IEICE TRANS. INF. & SYST., VOL.E106–D, NO.2 FEBRUARY 2023

PAPER

A Compression Router for Low-Latency Network-on-Chip

Naoya NIWA[†a)], **Yoshiya SHIKAMA**[†], ***Nonmembers***, **Hideharu AMANO**[†], ***Fellow***, ***and*** **Michihiro KOIBUCHI**[††,†††], ***Senior Member***

SUMMARY Network-on-Chips (NoCs) are important components for scalable many-core processors. Because the performance of parallel applications is usually sensitive to the latency of NoCs, reducing it is a primary requirement. In this study, a compression router that hides the (de)compression-operation delay is proposed. The compression router (de)compresses the contents of the incoming packet before the switch arbitration is completed, thus shortening the packet length without latency penalty and reducing the network injection-and-ejection latency. Evaluation results show that the compression router improves up to 33% of the parallel application performance (conjugate gradients (CG), fast Fourier transform (FT), integer sort (IS), and traveling salesman problem (TSP)) and 63% of the effective network throughput by 1.8 compression ratio on NoC. The cost is an increase in router area and its energy consumption by 0.22 mm^2 and 1.6 times compared to the conventional virtual-channel router. Another finding is that off-loading the decompressor onto a network interface decreases the compression-router area by 57% at the expense of the moderate increase in communication latency.

key words: *Network-on-Chips, router architecture, lossy data compression*

1. Introduction

The number of cores in a system is increasing, particularly in high-performance computing. A high degree of parallelism in these systems introduces various challenges to their applications and component architectures. The increase in bandwidth requirements for on-chip communication is one of the most serious problems. Network-on-Chip (NoC) is a popular method to support this high bandwidth requirement, which uses on-chip routers to build a network for scalable on-chip communication. Compared to the bus method, which uses global signal lines within the chip, NoC can achieve higher throughput, because communication can be initiated from multiple cores and a number of routers can transmit data simultaneously. On the contrary hand, NoC relies on complex routers, which increase the circuit area and energy consumption and cause increased latency because of multiple hops between routers. The major concern in NoC design is the reduction of latency without compromising scalability.

Manuscript received May 18, 2022.
Manuscript revised August 23, 2022.
Manuscript publicized November 8, 2022.

[†]The authors are with Keio University, Yokohama-shi, 223–8522 Japan.
[††]The author is with National Institute of Informatics, Tokyo, 101–8430 Japan.
[†††]The author is with PRESTO JST, Tokyo, 101–8430 Japan.
a) E-mail: naoya@am.ics.keio.ac.jp

DOI: 10.1587/transinf.2022EDP7080

Assuming that a packet that consists of L flits is transferred through H hops, the ideal zero-load communication latency, T, is calculated on a wormhole NoC as follows:

$$T = T_{lt} \times (H + 1) + T_{router} \times H + L, \tag{1}$$

where T_{router} and T_{lt} are the latency of the router and link transfer, respectively. To reduce the value H, the network topology needs to be changed, which tends to be fixed by the implementation requirements of the system. Thus, prior studies focused on reducing T_{router} with innovation on router architectures, and various low-latency routers have been proposed [1], [2].

This study attempts to reduce L in Eq. (1) by data compression. Data compression is a widely used technique that has been extensively researched. In NoCs, there is a severe restriction on latency. Assuming the number of cycles for data compression T_{comp} and that for data decompression T_{decomp}, Eq. (1) can be rewritten as follows:

$$\begin{aligned} T = {} & T_{lt} \times (H + 1) + T_{router} \times H + L_{compressed} \\ & + T_{comp} + T_{decomp}, \end{aligned} \tag{2}$$

That is, in order to reduce zero-load latency of NoC, $L - L_{compressed} > T_{comp} + T_{decomp}$ must be satisfied. In the practical NoC design, only a few cycles are allowed for $T_{comp} + T_{decomp}$. Thus, the compression algorithm must satisfy a strict compression-time limitation.

When the load of a network is heavy, the delay in the network increases significantly compared to the zero-load latency. Thus, the influence of the increased delay owing to data compression becomes small in such a case. In this study, a compression router that hides the latency by compressing and decompressing packets in the router pipeline, thus making $T_{comp} = T_{decomp} = 0$, is proposed.

A source router compresses the data body of a packet, while the destination router decompresses them. The control information required for packet transfer, for example, a destination, a virtual channel (VC) number, and a flit type are not compressed. Because it is a small portion of a packet, its influence on the packet length is low.

The compression router (de)compresses the data content of an incoming packet before completing the switch arbitration operations. Because the advanced on-chip router takes two to four pipeline stages, the compression latency should be completed in one or two cycles to achieve no latency penalty by the data (de)compression operation. In this

context, two simple bitwise compression algorithms, the frequent pattern compression(FPC) approach for integer numbers [3], and bit-cut approximation for floating-point numbers [4], [5] on the compression router, are considered. The FPC uses patterns effective for data sequences of positive integers with consecutive zeros in the upper bits. We then evaluated how the proposed method improved performance when the FPC worked efficiently. Although the proposed method can be efficient for different applications, investigating the area is outside the scope of this study.

The traditional data-compression approach (de) compresses the data at a network interface(NI) or processing element (PE) which heavily imposes the (de)compression latency to end-to-end communication latency on NoCs [6], [7].

The contributions of this paper are as follows.

- A compression router architecture that performs packet forwarding processing and (de)compression in parallel is proposed. The (de)compression overhead is two cycles, and its operation latency is hidden.
- The cycle-accurate network simulation shows that the compression router improved up to 63% of the effective network throughput under synthetic traffic patterns compared to the conventional NoC with no compression. Communication latency decreased up to 41%.
- The cost is an increase in router area and its energy consumption by 0.22 mm^2 and 1.6 times compared to the conventional virtual-channel router. Another finding is that off-loading the decompressor onto a network interface decreases the compression-router area by 57% at the expense of the moderate increase in communication latency.

This study extends the work of our prior publication [8] by exploring alternative compression router architectures and detailing a comprehensive analysis of NoCs using compression routers.

The remainder of this paper is organized as follows. Section 2 presents the background of the compression router. Section 3 describes the architecture of the compression router. Section 4 presents the results of the analysis and simulation. Section 5 concludes the paper with a summary of our findings.

2. Background

2.1 Compression in Network-on-Chip

Using compression to enlarge the network effective throughput is a common technique on networks outside chips. For example, well-known lossless compression algorithms such as gzip are widely used in common communications such as HTTP. In the field of High-Performance Computing (HPC), communication compression is also used to improve performance and reduce energy consumption [9], [10].

In such a network, because end-to-end communication latency is long and a message is composed of multiple packets, the time margin for data compression can be maintained and a certain number of packet reductions by compression can be expected. On the contrary, in NoC, its short end-to-end latency imposes severe restrictions on the compression overhead. To improve the system performance using data compression on NoCs, a method is to relax such severe design limitations. This study presents a proposal for communication compression of NoCs that overcomes these constraints.

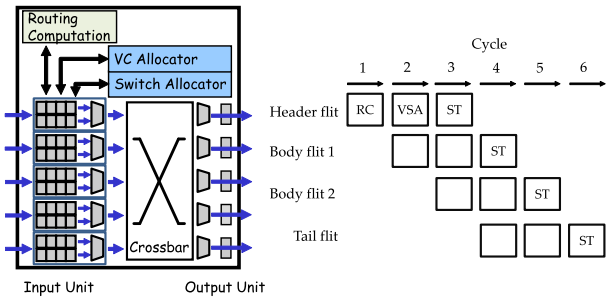


Fig. 1 (a) Block diagram of a baseline on-chip router, and (b) pipeline processing of a packet.

2.2 Baseline Router

The input-buffered router architecture shown in Fig. 1 (a) was introduced as the baseline. A wormhole router with p physical channels, each of which has two virtual channels (VCs) is assumed.

Each packet is forwarded along the pipelining, as shown in Fig. 1 (b). A header flit is forwarded along three pipeline stages: routing computation (RC), VC, and switch allocation (VSA) to obtain the access grant of an output channel, and switch traversal (ST) for transferring flits through the crossbar on a router.

The speculation technique was applied to an on-chip router. The speculative router performs multiple pipeline stages parallel to [1], [2]. If both stage operations succeed, the flit-transfer latency decreases. For example, in the speculation that performs VSA and ST operations in parallel, a router transfers a flit with two cycles. To perform this speculation, at least the input speedup is required [2].

2.3 Compression Algorithms

A compression algorithm that considers the similarity of the data to be compressed has been well researched. A low-latency compression algorithm that uses the prefix pattern of data appearing at high frequencies has been proposed for cache line and NoC communication [3], [11], [12]. There are also low latency compression algorithms used to enhance the performance of interconnection network [13], which is used in this study. An NoC forwards packets that contain data in a cache line between a processor element (PE) and a shared Level-2/3 cache on a chip multiprocessor

(CMP). Compression algorithms have been proposed for such NoC communications, and there are trade-offs in terms of compression ratio, latency, and accuracy. In our compression router, we chose existing compression algorithms that are appropriate for the application and adjusted their parameters.

Compression algorithms are classified as lossless and lossy. In terms of correctness, the lossless algorithm can be applied to any data, whereas the lossy algorithm can be applied only to applications that satisfy the requirement of accuracy.

This study follows the policy: lossless compression is used for accurate data transfer, and lossy compression is used for possible applications that accept degraded accuracy.

2.4 Compression Latency Hiding

Data compression can increase the effective throughput, while it may increase the communication latency owing to its operational overhead, which stretches the application execution time. Hiding the (de)compression operation delay is an essential consideration in the design.

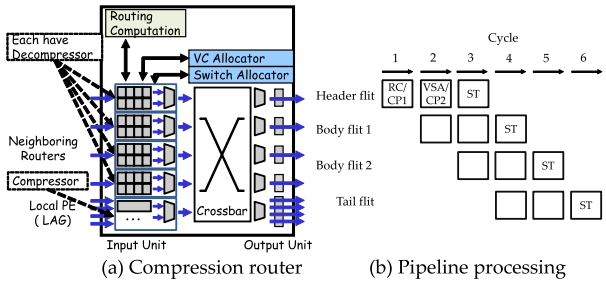
Jin et al. proposed embedding the compressor (encoder) and decompressor (decoder) in the network interface [14]. The overhead of encoding is only one clock cycle by the pipelined implementation of packet injection using streamlined encoding. Boyapati et al. introduced an approximation for NoC compression, similar to the proposed method [12]. They also proposed to compress the data with a VC allocation operation or the queuing time of the network interface in parallel. Both studies can hide the latency of compression; however, they do not touch the decompression.

In most studies, including this study, compression is performed when packets are injected into the network, and decompression is performed at the ejection. This allows the transfer of compressed data in the network and treatment of uncompressed data in the processing elements. In this framework, compression and decompression are asymmetric from the viewpoint of NoC. That is, at the injection, a packet is always the compression target, while the ejection of a packet is judged by the router at its final stage of packet processing. When a packet is judged to be ejected, the role of the NoC is simply ejecting it; thus, there is no time to hide the latency of decompression.

3. Compression Router

3.1 Router Components

The proposed compression router is an extension of the baseline input-buffered router, as shown in Fig. 1 (a). In the baseline router, a header flit waits for the completion of the RC and VSA operations at the input port. The compression router (de)compresses the contents of an incoming packet



(a) Compression router (b) Pipeline processing

Fig. 2 (a) Block diagram of the compression router, and (b) pipeline processing of a packet.

at the input port before completing the two pipeline operations. Our data compression targets are the cache line content of the packets. This includes some of the cash lines included in the flit. Figure 2 (a) illustrates the change in the compression router architecture from the baseline router in Fig. 1 (a).

The changes required for the compression router are as follows: (1) adding a compressor for an input channel corresponding to a local endpoint, that is, PE or cache, (2) adding a decompressor for each input channel from a neighboring router, and (3) aggregating five links from/to a local endpoint for compression latency hiding.

In (2), the decompressors are placed at each input port instead of the output port to the local endpoint to hide the latency caused by the decompressor. If the routing destination is a local endpoint, decompressed data is forwarded. If the routing destination is a router, it forwards undecompressed (compressed) data, and discards the decompressed data. From the packet's viewpoint, it is compressed once when it is injected to the network and decompressed each time it is forwarded to the router. In the case of a route with many hops, many of its decompressed values will be discarded. As described in Sect. 3.2, the proposed method hides the compression and decompression latency in the early part of the router pipeline. If a decompressor is placed on the output port from the neighboring router to the local endpoint, this latency concealment becomes impossible. Installing a decompressor at the input port means that an increasing area proportional to the number of routers is required. There is a trade-off relationship where the latency can be reduced at the cost of increasing the area. The proposed method first selects latency reduction.

In (3), link aggregation is necessary to work the compressor, which requires two cycles to compress the entire packet. The details are presented in Sect. 3.4.

The compressor and decompressor are equipped with multiple compression algorithm circuits and a mechanism to select one. This allows for adaptive compression depending on the cache line content. The details are described in Sect. 3.3.

3.2 Pipeline Structure

The data compression is independent of the operation of

the routing computation and VC/switch-allocation; thus, the compression can be executed in parallel with them. The compression router thus becomes a three-stage pipeline, [RC/CMP1] [VSA/CMP2] [ST], as shown in Fig. 2 (b).

At the time of switch traversal, data must be kicked out of the queue, therefore compression is not possible. Because the header flit also contains part of the data payload, the maximum latency that can be hidden is two cycles in the baseline router.

On an intermediate router, the incoming packet data body has already been compressed. Because the control information is not compressed on the packet, the pipelining of a packet can be the same as the conventional router in Fig. 1 (b), except for the case where the destination is a local endpoint. For such a case, a compression router is required to decompress an incoming packet.

3.3 Compressors

Two compressors, lossless and lossy, were designed. Both compressors only compress the cache lines contained in the packet, and not the headers containing packet destination and cache address information.

The compression router can provide multiple compressors and switch them using a compression mode field attached to the header. This proposal can be easily applied by changing the compression circuit accordingly.

3.3.1 The Lossless Compressor

We used the lossless compression algorithm and customized FPC [3] for integer values. The compression targets were 32-bit integer arrays. The array was divided into 32-bit integers, and each value was compared with the pattern shown in Tables 1, 2, and 3. The pattern on a sequence of zeros was prepared in the frequently occurring higher bits. We attempted up to three, seven, and fifteen compression patterns identified by a 2, 3, or 4-bit prefix. The data were compressed by combining the prefix corresponding to the matched pattern with the rest of the bits outside the pattern. There was a priority order of patterns, and patterns with smaller data sizes were prioritized. In the case of the "Uncompressed" pattern that did not match any of the patterns, the amount of redundant data increased by a purely redundant amount, from 32 to 35 bits with a 3-bit prefix.

Each pattern was identified to be a frequent pattern in preliminary experiments that observed NoC traffic. If the number of prefix bits is increased, the number of supported patterns can be increased, however, this increases the overhead of the prefix and complicates the circuit. It is difficult to set up complex patterns that require arithmetic operations because of the constraints on the number of cycles of compression. Because many integers with small values and consecutive zeros in the upper bits were transferred to the NoC traffic, a pattern that can be applied to patterns with particularly high frequencies was constructed.

Table 1 Customized Frequent Pattern Encoding (The pattern upper is the higher priority.)

Prefix	Pattern encoded	Data size after encoding
101	18 bits or more leading zero	17 bits
100	17 bits leading zero	18 bits
011	16 bits leading zero	19 bits
010	15 bits leading zero	20 bits
001	14 bits leading zero	21 bits
000	13 bits leading zero	22 bits
111	Uncompressed	35 bits
110	(Unused prefix)	

Table 2 Frequent Pattern Encoding (2-bit prefix) (The pattern upper is the higher priority.)

Prefix	Pattern encoded	Data size after encoding
10	18 bits or more leading zero	16 bits
01	17 bits leading zero	17 bits
00	16 bits leading zero	18 bits
11	Uncompressed	34 bits

Table 3 Frequent Pattern Encoding (4-bit prefix) (The pattern upper is the higher priority.)

Prefix	Pattern encoded	Data size after encoding
1110	24 bits or more leading zero	12 bits
1101	23 bits leading zero	13 bits
1100	22 bits leading zero	14 bits
1011	21 bits leading zero	15 bits
1010	20 bits leading zero	16 bits
1001	19 bits leading zero	17 bits
1000	18 bits leading zero	18 bits
0111	17 bits leading zero	19 bits
0110	16 bits leading zero	20 bits
0101	15 bits leading zero	21 bits
0100	14 bits leading zero	22 bits
0011	13 bits leading zero	23 bits
0010	12 bits leading zero	24 bits
0001	11 bits leading zero	25 bits
0000	10 bits leading zero	26 bits
1111	Uncompressed	36 bits

3.3.2 The Lossy Compressor

The lossy compression algorithm used in this study trims the lower bits for floating-point values [4], [5]. The compression targets are 64-bit IEEE 754 double-precision floating-point arrays, and the method reduces the size by trimming the lower bit data in the array. The number of bits to be trimmed was specified through the header. The trimmed bits were recovered on the receiver side. The pattern of 0b100... was used to make the completion close to the median of the possible values.

The characteristics of the application determine the bit length to be truncated. This number of bits was determined by finding the maximum number of bits to truncate that can actually run the application and maintain the quality of re-

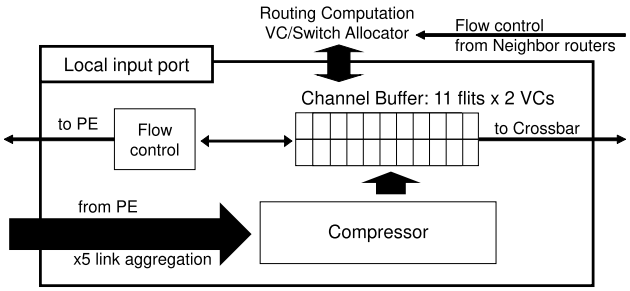


Fig. 3 The compressor on the compression router.

sults that the application will tolerate. The compression router used a compression circuit that allows the number of bits to be trimmed to be set arbitrarily to 2, 4, 6, . . . , 28 bits, and compresses the data with the accuracy required by the application.

3.4 Design of (De)Compressor

In the compression router, each compressor was independently implemented at a local input port and shared by all VCs on the compression router. The (de)compression must be completed within two cycles to avoid affecting the compression latency to the router latency.

In this study, we mainly discuss the implementation of the lossless compressor because the lossy compressor simply discards the leading LSBs for floating-point numbers. We used an FPC lossless algorithm that compressed only frequently appearing patterns of integer numbers, for example, leading 0 s and 1 s, and used three bits for the prefix, that is, up to seven patterns of values. The bit width on a link is constant on a router, and two cycles are required for compression. Thus, it was necessary to receive all flits in the two cycles during compression. Figure 3 shows the architecture of the compressor at the input port. When a packet is 10 flits, the router must receive 5 flits per cycle from the local PE through link aggregation. We need to prepare a 11-filt channel buffer for the increase in packet size after compression. The flow control is done based on the free buffer capacity as in common NoCs as well as other control mechanisms. If the compression ratio is low and time is needed to inject the packets after compression, the packets are held in a queue similar to the case of network congestion.

In the figure, for the compression of integer values, the prefix calculator was used to count the leading 0 s and 1 s to identify frequently appearing (input) patterns. In the pipeline, the first stage [CMP1] compresses the first four flits. A compression algorithm requires the serialization of the input data, and its operation is included in this stage. The second stage [CMP2] compresses the remaining flits and writes the header flit back to an input VC. The compression in the second stage was performed in two steps. First, the remaining six flits were divided into two fronts and four backs and compressed each one. The next two front flits are combined with the four flits compressed in [CMP1] and written back to the register. In stage [ST], the first six flits and the second half four flits that have been compressed separately so far are combined and written back to the input buffer. We maintain the operating frequency with the compression stage on the router, and the results are presented in the next section.

The decompressor is implemented on each input port from a neighbor, unlike the compressor. Because different VCs have the probability of receiving flits of multiple packets simultaneously, each VC has a decompressor that increases the amount of hardware for the router. Because the decompressor works only for a packet to a local endpoint at a compression router, it uses the results of RC. If the destination is not a local endpoint, the decompressor stops its operation. At this time, the first stage of the decompressor is already in operation, resulting in unnecessary energy consumption. It reduces latency, but with the trade-off of increased area and energy consumption. Because a header flit is not subject to compression, the routing itself does not require decompression.

3.5 Alternative Design of Compression Router

Since all the input ports from neighbor routers are equipped with the decompressors, the decompressors can be a dominant factor in the hardware amount of the compression router. To reduce the hardware amount, an alternative design is to off-load the decompressors onto a network interface. It only requires a single decompressor to a network interface instead. The drawback is that the operation latency of the decompression can not be hidden, thus increasing the end-to-end communication latency. We investigate this trade-off in the next section.

4. Evaluation

4.1 Condition

Similar to a conventional on-chip router with three pipeline stages, [RC], [VSA], and [ST], the compression router takes three pipeline stages, [RC/CMP1], [VSA/CMP2], and [ST]. For both routers, a cycle was maintained for the delay of link transfer. As a topology baseline, a 4×4 2-D mesh with minimal routing was used.

We considered IS, CG, and FT from NAS Parallel Benchmarks (OpenMP version) and TSP (traverse salesman problem) parallel applications for the evaluation. The TSP parallel program was extended to use a genetic algorithm available from [15], and "berlin52", which is a 52 cities problem included in TSPLIB, was used. The integer values in the communication data are a compression target of the lossless algorithm on IS and TSP, while floating-point values in CG and FT are for the lossy algorithm. As reported in the prior study [16], the fault tolerance of CG and FT is intrinsic as a result quality can remain acceptable even in the presence of soft errors in LSBs. The lossy compression marginally decreases the quality of the results on FT, whereas it converges the computation results on CG. From

the evidence of the prior study, we considered discarding 28 bits from Mantissa in floating-point communication data in CG and FT to accept the quality of results in NAS Parallel Benchmarks.

4.2 Data Compression Ratio

The data compression ratio was evaluated by applying each compression algorithm to a run-time dump of data that must be transferred to the NoC.

The application source codes and captured memory access dumps were analyzed to measure the compression ratio of the communication data. Each application used OpenMP for communication between the processing elements. We targeted the loop using the pragma of OpenMP (e.g., `#pragma omp parallel for`) for the analysis, and we only captured the data that would not be stored in the L1 cache or cache-coherent control messages between the processing elements. Note that we excluded multiple read traffic data from the same addresses in a loop because they are expected to be stored in the local L1 cache.

The variables in the loop parallelized by OpenMP were classified into several groups as follows, and it was determined whether they were required to transfer in the NoC. We then dumped only those variables that were transferred. We did not dump the variables that were incremented in the "for" loop to be parallelized because they were not exchanged between threads. The loop variables and temporary variables inside the "for" loop to be parallelized were not dumped for the same reason.

Access to the array prepared outside the loop was assumed to be a dump target. However, in the case of multiple writes to the same location, only the last write was included in the dump. Because the multicore processor employed a snoop cache, when one PE wrote to the same cache line continuously, the cache line was not transmitted unless other PEs accessed the area. Because the order of thread execution was not guaranteed in OpenMP, reading variables across threads without using barrier synchronization would result in race conditions. Therefore, while one thread wrote multiple times to the same location in a practical parallel application, no other thread read the variable. For this reason, only the last write to the same location was dumped, and multiple reads were only dumped in the first one.

Each application was executed to capture the memory dump on a real machine with 32 threads. Because the execution log was considered based on source code analysis, the result depended only on the parallelism of OpenMP and did not depend on the number of cores of the machine actually used. The problem with this memory access dump method was that it did not consider the increase or decrease of memory accesses owing to differences in cache configuration. However, what we want to get from this dump was only the compression ratio by target compression algorithms; thus, the amount of memory access did not affect the result. Because the tendency of the contents of memory reads and writes affected the result if accesses to arrays that account for most of the memory accesses could be dumped, sufficiently reasonable results with good accuracy could be obtained.

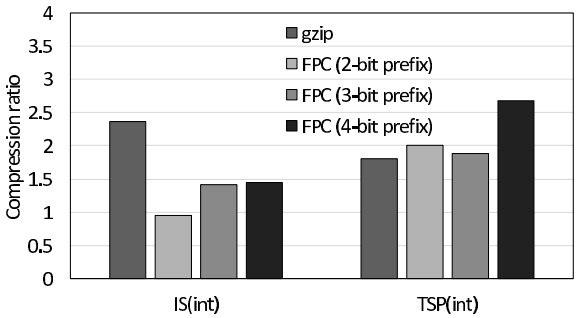


Fig. 4 The compression ratio of communication data. (Integer applications)

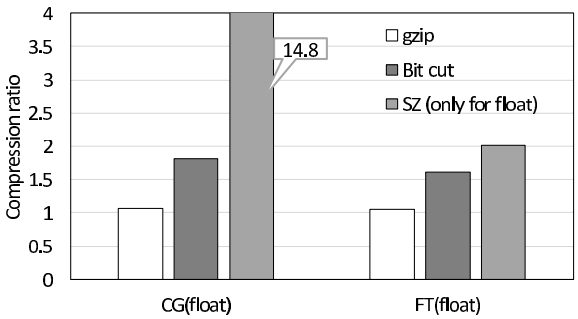


Fig. 5 The compression ratio of communication data. (Floating point applications)

After acquiring the memory dump, the target compression algorithm was applied to the dumped data, and the ratio of the data size before and after compression was used as the compression ratio.

Figures 4 and 5 show the compression ratio (original size/compressed size) of the communication data for each parallel application. The control information of a packet, including a destination, is not included; however, its size can be small to a packet data size. As a reference, the results for lossless compression, gzip, are added in the figure, although it is not applicable to the proposed router, as described in Sect. 2. As expected, the gzip has a low compression ratio (close to 1.0) in CG and FT. By contrast, the bit cut (truncating) provides 1.6 and 1.8 compression ratios of CG and FT, respectively. The FPC achieves a 1.4 and 1.9 compression ratio of IS and TSP, respectively. Note that the compression ratio of the compression algorithms in the compression router is determined by the ratio to discard the mantissa bits in floating-point values that satisfy the quality level defined by each benchmark.

Figure 4 shows a comparison of the FPCs with different prefix lengths. In IS, the compression ratio is only slightly higher for the FPC with a 4-bit prefix than for the 3-bit prefix, however, there is a large difference in the compression of the TSP, with a 4-bit prefix, and the compression ratio is approximately 2.7. This is because most of the content communicated in the case of TSP is an array of array indexes

and concentrates on positive integers with small values. By preparing patterns that match the workload trends, it is possible to go beyond the gzip. However, a 4-bit prefix FPC cannot be used in this study bacause its latency is beyond the acceptable latency of the proposed method.

In a 2-bit prefix FPC, the compression ratio is lower in the IS and higher in the TSP compared to the standard 3-bit prefix FPC. In IS, the compression ratio is below one, and the size increases owing to compression. This is because of the fact that the ratio of pattern matches decreased for IS and remained the same for TSP. The shorter the prefix, the smaller the overhead, however the fewer the patterns, the more radical the trend, and the fewer the applications that can be applied.

In this study, the 3-bit prefix is used as a case study for the following evaluations. This is because the 2-bit prefix has a low compression ratio and the 4-bit prefix decreases the operating frequency in our hardware design.

Figure 5 also shows SZ [17], a lossy compression algorithm for the floating-point number, as a comparison. The error bound of SZ is set to a value that is equivalent to Bit Cut. For floating-point numbers, SZ shows a higher compression ratio than gzip and the proposed method, especially for the CG. However, we must consider that most SZ compression algorithms require floating-point operations, and it is difficult to execute a few clock cycles.

4.3 Application Execution Time

4.3.1 Evaluation Set Up

A full-system CMP simulator, GEM5 [18] v21.2, was used. We modified a detailed network model of GEM5 to accurately simulate each NoC. We have added the feature to allow an independent setting of the latency to inject packets into the network and the latency to eject packets from the network. The source code with these changes is distributed at https://doi.org/10.5281/zenodo.6460109. Shared memory 16 or 64-tiled CMP was considered, in which each processor had private L1 data and instruction caches, while the unified L2 cache banks were shared by all the processors. The details are presented in Tables 4 and 5. Four memory controllers are attached to four corners of the bottom chip. These processors, L2 cache banks, and memory controllers are interconnected through on-chip routers. A directory-based cache coherence protocol that uses three message classes (or virtual channels) is running on the NoC.

The conventional NoC uses no compression to the NoC, in which each network interface compresses and decompresses the packet, only packet compression is performed by the router to hide latency, and decompression is performed by the network interface, and that with the compression routers proposed here were compared. When data compression is performed at network interfaces, it was assumed that the existing compression causes a delay of two cycles for performing (de)compression. Each uncompressed data packet was set to a 10-flit length. We set the data-packet length to $\lceil \frac{10}{r} \rceil$ when the compression ratio is r.

Table 4 GEM5 Simulation common parameters.

Processor	x86–64
Operating Frequency	3.0 GHz
L1 I/D cache size	32 Kib /128 KiB (line:64B)
L1 cache latency	1 cycle
L2 cache latency	6 cycles
Memory size	2 GB
Memory latency	160 cycles
Router pipeline	[RC/CMP1][VSA/CMP2][ST][LT]
Protocol	MOESI directory
Control / data packet size	1 flit / 10 flits (no compression)

Table 5 GEM5 Simulation topology parameters.

NoC Topology	4×4 Mesh	8×8 Mesh
# of Cores	16	64
L2 cache size	16 MiB (assoc:4)	64 MiB (assoc:4)

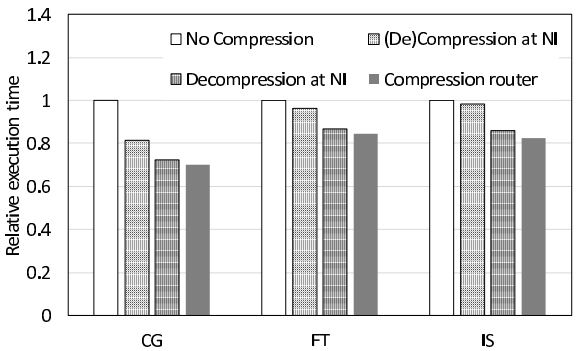


Fig. 6 The relative performance of the compression router on a full system simulation. (4×4 Mesh topology)

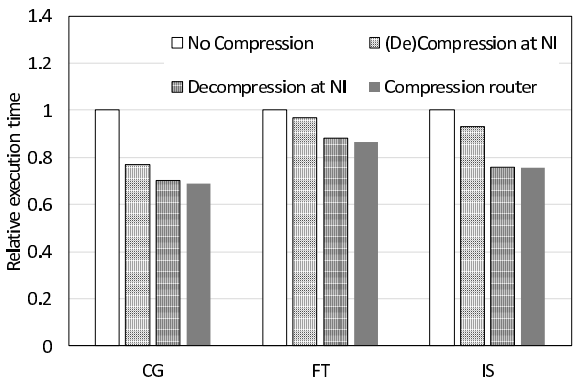


Fig. 7 The relative performance of the compression router on a full system simulation. (8×8 Mesh topology)

4.3.2 Results

Figures 6, 7, illustrate the application execution time of each NoC, and the value is relative to the baseline non-compression NoC (“Uncompressed”). In the figure, “(De)Compression at NI” is an NoC that (de)compresses each packet at the network interface, and “Decompression

at NI" is an NoC that decompresses each packet at the network interface but compresses it at the router (i.e., an alternative design of the compression router). This means that the latency of compression in case "Decompression at NI" is the same as in case "Compression router", and the latency of decompression is the same as in case "(De)Compression at NI". It is assumed that a data packet is compressed by compression ratios of 1.8, 1.6, and 1.4 on CG, FT, and IS, respectively, based on the results of the compression ratio in Figs. 4 and 5.

The compression router improves up to 31% of the execution time compared to the baseline non-compression NoC in 8×8 mesh topology. In contrast, the NoC that compresses a packet at the network interface ((De)Compression at NI) degrades up to 23% of the application execution time, and the NoC that hides the latency of only compressions (Decompression at NI) degrades up to 30%. The effect of reducing the packet length on the communication latency is relatively small compared to the negative impact of the (de)compression overhead(e.g., two cycles) on the communication latency.

Comparing the ratio of execution time for 4×4 Mesh and 8×8 Mesh, the trend is similar. This shows that the compression router is a scalable method that can speed up the system even when the NoC scale increases. However, the overhead of (De)Compression at NI is also less noticeable. This is because as the scale of the network increases, hop counts to transmit the flit and the associated router delay increase, and the compression delay ratio becomes smaller. A similar trend can be seen in Decompression at NI.

Note that our implementation of the NoC with Compression at NI attempts to compress not only data packets but also 1-flit control packets. Because a 1-flit control packet is the shortest, we would bypass compressing such packets at the network interface to improve the performance. However, we do not discuss this trade-off because it is beyond the scope of this study.

Through the full system simulation, we conclude that latency hiding for compression is essential, and the compression router is suitable for NoCs.

4.4 Network Throughput and Communication Latency

4.4.1 Evaluation Set Up

Our cycle-accurate network simulator written in C++ was used. A router model consisting of channel buffers, a crossbar, a link controller, and control circuits was used to simulate the switching fabric. Each node includes a router with a local processing element (PE).

The packet length was set to 10 or 17 flits when no data compression was applied. We simulate two synthetic traffic patterns that determine each source and destination pair: *random uniform* and *matrix-transpose*. These traffic patterns are commonly used to measure the performance of interconnection networks, as described in [2]. A node injects packets into the NoC, independently of each other. We used a 4×4 or 8×8 two-dimensional mesh with minimal routing.

This evaluation was performed not only for mesh but also for random topology. As with the mesh, 16 and 64 nodes were prepared, and the number of links is 31 and 127, respectively. Additional topologies were also evaluated to study the effect of different topologies.

As shown in the previous section's results, the target compression ratios were set to 1.4, 1.6, and 1.8 in the compression router. The traditional compression NoC, which compresses the contents at the network interface, has almost the same throughput as that in NoC with compression routers. Thus, the traditional compression NoC in the simulation was omitted; therefore, compression NoC means NoC with the compression router.

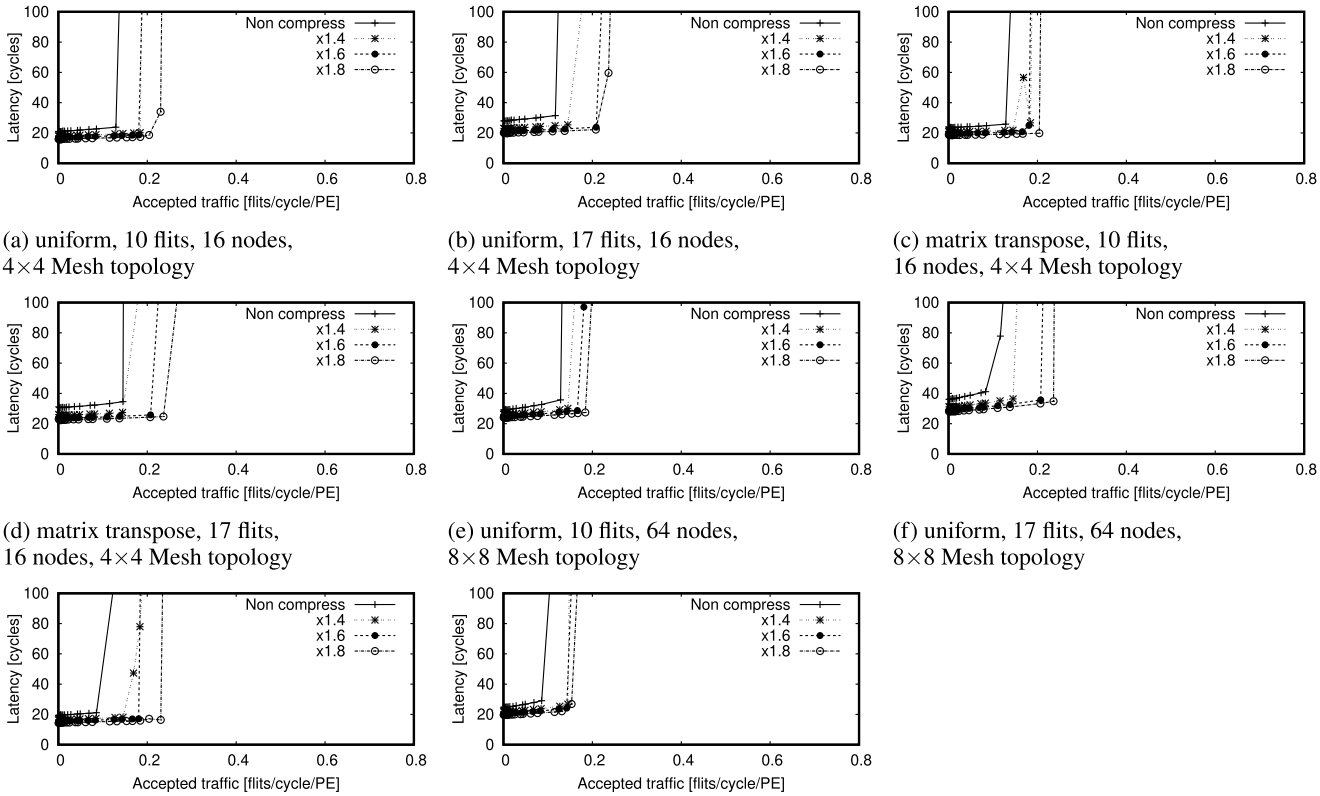
Our results show two important metrics: *communication latency* and *effective throughput*. Latency is the elapsed time between the generation of a packet at a source host and its delivery at a destination processing element. The communication latency in the simulation cycles was measured. The effective throughput is defined as the maximum accepted traffic, where the accepted traffic is the flit delivery rate.

Notice that we omit the case for the compression router without decompressors because the compression latency at a network interface does not affect the network throughput in the simulation.

4.4.2 Results

Figures 8 illustrate the relationship between communication latency and accepted traffic ratio for 16-node and 64-node NoCs. The plot of "x1.6" represents the compression NoC when achieving a compression ratio of 1.6. The compression router with any compression ratio always improves both effective throughput and communication latency. For example, when the packet length is 10 flits, the 16-node compression 4×4 Mesh NoC with a 1.8 compression ratio outperforms the conventional NoC by 77% of the effective throughput on uniform random traffic. At the light traffic load, the compression NoC outperforms up to 41% of the communication latency, because the reduced packet length decreases the injection-and-ejection latency.

As the packet size increases, the effective throughput and communication latency are significantly improved. As shown in Eq. (2), the effect of data compression on the reduction of the injection latency of a packet is proportional to the packet length when the accepted traffic is low. For example, the 16-node 4×4 Mesh compression NoC outperforms the conventional NoC by 59% and 63% of the effective throughput when the packet lengths are 10 and 17 flits, respectively, with a 1.8 compression ratio on matrix transpose traffic. There is no trend or significant difference in the effective throughput between the uniform random and matrix transpose traffic. The compression router improves the effective network throughput and communication latency for the two different traffic patterns and network sizes.



(a) uniform, 10 flits, 16 nodes, 4×4 Mesh topology

(b) uniform, 17 flits, 16 nodes, 4×4 Mesh topology

(c) matrix transpose, 10 flits, 16 nodes, 4×4 Mesh topology

(d) matrix transpose, 17 flits, 16 nodes, 4×4 Mesh topology

(e) uniform, 10 flits, 64 nodes, 8×8 Mesh topology

(f) uniform, 17 flits, 64 nodes, 8×8 Mesh topology

(g) uniform, 10 flits, 16 nodes, 16 Random topology

(h) uniform, 10 flits, 64 nodes, 64 Random topology

Fig. 8 Communication latency vs. accepted traffic for non-compression and compression NoCs

When compressing on the random topology, the same trend as for the mesh topology was observed: as the compression ratio increased, the latency decreased and the throughput increased. On the contrary, overall, the random topology had a lower upper throughput limit than the mesh topology. Although the random topology can reduce the hop count and zero-load latency compared to the mesh topology, it is more likely to cause congestion owing to the concentration of traffic when the entire network is overloaded.

4.5 Hardware Cost and Energy

4.5.1 Evaluation Set Up

First, a conventional five-port virtual-channel (VC) packet router with a fully four-stage pipelined architecture was implemented. The channel width was set to 64. Each pipeline stage had a buffer for storing one flit, and a FIFO buffer to store five flits is provided for each input VC. There were two VCs.

Second, a compression router based on the VC router was designed. To ensure that the compression logic works properly, five links were aggregated and there were 64(×5) channels on the input and output ports of the local endpoint. At the input port, additional working buffer space was reserved for storing up to 11 flits ($1 + 5 \times 2$ cycles). If the compressor missed the prediction of all the target integer values (i.e., the case of no compression), an input 10-flits packet becomes 11-flits length owing to the overhead of the prefix.

The buffer was used to perform data (de)compression. The other parameters were the same as those of the conventional VC router.

Using the Synopsys Design Compiler N-2017.09-SP1, the router was evaluated with a Nangate 45 nm open cell library. The switching activity of the running router was captured through a gate-level simulation of the synthesized router.

4.5.2 Amount of Hardware

Figure 9 illustrates the synthesis results for the network logic area of each router. “Comp. Router without Decompressor” is the area calculated by replacing the area of the input port with a decompressor in “Comp. Router” with that of a conventional input port with nothing attached. The compression router we design supports two compression algorithms, the 3-bit FPC and the lossy compression that cuts 2 to 28 bits of the floating-point number. It can dynamically control how many bits are trimmed in the lossy compression. The compression router works at 588 MHz and we confirm that VSA and decompression logic are on a critical

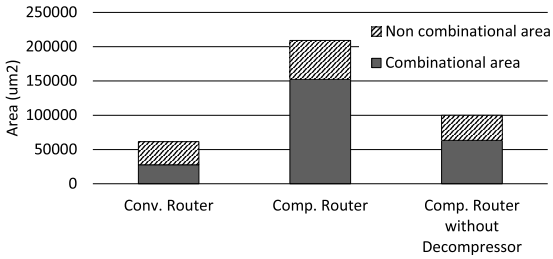


Fig. 9 Hardware amount for routers.

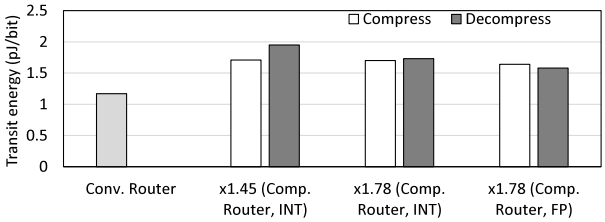


Fig. 10 Flit transfer energy at routers.

path in conventional routers and compression routers. In the case of 32-bit cut, the float data decompression operation is simplified and the decompression logic is left out of the critical path. The Bit Cut circuit should be used when the precision required by the application satisfies the precision provided by the Bit Cut circuit. The compression router can support multiple precisions by controlling the multiplexer with a header flit.

The results show that the additional hardware of the compression router is not negligible. In addition, most of them are combinational circuits, indicating that the increase in buffers is relatively small. However, compared with the area of CMPs, the amount of hardware required for the compression router is not very large. For example, under 45 nm CMOS technology, Intel Single-chip Cloud Computer (SCC) [19] has 48 cores on 567mm^2, and SPARC64 VIIIfx has eight cores on 22.7mm × 22.6mm. When the compression router is applied to each core on the CMPs, the total area of the compression routers would be only a few percentages on their chip areas. Our evaluation is on the same 45 nm generation scale, although not exact because we use different PDKs.

On the other hand, about 79% of the increase in area from the conventional router is the area of the decompressors. This is due to a large number of decompressors. If the decompressor is placed on the network interface instead of the input port of the router, the network interface will have a reasonable increase in area. If the number of routers is equal to the number of network interfaces, 57% of the increase in the area of the compression router is the overhead required to hide the decompression latency. However, as noted above, the router itself is a relatively small component, and whether this is an acceptable trade-off in the overall processor depends on the area of the processing element and cache.

4.5.3 Energy to Transmit a Flit

Figure 10 illustrates the energy consumption required to transmit a flit to each router. In the figure, "Conv. router" stands for energy consumption when a 10-flit packet is forwarded on the conventional router, while "x1.45 (Comp. Router, INT)" indicates that when the compression router compresses an input 10-flit packet whose data body is an integer value of 1.45. Because the conventional router does not perform compression, it outperforms the compression router in terms of energy. The decompression operation requires a more complex operation, that is, identifying each symbol's position and rounding for floating-point numbers, than compression. Thus, the energy at decompression is higher than that at compression. Another finding is that because the lossy compression for floating-point numbers simply discards LSBs, its energy is lower than the compression for integer numbers.

This increase is relatively small in terms of the overall chip. For example, in the Intel SCC, routers are only 12.1 W out of 125 W of power consumption in the fully loaded state, which is 10% of the total [19]. Therefore, the power increase in the compression router is expected to be only a few percent of the total chip power.

5. Conclusions

To reduce the communication latency, we proposed a compression router that (de)compresses the contents of an incoming packet for NoCs. To hide the (de)compression overhead from the communication latency, the compression router performs the (de)compression operation of an incoming packet before completing switch arbitration.

The compression router provided up to x1.8 compression ratios of communication data of four parallel applications, conjugate gradients, fast Fourier transform, integer sort, and traveling salesman problem, on a conventional CMP. A full-system simulation illustrated that shortening the packet length improved the execution time of parallel applications on a CMP up to 31%. The cycle-accurate network simulation showed that the compression NoC outperformed up to 41% of the communication latency, and it improved the effective network throughput by 63% compared to the conventional NoC.

The disadvantage is an increase in the amount of hardware and the energy of the router. In this context, we also explored alternative compression router design, off-loading the decompressor onto a network interface. It decreased the compression-router area by 57% at the expense of a moderate increase in communication latency.

Acknowledgments

This work was supported by JSPS KAKENHI 19H01106 and VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration with SYNOPSYS Corporation.

References

[1] L.-S. Peh and W.J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA), pp.255–266, Jan. 2001.
[2] W.D. Dally and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2003.
[3] A.R. Alameldeen and D.A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," Technical Report 1500, Computer Sciences Dept. UW-Madison, April 2004.
[4] F. Betzel, K. Khatamifard, H. Suresh, D.J. Lilja, J. Sartori, and U. Karpuzcu, "Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems," ACM Comput. Surv., vol.51, no.1, pp.1:1–1:32, 2018.
[5] M.F. Reza and P. Ampadu, "Approximate communication strategies for energy-efficient and high performance noc: Opportunities and challenges," Great Lakes Symposium on VLSI, GLSVLSI, pp.399–404, 2019.
[6] R. Das, A.K. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M.S. Yousif, and C.R. Das, "Performance and power optimization through data compression in network-on-chip architectures," International Symposium on High-Performance Computer Architecture (HPCA-14), pp.215–225, 2008.
[7] Y. He, H. Matsutani, H. Sasaki, and H. Nakamura, "Adaptive data compression on 3d network-on-chips," IPSJ Online Transactions, vol.5, pp.13–20, 2012.
[8] N. Niwa, Y. Shikama, H. Amano, and M. Koibuchi, "A case for low-latency network-on-chip using compression routers," The Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2021.
[9] B. Dickov, M. Pericàs, P.M. Carpenter, N. Navarro, and E. Ayguadé, "Analyzing performance improvements and energy savings in infiniband architecture using network compression," IEEE 26th International Symposium on Computer Architecture and High Performance Computing, pp.73–80, 2014.
[10] R. Filgueira, D.E. Singh, A. Calderón, and J. Carretero, "Compi: enhancing mpi based applications performance and scalability using run-time compression," European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting, vol.5759, pp.207–218, Springer, 2009.
[11] A.R. Alameldeen and D.A. Wood, "Adaptive cache compression for high-performance processors," International Symposium on Computer Architecture (ISCA), pp.212–223, 2004.
[12] R. Boyapati, J. Huang, P. Majumder, K.H. Yum, and E.J. Kim, "Approx-noc: A data approximation framework for network-on-chip architectures," ACM SIGARCH Computer Architecture News, vol.45, no.2, pp.666–677, June 2017.
[13] N. Niwa, H. Amano, and M. Koibuchi, "Boosting the performance of interconnection networks by selective data compression," IEICE Transactions on Information and Systems, vol.E105-D, no.12, pp.2057–2065, 2022.
[14] Y. Jin, K.H. Yum, and E.J. Kim, "Adaptive data compression for high-performance low-power on-chip networks," 2008 41st IEEE/ACM International Symposium on Microarchitecture, pp.354–363, 2008.
[15] rachit95arora, "A parallelised implementation of genetic algorithms for the travelling salesman problem using openmp." https://github.com/rachit95arora/travelling-salesman-openmp
[16] D. Fujiki, K. Ishii, I. Fujiwara, H. Matsutani, H. Amano, H. Casanova, and M. Koibuchi, "High-bandwidth low-latency approximate interconnection networks," International Symposium on High Performance Computer Architecture (HPCA), pp.469–480, Feb. 2017.
[17] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," international parallel and distributed processing symposium (ipdps), pp.730–739, 2016.
[18] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, and D.A. Wood, "The gem5 Simulator," ACM SIGARCH Computer Architecture News, vol.39, no.2, pp.1–7, May 2011.
[19] J. Howard, "A 48-core IA-32 processor with on-die message-passing and DVFS in 45nm CMOS," 2010 IEEE Asian Solid-State Circuits Conference, pp.108–109, Feb. 2010.

Naoya Niwa received the BE and ME degrees from Keio University, Yokohama, Japan, in 2018 and 2020, respectively. He is currently a Ph.D. student in the Department of Information and Computer Science, Keio University. His research interests are Approximate Computing and Network-on-Chips.

Yoshiya Shikama received the BE and ME degrees from Keio University, Yokohama, Japan, in 2020 and 2022, respectively. He is currently working on development work at ANRITSU CORPORATION.

Hideharu Amano received Ph.D. degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.

Michihiro Koibuchi received the BE, ME and Ph.D. degrees from Keio University, Yokohama, Japan, in 2000, 2002 and 2003, respectively. Currently, he is a professor in National Institute of Informatics and the Graduate University of Advanced Studies, Tokyo, Japan. His research interests include the areas of high-performance computing and interconnection networks. He is a senior member of the IEICE, IPSJ and IEEE.