Accelerating MPI All-to-All Communication with Online Compression on Modern GPU Clusters

Qinghua Zhou(✉), Pouya Kousha, Quentin Anthony, Kawthar Shafie Khorassani, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda

The Ohio State University, Columbus, OH 43210, USA
{zhou.2595,kousha.2,anthony.301,shafiekhorassani.1,shafi.16, subramoni.1,panda.2}@osu.edu

Abstract. As more High-Performance Computing (HPC) and Deep Learning (DL) applications are adapting to scale using GPUs, the communication of GPU-resident data is becoming vital to end-to-end application performance. Among the available MPI operations in such applications, All-to-All is one of the most communication-intensive operations that becomes the bottleneck of efficiently scaling applications to larger GPU systems. Over the last decade, most research has focused on the optimization of large GPU-resident data transfers. However, for state-of-the-art GPU-Aware MPI libraries, MPI_Alltoall communication for large GPU-resident data still suffers from poor performance due to the throughput limitation of commodity networks. However, the development of GPU-based compression algorithms with high throughput can reduce the volume of data transferred. The recent research of point-to-point-based online compression with these compression algorithms has shown potential on modern GPU clusters.

In this paper, we redesign an MPI library to enable efficient collective-level online compression with an optimized host-staging scheme for All-to-All communication. We demonstrate that the proposed design achieves benefits at both microbenchmark and application levels. At the microbenchmark level, the proposed design can reduce the All-to-All communication latency by up to 87%. For PSDNS, a traditional HPC application, our proposed design can reduce the All-to-All communication latency and total runtime by up to 29.2% and 21.8%, respectively, while ensuring data validation and not affecting the application convergence time. For Microsoft's DeepSpeed, a DL optimization library, the proposed design reduces the MPI_Alltoall runtime by up to 26.4% compared to a state-of-the-art MPI library with point-to-point compression while ensuring data validation. To the best of our knowledge, this is the first work that leverages online GPU-based compression techniques to significantly accelerate MPI_Alltoall communication for HPC and DL applications.

*This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, #2112606, and XRAC grant #NCR-130002.

A.-L. Varbanescu et al. (Eds.): ISC High Performance 2022, LNCS 13289, pp. 3–25, 2022.
https://doi.org/10.1007/978-3-031-07312-0_1

Keywords: All-to-All · GPU · Compression · GPU-Aware MPI · HPC · DL

1 Introduction

Emerging high-performance computing (HPC) and cloud computing systems are widely adopting Graphics Processing Units (GPUs) to support the computational power required by modern scientific and deep learning (DL) applications. By offering high-bandwidth memory, tensor processing, and massive parallelism, GPUs enable running complex applications such as weather forecasting, brain data visualization, and molecular dynamics. MPI is the de facto communication standard widely used in developing parallel scientific applications on HPC systems. To further enhance the high compute power of current generation of hardware, researchers are building large-scale GPU clusters to benefit from massive computation capabilities offered by these accelerators.

Due to the computing power offered by GPUs, a large range of applications have been adapted to scale on GPU-based systems by application developers. Communication performance plays a vital role in end-to-end application performance on such systems. In fact, at a large scale, the communication operations become the performance bottleneck for any massively parallel HPC and DL application. Over the last decade, researchers have significantly optimized data transfers in MPI for GPU-resident data [2,21,24]. Inter-node communication operations for large messages are highly optimized to saturate the bandwidth of the InfiniBand network by the state-of-the-art MPI libraries [9,24]. [32] has shown the saturated inter-node network bandwidth of the state-of-the-art MPI libraries. Although these MPI libraries are well optimized, the communication time at the application level is still a major bottleneck for many HPC and DL applications. Since the inter-node communication bandwidth is already saturated via optimizations implemented by major MPI libraries, we should seek other innovative ways to reduce the communication time of the HPC applications.

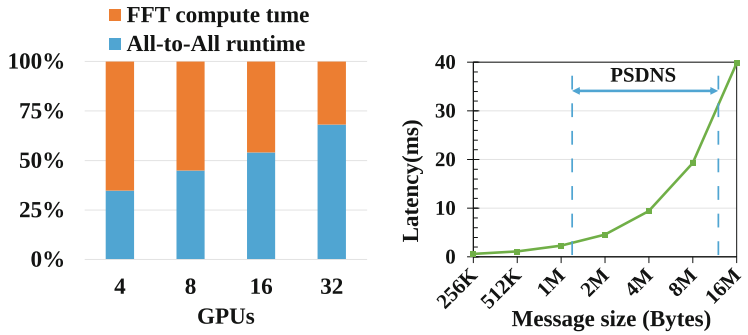
Thinking outside the box, we propose exploiting compression to aid with optimizing the performance of MPI stacks and HPC/DL applications, subsequently. Compression can reduce the amount of data that needs to be transmitted and/or stored helping to mitigate the cost of communication. Various compression techniques have been proposed in the literature diving into CPU-based algorithms and GPU-based algorithms. The common issue with CPU-based algorithms is the low throughput compared to GPU-based designs [14,31]. Existing GPU-based compression schemes such as MPC [31], SZ [3], and ZFP [14] are typically focused on achieving a high compression ratio and not absolute high performance.

1.1 Motivation

There are challenging aspects to consider when applying compression to the HPC domain. HPC requires low overhead while maintaining high throughput. Further, some HPC applications require that the underlying compression and

decompression operations are handled by the MPI library, leaving the HPC/DL application unchanged. We refer to this qualifier as "Online" compression. Online compression means the message should be compressed and decompressed in real-time inside the MPI library during a communication operation without any modifications to the end applications. This implies that the online compression algorithms should be self-contained with low overheads. Meeting these requirements first before maximizing the compression ratio and revamping the communication pattern/algorithm to fully exploit the HPC system's available transfer bandwidth is a challenging task that we undertake in this paper.

Since most MPI users are domain scientists first and programmers second, modifying the application to use compression is often out of reach. Adding support often involves understanding compression techniques and when to apply them based on message features such as size. Therefore, using compression directly in HPC/DL applications is a daunting task for domain scientists. In this context, [32] proposed an online compression enabled MPI library for point-to-point operations—this is an initial work in this direction.



(a) PSDNS time breakdown (b) All-to-All latency

Fig. 1. Motivating Example: All-to-all communication time for 8 GPUs on 2 Longhorn nodes. The observed message range in PSDNS is 1.2 MB to 9.5 MB. With more GPUs, All-to-all communication time becomes dominant within the overall runtime of PSDNS application.

In this paper, we use the online compression idea to optimize the performance of MPI collective operations and improve HPC/DL application performance. One of the most communication-intensive operations is MPI_Alltoall which is used in many applications like PSDNS [23] and DeepSpeed [22]. DeepSpeed depends on MPI_Alltoall to support the addition of Mixture-of-Experts [10]. As shown in the Fig. 1(a), with larger scale, the MPI_Alltoall time dominates the overall execution time of the PSDNS application. Figure 1(b) shows the message size range of MPI_Alltoall operations observed in PSDNS application. In this context, the MPI_Alltoall operation is ideally suited to benefit from compression since it is the most dense communication operation used in various HPC and DL applications.

1.2 Challenges

To design an efficient online compression scheme for MPI_Alltoall operation, following research challenges need to be addressed.

Challenge-1: The Limitation of Point-to-Point Based Online Compression Technique: MVAPICH2-GDR-2.3.6 is the only public library that has support for online compression. Table 1 summarizes the representative MPI_Alltoall algorithms in the MVAPICH2-GDR-2.3.6 MPI library and existing support for online compression.

Both the Scatter Destination (SD) and Pairwise Exchange (PE) algorithms rely on the GPU-based point-to-point communication to transfer data between GPUs. With the current point-to-point based online compression, these algorithms can leverage compression for both inter-node and intra-node communication. However, there are limitations in the existing point-to-point based online compression design. For the PSDNS application, we use the NVIDIA profiler Nsight to monitor the compression behavior of the existing GPU-based Scatter Destination and Pairwise Exchange All-to-All algorithms in the state-of-the-art MVAPICH2-GDR-2.3.6 library. Figure 2 shows the existing design that utilizes point-to-point operations in the MVAPICH2-GDR-2.3.6 library. The figure also proposes a design to overcome this limitation. As shown in the existing design section of Fig. 2, when a process sends data to other processes, the compression kernel in a single send operation does not overlap with kernels in other send operations even though they run on different CUDA streams.

Table 1. Comparison of existing online compression support in MVAPICH2-GDR-2.3.6 with proposed design

Algorithms	Compression support	Compression level	Inter-node data transfer	Intra-node data transfer	Multiple streams compression	Hide compression overhead	Overlap opportunity
GPU-aware Scatter Destination [28]	Y	Point-to-Point	GPUDirect	IPC	Within single Send/Recv	N	N
GPU-aware Pairwise Exchange [29]	Y	Point-to-Point	GPUDirect	IPC	Within single Send/Recv	N	N
CPU Staged Scatter Destination [28]	N	N	RDMA	Shared Memory	N	N	N
CPU Staged Pairwise Exchange [29]	N	N	RDMA	Shared Memory	N	N	N
Proposed Design	Y	Collective level	Staging + RDMA		Across multiple Send/Recv	Y	Y

This limitation is similar for the decompression kernels in receive operations. This essentially becomes a bottleneck for implementing dense collective operations like MPI_Alltoall efficiently.

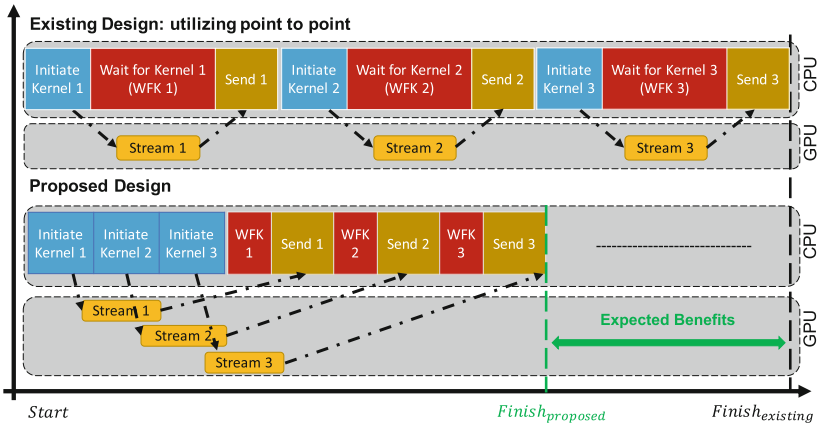


Fig. 2. Comparison between using existing compression method for point-to-point operations versus proposed design. The proposed design increases the overlap of kernel initialization and wait time by enabling compression at the collective level instead of the point-to-point level.

Challenge-2: Move the Point-to-Point Compression to the Collective-Level: The above limitation can be handled by utilizing compression at the collective level. In MPI libraries, collectives operations are typically built using point-to-point operations. In collective-level compression, the compression/decompression is done at the collective algorithm level before calling the underlying point to point send/receive operation to transfer data. This provides us the opportunity that CUDA kernels across multiple send/receive operations can be overlapped to reduce the compression overheads—this is depicted in Fig. 2 and forms the primary motivation of our proposed design in this paper. However, the underlying mechanism of Scatter Destination and Pairwise Exchange algorithms prevents us from moving the compression to the collective level efficiently. This pushes us to explore other MPI_Alltoall algorithms. The CPU Staging algorithm [28] moves the data from GPU to host and leverages the host-based Scatter Destination, Pairwise Exchange, or other MPI_Alltoall algorithms to transfer the data. Since the send and receive operations are host-based, we cannot merely use the existing GPU-based point-to-point compression. We need to co-design the GPU-based compression at the collective level (Sect. 3).

Challenge-3: Revamp and Optimize GPU-Based Compression for the Collective-Level Online Compression: While point-to-point compression focuses on reducing the inherent compression-related overheads, collective-level compression aims to further reduce the effective kernel computing time by co-designing the compression with the collective operations. This needs the

enhancement of designing interfaces for the existing collective operations to support GPU-based compression. Furthermore, naive integration of the compression algorithms at the collective level may not achieve optimal performance (Sect. 3.1). We have to analyze the bottlenecks of such naive compression designs, revamp the existing GPU-based algorithm, upgrade the naive design to support the new interface, and optimize the collective operations. The implementations of each optimization will be proposed (Sect. 4).

1.3 Contributions

In this paper, we design and implement high-performance online message compression for the MPI_Alltoall communication operation on modern GPU clusters. To the best of our knowledge, this is the first work that leverages GPU-based compression techniques to significantly improve MPI_Alltoall communication performance while maintaining data validation and not affecting the convergence time. To summarize, this paper makes the following main contributions:

- We conduct a thorough analysis of the limitations and possible optimization opportunities for existing MPI_Alltoall algorithms with online compression support on modern GPU systems.
- We propose an online compression design that is integrated into the underlying communication libraries (e.g., MPI) for host-staging based MPI_Alltoall communication. Later, we analyze the limitations of naively integrating the existing ZFP compression library.
- We optimize the ZFP compression library to enable execution of compression/decompression kernels on multiple CUDA streams. These strategies reduce the overhead of compression/decompression kernels and improve overall performance.
- We use the OSU Micro Benchmark (OMB) suite to evaluate MPI_Alltoall communication and show that the proposed design can achieve up to 87% improvement in performance. We also enhance OMB to use real data sets and get up to 75% improvement in the MPI_Alltoall operation.
- We evaluate the effectiveness of the proposed design through application studies. In the PSDNS application, we can gain up to 29.2% and 21.8% reduced MPI_Alltoall runtime and total execution time, respectively, compared to the existing MVAPICH2-GDR-2.3.6 with point-to-point compression. In the Deep Learning framework DeepSpeed, the proposed design reduces the MPI_Alltoall runtime by up to 26.4% and improves throughput by up to 35.8%.

2 Background

In this section, we provide the necessary background knowledge including the recent development of GPU based compression algorithms, MPI_Alltoall algorithms in MPI libraries, GPUDirect technology, and GPU-aware communication middlewares.

2.1 Compression Algorithms for HPC Applications

In recent years, lossy compression libraries have shown acceptable error-bounds [6] for HPC applications. Among them, ZFP [14] is a well-known public compression library with user-friendly interfaces and supports CUDA-enabled fixed-rate compression. ZFP deconstructs a d-dimensional array into 4^d blocks. The resulting compression rate is the number of amortized compressed bits among these blocks. For example, for single-precision (32-bit) floating-point data, a compression rate of 8 bits/value can get a compression ratio of 4. In this work, we use the ZFP compression library.

NVIDIA recently proposed nvCOMP [19], a CUDA-based lossless compression interface to achieve high-performance compression kernels. nvCOMP supports Cascaded, LZ4, and Snappy compression methods. However, the burden of integrating nvCOMP APIs and using them for HPC applications requires changing application code. Since nvCOMP is a user-level library, we don't consider it for online compression.

2.2 Algorithms for MPI_Alltoall Communication

Different MPI libraries have their own implementations of MPI_Alltoall algorithms and often tune their library to pick up the most efficient MPI_Alltoall algorithm for a given system and message size at runtime. In existing MPI libraries, there are three representative MPI_Alltoall algorithms for large-message data transfers. (a) In the Scatter Destination algorithm [28], each process posts a series of MPI_Isend and MPI_Irecv operations and waits for these operations to complete. (b) In the Pairwise Exchange algorithm [29], each process runs MPI_Sendrecv to communicate with only one source and one destination. These send and receive operations will reply with GPU-based point-to-point communication schemes to transfer data between GPUs. (c) The CPU staging algorithm [28] leverages the host-based send and receive operations to transfer the data. The GPU data will be moved from GPU to host before the MPI_Isend operation, and will be copied back from host to GPU after MPI_Irecv.

2.3 GPU-Aware Communication Middleware

GPU-aware MPI libraries like SpectrumMPI [5], OpenMPI [20], and MVAPICH2 [17] can distinguish between host buffers and GPU buffers. These libraries have been optimized with GPU-based point-to-point communication schemes like CUDA Inter-Process Communication (IPC) [25] and NVIDIA GPUDirect technology [18] which supports direct reading and writing to host and device memory by the CPU and GPU. Such technologies provide optimal performance across varied communication paths.

3 Proposed Online Compression Design for MPI_Alltoall Communication

To tackle the limitation of using point-to-point based compression (Challenge-1) for MPI_Alltoall communication and move the point-to-point compression to collective level (Challenge-2), we redesign the host-staging based MPI_Alltoall algorithm in the MPI library to implement efficient MPI_Alltoall communication of GPU data with online compression. Figure 3 depicts the data flow of host-staging based MPI_Alltoall operations with compression. GPU data are exchanged among four GPUs. In GPU0, the device buffer sendbuf contains data A0, A1, A2, A3 which will be sent to the recvbuf in GPU0, GPU1, GPU2 and GPU3 respectively.

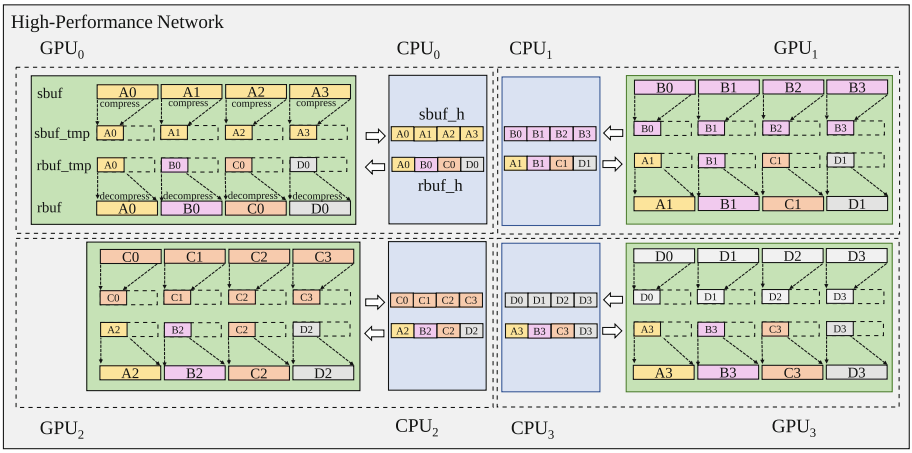


Fig. 3. Host-Staging based MPI_Alltoall with compression. GPU data will be compressed to the temporary device buffer sendbuf_tmp and copied by cudaMemcpyAsync to the host buffer sendbuf_host. MPI_Isend sends out the data in the host buffer to other CPUs. MPI_Irecv collects the data to the host buffer recvbuf_host from other CPUs. The received data will be copied by cudaMemcpyAsync to the temporary device buffer recvbuf_tmp and decompressed to the recvbuf.

Algorithm 1 provides a high-level overview of the compression design for host-staging based MPI_Alltoall. Before staging the GPU data to the CPU, a compression kernel will be launched on the send buffer for each process (Line 4). The compressed data will be stored into the corresponding part of a temporary device buffer sendbuf_tmp. Once the compression kernel finishes, the compressed data will be copied to the host buffer sendbuf_host using cudaMemcpyAsync on a specific CUDA stream $Stream1$ (Line 5). After each cudaMemcpyAsync, a CUDA event will be recorded on the same CUDA stream (Line 6).

With compression, the data size of the transferred data is changed. The MPI_Isend operation needs to specify the compressed data size instead of the original data size. We use a data size array B to record the data size of each compressed data after compression. For the peer processes on other GPUs, they

also should specify the correct data size for the upcoming data in MPI_Irecv. To transfer such data size information before transferring the compressed data, we run an MPI_Alltoall to exchange the elements in the data size array between all the CPUs (Line 7). Since each element is only a 4 bytes integer, the overhead of such operation is negligible compared to the large data transfer.

The multiple MPI_Irecv operations for all the peer processes will be issued ahead of MPI_Isend (Line 9). Each MPI_Irecv is associated with a receive request. Before MPI_Isend, we use cudaEventSynchronize to indicate the completion of related cudaMemcpyAsync from device to host (Line 11). The MPI_Isend will be issued to send out data in the host buffer S_H to the buffer address in another CPU (Line 12).

Once a receive request is completed, the related compressed data is stored in the host buffer R_H. Similar to the send operation, the data will be copied to a temporary device buffer R_tmp using cudaMemcpyAsync on a specific CUDA stream (Line 14). The decompression kernel will be launched on the data of each process in R_tmp after the corresponding cudaMemcpyAsync is finished (Line 19). The compressed data will be restored to the receive buffer R.

Algorithm 1: Online Compression/Decompression Design for Host-Staging based MPI_Alltoall Communication

Input : Send buffer S, Control parameters A, Number of MPI processes N, Preallocated GPU buffer S_tmp, R_tmp, Preallocated Host buffer S_H, R_H, CUDA events for send E_S, CUDA events for receive E_R

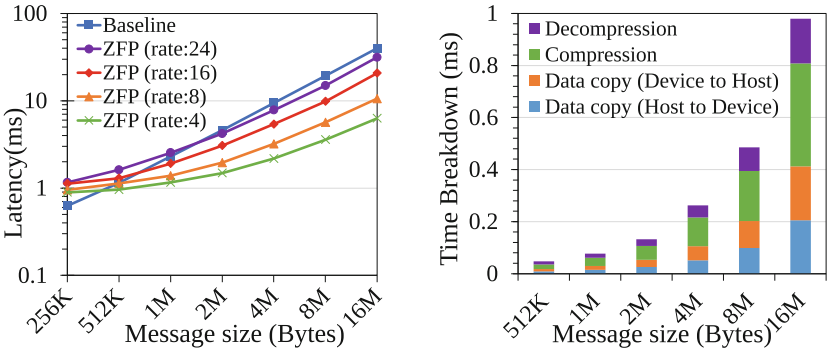
Output: Receive buffer R, Compressed data size B for send buffer, Compressed data size C for receive buffer

1 **for** $i = 1$ *to* N **do**
2 Construct zfp_stream and zfp_field;
3 Attach A to zfp_stream and zfp_field;
4 (B_i, S_tmp_i)=zfp_compress(S_i, A_i); //Runs on default CUDA Stream0
5 cudaMemcpyAsync(S_tmp_i, S_H_i, B_i, cudaMemcpyDeviceToHost, Stream1);
6 cudaEventRecord(E_S_i, Stream1);
7 MPI_Alltoall(B, 1, MPI_INT, C, 1, MPI_INT, MPI_COMM_WORLD); // Exchange the compressed data size
8 **for** $i = 1$ *to* N **do**
9 MPI_Irecv(R_H_i, C_i, ...) //Receive compressed data;
10 **for** $i = 1$ *to* N **do**
11 cudaEventSynchronize(E_S_i);
12 MPI_Isend(S_H_i, B_i, ...); // Send compressed data;
13 **if** *MPI_Irecv finishes for* R_H_i **then**
14 cudaMemcpyAsync(R_tmp_i, R_H_i, C_i, cudaMemcpyHostToDevice, Stream2);
15 cudaEventRecord(E_R_i, Stream2);
16 **for** $i = 1$ *to* N **do**
17 cudaEventSynchronize(E_R_i);
18 Construct zfp_stream and zfp_field based on control parameter A;
19 R_i = zfp_decompress(R_tmp_i, C_i, A_i); //Runs on default CUDA Stream0

We define runtime parameters to enable/disable compression in the host-staging based MPI_Alltoall design. We also define several control parameters such as compression rate, dimensionality, and data type to run the ZFP compression library,

3.1 Analysis of the Benefits and Limitation for the Naive Compression Design

In this section, we analyze the compression-related benefits and costs to find out the bottleneck (Challenge-3) in the naive compression design. With compression, there will be less data movement by cudaMemcpyAsync between CPU and GPU in the staging operations. The run time of the staging operation will be reduced. MPI_Isend can send out the data in the host buffer much earlier. Similarly, the run time of transferring data between the CPUs will be reduced. On the receiver side, it will take less time to copy data from the host buffer to the device buffer. However, similar to the point-to-point based compression [32], there is also extra compression/decompression kernel execution time and related kernel launching overheads in the naive host-staging based compression. When the compression ratio is not high enough, the benefits brought by the reduced data size may not compensate for these extra running time costs. We need to optimize the compression design to reduce such costs.



(a) Performance of MPI_Alltoall with naive compression design

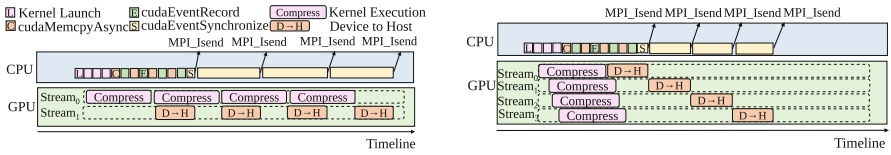
(b) Time breakdown of key operations with ZFP (rate:24)

Fig. 4. Performance of host-staging based MPI_Alltoall with naive compression design on 2 nodes (4 ppn) of the Longhorn cluster. Higher compression rate (16, 24) indicates a lower compression ratio. The design only starts to outperform the baseline from larger message size 1 MB for rate = 16 and 2 MB for rate = 24. The time breakdown shows the latency of single compression/decompression kernel, and data copy from host to device and device to host.

We evaluate the proposed compression design using the OSU Micro-Benchmark suite (OMB) on 2 nodes with 4ppn (4GPUs/node) of the TACC

Longhorn cluster. As shown in Fig. 4(a), the proposed host-staging based naive ZFP compression design can achieve benefits from 512 KB with low compression rates 8 and 4. However, with a higher compression rate (and consequently a lower compression ratio), it only starts to outperform the baseline for larger message size. Since ZFP is a lossy compression algorithm, this shortage will prevent the design from applying to those applications which need higher accuracy. Figure 4(b) depicts the time breakdown of some key operations in the naive compression design with ZFP (rate:24). The results show the latency of every single operation.

In the existing ZFP library, compression kernel cuZFP::encode runs on the default CUDA stream. In the naive compression design, although the cudaMemcpyAsync executing on a non-default stream with a non-blocking flag cudaStreamNonBlocking can achieve overlap with the compression kernels for other ranks, each cudaMemcpyAsync still needs to wait for the completion of compression kernel for its rank. As we can see in Fig. 5(a), since the compression kernels run serially in the default stream, there is a long waiting time for the MPI_Isend operation to send out the data since MPI_Isend must wait for the finish of compression kernel and memory copy from device to host.

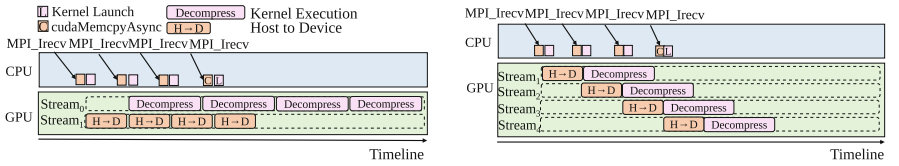


(a) Send operations with naive compression on default stream

(b) Send operations with optimized compression on multiple streams

Fig. 5. Comparison between compression on the default CUDA stream and multiple CUDA streams for send operations in the host-staging based All-to-All. Overall compression time is reduced due to the overlap between the compression kernels. The data will be sent out faster since the cudaMemcpyAsync and MPI_Isend can be executed much earlier.

There is also a similar limitation for the decompression phase. As shown in Fig. 6(a), the decompression kernel cuZFP::decode also runs on default CUDA stream. Although the cudaMemcpyAsync can be overlapped with the decompression kernel, it will cost a long operation time to restore data in the GPU due to the serial operations among the decompression kernels.



(a) Receive operations with naive decompression on default stream

(b) Receive operations with optimized decompression on multiple streams

Fig. 6. Comparison between ZFP decompression on the default CUDA stream and multiple CUDA streams. Explicit calling of cudaEventSynchronize is not needed. Overall decompression time is reduced due to the overlap between the decompression kernels.

4 Optimization Strategies in the Host-Staging Based MPI_Alltoall

Based on the previous analysis of the limitation of the naive compression design, we propose the following optimizations to address the Challenge-3.

4.1 Enabling Multiple CUDA Streams in ZFP Library

To reduce the overall compression and decompression time, we aim to achieve overlap between the kernels. However, the current ZFP library does not provide such an interface to run the kernels concurrently on non-default CUDA streams. Therefore, we enhance the existing ZFP library to allow compression and decompression kernels to run on multiple streams. We define two new functions, zfp_compress_multi_stream and zfp_decompress_multi_stream. A new parameter of CUDA stream object cudaStream_t is added to these functions. At the user level, we can assign a specific stream to the compression and decompression. ZFP uses a function table to select the correct low-level compression and decompression functions according to the execution policy (Serial, OpenMP, CUDA), stride, dimensionality, and scalar type. We extend the function table and introduce a new execution policy named zfp_exec_cuda_multi_stream to allow the selection of new lower-level APIs with a stream parameter. We add a new cudaStream_t parameter to all the related lower-level APIs.

In the proposed compression design, we use the 1D array type for ZFP compression with the number of floating-point values as the dimensionality. The compression kernel cudaEncode1 and decompression kernel cudaDecode1 will be launched to the CUDA stream specified by the new High-level APIs. In the existing compression kernel, launch function, and constant setup function, two synchronous CUDA memory copy functions (cudaMemset and cudaMemcpyToSymbol) are used to prepare for the compression and decompression on the default stream. We change them to cudaMemsetAsync and cudaMemcpyToSymbolAsync, respectively, with the same CUDA stream used for compression or decompression.

4.2 Proposed Optimization Metrics

With the enhanced ZFP library (ZFP-OPT), we use two new API calls in the compression design: zfp_compress_multi_stream and zfp_decompress_multi_stream.

Algorithm 2: Proposed optimized multi-stream compression/decompression for MPI_Alltoall Communication

Input : Send buffer S, Control parameters A, Number of MPI processes N, Preallocated GPU buffer S_tmp, R_tmp, Preallocated Host buffer S_H, R_H, CUDA events for send E_S, CUDA events for receive E_R
$[S_1, ..., S_N]$= Send buffers for peer processes in Send buffer S;
$[S\_tmp_1, ..., S\_tmp_N]$= Divided N partitions of S_tmp;
$[R\_H_1, ..., R\_H_N]$= Receive buffers for peer processes in R_H;
$[R\_tmp_1, ..., R\_tmp_N]$= Divided N partitions of R_tmp

Output: Receive buffer R, Compressed data size B for send buffer, Compressed data size C for receive buffer

```
1  Multi-stream compression for send operation:
2  for i = 1 to N do
3      Construct zfp_stream and zfp_field based on control parameter A;
4      zfp_stream_set_execution(zfp_stream, zfp_exec_cuda_multi_stream);
5      (B_i, S_tmp_i)=zfp_compress_multi_stream(S_i, A_i, Stream_i); //Runs on
         non-default CUDA Stream
6      cudaMemcpyAsync(S_H_i, M_i, B_i, cudaMemcpyDeviceToHost, Stream_i);
         //Run on the same CUDA stream
7      cudaEventRecord(E_i, Stream_i);
8  MPI_Isend, MPI_Irecv operations;
9  Multi-stream decompression for receive operation:
10 for i = 1 to N do
11     cudaMemcpyAsync(R_H_i, R_tmp_i, C_i, cudaMemcpyHostToDevice,
         Stream_i); // Runs on non-default CUDA stream
12     Construct zfp_stream and zfp_field based on control parameter A;
13     zfp_stream_set_execution(zfp_stream, zfp_exec_cuda_multi_stream);
14     R_i=zfp_decompress_multi_stream(R_tmp_i, A_i, Stream_i); //Runs on the
         same CUDA Stream
```

Algorithm 2 provides a high-level overview of the multi-stream compression and decompression for the for MPI_Alltoall operation. For the compression on send operation side, we set a new execution policy zfp_exec_cuda_multi_stream (Line 4). Then we launch the compression kernels to different CUDA streams (Line 5). Each corresponding cudaMemcpyAsync also runs on the same stream as the kernel (Line 6). The benefits of concurrent kernel execution are two-fold. Due to the overlap between the compression kernels, the overall compression time is reduced. Furthermore, since cudaMemcpyAsync can copy the compressed data to CPU earlier, MPI_Isend can send out the data from CPU in advance. Figure 5(b) depicts the optimized send operations with this mechanism. Note that, the

overlapping situation among the kernels and data copy operations depends on the number of processes in the MPI_Alltoall operation and the compression rate.

Similarly, on the receive operation side, we optimize decompression using multiple CUDA streams. Once a receive request is finished, we run the cudaMemcpyAsync on a non-default stream to copy the compressed data from host to device (Line 11). To enable the multi-stream decompression, we also need to use the execution policy of zfp_exec_cuda_multi_stream (Line 13). The related decompression kernel will also run on the same stream (Line 14). In this way, we do not need to explicitly launch cudaEventSynchronize to wait for the completion of cudaMemcpyAsync. As shown in Fig. 6(b), the overlap between the decompression kernels will reduce the overall decompression time and thus, accelerate the data restoration phase in the GPU. In the proposed design, we define wrapper functions to execute the compression/decompression kernels. Such optimization metrics can be easily applied to compression/decompression kernels of other compression algorithms.

5 Microbenchmark Results and Analysis

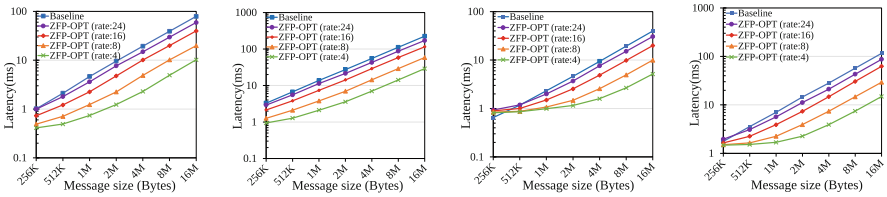
We run the experiments on three GPU-enabled clusters: Longhorn [16] and the Liquid [15] subsystem at the Texas Advanced Computing Center, and the Lassen [13] system at Lawrence Livermore National Laboratory. Each computing node on the Longhorn and Lassen systems is equipped with IBM POWER9 CPUs and 4 NVIDIA V100 GPUs. They use RHEL operating system. Both systems enable NVLink2 interconnection between CPU and GPU, and Infiniband EDR between nodes. Each node on Frontera Liquid is installed with Intel Xeon E5-2620 CPUs and 4 NVIDIA Quadro RTX5000 GPUs. Frontera Liquid uses PCIe Gen3 interconnection between CPU and GPU, and Infiniband FDR between nodes. It installs CentOS operating system. More details about the system configurations can be found in their respective specification documents.

We used osu_alltoall in the OSU Micro-Benchmark suite (OMB) to evaluate the MPI_Alltoall communications of GPU data on multiple nodes. We also enhanced OMB to use real data sets for the MPI_Alltoall communication tests.

5.1 MPI_Alltoall Communication Latency on Micro-Benchmark

We run the OSU Micro-Benchmark suite (OMB) to evaluate the MPI_Alltoall communication latency. Figures 7(a) and 7(b) show the MPI_Alltoall communication latency of message size from 256 KB to 16 MB on the Frontera Liquid system. Since the proposed design is aimed at the transfer of large GPU messages, the performance results of smaller message sizes are not shown in the figures. We observe performance improvement with the optimized compression design in the 256 KB to 16 MB message range. With a lower compression rate, ZFP-OPT achieves a higher compression ratio and a further reduced communication latency. Compared to the baseline, ZFP-OPT (rate:4) can achieve up to 87.1% reduced latency at 16 MB on both 2nodes and 4nodes with 4ppn

(4 GPUs/node). Figures 7(c) and 7(d) show the MPI_Alltoall communication latency on the Longhorn system. On 2 nodes with 4ppn, ZFP-OPT starts to outperform the baseline from around 512 KB. Compared to Fig. 4(a), Fig. 7(c) demonstrates the performance improvement with the optimization strategies discussed in Sect. 4. On 4 nodes, except for rate = 24, ZFP-OPT has performance benefits starting from 256 KB. Similar to the Frontera liquid system, ZFP-OPT (rate:4) can achieve up to 87.1% reduced latency at 16 MB on 2 nodes and 4 nodes.

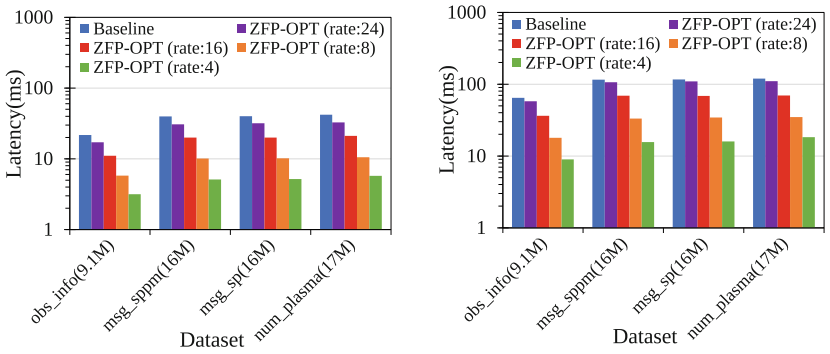


(a) Frontera: 8GPUs (2nodes, 4ppn) (b) Frontera: 16GPUs (4nodes, 4ppn) (c) Longhorn: 8GPUs (2nodes, 4ppn) (d) Longhorn: 16GPUs (4nodes, 4ppn)

Fig. 7. Latency of MPI_Alltoall on Frontera Liquid and Longhorn. On Frontera Liquid, ZFP-OPT starts to show benefits from 256 KB on both 2 nodes and 4 nodes. With a lower compression rate, ZFP-OPT achieves a higher compression ratio and reduces communication latency. On Longhorn, ZFP-OPT shows performance improvement from about 512 KB on 2 nodes. On 4 nodes, except for rate = 24, ZFP-OPT achieves benefits from 256 KB. On both systems, ZFP-OPT (rate:4) can achieve up to 87.1% reduced latency at 16 MB on 2 nodes and 4 nodes.

5.2 MPI_Alltoall Communication Latency with Real Data Sets

This section evaluates the impact of the proposed design on the MPI_Alltoall communication performance on the Longhorn system with real data sets from [31]. Figures 8(a) and 8(b) show the results of MPI_Alltoall communication latency on 2 nodes and 4 nodes respectively. In the fixed-rate compression mode, with the same compression rate, ZFP will have the same compression ratio it has in the micro-benchmark test. The proposed design achieves similar benefits as the Micro-benchmark test. With lower compression rate, it reduces communication latency further. ZFP-OPT (rate:4) reduces the MPI_Alltoall communication latency by up to 75% (num_plasma) on 2 nodes, 72% (obs_info) on 4 nodes respectively.



(a) 8 processes (2nodes, 4ppn) (b) 16 processes (4nodes, 4ppn)

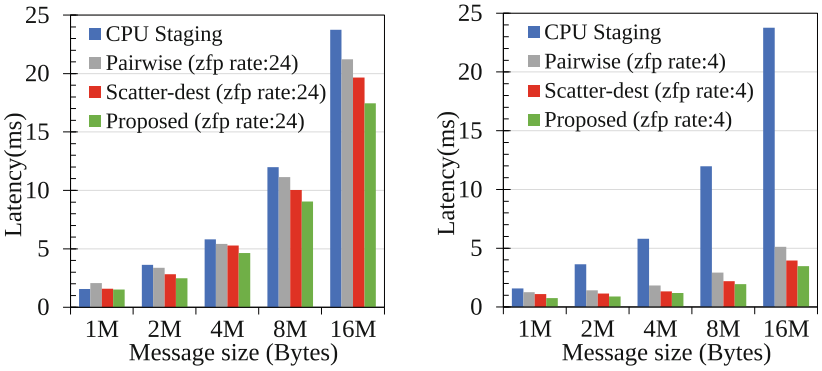
Fig. 8. Latency of MPI_Alltoall with real datasets on Longhorn. With a lower compression rate, ZFP-OPT achieves greater performance benefit. ZFP-OPT (rate:4) reduces the MPI_Alltoall communication latency by up to 75% (data set num_plasma) on 2 nodes and 72% (data set obs_info) on 4 nodes, respectively.

5.3 Comparison of the Proposed Design and Existing MPI_Alltoall Algorithms with Point-to-Point Compression

In this section, we compare our proposed design with different algorithms: CPU Staging (No compression), Scatter Destination, and Pairwise Exchange in MVAPICH2-GDR-2.3.6. We use the runtime parameters provided by the MVAPICH2-GDR-2.3.6 to trigger the point-to-point compression for Scatter Destination and Pairwise Exchange.

On the Lassen system, for 8 GPUs on 2 nodes, our proposed design performs better than these algorithms starting from 1 MB as shown in Fig. 9(a) and 9(b). Figure 9(a) shows, for 16 MB data, the proposed design reduces the MPI_Alltoall latency by up to 11.2%, 17.8% and 26.6% compared to the Scatter Destination(zfp rate:24), Pairwise Exchange(zfp rate:24), and CPU Staging (No compression), respectively. In Fig. 9(b), with zfp compression (rate:4), the latency is reduced by up to 12.4%, 32.3%, and 85.4% compared to the Scatter Destination, Pairwise Exchange, and CPU Staging (No compression), respectively.

In application tests, we observe greater benefit compared to the Scatter Destination and Pairwise Exchange on larger scales.



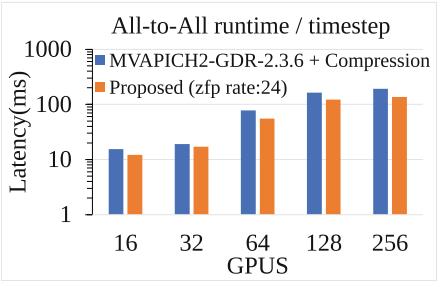
(a) Comparison with zfp (rate:24) (b) Comparison with zfp (rate:4)

Fig. 9. MPI_Alltoall latency with different algorithms for 8 GPUs on 2 Lassen nodes. With zfp compression (rate:24), the proposed design reduces the MPI_Alltoall latency by up to 11.2%, 17.8%, and 26.6% compared to the Scatter Destination, Pairwise Exchange, and CPU Staging (No compression), respectively. With zfp compression (rate:4), the latency is reduced by up to 12.4%, 32.3%, and 85.4% compared to the Scatter Destination, Pairwise Exchange, and CPU Staging (No compression), respectively

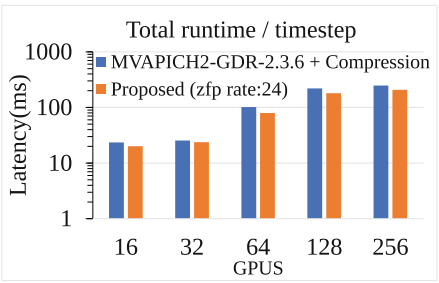
6 Application Results and Analysis

6.1 PSDNS

We evaluate the proposed design with a modified 3D-FFT kernel of the Fourier pseudo spectral simulation of turbulence (PSDNS) application [23]. The code was written in Fortran with a hybrid MPI+OpenMP approach and compiled with the IBM XL compiler. We run PSDNS on the Lassen system which uses the IBM Power9 CPU architecture. In the 3D-FFT kernel, MPI_Alltoall is used to transfer the transposed data among the multiple GPUs. The kernel will also generate a timing report about the runtime per timestep of MPI_Alltoall operations, FFT computing, and other operations. It also checks the max global difference of the sinusoidal velocity field as an accuracy criteria. The underlying different algorithms of MPI_Alltoall can be triggered by runtime parameters. Note that the Scatter Destination and Pairwise Exchange algorithms are built on top of point-to-point operations. We compare our proposed design with the state-of-the-art MVAPICH2-2-GDR-2.3.6 with point-to-point compression.



(a) MPI_Alltoall runtime per timestep, 4 GPUs/node (Lower is better)



(b) Total runtime per timestep, 4 GPUs/node (Lower is better)

Fig. 10. MPI_Alltoall runtime in the 3D-FFT kernel of the PSDNS application on the Lassen system. The proposed design with optimized ZFP(rate:24) can reduce the MPI_Alltoall runtime and total runtime by up to 29.2% and 21.8%, respectively, on 64 GPUs compared to the state-of-the-art MVAPICH2-GDR-2.3.6 with point-to-point compression.

As shown in Fig. 1(a), the MPI_Alltoall communication is dominant when the application runs on large scale. In this section, by increasing the grid size of nx, ny, nz along with the number of GPUs, we can evaluate our compression design on different problem scales. For 128 GPUs, the grid size (nx, ny, nz) is (1536, 1536, 1536).

Figure 10(a) depicts the MPI_Alltoall runtime per time step in the application. The proposed design with optimized ZFP (rate:24) is able to reduce the latency up to 29.2% on 64 GPUs(4 GPUs/node) compared to the state-of-the-art MVAPICH2-GDR-2.3.6 with point-to-point based compression. For MVAPICH2-GDR-2.3.6, we report the best result of either Scatter Destination or Pairwise algorithms with point-to-point based compression. Note that we set the same rate:24 for MVAPICH2-GDR-2.3.6. Since ZFP compression is lossy, we have ensured by working with application developers that the data generated with compression rate (>= 24) maintains acceptable precision for the FFT computation. Table 2 shows the max global difference of the proposed design reported in the 3D-FFT kernel. The tolerance of this value is 1.0E–05.

Table 2. Max global difference error

GPUs	No compression	Compression (rate:24)
16	3.492E–06	5.257E–06
32	3.721E–06	5.050E–06
64	3.275E–06	5.133E–06
128	2.943E–06	4.886E–06
256	3.218E–06	5.173E–06

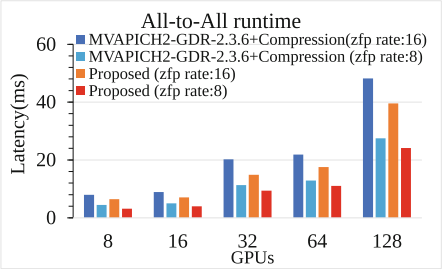
Figure 10(b) depicts the total runtime per time step in the application. Despite the use of ZFP (rate:24) with low compression ratio for PSDNS, we are still able to show overall improvements in the application execution time. The proposed design with optimized ZFP (rate:24) reduces the total runtime by up to 21.8% on 64 GPUs compared to the MVAPICH2-GDR-2.3.6 with compression. These results demonstrate the scalability of the proposed design. The proposed design could be applied to larger scales due to the straightforward send/receive operations.

6.2 Deep Learning Application

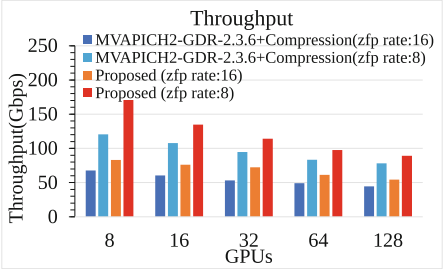
Given DeepSpeed's addition of Mixture-of-Experts support [10] which depends on All-to-All operations, we have evaluated our compression designs at the PyTorch level. To measure potential deep learning training benefits, we have implemented a communication benchmark in PyTorch and DeepSpeed [22]. Specifically, our benchmark initializes MPI through DeepSpeed, initializes PyTorch tensors of varying sizes, and calls MPI_Alltoall on each tensor. We conduct the experiments on the Lassen system. For different numbers of GPUs, we use the following tensor sizes as shown in Table 3.

Table 3. Tensor size and message size

GPUs	Tensor size (Bytes)	Message size (Bytes)
8	$2097152 \times 8 \times 4$	8M
16	$1048576 \times 16 \times 4$	4M
32	$1048576 \times 32 \times 4$	4M
64	$524288 \times 64 \times 4$	2M
128	$524288 \times 128 \times 4$	2M



(a) MPI_Alltoall runtime (Lower is better)



(b) Throughput (Higher is better)

Fig. 11. MPI_Alltoall runtime and throughput in DeepSpeed benchmark on Lassen. The proposed design with optimized ZFP (rate:16) reduces the MPI_Alltoall latency by up to 26.4% and improves the throughput by up to 35.8% on 32 GPUs compared to the MVAPICH2-GDR-2.3.6 with point-to-point based compression.

Figure 11(a) shows MPI_Alltoall runtime in DeepSpeed on Lassen system with 4 GPUs per node. Figure 11(b) shows the throughput result. Similar to the PSDNS application, we compare our proposed design with the state-of-the-art MVAPICH2-GDR-2.3.6. The proposed design with optimized ZFP (rate:16) reduces the MPI_Alltoall latency by up to 26.4% and improves the throughput by up to 35.8% on 32 GPUs compared to MVAPICH2-GDR-2.3.6 with point-to-point based compression support. These results demonstrate the potential benefits for deep learning training.

7 Related Work

MPI_Alltoall communication operations [7] are data-intense operations in modern HPC and Deep Learning applications. In [1], Bruck et al. evaluate MPI_Alltoall collective algorithms, and propose efficient MPI_Alltoall operation implementations for multi port message-passing systems. In [26,27], Singh et al. utilize CUDA-aware MPI to implement the GPU-based MPI_Alltoall collective operations. More recently, with the advent of NVLINK interconnects on modern GPU clusters, additional design challenges are incorporated in the adaptive MPI_Alltoall design [8]. However, no work has been done to optimize GPU-based MPI_Alltoall operations using a GPU-based compression in MPI run-time. In previous work, Filgueira et al. [4] use CPU-based lossless compression algorithms for MPI communication, CoMPI, to show host-based benefits of compression. Jin et al. [6] show high compression throughput for large-scale HPC applications through using GPU-based lossy compression algorithms. Zhou et al. [32] proposed a framework to integrate the GPU-based compression algorithms MPC [31] and ZFP [14] into MPI library to realize online compression for point-to-point based GPU communication. Recently, Tian et al. proposed cuSZ [30] with dual-quantization schemes for NVIDIA GPU architectures. A recent lossless GPU-based compression library built by NVIDIA, nvCOMP [19], provides a compression interface for applications.

8 Conclusion

In this paper, we propose a host-staging based scheme with online compression in the MPI library for MPI_Alltoall communication of large GPU data. Moreover, we move the compression to the collective level and optimize the existing ZFP compression library to enable the compression/decompression kernels to run on multiple CUDA streams to achieve overlap across the send/receive operations and improve the performance of MPI_Alltoall while maintaining data validation and not affecting the convergence time. The proposed design demonstrates up to 87.1% reduced MPI_Alltoall communication latency at the benchmark-level. At the application level, we compare the proposed design to the state-of-the-art MPI library MVAPICH2-GDR-2.3.6 with point-to-point compression. In the PSDNS application, the proposed design yields up to 21.8% reduced overall

running time. In the DeepSpeed benchmark, the proposed design reduces the MPI_Alltoall runtime by up to 26.4%.

As future work, we plan to study and incorporate more GPU-based compression algorithms, like cuSZ [30] and NVIDIA's nvCOMP [19]. To analyze the communication time in the compression design, we plan to utilize real-time monitor tools like OSU INAM [11,12]. Also, we plan to exploit the online compression design for various collective communications like All-Reduce and study the impact on more HPC and Deep Learning applications.

Acknowledgment. The authors would like to thank Kiran Ravikumar and Prof. P.K. Yeung from Georgia Institute of Technology for guiding conducting experiments with the 3D-FFT kernel of application PSDNS.

References

1. Bruck, J., Ho, C.T., Kipnis, S., Upfal, E., Weathersby, D.: Efficient algorithms for All-to-All communications in multiport message-passing systems. IEEE Trans. Parallel Distrib. Syst. **8**(11), 1143–1156 (1997)
2. Chu, C.H., Kousha, P., Awan, A.A., Khorassani, K.S., Subramoni, H., Panda, D.K.: NV-group: link-efficient reduction for distributed deep learning on modern dense GPU systems. In: Proceedings of the 34th ACM International Conference on Supercomputing (2020)
3. Di, S., Cappello, F.: Fast error-bounded lossy HPC data compression with SZ. In: International Parallel and Distributed Processing Symposium (IPDPS) (2016)
4. Filgueira, R., Singh, D., Calderón, A., Carretero, J.: CoMPI: enhancing MPI based applications performance and scalability using run-time compression. In: European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting, pp. 207–218 (2009)
5. IBM: IBM Spectrum MPI: accelerating high-performance application parallelization (2018). https://www.ibm.com/us-en/marketplace/spectrum-mpi. Accessed 13 May 2022
6. Jin, S., et al.: Understanding GPU-Based Lossy Compression for Extreme-Scale Cosmological Simulations. ArXiv:abs/2004.00224 (2020)
7. Kale, L., Kumar, S., Varadarajan, K.: A framework for collective personalized communication. In: Proceedings International Parallel and Distributed Processing Symposium, p. 9 (2003). https://doi.org/10.1109/IPDPS.2003.1213166
8. Khorassani, K.S., Chu, C.H., Anthony, Q.G., Subramoni, H., Panda, D.K.: Adaptive and hierarchical large message All-to-All communication algorithms for large-scale dense GPU systems. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 113–122 (2021). https://doi.org/10.1109/CCGrid51090.2021.00021
9. Khorassani, K.S., Chu, C.H., Subramoni, H., Panda, D.K.: Performance evaluation of MPI libraries on GPU-enabled OpenPOWER architectures: early experiences. In: International Workshop on OpenPOWER for HPC (IWOPH 19) at the 2019 ISC High Performance Conference (2018)
10. Kim, Y.J., et al.: Scalable and efficient MOE training for multitask multilingual models (2021)

11. Kousha, P., et al.: Designing a profiling and visualization tool for scalable and in-depth analysis of high-performance GPU clusters. In: 2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC), pp. 93–102 (2019). https://doi.org/10.1109/HiPC.2019.00022
12. Kousha, P., et al.: INAM: Cross-Stack Profiling and Analysis of Communication in MPI-Based Applications. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3437359.3465582
13. Lawrence Livermore National Laboratory: lassen—high performance computing (2018). https://hpc.llnl.gov/hardware/platforms/lassen. Accessed 13 March 2022
14. Lindstrom, P.: Fixed-rate compressed floating-point arrays. IEEE Trans. Visualiz. Comput. Graph. **20** (2014). https://doi.org/10.1109/TVCG.2014.2346458
15. Liquid Submerged System - Texas Advanced Computing Center, Frontera - Specifications. https://www.tacc.utexas.edu/systems/frontera
16. Longhorn - Texas Advanced Computing Center Frontera - User Guide. https://portal.tacc.utexas.edu/user-guides/longhorn
17. Network-Based Computing Laboratory: MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE (2001). http://mvapich.cse.ohio-state.edu/. Accessed 13 March 2022
18. NVIDIA: NVIDIA GPUDirect (2011). https://developer.nvidia.com/gpudirect. Accessed 13 March 2022
19. NVIDIA: nvCOMP (2020). https://github.com/NVIDIA/nvcomp. Accessed 13 March 2022
20. Open MPI: Open MPI: Open Source High Performance Computing (2004). https://www.open-mpi.org/. Accessed 13 March 2022
21. Potluri, S., Hamidouche, K., Venkatesh, A., Bureddy, D., Panda, D.K.: Efficient inter-node MPI communication using GPUDirect RDMA for infiniBand clusters with NVIDIA GPUs. In: 42nd International Conference on Parallel Processing (ICPP), pp. 80–89. IEEE (2013)
22. Rasley, J., Rajbhandari, S., Ruwase, O., He, Y.: Deepspeed: system optimizations enable training deep learning models with over 100 billion parameters. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 3505–3506. KDD 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3394486.3406703
23. Ravikumar, K., Appelhans, D., Yeung, P.K.: GPU acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3295500.3356209
24. Sharkawi, S.S., Chochia, G.A.: Communication protocol optimization for enhanced GPU performance. IBM J. Res. Develop. **64**(3/4), 9:1–9:9 (2020)
25. Shi, R., et al.: Designing efficient small message transfer mechanism for inter-node MPI communication on InfiniBand GPU clusters. In: 2014 21st International Conference on High Performance Computing (HiPC), pp. 1–10 (2014)
26. Singh, A.K., Potluri, S., Wang, H., Kandalla, K., Sur, S., Panda, D.K.: MPI AlltoAll personalized exchange on GPGPU clusters: design alternatives and benefit. In: 2011 IEEE International Conference on Cluster Computing, pp. 420–427 (2011)
27. Singh, A.K.: Optimizing All-to-All and Allgather Communications on GPGPU Clusters. Master's thesis, The Ohio State University (2012)

28. Singh, A.K., Potluri, S., Wang, H., Kandalla, K., Sur, S., Panda, D.K.: MPI AlltoAll personalized exchange on GPGPU clusters: design alternatives and benefit. In: 2011 IEEE International Conference on Cluster Computing, pp. 420–427 (2011). https://doi.org/10.1109/CLUSTER.2011.67
29. Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of collective communication operations in MPICH. Int. J. High Perform. Comput. Appl. **19**(1), 49–66 (2005). https://doi.org/10.1177/1094342005051521
30. Tian, J., et al.: CUSZ: an efficient GPU-based error-bounded lossy compression framework for scientific data. In: Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques, pp. 3–15. PACT 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3410463.3414624
31. Yang, A., Mukka, H., Hesaaraki, F., Burtscher, M.: MPC: a massively parallel compression algorithm for scientific data. In: IEEE Cluster Conference (2015)
32. Zhou, Q., et al.: Designing high-performance MPI libraries with on-the-fly compression for modern GPU clusters*. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 444–453 (2021). https://doi.org/10.1109/IPDPS49936.2021.00053