Low-Latency High-Bandwidth Interconnection Networks by Selective Packet Compression

Naoya Niwa
Keio University
Yokohama, Japan
naoya@am.ics.keio.ac.jp

Hideharu Amano
Keio University
Yokohama, Japan
hunga@am.ics.keio.ac.jp

Michihiro Koibuchi
National Institute of Informatics
Chiyoda-ku, Japan
koibuchi@nii.ac.jp

***Abstract*—Interconnection network ideally transfers the maximum amount of communication dataset within the least amount of time to fully exploit the parallelism of target applications on parallel computer systems. To this goal, we propose a selective data-compression interconnection network. Data compression virtually increases the effective network bandwidth, while each compute node introduces additional latency overhead to perform (de-)compression operation to end-to-end communication latency. To minimize the effect of the compression latency overhead on the end-to-end communication latency, we selectively apply a compression technique to a packet. The compression operation is taken for long packets and is also taken when network congestion is detected at a network interface. Evaluation results show that simple lossless and lossy compression algorithms have up to 3.0 and 1.8 compression ratios for integer and floating-point communication data in some parallel applications, respectively, while the lossy compression algorithm successfully satisfies the required quality of results. Through a cycle-network simulation, the selective compression method using the above compression algorithms improves by up to 46% the network throughput with the moderate increase of the communication latency of short packets.**

I. Introduction

A way to virtually increasing the effective network bandwidth is by reducing the redundancy of communication data as possible. Some parallel scientific applications repeatedly generate a similar communication dataset, e.g., consecutive values of the array in a Fluid computation application [1]. For these communication patterns, each network interface of a compute node has a chance to efficiently reduce a packet length by compressing the data.

An essential design issue of a compressor is to minimize the operation latency overhead at a network interface. Interconnection networks are latency-sensitive, i.e., time to across the system, since their communication latency strongly affects the application performance.

Assuming that a packet that consists of L flits is compressed by C compression rate, and it is transferred through H intermediate switches. The zero-load communication latency T is calculated as follows.

$$T = T_{lt} \times (H+1) + T_{switch} \times H + L/C + D, \qquad (1)$$

where T_{switch}, T_{lt}, and D are the latencies of switch, link transfer, and the sum of compression and decompression overhead at network interfaces, respectively.

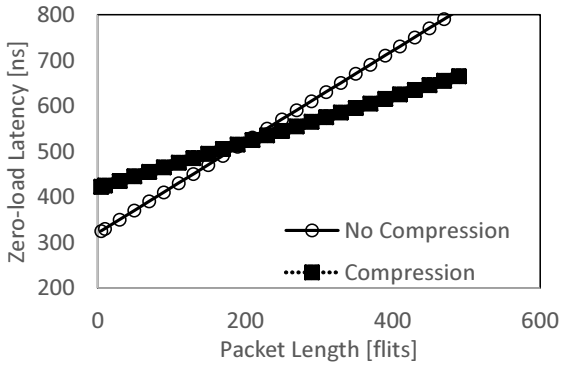


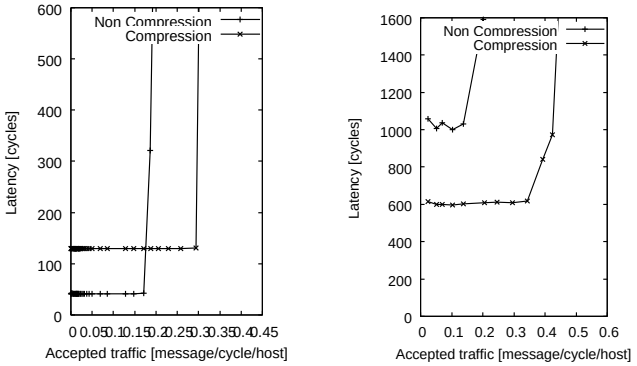
Fig. 1. Zero-load communication latency vs. packet length for data compression.

Figure 1 plots the relationship between a packet length versus zero-load communication latency according to Equation 1, where H, T_{switch}, T_{lt}, and D are set to 4, 60ns, 20ns, and 100ns, respectively. The compression ratio is set to 2.0.

We observe that the data compression is advantageous as an original packet length becomes longer. The data compression shortens a packet length, and it reduces the injection time in the zero-load communication latency at the expense of the (de)compression latency overhead.

For even short packets, the compression is efficient for improving the end-to-end communication latency when the network is congested. Figures 2 (a) and (b) illustrate the detailed simulation results of a cycle-accurate network simulation corresponding to the short- and long-packet cases in Figure 1, respectively. We assume random (uniform) traffic on 8×8 2-D mesh. The (de)compression operational overhead at a network interface is set to 100 cycles, and the compression ratio is set to 2.0. The other simulation parameters are described in Section IV.

The results are consistent with Figure 1, and the data compression always reduces the end-to-end communication latency when the network load is high. Compressed packets mitigate the network congestion by reducing the total amount of flits around the congestion. When the latency penalty blocked by another packet becomes relatively small, it leads to lower communication latency. Another finding is that the data



(a) Short packet (10 flits) (b) Long packet (400 flits)

Fig. 2. Accepted traffic vs. communication latency on a 64-switch network.

compression always improves the network throughput defined by the maximum amount of the accepted traffic.

Through these observations, in this study, we exploit a technique to selectively apply a compression technique to a packet by taking into account the network congestion and packet length to achieve lower communication latency and higher throughput. To detect the network congestion, each network interface counts the number of flits recently received and monitors the utilization of the input packet queue. Each source network interface then decides whether incoming packets would be compressed or not.

The contributions of this study are listed as follows.

- A simple lossless compression algorithm obtains up to 3.0 compression ratio for the communication data in integer sort and ant colony optimization that treat mostly integer data. A lossy compression algorithm truncating multiple bits obtains up to 1.8 compression ratio for floating-point communication data in CG and FT with the required quality of results of NAS Parallel Benchmarks (MPI version).
- We propose the selective packet compression method that selectively compresses packets according to the network load. Our network simulation results show that the selective compression method provides similar low communication latency to that of the counterpart original interconnection network, and a similar high network throughput to that of the ever-compression interconnection network.

The remainder of this paper is organized as follows. Section II describes background information on the data compression on interconnection networks. Section III presents a technique to selectively apply a compression method to packets. Section IV presents simulation results of the selectively compression method in terms of the compression ratio and network performance. Section V concludes this study with a summary of our findings.

II. Background

A. Data Compression for Interconnection Networks

There are a large number of data compression algorithms for various fields, including archiving and multimedia. The unique requirement of interconnection networks is to perform a low-latency compression operation. Interconnection networks are latency-sensitive, i.e., one-microsecond communication across the system is required. Thus, the compression operation latency should be the order of nanoseconds.

The target communication data usually form a one-dimensional array of floating-point numbers and integer numbers. In the context of the low compression latency overhead, a compression technique considered for data in the cache of a processor is attractive for interconnection networks as treated in the FPC [2]. FPC is a lossless compression that only picks up and compresses frequently appeared values. Reversely, the remaining values are uncompressed and are transferred with additional prefix bits.

By contrast, floating-point numbers are often not compressible by a general-purpose lossless data compression algorithm. A lossy compression is a way to efficiently compress them at the expense of the accuracy. For example, SZ attempts to approximate the communication data value by either a linear or pseudo-approximate quadratic curve based on the values of the previously transferred data [3]. A similar approach is taken in the data compression on inter-FPGA networks [1]. It reported that its custom design interestingly obtains an extremely high-ratio lossless compression for floating-point numbers in fluid dynamics applications.

Another example of a lossy compression is a bit cut of a floating-point number. The values are simply compressed by trimming its lower bits. It is reported that the energy efficiency of an interconnection network can be improved, although the quality of results is not assessed on MPI parallel applications [4].

In this study, we apply FPC for integer numbers and the bit cut for floating-point numbers, respectively.

B. Congestion Control

The increase in network load leads to its congestion. Once the congestion occurs, it can cause a performance degradation. There are two approaches to handle the congestion: end-to-end and switch-to-switch. A typical way to end-to-end congestion control is the use of injection limitation and throttling [5]. For example, each node judges whether the congestion occurs or not independently using the utilization of queues. When detecting the congestion, a node stops inserting packets temporarily. A switch-to-switch congestion control relies on adaptive routing.

To detour the congestion, a bypassing path should be taken by an adaptive routing [6]. However, J. Won et al. pointed out the difficulty in the congestion problem in large interconnection networks. It is difficult for a switch forwarding packets to detect the congestion when the congested region is far [6]. A significant communication latency between routers and the temporal congestion caused by the instability of the routes by using the adaptive routing often introduce the confusing values of the congestion indicators, such as queue length. To override the problem, they proposed a history-window approach, which uses the number of flits passed in the last fixed cycles as

a metric. With the approach, they improved the latency and throughput of adaptive routing in the Dragonfly network.

All the above congestion controls result in the reduction of the number of flits around the congestion. In this context, the selectively compression method in this study can be categorized as a congestion control, because the selectively compression method decreases a packet length when the network is congested.

III. Selective Packet Compression Method

A. Compression Algorithm

We apply an existing lossless FPC compression for integer values of communication data, and apply a lossy LSB-cut compression for floating-point values in the selectively packet compression method. Since there are various compression algorithms, we tune them for interconnection networks based on the method described in [7].

Compression is performed for the message payload at a source network interface, and decompression is performed at a destination network interface. We do not compress the header information that is essentially needed for routing and flow control. Since no changes are made to the information necessary to control packets, intermediate network switches can be used as existing ones.

1) FPC for Integer Communication Data: We used a compression algorithm that adapts FPC for communication data of MPI parallel applications. Assume that each application controls compression and decompression through the type information specified in MPI when sending data.

The compression target is 32-bit integer arrays. It is performed when the Datatype is set to "MPI_INT" for MPI communication.

The array is divided into 32-bit integers and each value is compared with the prepared patterns in Table I. We only compress a sequence of zeros from MSB.

TABLE I
Compression pattern of FPC

Priority	Prefix	Pattern encoded	Data size (prefix + data)
1	101	18-bit zero run	17bit
2	100	17-bit zero run	18bit
3	011	16-bit zero run	19bit
4	010	15-bit zero run	20bit
5	001	14-bit zero run	21bit
6	000	13-bit zero run	22bit
7	110	(unused)	
8	111	(non-compress)	35bit

A 3-bit prefix identifies each pattern. We take the longest *zero* matching to minimize the compressed data size by the priority in Table I. The "non-compress" pattern is a case in which input value did not match any of the patterns. The amount of data then increases from 32 bits to 35 bits.

2) LSB-Cut for Floating-Point Communication Data: Here, the compression target is 64-bit IEEE 754 double-precision floating-point arrays. We borrow a lossy compression technique called LSB-cut from [4]. It truncates multiple least-significant bits.

In MPI communication, if "MPI_DOUBLE" or "MPI_DOUBLE_COMPLEX" is specified in Datatype, compression is performed. The number of truncated bits is set by calling the original API added to MPI at the time of initialization. On the receiver side, the truncated bits are padded to the received data by the pattern of 0b100…0. The value is close to the median of the possible values by truncating.

B. Selective Packet Compression Algorithm

We introduce a selective compression algorithm for compressing a part of packets considering the packet length and network congestion. The goal is to maximize the network throughput within the allowed communication latency. We dynamically select the two lossless and lossy compression algorithms described in the previous subsection.

Each network interface performs the selective compression algorithm independently. The selective compression algorithm for communication data is described in Algorithm 1.

Algorithm 1 Compression selection algorithm.

Require: Number of received flits in the latest time period (N), number of packets in input queue without the target packet (Q), and an incoming packet length (L),
Ensure: Selected compression algorithm.

```
1: procedure :
2:     Pick up a packet from injection queue
3:     if L ≥ L_th then
4:         if data type = int then
5:             FPC algorithm
6:         else if data type = float then
7:             LSB-cut algorithm
8:         else
9:             No compression
10:        end if
11:    else if (N > 0) or (Q > 0) then
12:        if data type = int then
13:            FPC algorithm
14:        else if data type = float then
15:            LSB-cut algorithm
16:        else
17:            No compression
18:        end if
19:    else
20:        No compression
21:    end if
22: end procedure
```

The compression selection algorithm compresses long packets unconditionally and selects short packets using a conditional expression $N > 0$ or $Q > 0$. The threshold of the packet length, L_{th}, is roughly computed by Equation 1. The conditional expression dynamically identifies the network congestion, and it should depend on the network implementation.

The hardware implementation of the selective compression method at a network is not discussed in this paper. However, it is not impractical, as reported in [8].

IV. Evaluation

A. Compression Ratio

We modified MPI communication in each program to compress the communication data at the transmission and decompress at the reception. We ran the modified application on the environment shown in Table II in two processes on two PCs, and measured the compression ratio by tracing the communication. The compression ratio is calculated as "total size of the original data / total size of the compressed data". We also evaluated gzip 1.10 and SZ 2.1 [9] for the comparison. SZ was applied only to data for floating point numbers, and the precision constraint was set close to LSB-Cut. However, it is not practical to use SZ for streaming compression because it requires long buffering and complex computation, and the latency is extended by an order of magnitude compared to LSB-Cut. We evaluated it as the compression ratio of practical lossless compression without time constraints.

The compressed data size was computed by excluding metadata such as headers and CRC from the communication data size to make the comparison fair.

TABLE II
Environment for evaluating compression ratio.

CPU	Intel Core i7–3770
Memory	32GiB
NIC	Mellanox ConnectX-5
Operating system	Ubuntu 18.04 LTS
Linux kernel	5.0.0-36
MPI	OpenMPI 2.1.1-8

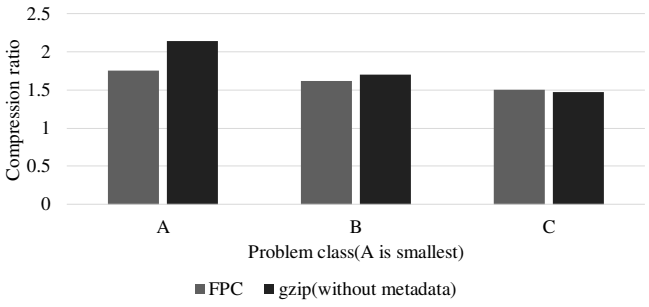


Fig. 3. Compression ratio of NPB IS (integer).

Figures 3 and 4 illustrate the compression ratio of the FPC for communication data. We use integer sort (IS) of NAS Parallel Benchmark (MPI version) [10] and ant colony optimization (ACO) for the traveling sales problem (TSP) with 105-city named "lin105" in TSPLIB [11]. In the ACO program, each process exchanges an index value of an array that stores a city identifier. Since the index value tends to be small, FPC achieves 3.0 compression ratio. Both results illustrate that FPC has a similar average compression ratio to that in gzip.

Figures 5 and 6 illustrate the compression ratio of LSB-Cut for communication data. We preliminarily investigate the

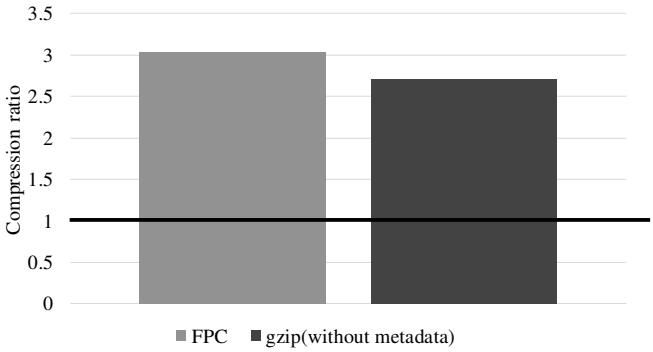


Fig. 4. Compression ratio of ant colony optimization (integer).

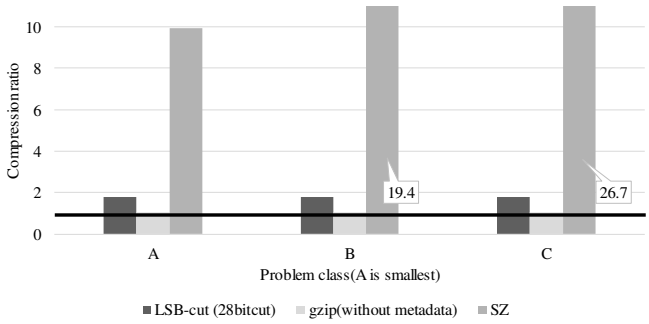


Fig. 5. Compression ratio of NPB CG (floating-point).

maximum number of truncating bits to meet the required quality of results of two applications. We found that 28-bit truncating from LSB for IEEE 754 double-prevision floating-point number works well on both parallel applications. Thus, the compression ratio becomes approximately 1.8.

However, the compression ratio of LSB-Cut is much lower than that of SZ. The compression ratio of SZ is obtained on a per-message basis at the application level, which is a big advantage compared to the network level like LSB-Cut, especially when exchanging long data. However, SZ has to pay a large cost regarding latency.

B. Network Throughput and Latency

1) Environment: We use a cycle-accurate network simulator written in C++. A switch model consisting of channel buffers, a crossbar, a link controller and the control circuits is used to simulate the switching fabric. On a conventional packet switch, a header flit transfer requires four cycles that include the routing, virtual-channel allocation, switch allocation, and flit transfer from an input channel to an output channel through a crossbar. We use dimension-order routing on 2-D meshes and Up*/Down* routing on random topologies. The compression ratio is set to 2.0 from the results of the compression ratio in Section IV-A. The latency is set to 100 cycles, which is constant regardless of the conditional branching in the selection algorithm, assuming a simple implementation in hardware.

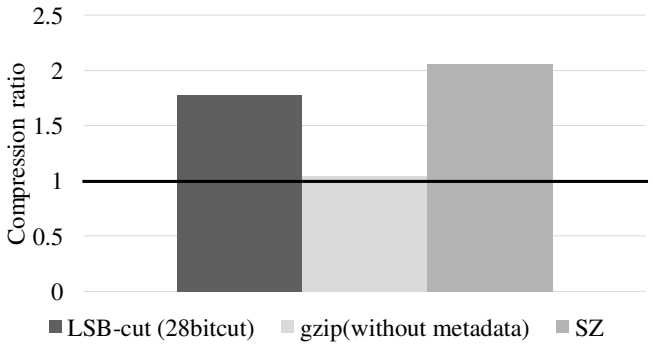


Fig. 6. Compression ratio of NPB FT (floating-point).

Unless particularly mentioned, the switch latency is assumed to be four cycles. In Algorithm 1, we counted the number of recently received flits at the interval of 1,000 cycles.

Notice that the compression selection algorithm compresses all longer packets than L_{th}. We omit the evaluation when all packets are longer than L_{th}. Based on Equation 1, L_{th} was set to 200 in our simulation, and packets longer than 200 flits are compressed unconditionally. In the evaluation, the packet length is assumed to be uniformly distributed from 10 to 200.

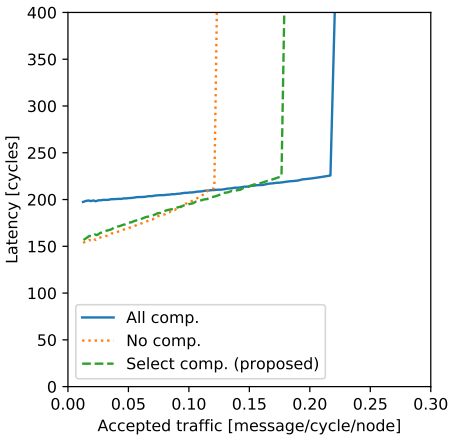
Three synthetic traffic patterns, uniform (random), matrix transpose, and bit reversal, are used [5]. Each compute node generates a packet independently. Our results use two important metrics: *latency* and *throughput*. The latency is the elapsed time between the generation of a packet at a source compute node and its delivery at a destination compute node. We measure latency in simulation cycles. The throughput is defined as the maximum accepted traffic, that is, the maximum flit delivery rate.

2) Simulation Results: We evaluate the selective compression method, ever-compression method, and the original non-compression method.

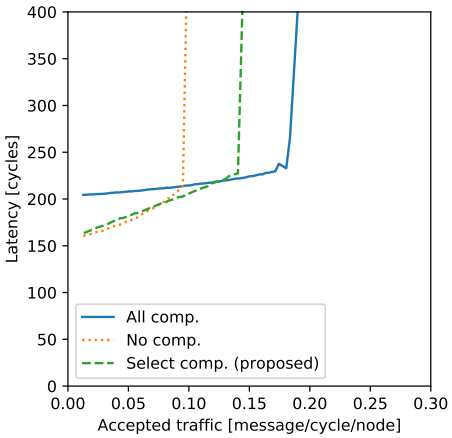
Figures 7 illustrates the relationship between communication latency and accepted traffic rate of each method for each traffic pattern. The network topology was 8×8 2-D mesh. The selective compression method outperforms up to 38% network throughput, and it has a similar low communication latency to the non-compression method on all the traffic patterns. The ever-compression method outperforms by up to 26% network throughput to the original non-compression method. However, it has 1.3-times larger communication latency at the low traffic load. We thus conclude that the selective compression method is advantageous.

Figure 8 illustrates the relationship between compressed packets number percentage and accepted traffic rate. The conditions are same as those in Figure 7. The ratio of compressed packets is mostly proportional to the accepted traffic. No packets are compressed at low load, while almost half of the packets are compressed at high load. It can be seen that the selective compression method successfully selects the compression target packets depending on the traffic load.

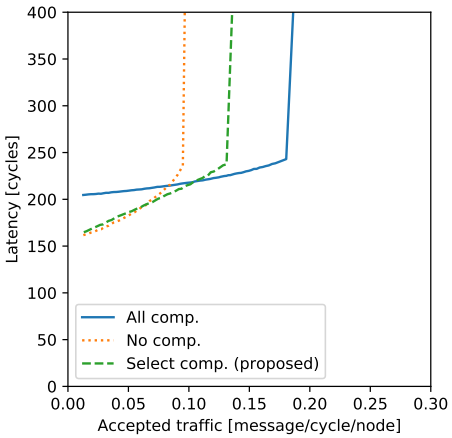
We shift to investigate the impact of the network topology, network size, and the switch delay on the selective compression method. Figure 9 illustrates the relationship between communication latency and the accepted traffic of each compression method on mesh, torus, and random network



(a) Uniform random

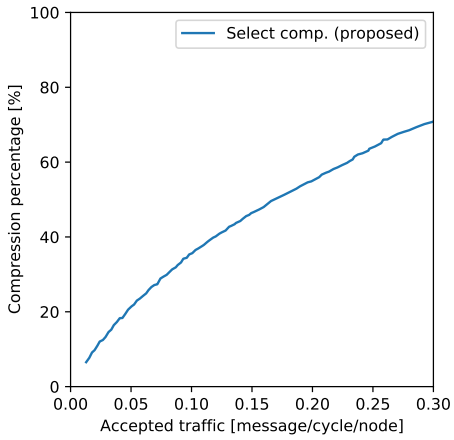


(b) Matrix transpose

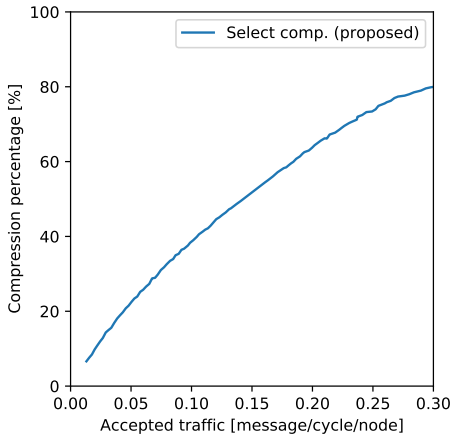


(c) Bit reversal

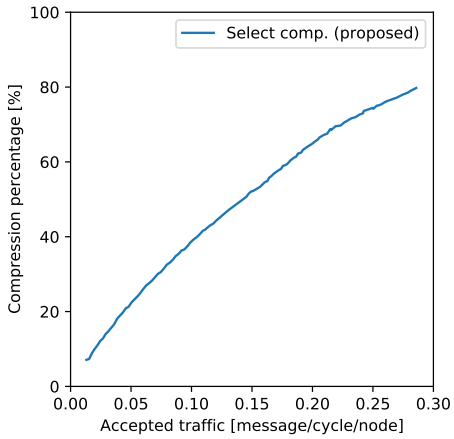
Fig. 7. Accepted traffic vs. communication latency on compression selection algorithms (8×8 Mesh).



(a) Uniform random

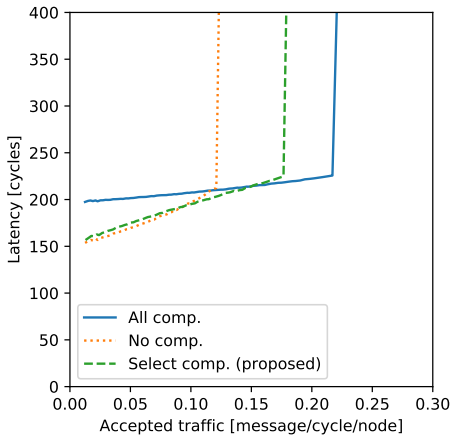


(b) Matrix transpose

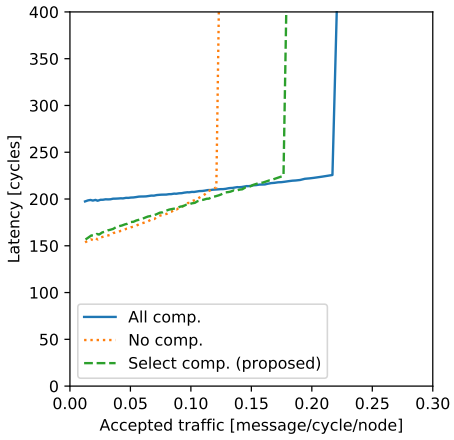


(c) Bit reversal

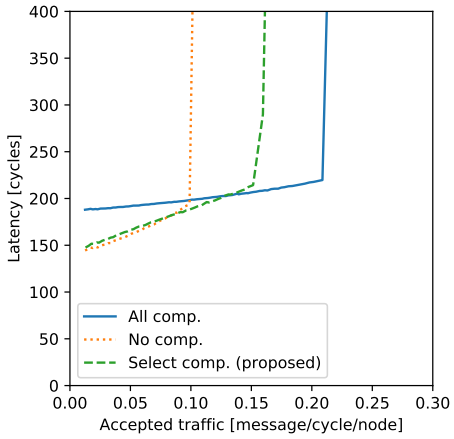
Fig. 8. Accepted traffic vs. compressed packets number percentage on compression selection algorithms (8×8 Mesh).



(a) 2-D Mesh



(b) 2-D Torus.

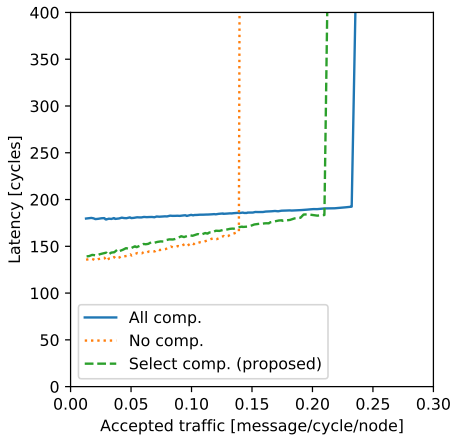


(c) Random.

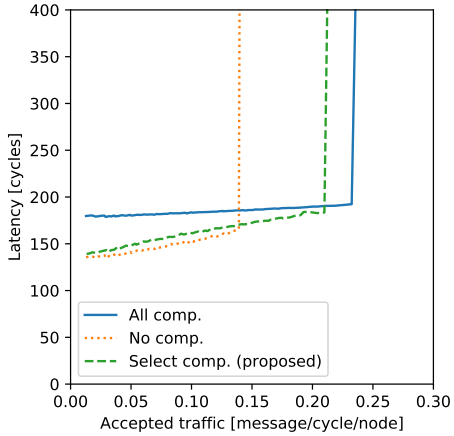
Fig. 9. Accepted traffic vs. communication latency on compression selection algorithms (64 nodes).

topologies with a degree of four. The uniform traffic is taken. Our main finding is that the network topology hardly affects the performance of the compression method. The selectively compression method successfully achieves low latency and high throughput on all the network topologies considered in this evaluation.

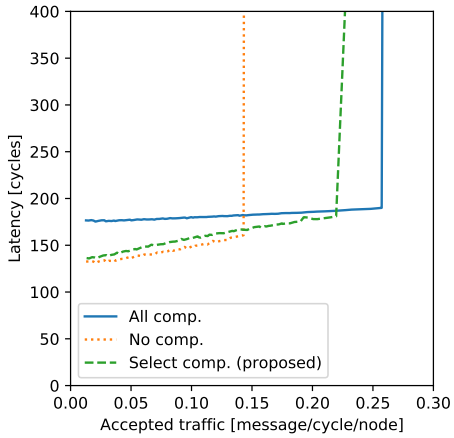
Figure 10 illustrates the communication latency and accepted traffic of each method on 16 nodes. We used uniform traffic. The selective packet compression achieves up to 46%



(a) 2-D Mesh.



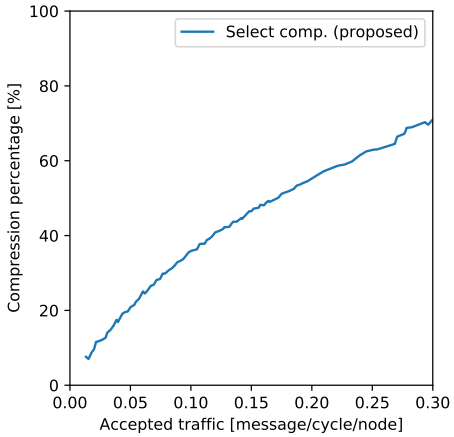
(b) 2-D Torus.



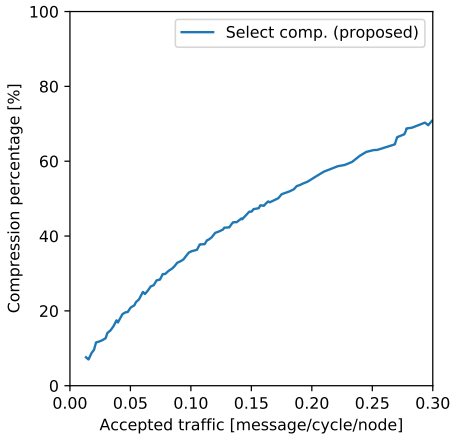
(c) Random.

Fig. 10. Accepted traffic vs. communication latency on compression selection algorithms (16 nodes).

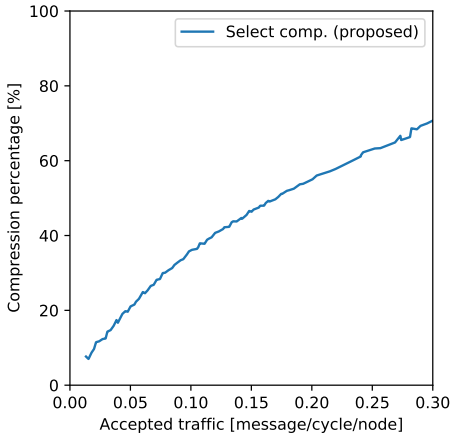
higher network throughput than the original non-compression method. Their impact becomes higher than that in 64-node interconnection networks. This is because the smaller network has lower hop counts (H) in Equation 1, and the compression influence (L/C) relatively becomes crucial in the communication latency.



(a) 2-D Mesh.



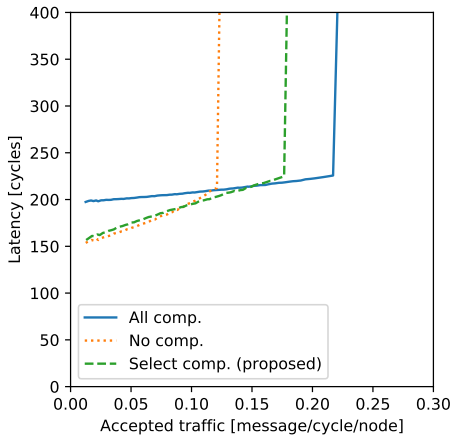
(b) 2-D Torus.



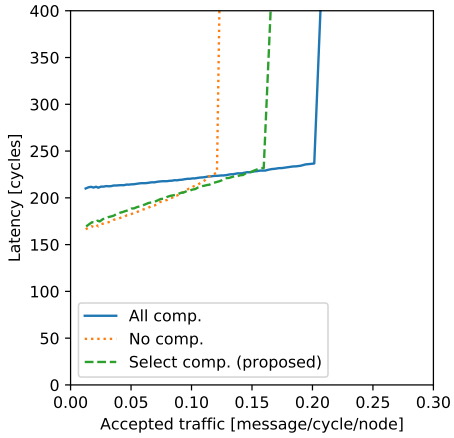
(c) Random.

Fig. 11. Accepted traffic vs. compressed packets percentage on compression selection algorithms (16 Nodes).

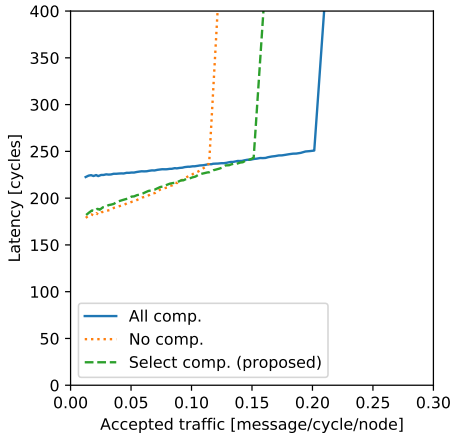
We illustrate the ratio of compression packets in Figure 11. Just before the network is saturated, the selective packet compression compresses 60% of packets resulting to the high network throughput.



(a) Switch latency = 5.



(b) Switch latency = 7.



(c) Switch latency = 9.

Fig. 12. Accepted traffic vs. communication latency on different network switch latency (8×8 Mesh).

Figure 12 illustrates the relationship between accepted traffic and the communication latency on different network switch latencies. We take different switch latencies, which leads to different zero-load communication latency in Equation 1. That is, T_{switch} is varied in Equation 1. As the switch latency becomes large, the effect of the compression relatively decreases in Equation 1. However, the simulation results interestingly illustrate that its effect is limited in the selective compression method.

V. Conclusions

We proposed a selective compression technique for an incoming packet to achieve a low-latency, high-throughput interconnection network of parallel computers. The compression operation is taken for long incoming packets, and is also taken when a network congestion is detected at a network interface of a compute node.

Since an interconnection network is latency-sensitive, we have to select a fast compression algorithm. Therefore, we selected a compression algorithm according to the data type, applying lossless compression for integers and lossy compression for floating-point value. A common lossless compression algorithm, e.g., gzip, usually provides a low compression rate for a floating-point value with a high operational overhead. We aggressively apply a lossy compression truncating multiple bits from LSBs, while maintaining the quality of results of parallel applications.

Evaluation results show that simple lossless and lossy compression algorithms have up to 3.0 and 1.8 compression ratios for integer and floating-point communication data of some parallel applications, respectively. Through a cycle-network simulation, the selective compression method with 2.0 compression ratio improves by up to 46% the network throughput with the moderate increase of the communication latency of short packets.

Acknowledgements

This work was supported by JSPS KAKENHI 19H01106 and JST SPRING, Grant Number JPMJSP2123.

References

[1] T. Ueno, K. Sano, and S. Yamamoto, "Bandwidth compression of floating-point numerical data streams for fp ga-based high-performance computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, pp. 1–22, May 2017.

[2] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme f or l2 caches," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2004.

[3] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *2016 ieee international parallel and distributed processing symposi um (ipdps)*, pp. 730–739, IEEE, 2016.

[4] B. Dickov, M. Pericàs, P. M. Carpenter, N. Navarro, and E. Ayguadé, "Analyzing performance improvements and energy savings in infiniband architecture using network compression," in *IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, pp. 73–80, 2014.

[5] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: an engineering approach*. Morgan Kaufmann, 2002.

[6] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott, "Overcoming far-end congestion in large-scale networks," in *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 415–427, 2015.

[7] N.Niwa, Y. Shikama, H. Amano, and M. Koibuchi, "A case for low-latency network-on-chip using compression routers," in *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP*, pp. 134–142, 2021.

[8] H. Shimura, H. Noda, and H. Amano, "C4: an fpga-based compression algorithm for expether," in *Sixth International Symposium on Computing and Networking, CANDAR Workshops*, pp. 356–362, IEEE Computer Society, 2018.

[9] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 438–447, 2018.

[10] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performane," in *NAS Technical Report NAS-99-011*, Oct. 1999.

[11] G. Reinelt, "Tsplib–a traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.