussions on the practical method based on the results are our future work.

8. Discussions and Future Work

8.1 GA Parameters

Figure 4 depicts some typical behaviors in the GA operation. This figure shows the time series variations of the average (*avg*) and the best duration times in some typical GA runs. The horizontal axis shows the GA generation number and the vertical axis shows the duration time as GA evaluation value. Note that the vertical axis starts from 100 [cycles] (not zero) for a magnification purpose. In this figure, prefixes *w* and *l* correspond to the watchdog and lifetime in the GA parameters, respectively, and the following number is simply an identification.

The average value only includes the scores of newly evaluated chromosomes, where values of un-updated ones (which have survived) are excluded. Periodical lower peaks, which appear in *avg(l0)* and *avg(l1)* curves, correspond to the crossover operation, where the operation overwrites many of chromosomes. From a different viewpoint, these curves suggest that many of mutation operations fail to improve the duration times, however, some survivals conduct the gradual improvement, which leads to successful results.

The *avg(l0)* curve gradually increases at around the 540th generation. This shows that the survived chromosomes are killed by the lifetime limitation[†] at that time. The curve increases again at around the 840th generation. This behavior suggests that the GA run (*l0*) fell into strong local-minima. On the other hand in the *l1* case, the behavior of *avg(l1)* is rather stable and the best duration time is steadily updated as the GA generations go on. Note that there are no differences between *l0* and *l1* in evaluation parameters and the only difference is the random seed. The *avg(w3)* curve shows a sudden increase (around the 590th generation). This shows that the watchdog timer invokes at the time and re-initializes all chromosomes.

Through the discussions on Fig. 4, GA functions of

[†]200 generations in this case.

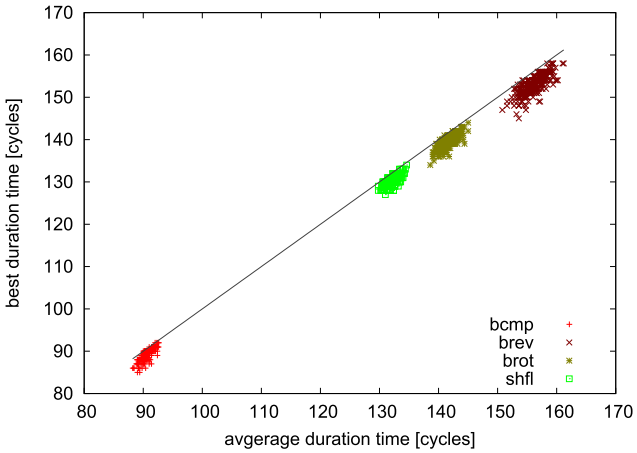


Fig. 5 Best duration time versus average one.

watchdog and *lifetime* work appropriately for avoiding local-minima situations. However, to obtain satisfactory results, we should run multiple GA operations, due to strong attraction of local-minima solutions.

Then, we discuss the degree of variations in the GA results. Figure 5 shows the variation in the extended Cup-Stacking method. In this figure, each traffic pattern has 288 combinations of the GA parameters and each combination executes five GA runs (a total of 1440 runs). Each dot represents the average (in the horizontal axis) and the best (vertical axis) results in its five GA runs. The solid line simply represents $y = x$. We can find that the best duration time has some levels of correlation to the average duration, however, a low average does not guarantee the best duration.

Throughout the vast GA evaluations in this study, we found out some levels of tendencies in the GA parameters. Mutation ratio (r_m) is set $r_m \in \{0.1, 0.05, 0.02\}$ as described in Appendix A.3 (3). In our experiments, larger mutation rates did not work well. This implies that a gradual improvement is effective.

On the other hand, crossover rate (r_c) varies from 0.1 to 0.9 as Appendix A.3 (4) describes. The crossover rate represents the ratio of the exchanged genes. In the $r_c = 0.10$ case, ten percent of genes are exchanged and the remaining 90 percent are inherited to the child chromosome, and in the $r_c = 0.9$ case, 90 percent of genes are replaced and only 10 percent are inherited. A small crossover rate preserves many of genes in the original chromosome and it keeps a wide variety of chromosomes that lead to wide search capability. On the other hand, a large crossover rate generates almost copy of the strong chromosome and it leads to a steep search capability that results in good solutions, whereas it possibly fall into a local-minima situation. In our experiments, we could not find the best crossover rate that is applicable to any conditions of traffic patterns and slices.

With respect to the crossover operation, the interval (i_c) is used as a GA parameter, and this paper sets $i_c \in \{10, 20\}$. A crossover operation sometimes triggers a seed of the next best duration time, which is improved by the following mutation operations. In our experiments, we found that 10 and 20 generations of crossover intervals are appropriate. Furthermore, *watchdog* and *lifetime* parameters strongly correlate to the crossover interval. Short values of these parameters disturb the gradual improvements by the crossover and mutation operations. Thus, we use 100 and 200 [generations] for these parameters.

Despite the discussions above, we could not find the silver bullet (optimal set of GA parameters) at this stage, and we should still execute multiple runs of GA operations. We leave this issue as our future work.

8.2 Search Space

Next, we discuss the search space in the Cup-Stacking method, since this paper introduces many genes that widen the search space in the GA operation. The major issue to discuss is effective exploitation in the different kinds of genes.

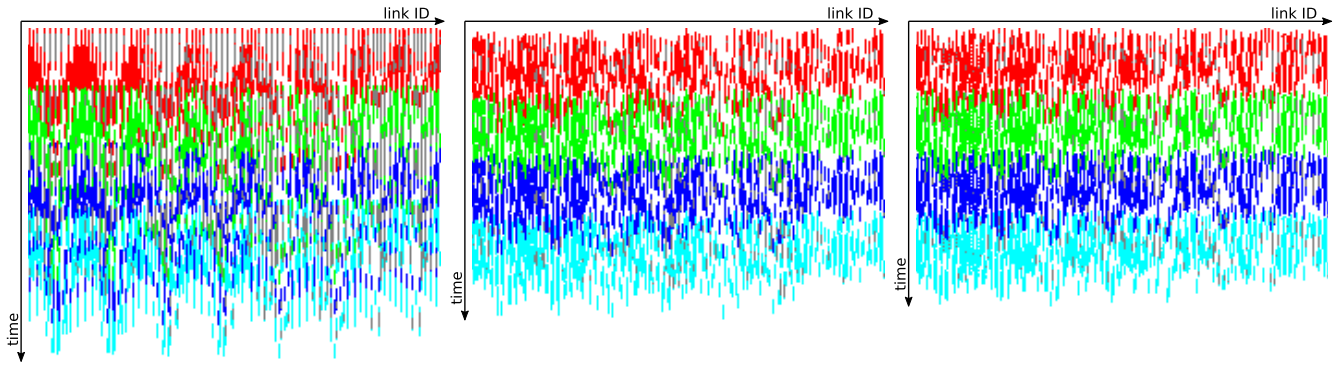
As Sect. 5.4 (2) described, a subset of the extended Cup-Stacking method (i.e., without timing adjustment) exceeds the performance in the brev traffic pattern. Similar result is found in the optimal inter-slice interval discussions (as the perturbation method) in Sect. 7.1.

These results clearly show that the optimal solutions of the Cup-Stacking stay at unbalanced locations in the search space in some traffic patterns (specifically brev traffic in this paper). The *w/o* T_{adj} case sometimes exceeds the full-GA, where the injection timing I_i is set to zero. This fact means that the remaining genes sufficiently affect the shape of the slice. The GA operation employed in this paper consists of the crossover and mutation as described in Appendix A.3. The GA operation widely searches in the injection timing, although, it rarely visits near $I_i = 0$ situations. This discussion leads us to different meta-heuristic methods, such as particle swarm optimization (PSO) as literature [16] employed. However, PSO methods are not well applicable to other genes in Sect. 5.3. We will further discuss effective methods in our future work.

8.3 Visualizing Behaviors

The effectiveness of the extended Cup-Stacking method is shown in Table 4, although, internal behaviors in the network are not discussed. Despite this paper introduced some metrics that can quantitatively represent internal situations and behaviors of the network in Sect. 6.3, we still cannot grasp the network situation intuitively.

Figure 6 visualizes the communication behaviors in bcmp traffic in a 16×16 2D-torus network. In this figure, each horizontal position shows the corresponding (physical) link of the router. For example, the leftmost position corresponds to the north link of the router (0,0). The vertical axis shows time (simulation step). This figure draws a colored dot when a flit is transferred via the link. Different colors of red, green, blue, and cyan correspond to the first, second, third, and fourth slices, respectively. A gray dot represents that the packet transfer is suspended because of the busy



(a) Simple stacking (b) Cup-Stacking (c) extended Cup-Stacking

Fig. 6 Visualized communication behaviors. (16×16 2D-torus, bcmp traffic, 4 slices)

buffer and a white dot shows the link is not used at the time.

Figure 6 shows one-fourth of the total drawing in the x-axis direction, due to the space limitation. Figures 6 (a), (b), and (c) show the simple stacking without re-shaping slice, the (original) Cup-Stacking result, and the extended Cup-Stacking case, respectively. As the figures show, the extended Cup-Stacking method can effectively *stack* the slices to reduce the total duration time.

We then place a different point of view, i.e., slice shapes. As the fundamental idea of the Cup-Stacking method is simply to stack thin cups (slices) to shorten the total height (duration time) as Sect. 3 stated, the idea implies that we expect an identical shape of slices. As Fig. 6 (c) depicts, the shape of the first slice well keeps in the succeeding slices. Slice shapes are not exactly identical, since slices are *pressed* by the short inter-slice interval, however, the distortion portion is limited.

9. Conclusions

Congestion control is one of the most crucial subjects to achieve peak network performance. The Cup-Stacking method intends to improve collective communication performance by splitting a large packet into small slices and re-forming the slices to stack in a short duration time. Preliminary discussions and evaluation results are drawn in the succeeding paper [3], however, some rooms are left open.

This paper discussed possible extensions to the Cup-Stacking method and proposed the extended one, which improves performance at most seven percent in duration time over the original method and accelerates at most 1.9 times over the non-stacking communication.

This paper further discussed practical and ideal issues: large computing complexity and optimal (or better) solutions, respectively. These discussions intend to offer the wide knowledge on the Cup-Stacking principle obtained during the study, which will benefit wide readers. The former discussion leads to three meaningful methods, slice extrapolation, rough estimation, and post-stacking. The latter discussion includes an ideal inter-slice interval (as the perturbation method) and additional evaluation items.

Despite the comprehensive discussions on the Cup-Stacking principle, many issues are left for future work; efficient search method with some meta-heuristic methods for the vast search space and further extension. Robustness of the Cup-Stacking methods is another important issue, since, in practical situations, exact reproduction of the communication situation is quite difficult.

Acknowledgments

This work was partly supported by JSPS KAKENHI Grant Number 20K11726.

References

[1] W.J. Dally and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann Pub., 2004.
[2] J. Duato, S. Yalamanchili, and L. Ni, Interconnection Networks: An Engineering Approach, Morgan Kaufmann Pub., 2003.
[3] T. Yokota, K. Ootsu, and S. Kojima, "On a cup-stacking concept in repetitive collective communication," IEICE Trans. Inf. & Syst., vol.E105-D, no.7, pp.1325–1329, July 2022. DOI: 10.1587/transinf.2021EDL8098
[4] G.F. Pfister and V.A. Norton, ""Hot spot" contention and combining in multistage interconnection networks," IEEE Trans. Comput., vol.C-34, no.10, pp.943–948, 1985. DOI: 10.1109/TC.1985.6312198
[5] E. Baydal, P. Lopez, and J. Duato, "A family of mechanisms for congestion control in wormhole networks," IEEE Trans. Parallel and Distributed Systems, vol.16, no.9, pp.772–784, 2005. DOI: 10.1109/TPDS.2005.102
[6] H. Shibamura, "Active packet pacing as a congestion avoidance technique in interconnection network," Parallel Computing: On the Road to Exascale, pp.257–264, IOS Press, 2016. DOI: 10.3233/978-1-61499-621-7-257
[7] R. Takano, M. Matsuda, T. Kudoh, Y. Kodama, F. Okazaki, and Y. Ishikawa, "Effects of packet pacing for mpi programs in a grid environment," Proc. 2007 IEEE International Conference on Cluster Computing, pp.382–391, 2007. 10.1109/CLUSTR.2007.4629253
[8] Y. Morie, N. Sueyasu, T. Matsumoto, T. Nanri, H. Ishihata, K. Inoue, and K. Murakami, "Optimization of MPI rank allocation considering communication timing for reducing contention," IPSJ Transactions on Advanced Computing Systems, vol.48, no.SIG13(ACS19), pp.192–202, Aug. 2007 (in Japansese).
[9] J.H. Ju and V.O.K. Li, "An optimal topology-transparent scheduling method in multihop packet radio networks," IEEE/ACM Trans.

Netw., vol.6, no.3, pp.298–306, June 1998. DOI: 10.1109/90.700893
[10] D. Rajan, A. Sabharwal, and B. Aazhang, "Delay-bounded packet scheduling of bursty traffic over wireless channels," IEEE Trans. Inf. Theory, vol.50, no.1, pp.125–144, Jan. 2004. DOI: 10.1109/TIT.2003.821989
[11] J. Yang and S. Ulukus, "Optimal packet scheduling in an energy harvesting communication system," IEEE Trans. Commun., vol.60, no.1, pp.220–230, Jan. 2012. DOI: 10.1109/TCOMM.2011.112811.100349
[12] Z. Zhang, Z. Guo, and Y. Yang, "Bounded-reorder packet scheduling in optical cut-through switch," IEEE Trans. Parallel Distrib. Syst., vol.26, no.11, pp.2927–2941, Nov. 2015. DOI: 10.1109/TPDS.2014.2363668
[13] A. Agarwal, "Limit on interconnection network performance," IEEE Trans. Parallel Distrib. Syst., vol.2, no.4, pp.398–412, Feb. 1991. DOI: 10.1109/71.97897
[14] M. Matsuda, T. Kudoh, Y. Kodama, R. Takano, and Y. Ishikawa, "Efficient mpi collective operations for clusters in long-and-fast networks," pp.1–9, 2006. DOI: 10.1109/CLUSTR.2006.311848
[15] B. Tu, J. Fan, J. Zhan, and X. Zhao, "Performance analysis and optimiation of MPI collective operations on multi-core clusters," J. Supercomput., vol.60, no.1, pp.141–162, 2012. DOI: 10.1007/s11227-009-0296-3
[16] T. Yokota, K. Ootsu, and T. Ohkawa, "A static packet scheduling approach for fast collective communication by using PSO," IEICE Trans. Inf. & Syst., vol.E100-D, no.12, pp.2781–2795, Dec. 2017. DOI:10.1587/transinf.2017PAP0015
[17] T. Yokota, K. Ootsu, and T. Ohkawa, "Performance impacts of arbitration functions of interconnection network router for steady/unsteady communications," Proc. 2014 International Conference on Computational Science and Computational Intelligence (CSCI'14), pp.28–33, March 2014. DOI 10.1109/CSCI.2014.13
[18] T. Yokota, K. Ootsu, and T. Ohkawa, "Accelerating large-scale interconnection network simulation by cellular automata concept," IEICE Trans. Inf. & Syst., vol.E102-D, no.1, pp.52–74, Jan. 2019. DOI: 10.1587/transinf.2018EDP7131

Appendix A: Common Evaluation Environment and Conditions

A.1 Evaluation Environment

We extended our interconnection network simulator that is based on the unique concept of cellular automata [18]. This simulator originally intends to accelerate (ultra-)large-scale simulation in a GPGPU environment, however, it achieves sufficient performance in simulation speed even in an ordinary processor (i.e., not GPU) system.

Major extension points are (1) to fit to router behavior to the precise control of parameters that correspond to the chromosome expression (as given in Sect. 5.3), and, (2) to add two operation modes for GA operation and post-stacking. The behavior of the extended evaluation system is precisely specified by run-time parameters. The evaluation system runs in one of the two operation modes: GA mode and parameter-loading mode.

Figure 2 shows the logical organization of the router. The simulator models a simple non-pipelined router and it assumes two-dimensional torus topology. Throughout the evaluations, the number of virtual channels (VC) is three and each VC has a four-flit buffer at every input port of the router. The network employs a deterministic routing algorithm, dimension order routing, where each packet firstly traverses along the x-axis then goes in y-direction until it reaches the destination. Every packet is injected in the first virtual channel (VC-0). When a packet goes across the dateline at $x = 0$, $y = 0$, $x = N/2$, and $y = N/2$, its virtual channel number is increased, where the network forms an $N \times N$ torus.

We use six traffic patterns: bit-complement (bcmp), bit-reverse (brev), bit-rotation (brot), perfect shuffle (shfl), tornado (torn), and transpose (trns). Table A· 1 summarizes the traffic patterns. In this table, X and Y represent x- and y- addresses, respectively. Small letters show bit representation: x_i means the i-th bit in X. W is the concatenation of Y and X: $W = w_{2n-1} \cdots w_0 = y_{n-1} \cdots y_0 x_{n-1} \cdots x_0$. In this paper, we use 8-flit packets.

A.2 Chromosome Expression

Each chromosome expression consists of the following items (genes): (a) inter-slice intervals (V) for each node, (b) injection timing (delay from the inter-slice interval, I_i) for each node, (c) virtual channel selection scheme (C_{id}) for every output port in every node, and, (d) queue selection order (Q_{idc}) for every virtual channel in every output port in every node. As the chromosome expression is designed for flexibility, thus, for some evaluation conditions, a group of chromosome expressions are set to the same value. For example, the extended Cup-Stacking model without timing adjustment was evaluated on the evaluation system with the injection timing parameters (I_i) are forced to zero. Note that, in this paper, inter-slice intervals are set to the same value for all the nodes.

A.3 GA Operation Mode

As described in Appendix A.1, the evaluation system has two operation modes. One of them is the GA operation mode. This mode executes GA operations of the specified number of chromosomes for the specified number of generations. We unalterably set the number of chromosomes to fifty (50) and the number of generations to one thousand (1,000) throughout this paper. The GA mode inhibits crossover function in the first one hundred (100) genera-

Table A·1 Traffic patterns used.

abbreviation	description
bcmp	bit-complement. $w_{2n-1}w_{2n-2}\cdots w_0 \longrightarrow \overline{w_{2n-1}}\,\overline{w_{2n-2}}\cdots\overline{w_0}$
brev	bit-reverse. $w_{2n-1}w_{2n-2}\cdots w_0 \longrightarrow w_0\cdots w_{2n-2}w_{2n-1}$
brot	bit-rotation. $w_{2n-1}\cdots w_1w_0 \longrightarrow w_0w_{2n-1}\cdots w_1$
shfl	perfect shuffle. $w_{2n-1}w_{2n-2}\cdots w_0 \longrightarrow w_{2n-2}\cdots w_0w_{2n-1}$
torn	tornado. $W \longrightarrow \mathrm{mod}(W + N/2, N^2)$
trns	transpose. $(X, Y) \longrightarrow (Y, X)$

tions. In this period of generations, each chromosome *trains* by a mutation operation. This intends to a preferable set of initial chromosomes for the succeeding GA operations. After the initialization phase is finished, the GA operation runs mutation and crossover, according to the given parameters.

Every chromosome is marked with a 32-bit signature to distinguish identical sets of parameters and avoid cloning. This paper uses CRC-32 code, which is generated from all of the genes in the chromosome. The evaluation system records all the signatures of the best-score cases. When a chromosome marks the same score with the current best, the evaluation system calculates its CRC-32 code and checks whether the identical CRC-32 code is already recorded. If the identical code is recorded, the corresponding chromosome is initialized.

In the GA operation mode, the evaluation system firstly runs a single-slice simulation with randomly selected C_{id} and $I_i = 0$, and measures the duration. That duration value (t_{d0}) is used for the upper limit of the inter-slice interval (V) and injection timing (I_i).

(3) Mutation

Mutation operation selects the given ratios of the alterable chromosome expressions (genes) and alters their values within the defined range to generate a child chromosome. For example, injection timing (I_i) may be changed within the range of $[0 : I_{max}]$, where $I_{max} = t_{d0} - l_c$ and l_c is the packet length. If the altered chromosome (child) performs better than its parent, the child survives for the next generation. Otherwise, the parent survives and the child is killed. In this paper, we set the mutation ratios $r_m \in \{0.1, 0.05, 0.02\}$.

(4) Crossover

Crossover operation is executed intermittently in the given intervals of generations: $i_c \in \{10, 20\}$. The operation firstly sorts the chromosomes in the evaluation value order. For the tie chromosomes, which have the same evaluation score, we used two options: *random* and *age*. The former arranges tie chromosomes randomly and the latter arranges the chromosomes by the age order.

Then, after chromosome ordering is fixed, it systematically selects two chromosomes from the sorted order as $(1, 2)$, $(1, 3)$, $(2, 3)$, $(1, 4)$, $(2, 4)$, $(3, 4)$, and so on, where (a, b) means a combination of two chromosomes at the a- and b-th orders. The child chromosome overrides the weaker parent and the stronger one survives. The crossover function substitutes randomly-selected genes in the a-th chromosome (I^a) to the corresponding genes in the b-th chromosome (I^b) as $I^a_i = I^b_i$. Every gene in the chromosome is selected to crossover at a given possibility $r_c \in \{0.1, 0.2, 0.4, 0.6, 0.8, 0.9\}$. When the crossover operation is not carried out, a mutation operation is applied.

(5) Watchdog and Lifetime

Furthermore, to prevent local-minima situations, we introduce *lifetime* and *watchdog timer* in the GA operation. The former restricts the maximum number of generations of a chromosome, and the latter re-sets a timer at every update of the best score of the evaluation value. The timer re-initializes all the chromosomes at their expiration time. Lifetime is given as $lt \in \{0, 100, 200\}$ [generations] and the watchdog timer is set as $wd \in \{0, 100, 200\}$ [generations], where $lt = 0$ and $wd = 0$ mean that the corresponding functions are inhibited.

(6) GA Operation Run

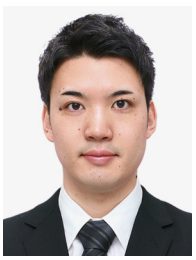
We ran possible combinations of parameter values and sorting options (*random* and *age*, described in Appendix A.3 (4)), where each combination has five simulation runs. Since the watchdog and lifetime operations are mutually exclusive, thus, when the lifetime $lt > 0$, the watchdog operation is prohibited, and vice versa.

A.4 Parameter-Loading Mode

The other mode of the evaluation system is the *parameter-loading* mode. This mode reads a set of chromosome information that is recorded in a file during a GA-mode run. Once the chromosome loading is succeeded, the evaluation system runs a network simulation according to the loaded chromosome and run-time parameters and it measures necessary metrics such as the duration time and occupation time. For example, the Post-Stacking method was evaluated in the parameter-loading mode, varying the inter-slice intervals by the corresponding run-time parameter.

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 was a research associate and then an assistant professor at Utsunomiya University. From 2009 to 2020, he was an associate professor at Utsunomiya University. Since 2020, he has been a professor at Utsunomiya University. His research interests are in high-performance computer architecture, multi-core/multithread processor architecture, binary translation and run-time optimization, mobile computers, and embedded systems. He is a member of IPSJ (Information Processing Society of Japan), IEICE (Institute of Electronics, Information and Communication Engineers) and ISCIE (Institute of Systems, Control and Information Engineers).



Shun Kojima received the B.E., M.E., and Ph.D. degrees in electrical and electronics engineering from Chiba University, Japan, in 2017, 2018, 2021, respectively. From 2021 to 2022, he was an Assistant Professor in the Department of Fundamental Engineering, Utsunomiya University, Tochigi, Japan. Currently, he is a Project Research Associate with the Institute of Industrial Science, The University of Tokyo, Tokyo, Japan. His research interests include OFDM, MIMO, cooperative communications, adaptive modulation and coding, visible light communications, channel estimation, and machine learning. He is a member of IEEE and the Institute of Electronics, Information and Communication Engineers (IEICE). He received the Best Paper Award at the 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM2018) in 2018, the Best Poster Award at the 3rd Communication Quality Student Workshop in 2019, IEEE VTS Tokyo/Japan Chapter 2020 Young Researcher's Encouragement Award in 2020, RISP Best Paper Award in 2021, and IEICE Radio Communication Systems (RCS) Active Researcher Award in 2021.