Accelerating MPI Communication Using Floating-point Compression on Lossy Interconnection Networks

Yao Hu, Michihiro Koibuchi
National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan 101-8430
{huyao, koibuchi}@nii.ac.jp

***Abstract*—Approximate communication has arisen as a new opportunity for improving the efficiency of communication in parallel computer systems, which can significantly reduce the communication time by transmitting partial or imprecise messages. In this study, we explore application-level approximate communication techniques on lossy high-performance interconnection networks that leave bit flips. Although existing compression techniques do not assume lossy interconnection networks, our approximate-communication challenge is to co-design a lossy floating-point compression algorithm and a critical bit-flip recovery scheme optimized to a given bit error rate (BER) on a target lossy interconnection network. Our objective is to transfer the maximum amount of approximate data over the least amount of compression overhead and bit-flip recovery time. Our scheme is implemented into several representative communication-intensive MPI (Message Passing Interface) applications, and it is shown that our approximate communication scheme effectively speeds up the total execution time without much loss in the quality of the result.**

***Index Terms*—Interconnection network, parallel computing, approximate computing, lossy compression, error check**

I. Introduction

A way to virtually increase the network bandwidth is the reduction of redundancy of communication data themselves on lossy interconnection networks. In this context, approximate computing [1] is gaining traction as a computing paradigm to introduce a new trade-off between the quality and the speed. The parallel applications can improve the system efficiency while retaining an acceptable level of accuracy. In these parallel applications, a large amount of floating-point datasets are frequently exchanged between compute nodes via an interconnection network.

A large fraction of applications running on supercomputing systems primarily use MPI (Message Passing Interface) parallelism to explore execution efficiency. It is reported that a reasonably high number of applications spend more than half their time in MPI [2]. Moreover, the fraction of time spent on communication increases significantly with the number of processes [3]. This large communication overhead limits the scalability of parallel applications. Hence, speeding up the MPI communication improves the performance of applications. To this end, data compression brings shorter MPI communication time. For compression of floating-point datasets, a simple lossless algorithm generally brings a low compression ratio, while a complicated lossy algorithm usually introduces a large latency overhead to obtain a high compression ratio. Because interconnection networks are latency-sensitive, e.g., less than one microsecond for inter-process communication in parallel computer systems [4], it is harmful to append heavy load of data compression on MPI communication.

In this study, we co-design an application-level fast lossy compression algorithm and a critical bit-flip recovery scheme optimized to a given bit error rate (BER) on lossy interconnection networks.

II. Bit-wise Lossy Floating-point Compression

A. Lossy Bit-zip Compression

1) Bit Rounding: We employ a lossy prediction-based compression algorithm [5] to exploit the continuity of numerical floating-point data. If the prediction fails, we round the floating-point value d_i. Since the LSBs in the *mantissa* have less impact on the value of a floating-point number, we discard the LSBs while maintaining the error bound. In other words, we only retain the necessary b bits in the *mantissa* to maintain the error bound. Through Equations 1 - 3, we figure out the value of b for the floating-point value d_i according to the error bound *E*.

$$2^{-x} \leq E < 2^{-x+1}\ (x > 0) \tag{1}$$

$$2^{y} \leq d_i < 2^{y+1} \tag{2}$$

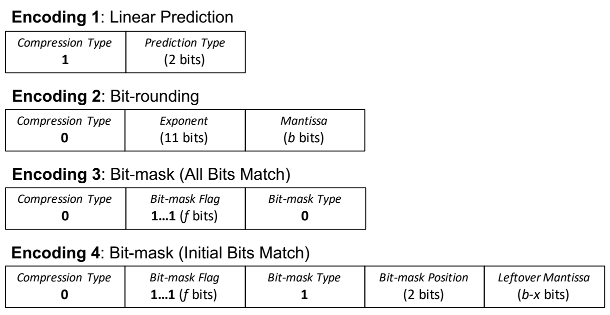
$$b = x + y\ (b = 0\ \textit{if}\ x + y < 0) \tag{3}$$

Thus, the least necessary number of bits for the lossy error-bounded compression is $1 + 11 + b = 12 + b$ for a double-precision floating-point value, and the compression ratio is $64/(12 + b)$ for that value.

2) Encoding and Decoding: All the encoded data bits are concatenated to a continuous output bit-stream in accordance with their original order in the input dataset. In other words, our lossy bit-zip compression scheme does not require any displacement information and thus reduces the communication overhead. For the bit-zip decompression, we can easily reconstruct the data extracted from the received bit-stream.

B. Lossless Bit-mask Compression

1) Mask Selection: We use a bit-mask pattern to further improve the compression ratio by creating a matching sequence. This is to improve the compression ratio without adding significant cost (extra bits) or introducing much decompression penalty. In other words, this bit-mask compression technique provides both good compression ratio and fast decompression. The basic idea is similar to the works [6] [7], and it takes advantage of commonly occurring bit sequences and creates matches by remembering a few bit positions. The efficiency of the bit-mask compression is limited by the number of bit changes when compared to the matching bit sequence. Obviously, if more bit changes are allowed, more matching sequences will be generated. Considering the balance of cost and benefit, we use the middle value in the dataset to create a matching sequence, which is employed as the bit-mask to replace a few bit strings by several predefined shorter bit-string symbols.



Encoding 1: Linear Prediction

Compression Type	*Prediction Type*
1	(2 bits)

Encoding 2: Bit-rounding

Compression Type	*Exponent*	*Mantissa*
0	(11 bits)	(*b* bits)

Encoding 3: Bit-mask (All Bits Match)

Compression Type	*Bit-mask Flag*	*Bit-mask Type*
0	**1…1** (*f* bits)	**0**

Encoding 4: Bit-mask (Initial Bits Match)

Compression Type	*Bit-mask Flag*	*Bit-mask Type*	*Bit-mask Position*	*Leftover Mantissa*
0	**1…1** (*f* bits)	**1**	(2 bits)	(*b-x* bits)

Fig. 1. The generic encoding scheme in our compression algorithm.

2) Encoding and Decoding: Figure 1 shows the generic encoding scheme used by our compression algorithm. Notice that, the bit-mask compression technique is an extension of the lossy bit-zip compression scheme (Section II-A), thus the encoding scheme applies to both the bit-zip compression and the bit-mask compression. The initial bit of the generic encoding refers to the compression type: '0' indicates the bit-rounding compression or the bit-mask compression, and '1' indicates the linear-prediction compression. Besides, the generic encoding requires one flag bit '1' or more to identify the bit-mask encoding. The number of *bit-mask flag* bits (f) depends on the maximum value d_{max} of the dataset d. This is to ensure that the leading *exponent* bit(s) of each value in the dataset d are not all '1' such that they do not conflict with the all-'1' *bit-mask flag* bits. Concretely, we get the value of f as follows.

$$f = \begin{cases} 1 & d_{max} < 2^{2^{10}-1023} \\ 2 & 2^{2^{10}-1023} \leq d_{max} < 2^{2^{10}+2^{9}-1023} \\ 3 & 2^{2^{10}+2^{9}-1023} \leq d_{max} < 2^{2^{10}+2^{9}+2^{8}-1023} \end{cases}$$

Obviously, using up to three bit-mask flag bits ($f \leq 3$) is sufficient in most cases ($d_{max} < 2^{769}$). In addition, we use one *bit-mask type* bit to represent two types of the bit-mask compression: '0' indicates that no mismatch occurs for all bits after XORing the value and the bit-mask, and '1' indicates that no mismatch occurs for initial bits before the bit location where mismatch (i.e., '1') occurs. The following two *bit-mask position* bits determine the bit location: '00' indicates that no mismatch occurs for the initial 12 bits, '01' indicates the initial $12 + 2^1 = 14$ bits, '10' indicates the initial $12 + 2^2 = 16$ bits and '11' indicates the initial $12+2^3 = 20$ bits. In this study, the bit-mask compression apparently benefits a dataset that has many data around the middle value so that their MSBs have more identical occurring bits to be compressed.

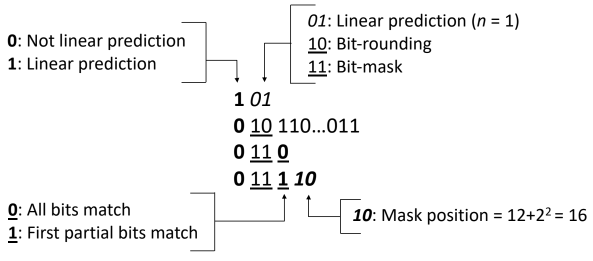


Fig. 2. An example of the encoded bits by our compression approach (the number of bit-mask flag bits $f = 2$).

Figure 2 shows an example of the encoded bits by our approach (the number of *bit-mask flag* bits $f = 2$). All the encoded bits are concatenated to form a bit-stream to be transferred at the sender side. For different compression types, they employ differential prefix bit(s) in the bit-stream, which are identified for smooth decompression at the receiver side.

III. Bit-flip Check and Correction

A. Co-design with Lossy Compression

We use CRC-32 to provide effective and high level of protection for communication data on lossy interconnection networks, which detects bit-flip errors in the received data. In addition, Hamming code is used conditionally to correct single bit errors if any. Because a larger bit-stream suffers a higher possibility of bit flipping during transmission on lossy interconnection networks, we adaptively cut the whole bit-stream into multiple data blocks (DBs) and apply Hamming code to each DB accordingly if it is estimated that multiple bit flips would occur in the bit-stream. The data block size depends on the bit error rate (BER) on the target lossy interconnection network. Especially, the data block size should be small enough to tolerate high BER. For instance, if BER = 10^{-5}, we set the data block size to 10^5 bits and apply Hamming code to each DB. In this case, if a single bit flip is detected in a DB, the error can be directly corrected on the receiver side without requiring retransmission from the sender side. However, if there are multiple bit flips, e.g., burst errors with the burst length ≤ 31 bits, detected in a DB, the DB will be required to be resent from the sender side.

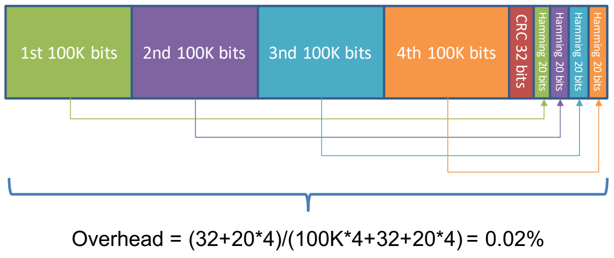


Fig. 3. Bit-flip check and correction of 400K-bit compressed data.

A 400K-bit communication stream using a Hamming code scheme in addition to CRC-32 looks like Fig. 3. Assuming BER = 10^{-5}, CRC-32 requires 32 overhead bits for bit-flip check in the whole communication data. Besides, Hamming code requires 20 overhead bits per 100K-bit DB for bit-flip correction, and it requires 80 overhead bits in total for bit-flip correction of the whole 400K-bit communication data. In this example, the bit-flip check and correction overhead is $32 + 20 \times 4 = 112$ bits, occupying only about 0.02% of the whole communication bit-stream. As the BER becomes lower, i.e., smaller than 10^{-5}, the data block size can be enlarged accordingly and the overhead rate turns out to be even smaller.

B. MPI Implementation

The bit-wise lossy floating-point compression uses basic MPI functions and basic MPI data types for high portability to various MPI implementations.

First, the sender only needs to send the size of the compressed data (constructed with a single MPI_INT) before transferring the compressed data (constructed with MPI_UNSIGNED_CHAR). Afterwards, the sender transfers the difference information (constructed with a single MPI_DOUBLE) generated at the difference preprocessing phase. For point-to-point communication, we use MPI_Isend, MPI_Irecv and MPI_Waitall between the corresponding processes; for collective communication such as broadcast, we use MPI_Bcast among the involved processes. Notice that, all the compressed data bits are concatenated seamlessly at the sender side and the prefix bits guide smooth decompression at the receiver side, thus there is no additional overhead for any bit displacement information.

As a co-design of the bit-wise lossy floating-point data compression on lossy interconnection networks, CRC verification is conducted at the receiver side. The compressed data is decompressed if the verification is passed. Hamming code is conditionally applied to correct single bit errors at the receiver side. The compressed data can be still successfully decompressed after the correction procedure. The sender is required to resend the compressed data if multiple bit flips in a DB are detected at the receiver side.

IV. Evaluation

A. Time Cost

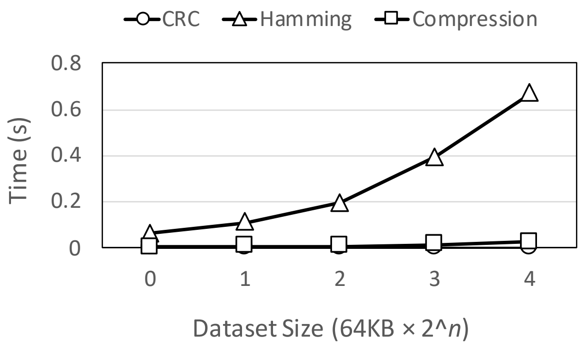
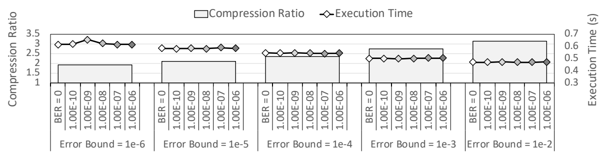


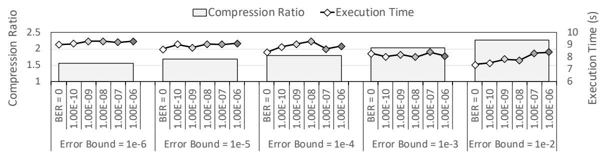
Fig. 4. Time cost for different dataset sizes.

In Section III-A, we analyzed the overhead bits appended to the compressed data for bit-flip check and correction on lossy interconnection networks. In this section, we evaluate the time cost of the bit-flip check and correction techniques, i.e., CRC and Hamming code. As shown in Fig. 4, for different dataset sizes CRC keeps a lower time cost than the bit-wise lossy compression algorithm. Comparatively, it takes a longer time for Hamming code to perform the bit-flip correction especially for a larger dataset size. Such time cost is not allowable for time-sensitive communication on lossy interconnection networks. We thus recommend using only CRC for bit-flip check if the dataset size is large. In this case, if any bit flip is detected on the receiver side, it will require the retransmission of the errorous data block (DB) from the sender side.

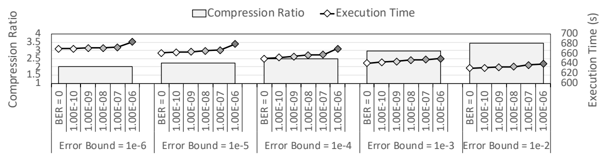
B. *Benchmark Performance*



(a) MM



(b) LU



(c) K-means

Fig. 5. Execution time of the MPI applications with bit-flip recovery.

We integrate the bit-flip recovery schemes with the bit-wise lossy compression algorithm in the MPI applications: MM, LU and K-means. Because the datasets used in these applications are large, we rely on only CRC and retransmission for bit-flip check on the target lossy interconnection network. Figure 5 shows the execution times of the MPI applications including bit-flip recovery. Overall, the speedup by the bit-wise lossy compression algorithm is incrementally obtained as the error bound becomes relaxed, which is consistent with our previous observation. On the other hand, although a large BER usually sacrifices the speedup, the time overhead is marginal when compared to a lossless interconnection network (BER = 0).

V. Conclusion

Data compression increases the effective network bandwidth on an interconnection network of parallel computers. Instead of hardware compression at network interfaces, we introduce an application-level bit-wise error-bounded lossy compression algorithm for floating-point data communication to improve the performance of parallel applications. The compressed floating-point values are concatenated in a bit-stream encapsulated in a byte array, corresponding to an MPI unsigned char type for providing high portability. For the purpose of recovering bit flips on lossy interconnection networks, we co-design with a simple bit-flip recovery scheme for the compressed bit-stream. We propose using CRC for bit-flip detection and using Hamming code for single bit-flip correction without sacrificing the merit of the bit-wise lossy compression algorithm.

Evaluation results demonstrated that our bit-wise lossy compression algorithm successfully improves the application performance. Rather than an application-level bit-flip correction technique such as Hamming code, CRC with retransmission is more effective and efficient for a large dataset size to recover bit flips on lossy interconnection networks.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 19H01106, and JST PRESTO JPMJPR19M1.

References

[1] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, "Approximate computing: Challenges and opportunities," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–8.

[2] S. Chunduri, S. Parker, P. Balaji, K. Harms, and K. Kumaran, "Characterization of mpi usage on a production supercomputer," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 386–400.

[3] Q. Fan, D. J. Lilja, and S. S. Sapatnekar, "Using dct-based approximate communication to improve mpi performance in parallel clusters," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, 2019, pp. 1–10.

[4] J. Tomkins, "Interconnects: A Buyers Point of View," ACS Workshop, 2007.

[5] Y. Hu and M. Koibuchi, "The case for error-bounded lossy floating-point data compression on interconnection networks," in *10th International Conference on Parallel, Distributed Computing Technologies and Applications (PDCTA-2021)*, 2021, pp. 55–76.

[6] S. Seong and P. Mishra, "Bitmask-based code compression for embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 673–685, 2008.

[7] L. A. B. Gomez and F. Cappello, "Improving floating point compression through binary masks," in *2013 IEEE International Conference on Big Data*, 2013, pp. 326–331.