*gh*ZCCL: Advancing GPU-aware Collective Communications with Homomorphic Compression

Jiajun Huang*
University of South Florida
Tampa, FL, USA
jiajun.huang.cs@gmail.com

Sheng Di
Argonne National Laboratory
Lemont, IL, USA
sdi1@anl.gov

Yafan Huang
University of Iowa
Iowa City, IA, USA
yafan-huang@uiowa.edu

Zizhong Chen
University of California, Riverside
Riverside, CA, USA
chen@cs.ucr.edu

Franck Cappello
Argonne National Laboratory
Lemont, IL, USA
cappello@mcs.anl.gov

Yanfei Guo
Argonne National Laboratory
Lemont, IL, USA
yguo@anl.gov

Rajeev Thakur
Argonne National Laboratory
Lemont, IL, USA
thakur@anl.gov

ABSTRACT

In the exascale computing era, collective communication has emerged as a significant bottleneck for GPU-based applications, as network bandwidth lags behind rapid GPU advancements. While traditional GPU-aware approaches employ error-bounded lossy compression to mitigate this issue, they incur substantial decompression-operation-compression (DOC) overhead. To overcome these limitations, we introduce *gh*ZCCL, a GPU-aware homomorphic compression-accelerated collective communications library that enables direct computation and communication on compressed data, eliminating the DOC workflow. We design the first GPU homomorphic compressor, surpassing the fastest existing GPU lossy compressor, cuSZp2, by 3.47–3.89× for DOC workloads. We also propose co-design strategies to further optimize GPU-aware collective communications with homomorphic compression. Experiments on up to 512 NVIDIA A100 GPUs show that *gh*ZCCL significantly outperforms three state-of-the-art communication libraries—gZCCL, NCCL, and Cray MPI—by achieving speedups of up to 2.29×, 5.81×, and 188×, respectively, while maintaining high data accuracy.

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**; **Parallel computing methodologies**; • **General and reference** → **Performance**.

*Corresponding author.

ICS '25, June 8–11, 2025, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/2025/06
https://doi.org/10.1145/3721145.3733642

KEYWORDS

GPU, Collective Communications, Compression

ACM Reference Format:
Jiajun Huang, Sheng Di, Yafan Huang, Zizhong Chen, Franck Cappello, Yanfei Guo, and Rajeev Thakur. 2025. *gh*ZCCL: Advancing GPU-aware Collective Communications with Homomorphic Compression. In *2025 International Conference on Supercomputing (ICS '25), June 8–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3721145.3733642

1 INTRODUCTION

As network bandwidth struggles to keep pace with the rapid growth of GPU computing power, the efficiency of collective communication has become a critical bottleneck for exascale distributed and parallel GPU applications. This bottleneck is particularly prominent in scientific computing and deep learning tasks, where extensive data processing and exchange are required [1, 2, 4, 5, 16]. For example, the Allreduce collective communication alone consumes 73.2% of the training time for the GPT-2-Large model [9]. This inefficiency in GPU-aware collective communication leads to considerable resource wastage, including computing power, energy, and financial costs. Therefore, optimizing GPU-aware collective communication, particularly for large message sizes, is an urgent priority [7, 10].

1.1 Motivation for Compression-Accelerated Collectives

The internode collective communication performance is often the major bottleneck for the efficiency of collective communications because of the limited network bandwidth. With the vast amounts of data processed and exchanged in the large-scale applications, the network is easily saturated, which significantly limits the efficiency of collective communications and the scalability of these applications. To

facilitate the saturated networks, in the past, researchers proposed different bandwidth-optimized collective algorithms to decrease the overall collective communication volume [3, 37, 39]. However, these algorithms have already approached their limits in enhancing the collective performance and have limited space for further optimization. With the recent development of GPU-based ultra-fast error-bounded lossy compression [24, 25, 46], it is now possible to utilize error-bounded lossy compression to significantly reduce message sizes and accelerate collective communications while preserving high data quality.

1.2 Limitations of Existing Works and Goal

Recently, researchers have proposed several compression-accelerated collective communication libraries that achieve sound speedups over the previous approaches without compression support while maintaining high data accuracy as shown in Table 1. However, the existing state-of-the-art communication libraries all demonstrate certain limitations. The HPE Cray-MPI [17] is a GPU-aware MPI that reaches high performance in Cray Systems (two out of three US exascale supercomputers communicate with Cray MPI [41]). Although its collectives are GPU-aware, they still rely on intermediate CPU buffers, which leads to suboptimal performance [42]. NCCL [12] is another high-performance collective communications library on systems with NVIDIA GPUs. It is GPU-centric but lacks compression support, which significantly limits its collective performance. The state-of-the-art C-Coll [21, 22] and gZCCL [19] utilize error-bounded lossy compression to accelerate collective communications on CPU and GPU clusters, respectively. However, they are subjected to time-consuming Decompression-Operation-Compression (DOC) workflow, in which each process has to decompress the compressed data before applying operations and then recompress the operated data.

Collective Commu. Libraries	GPU-centric Design?	Compression Support?	Accuracy Control?	Co-designed Compression?	Homomorphic Capability?
Cray-MPI	✗	✗	—	✗	✗
NCCL	✓	✗	—	✗	✗
C-Coll	✗	✓	✓	✗	✗
gZCCL	✓	✓	✓	✗	✗
***gh*ZCCL (our work)**	✓	✓	✓	✓	✓

Table 1: Key designs of state-of-the-art collective communications libraries. The Homomorphic Capability means that GPUs can directly operate on compressed data.

An ideal GPU-aware, compression-accelerated collective communications library should meet the following criteria:

- GPU-centric design that avoids host-device data transfers and CPU computation overheads.
- Compression support with accuracy control to achieve performance improvements with high data quality.
- Co-designed compression to maximize both throughput and compression ratio.
- Direct GPU operations on compressed data during intensive communications, eliminating the need for expensive DOC workflows.

To propose such an ideal solution, several new challenges must be addressed:

- Homomorphic compression: How can GPUs compute with compressed data during communication, removing the need for costly DOC workflows? *Currently, no existing GPU compressors offer this functionality.*
- Performance vs. quality: How can we ensure that a GPU homomorphic workflow delivers high compression performance without sacrificing quality?
- Co-design with collective communications: How can we co-design this new compression workflow with collective communications to achieve the best overall performance for GPU-centric communication?

1.3 Our solution: *gh*ZCCL

To address the aforementioned limitations of existing works and new challenges, we propose ***gh*ZCCL**, which is a **G**PU-aware **h**omomorphic **compression**-accelerated **c**ollective **c**ommunications **l**ibrary that allows GPUs to directly compute and communicate with compressed data. To the best of our knowledge, *gh*ZCCL is the *first-ever* GPU-aware homomorphic compression-communication co-design. To be specific, there are three key designs in *gh*ZCCL: (1) Novel workflow: *Pioneering GPU homomorphic compression pipeline.* This ultra-fast lossless homomorphic compression pipeline diminishes the needs for complete decompression and recompression while maintaining the same data quality compared with the original DOC workflow, allowing *gh*ZCCL's homomorphic compressor—*ghZ* achieves extreme compression throughput. (2) Throughput optimization: *Fused lightweight compression kernel.* It conducts partial decompression, operation, and partial recompression with a single kernel, significantly decreases the kernel launching overheads and increases memory access efficiency. (3) Communication co-design: *GPU-centric homomorphic compression-communication co-design.* This GPU-centric co-design supports different collective computation operations and soundly improves the collective communication efficiency on modern GPU clusters. We evaluate *gh*ZCCL with various application datasets across up-to 512 NVIDIA A100 GPUs and present some key findings below:

- *gh*Z achieves 531.91–942.44 GB/s averaged DOC-handling throughput on NVIDIA A100 GPU that is 3.47–3.89× faster than the current fastest lossy compressor—cuSZp2.

- *gh*Z maintains the same high compression quality and compression ratio compared to the traditional error-bounded lossy compressor across different application datasets.
- *gh*ZCCL significantly outperforms state-of-the-art communication libraries. Specifically, *gh*ZCCL-accelerated Reduce delivers up to 1.34×, 3.85×, and 188× performance improvements and *gh*ZCCL-accelerated Allreduce achieves up to 2.29×, 5.81 ×, and 6.01 × speedups compared with gZCCL, NCCL, and Cray-MPI, respectively.
- The practical use case — image stacking analysis shows that *gh*ZCCL surpasses gZCCL and reaches 1.91× and 2.34× performance improvements over gZCCL and NCCL while preserving both statistical and visual accuracies.

The remainder of this paper is structured as follows: Section 2 provides an overview of the background and related work. Sections 3, 4, and 5 detail our design and optimization strategies. Section 6 discusses the evaluation findings. Finally, Section 7 concludes the paper and outlines directions for future work.

2 BACKGROUND AND RELATED WORK

In the exascale era, researchers are actively developing high-speed GPU lossy compressors due to two reasons: (1) error-bounded lossy compressors can provide a much higher compression ratio than the loss compressors while strictly bounding the compression error between the reconstructed data and the original data within a user-specified threshold [23, 31, 47]. (2) GPU compressors can provide orders of magnitude higher compression throughputs than the CPU compressors because of the tremendous computing power of modern GPUs. There are many types of GPU lossy compressors [24, 25, 33, 34, 40, 45]. Among them, the cuSZp2 [24] is considered to be the state-of-the-art error-bounded GPU compressor that provides significantly higher compression throughput than other compressors (2× and 200× faster than pure-GPU and CPU-GPU lossy compressors, respectively) with a high compression ratio and reconstructed data quality. However, none of these GPU compressors can apply operations on compressed data. In contrast, our GPU homomorphic compressor, *gh*Z, performs direct calculations on compressed data, which provides 3.89× speedup compared with the traditional decompression-operation-compression (DOC) workflow of the fastest cuSZp2 GPU compressor.

Error-bounded lossy compression has been demonstrated to effectively accelerate numerous scientific applications and AI workloads, including checkpointing [38], quantum simulations [44], partial differential equation (PDE) simulations [8], and deep learning [26, 27], all yielding validated application results. More recently, researchers have leveraged compression techniques to improve the communication performance of high-performance clusters [18–22, 32, 43, 48, 49]. Among these, error-bounded lossy compression-based solutions [19, 21, 22] have gained significant attention for their ability to precisely control error propagation, as demonstrated through both theoretical and experimental analyses in [22]. However, existing compression-accelerated communication methods, including the state-of-the-art GPU-aware gZCCL, rely on a traditional DOC workflow, which introduces significant overheads in collective communications. In contrast, our *gh*ZCCL supports direct computation and communication on compressed data on GPUs, significantly improving DOC-handling efficiency and enhancing collective communication performance, as detailed in Section 6.

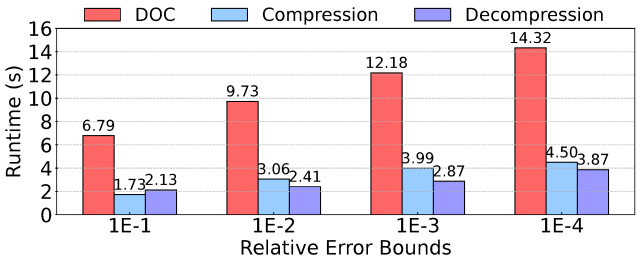
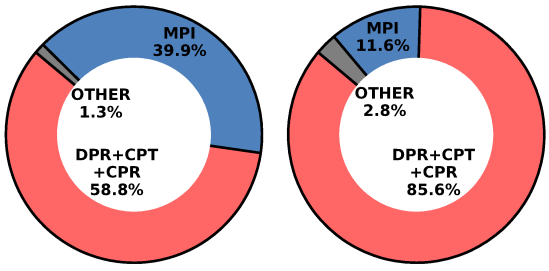


Figure 1: Compare the runtime of DOC workflow with a single compression/decompression.

3 RETHINKING GPU COMPRESSION: WHY HOMOMORPHIC COMPRESSION?

GPU compressors have been widely-used in many data-intensive applications/routines, including the collective computation operations (Reduce_scatter, Allreduce, etc.) we focus on in this paper. To utilize GPU compressors in real-world applications, users typically compress the original data to reduce the storage space, memory footprint and data movement costs [19, 35, 44]. Then, the compressed data need to be fully decompressed if users need to analyze or conduct operations on the whole original data. After that, if the data has been modified, users need to fully recompress the operated data into compressed format to reduce the storage cost or accelerate the applications/routines. Thus, this process is rather time-consuming compared with a single compression/decompression operation. This scenario is even worse when users need to compute with two compressed data inputs, since we need two decompression operations to decompress the two compressed data inputs, one computation operation to compute with the two decompressed data, and another compression operation to recompress the operated data. For instance, in Figure 1, we evaluate the runtime of the DOC workflow compared with a single compression/decompression using the state-of-the-art GPU error-bounded lossy compressor, cuSZp2. We can notice that DOC consumes



(a) gZCCL (Ring) (b) gZCCL (ReDoub)

Figure 2: Performance breakdown of gZCCL's ring-based and recursive_doubling-based Allreduce operations.

3–4× more runtime compared with one compression/decompression operation. This finding prompts a question: would the DOC workflow becomes a major bottleneck for GPU-aware compression-accelerated collective communications?

In the state-of-the-art gZCCL framework, GPUs need to decompress the received data before applying operations and recompress the operated data before send them to other GPUs. This process is exactly the same as the DOC workflow mentioned earlier. To understand the overheads of the DOC workflow inside of the collective communications, we breakdown the performance of the gZCCL framework with both the ring-based and recursive_doubling-based Allreduce operations. In Figure 2, DPR+CPT+CPR represents the total time spent on decompression, computation, and compression, which corresponds to the DOC workflow. MPI stands for the runtime of MPI communications, while OTHER encompasses the time consumed by other operations. We can observe that the DPR+CPT+CPR dominates the two cases, with a significant proportion of 58.8% in the ring-based approach and 85.6% in the recursive_doubling-based approach. This observation underscores the limitations of the traditional DOC workflow in enhancing the overall performance of collective communications.

Design Takeaway 1: The DOC workflow is the *main bottleneck* in GPU-aware compression-accelerated collective communications. To resolve this, we need a high-throughput *GPU homomorphic compression kernel* that efficiently processes the DOC workflow.

4 *GHZCCL*: HIGH-LEVEL OVERVIEW

We propose *gh*ZCCL, a pioneering GPU-aware homomorphic compression-accelerated collective communications library that allows GPUs to directly compute and communicate with compressed data. There are two key components of *gh*ZCCL: (1) a lightning-fast single-kernel GPU homomorphic compressor—*gh*Z. (2) a GPU-aware homomorphic compression-communication co-design. In this section, we present the high-level overview of *gh*Z and the co-design.

4.1 *ghZ* Overview

Figure 3 shows the high-level overview of the *gh*Z workflow when compared with the traditional DOC workflow of the state-of-the-art cuSZp2. In the traditional DOC workflow, the GPU first initiates a GPU decompression kernel to decompress the compressed byte array into a float array. This decompression process started with a Global Synchronization (1) that retrieves the index of each compressed block within the compressed byte array. Then the Lossless Decoding (2) decodes the byte array to an integer array. After that, the Lossless Transform (3) converts the integer array to a decompressed float array. The GPU needs to repeat steps 1–3 to decompress another compressed byte array. Later, the two decompressed float arrays are combined together through a GPU Computation Kernel (4). Next, the operated float array is compressed by the GPU compression kernel into the compressed format. The compression kernel starts with a Error-bounded Transform (5) that converts the float array into integer array. Then, the Global Synchronization (1) obtains the index of compressed blocks and the Lossless Encoding (6) transforms the Int array into the compressed bytes. In total, the DOC workflow consists of *four* kernel launching and *ten* individual processing stages.

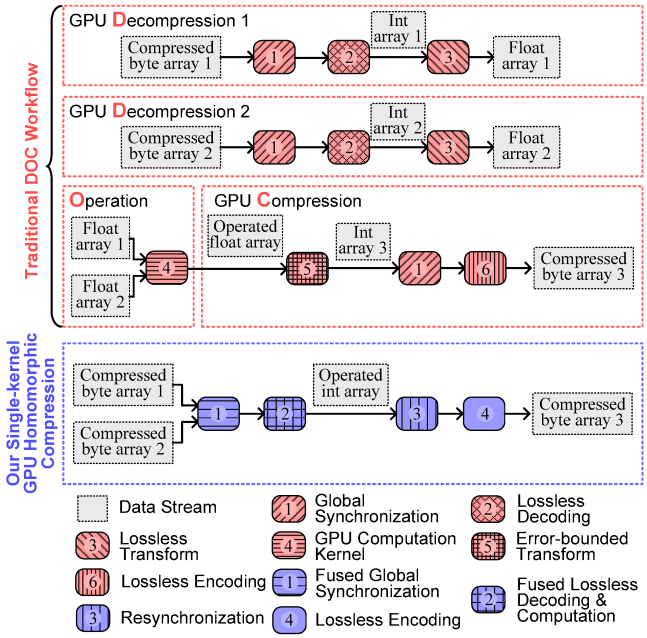


Figure 3: High-level overview of *ghZ* when compared with the traditional DOC workflow.

In contrast, our *gh*Z features a *single*-kernel extreme-throughput homomorphic compression workflow that is able to directly operate with compressed data inputs with only *four* processing stages. First, the two compressed byte arrays

undergo a **Fused Global Synchronization** (1), which retrieves offsets of the two groups of compressed data blocks with a fused device-level parallel prefix-sum. Then, based on the two sets of indexes obtained, *gh*Z decode the two compressed byte arrays using a **Fused Lossless Decoding & Computation** (2) that transforms and combines the byte arrays into a single operated integer array. Then, the GPU threads synchronize with each other in a **Resynchronization** (3) to obtain the new compressed bytes offsets. Finally, the integer array is encoded into a compressed byte array through a **Lossless Encoding** (4).

Design Takeaway 2: The GPU homomorphic compression workflow of *gh*Z surpasses the DOC workflow of cuSZp2. By cutting kernel launches from *four* to *one* and processing stages from *ten* to *four*, it significantly boosts DOC-handling latency and throughput.

4.2 *gh*ZCCL Co-design Overview

As shown in Figure 4, we present a high-level overview of our compression-communication co-design. To fully exploit the unique benefits of GPU homomorphic compression in collective communications, we propose a GPU-centric homomorphic compression-accelerated collective communication framework specifically designed for computation-intensive collective operations. First, we develop **Co-designed Algorithms** (5) to improve performance and GPU utilization across diverse input data sizes and GPU counts for GPU-aware homomorphic compression-accelerated collective communications. These algorithms outperform the gZCCL's collective algorithms that rely on the traditional DOC workflow. Next, we further enhance the efficiency of these algorithms with a co-designed **In-place *gh*Z** (6), which effectively minimizes GPU memory usage and data copying overhead. Additional key optimizations include **Adaptive Vectorized Memory Access** (7), which allow both *gh*Z and cuSZp2 to adaptively access GPU main memory in a vectorized manner during intensive collective communications, and **Multi-stream Compression** (8), which overlaps compression kernels to significantly reduce runtime.

5 *GH*ZCCL: KEY DESIGNS

In this section, we present the eight key designs of *gh*ZCCL. For clarity, the step numbers (e.g., 1) in the text continue to refer to Figures 3 and 4.

5.1 *gh*Z: Design Details

We now explore the design of *gh*Z in detail. The components 1–4 correspond to those labeled 1–4 in Figure 3.

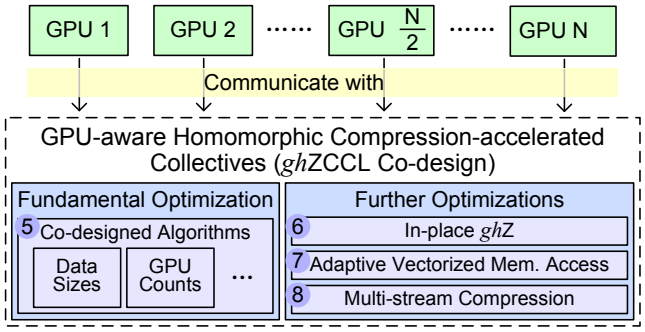


Figure 4: High-level overview of *gh*ZCCL.

5.1.1 Fused Global Synchronization 1. To directly operate on compressed data inputs, the first step is to retrieve the offsets for each compressed data block through a fused synchronization strategy, enabling further processing. To better introduce this synchronization approach, we first explain the high-level compression workflow. The original data is divided into small blocks (e.g., 32 floating-point data points per block), with each thread compressing a single block per iteration. To ensure memory coalescing, threads within the same warp process neighboring blocks. This cycle repeats 32 times, resulting in each warp compressing a total of $32 \times 32 = 1024$ blocks. Since compressed data blocks may have varying sizes, the exact locations of each block within the compressed data cannot be predetermined. Consequently, threads must synchronize to communicate and calculate offset information.

In our *gh*Z, we employ a Fused Global Synchronization to determine the specific locations of compressed blocks, eliminating the need for independent synchronizations for each input, as required in the DOC workflow. First, each GPU thread calculates the total compressed data offsets for two sets of 32 data blocks using fixed-rate information, which specifies the number of bits used to encode each data point within a block. After determining the thread-level offsets, an inclusive warp-level prefix sum is applied to calculate the two total compressed data sizes for a warp of the compressed byte arrays. This step is optimized using __shfl_up_sync, enabling efficient in-warp communication. Subsequently, the last thread of each warp writes the total compressed data sizes to temporary global memory buffers. An exclusive global-level prefix sum is then performed to compute the global compressed data offsets for each warp in the byte arrays. This process utilizes the decoupled look-back technique described in [13, 24]. By employing Fused Global Synchronization, we significantly reduce synchronization latency compared to the DOC workflow, enhancing overall efficiency.

5.1.2 Fused Lossless Decoding & Computation 2. In *gh*Z, we introduce a Fused Lossless Decoding & Computation approach to partially decode compressed byte arrays and perform calculations directly on integer data. In contrast, the

traditional DOC workflow requires fully decompressing the compressed data into floating-point arrays and launching an additional GPU kernel to process the two floating-point data inputs, as illustrated in 3. For each data block, the GPU thread first performs warp-level communication to determine the compressed byte offsets of the two compressed data blocks. This is achieved using the previously obtained fixed-rate information and the __shfl_up_sync primitive. If both fixed rates are zero, the two blocks are skipped to save processing time. If at least one block has a non-zero fixed rate, *gh*Z retrieves the sign flags (e.g., 32 sign bits per block) for the two compressed data blocks in a vectorized manner.

Next, for each compressed block, *gh*Z's bit-shuffle-based fixed-length decoder partially decompresses the corresponding compressed bytes into two 32-element integer arrays. It then directly computes using the integer arrays to generate a new sign-bit array, an operated integer array, and an updated fixed rate, all within a single loop optimized with loop unrolling. The fixed-rate information is subsequently stored in the operated compressed data, and __shfl_sync is employed to update the compressed data offsets across different iterations of data blocks. This Fused Lossless Decoding & Computation method delivers significantly higher throughput compared to the DOC workflow, which always requires fully decompressing the two compressed data arrays and operating on floating-point data.

5.1.3 Resynchronization and Lossless Encoding 3 *&* 4*.* After obtaining the operated integer array, *gh*Z performs a Resynchronization step to determine the compressed byte offsets of the newly compressed operated data blocks and applies a lossless encoding to encode the integer blocks into compressed bytes. This process is significantly more lightweight compared to the full recompression required by the traditional DOC workflow, as shown in Figure 3. The Resynchronization process involves a warp-level synchronization followed by a global-level synchronization, akin to the Fused Global Synchronization in *gh*Z. During the Lossless Encoding phase, the previously obtained fixed-rate information for each block is utilized to store only the fixed number of bits required for each element in the block (e.g., 4 bits per element). This approach significantly reduces storage space compared to storing the full 32 bits for each element.

Design Takeaway 3: The lightweight design of *gh*Z *optimizes memory access* and *reduces computational costs*, significantly outperforming the traditional DOC workflow in compression efficiency.

5.2 *gh*ZCCL: Co-design Details

In this section, we delve into the co-design details of *gh*ZCCL, with components 5–8 corresponding to those in Figure 4.

5.2.1 GPU-aware homomorphic compression-accelerated collective algorithms 5*.* To effectively leverage homomorphic compression in GPU-aware collective communications, the foundational step is to co-design the collective communication algorithms. The state-of-the-art GPU-aware compression-accelerated collective framework, gZCCL [19], was specifically designed for the DOC workflow and is not compatible with homomorphic compression. To address this limitation, we propose different co-designed algorithms (e.g., ring-based and recursive_doubling-based Allreduce) to optimize collective performance across varying data sizes and GPU counts. In this subsection, we use the recursive_doubling-based Allreduce as an exemplar, and the same methodology can be easily applied to other algorithms.

While the ring-based Allreduce is widely employed for processing large messages in leading collective communication libraries such as MPICH [30] and NCCL [12], it can encounter GPU underutilization when the GPU count is large. This inefficiency arises because each GPU processes only D/N data per compression task, where D is the input data size and N is the number of GPUs [19]. To address this scalability challenge, we propose the GPU Homomorphic Compression-Accelerated Recursive_Doubling-Based Allreduce Algorithm. In Figure 5, we compare the high-level design of *gh*ZCCL with gZCCL in the recursive_doubling-based Allreduce algorithm for four GPUs. In gZCCL, each GPU first compresses its original data and sends the compressed data to the target GPU. Upon receiving the data, the target GPU decompresses it to reconstruct the original data, then launches a reduction kernel to operate on the two original data inputs. After obtaining the reduced result, the data is recompressed into a compressed format using another compression kernel. This DOC workflow repeats $\log N - 1$ times, where N is the number of GPUs. In the final round, the last received data is decompressed and combined with the previously reduced output. This round does not involve recompression since it produces the final reduced output. If the compression cost of the original data is CPR, the decompression cost is DPR, and the operation cost is OPR, the total computational cost in gZCCL's recursive_doubling-based Allreduce algorithm is: $T^{AR}_{gZCCL} = \log N \times (DPR + OPR + CPR)$.

In contrast, our *gh*ZCCL co-design, built around our GPU homomorphic compressor—*gh*Z—eliminates the costly DOC workflow used by gZCCL. In *gh*ZCCL, each GPU first concurrently compresses its original data, then exchanges the compressed data with other GPUs. Following the communication step, each GPU directly operates on the compressed data using our GPU homomorphic compressor, bypassing the need to decompress it. The resulting newly operated compressed data is then transmitted among GPUs. This communication and homomorphic compression process repeats

for $\log N - 1$ rounds. After the intensive communications, during the final round, the last received compressed data is directly computed with the previously reduced compressed output using the GPU homomorphic compressor. Finally, the reduced compressed data is decompressed to retrieve the original reduced output, completing the algorithm. The total cost of this algorithm is: $T^{AR}_{GHCL} = CPR + (\log N - 1) \times HPR + HPR + DPR = CPR + \log N \times HPR + DPR$, where HPR represents the homomorphic processing cost. The cost difference between gZCCL and *gh*ZCCL is: $T^{AR}_{gZCCL} - T^{AR}_{GHCL} = \log N(DPR + OPR + CPR - HPR) - CPR - DPR$. Since the traditional DOC cost $(DPR + OPR + CPR)$ is significantly higher than HPR, we conclude that *gh*ZCCL achieves much higher collective performance than gZCCL.

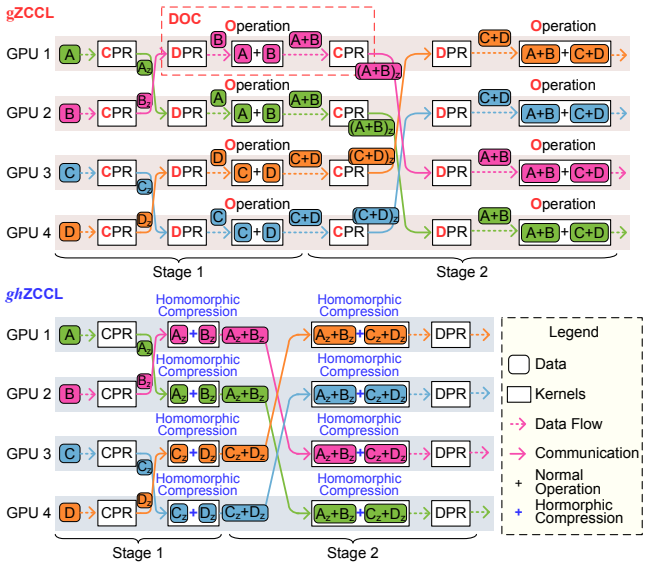


Figure 5: Compare the high-level design of *gh*ZCCL with gZCCL in the recursive_doubling-based Allreduce algorithm. This example uses four GPUs/processes.

5.2.2 In-place ghZ and Adaptive Vectorized Memory Access 6 *&* 7. In this section, we detail the designs of the In-place *gh*Z and Adaptive Vectorized Memory Access, both specifically co-designed to meet the needs of GPU-aware collective communications. Initially, the *gh*Z processes two compressed byte arrays as inputs and directly operates on them to produce an operated compressed byte array, which is stored in an additional GPU buffer. While this design already outperforms the DOC workflow, it leads to suboptimal performance and memory management in collective communication scenarios, particularly on GPUs where memory resources are constrained. In *gh*ZCCL, after a GPU receives compressed data from another GPU, the data is stored in a temporary GPU buffer called tmp_buf. This buffer, along with another input buffer outputBytes (which stores the previously reduced compressed data), is fed into the GPU homomorphic compression kernel. With the original *gh*Z, an additional GPU buffer would be required to store the newly operated compressed output. Subsequently, the data would need to be copied back to outputBytes for either local storage or further communication. This process increases both the memory footprint and runtime. To address this inefficiency, we co-designed an in-place *gh*Z, capable of directly writing the homomorphically compressed data into one of the input GPU buffers during the compression process. This approach reduces memory usage and improves runtime efficiency, enhancing performance for GPU-aware collectives.

We also propose the Adaptive Vectorized Memory Access to enable the vectorized memory access capability for compression tasks (including both normal compression and homomorphic compression) during collective communications to better exploit the GPU global memory bandwidth. In the collective communication scenario, it is common to divide input data into smaller data chunks for data communications. For example, the ring-based Reduce_scatter divides the input data of size D into N chunks, where N is the number of processes/GPUs. Then, each chunk will be compressed and communicated during the intensive communications. However, this can possibly result in the misaligned memory access issue if using vectorized memory access during the compression tasks because the start index for each chunk in the GPU receive buffer may not be a multiple of 4. Figure 6 illustrates the workflow of Adaptive Vectorized Memory Access during the normal compression process. Prior to the compression task, the data undergoes a three-step preprocessing procedure to prepare it for efficient handling from GPU global memory: (1) The starting address and length of the data chunk designated for compression and communication are analyzed to determine its suitability for vectorized processing. (2) If the data chunk is not vectorizable, the remainders at the starting and ending locations of the chunk are identified and retrieved to facilitate scalar operations. (3) If the data chunk is vectorizable, this step is bypassed, and the data is directly accessed and processed in a vectorized manner before proceeding to cuSZp compression. During the homomorphic compression process, these three steps are executed in the same sequence before the Fused Lossless Decoding & Computation phase and after the Lossless Encoding phase. Figure 7 presents a running example of a data

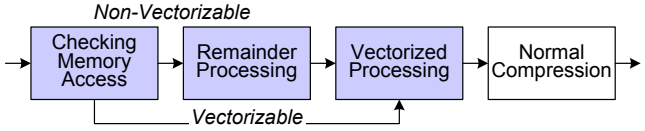


Figure 6: The Adaptive Vectorized Memory Access workflow in normal compression.

chunk processed using the proposed Adaptive Vectorized Memory Access technique with the normal compression process. The original application data is stored in a 1D layout within computer systems. When processed by our technique, any remainders at the boundaries of the data chunk are identified and handled through scalar operations. This ensures that the remaining data points are fully vectorized, enabling ultra-fast processing speeds.

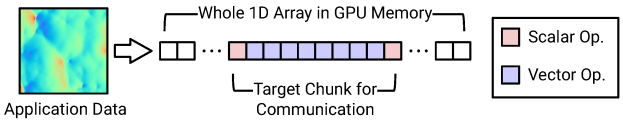


Figure 7: A running example of Adaptive Vectorized Memory Access in normal compression.

5.2.3 Multi-stream Compression 8. In the ring-based Reduce_scatter algorithm, we utilize our developed multi-stream compression to overlap compression kernels with each other, improving GPU utilization in our co-designed algorithm. In the original ring-based Reduce_scatter algorithm within gZCCL, each GPU compresses a single chunk of data, sends it to the next GPU, receives compressed data from the previous GPU, performs a DOC workflow on the received data, and sends the updated data to the next GPU. This process repeats for $N-1$ rounds, where N is the number of GPUs. Consequently, the total computational cost can be expressed as: $T^{RS}_{gZCCL} = (N-1)\times(DPR+OPR+CPR)$, where DPR, OPR, and CPR are the runtimes for decompression, operation, and compression of a single data chunk of size D/N, where D is the input data size.

By comparison, our *gh*ZCCL co-design compresses all small chunks of the original data at the very beginning, prior to intensive communications. Subsequently, all GPUs engage in a ring-like communication pattern, directly transmitting and operating on compressed data chunks. Thus, we have the opportunity to overlap the compression kernels using our developed multi-stream compression. Since compressing a single chunk often leads to underutilized GPU device, overlapping compression tasks further enhances performance, complementing the improvements achieved through homomorphic compression. As a result, the total cost for our *gh*ZCCL can be expressed as: $T^{RS}_{GHCL} = (N-1)\times HPR+DPR+CPR_{Overlap} \approx (N-1)\times HPR+DPR+CPR$, where HPR represents the homomorphic processing runtime, and $CPR_{Overlap}$ reflects the reduced compression cost due to overlapping. Accordingly, the time difference between gZCCL and *gh*ZCCL is: $T^{RS}_{gZCCL} - T^{RS}_{GHCL} = (N-1)\times(DPR+OPR+CPR-HPR) - DPR-CPR = (N-1)\times(DOC-HPR) - DPR-CPR$. Since the DOC cost is considerably higher than HPR, the relative performance of *gh*ZCCL improves as the number of GPUs (N) increases. A larger GPU count further amplifies the performance advantage of *gh*ZCCL over the SOTA gZCCL.

Design Takeaway 4: The co-design and optimizations of *gh*ZCCL enhance the *efficiency* and *scalability* of GPU-based collective communications, outperforming gZCCL, which relies on the DOC workflow.

6 EXPERIMENTAL EVALUATION

The following sections discuss the evaluation results.

6.1 Experimental Setup

We evaluate on a GPU supercomputer with 512 NVIDIA A100 80G GPUs (128 nodes, 4 GPUs each). Nodes are connected with HPE Slingshot 11 (200 Gbps). The default compression error bound is 1E-4, unless stated otherwise.

Seven real-world scientific applications from diverse domains are evaluated, as summarized in Table 2. These include two different RTM application datasets generated under distinct simulation settings of the 3D SEG/EAGE Overthrust model [28], and the CESM-ATM application dataset derived from the atmospheric model of the CESM climate simulation package [11]. The Nyx application dataset is produced from a cosmological hydrodynamics simulation using adaptive mesh [36], while the JetIn application dataset represents the Q-criterion of a jet in crossflow, created through direct numerical simulation [14]. The SynTruss application dataset simulates a CT scan of an 8x8x8 octet truss with five defects on its front side [29], and the HCCI application dataset captures the first timestep of a direct numerical simulation of autoignition in stratified dimethyl-ether/air turbulent mixtures [6]. These application datasets span a wide range of scientific challenges and computational domains, ensuring comprehensive evaluation.

Simulation Setting 1: Seismic Wave Application		
151 fields	dim: $512_x \times 512_y \times 512_z$	total: 95.3 GB
Simulation Setting 2: Seismic Wave Application		
3 fields	dim: $1008_x \times 1008_y \times 352_z$	total: 4.0 GB
CESM-ATM: Climate Simulation		
33 fields	dim: $26_x \times 1800_y \times 3600_z$	total: 20.7 GB
Nyx: Cosmological Hydrodynamics Simulation		
6 fields	dim: $512_x \times 512_y \times 512_z$	total: 3.1 GB
JetIn: Computational Fluid Dynamics		
1 fields	dim: $1408_x \times 1080_y \times 1100_z$	total: 6.2 GB
SynTruss: Computed Tomography Scan		
1 fields	dim: $1200_x \times 1200_y \times 1200_z$	total: 6.4 GB
HCCI: Computational Fluid Dynamics		
1 fields	dim: $560_x \times 560_y \times 560_z$	total: 669 MB

Table 2: Information of evaluated application datasets.

We also summarize all the compression and collective communication solutions evaluated in Table 3.

cuSZp2: The fastest GPU error-bounded lossy compressor [24]
ghZ: The proposed first-ever GPU homomorphic compressor
Cray MPI: The state-of-the-art MPI library used in 2 of 3 Exascale supercomputers in the world [17, 41]
NCCL: The fastest collective communications library for NVIDIA GPUs [12]
gZCCL: The state-of-the-art compression-accelerated collective communications library for GPUs [19]
ghZCCL: The proposed first-ever GPU-aware homomorphic compression-accelerated collective communications library

Table 3: Information of evaluated compression and collective communication solutions. We highlight our solutions and baselines with blue and red colors, respectively.

6.2 Evaluating GPU Homomorphic Compressor–*ghZ*

In this section, we conduct a comprehensive evaluation of our *gh*Z, focusing on DOC-handling throughput, compression ratio, and compression quality.

6.2.1 DOC-handling throughput of ghZ. In Figure 8, we evaluate the DOC-handling throughput of *gh*Z compared to the DOC workflow using the fastest error-bounded lossy compressor, cuSZp2. Across all application datasets, *gh*Z consistently outperforms cuSZp2, regardless of the relative error bounds. On average, *gh*Z achieves DOC-handling throughputs ranging from 531.91 to 942.44 GB/s, while cuSZp2 achieves compression throughputs between 153.32 and 252.24 GB/s, making it 3.47–3.89× slower than *gh*Z.

Notably, *gh*Z demonstrates its highest throughput with the JetIn application dataset, reaching 1323.40 GB/s with a 1E-1 error bound and 1056.89 GB/s with a 1E-4 error bound. These values far exceed the 302.45 GB/s and 294.52 GB/s achieved by cuSZp2, corresponding to speedups of 4.38× and 3.59×, respectively. This superior performance can be attributed to the high sparsity of the JetIn, which consists of many zero data blocks (i.e., blocks containing only zero values). In *gh*Z, these zero blocks are skipped by directly setting the lossless decoded integer values to zero, avoiding unnecessary retrievals from the compressed data inputs. This adaptive homomorphic compression strategy further boosts the speed of *gh*Z.

Additionally, we observe that both *gh*Z and cuSZp2 experience lower compression throughputs as the error bound decreases. This is because smaller error bounds make the data harder to compress, increasing computational and memory access costs, which in turn reduces performance. However, even with a 1E-4 error bound, *gh*Z maintains a significantly higher throughput than cuSZp2 with a 1E-1 error bound when processing the same application dataset. For example, when processing the NYX application dataset, *gh*Z achieves a throughput of 382.64 GB/s with a 1E-4 error bound, while cuSZp2 reaches only 238.10 GB/s with a far less restrictive 1E-1 error bound. This highlights that our *gh*Z can effectively tackle considerably more challenging compression scenarios while achieving higher compression performance compared to cuSZp2, even when cuSZp2 operates under much simpler compression conditions. This advantage greatly expands the potential use cases of *gh*Z in DOC and similar workflows.

6.2.2 Compression ratio and quality of ghZ. In Table 4, we evaluate the compression ratio and quality of *gh*Z across a range of application datasets. Since *gh*Z operates losslessly, any compression accuracy loss stems solely from the already lossyly compressed data inputs. Consequently, the compression ratio and quality of *gh*Z are identical to those of the traditional DOC workflow with cuSZp2, as confirmed by our comprehensive experiments. Thus, we only report values of *gh*Z in Table 4. This demonstrates that *gh*Z maintains the same compression ratio and quality as the traditional DOC workflow while significantly outperforming it in compression performance, as shown in 6.2.1.

The evaluation results reveal that *gh*Z exhibits varying compression ratios across different application datasets and relative error bounds. For a 1E-1 error bound, the average compression ratio ranges from 35.28 to 127.94 across datasets. For a more restrictive 1E-4 relative error bound, the average compression ratio ranges from 3.81 to 77.44. This trend indicates that smaller error bounds result in lower compression ratios because more data features must be preserved, making the data harder to compress.

Regarding compression quality, *gh*Z achieves excellent results across all application datasets. For example, in the JetIn dataset, the Peak Signal-to-Noise Ratio (PSNR) ranges from 66.58 to 101.61 for error bounds between 1E-1 and 1E-4, indicating high-quality compression. Additionally, smaller error bounds lead to higher compression quality because more data features are preserved. These findings confirm that *gh*Z delivers high compression throughput and impressive compression ratios without sacrificing data accuracy.

Evaluation Takeaway 1: On average, cuSZp2 achieves 153.32–252.24 GB/s, while *gh*Z reaches 531.91–942.44 GB/s, delivering a 3.47–3.89× speedup without compromising compression ratio or accuracy, due to its lossless homomorphic compression design.

6.3 Comparing *ghZCCL* with SOTA Collective Communications Libraries

After demonstrating the high compression performance of *gh*ZCCL's *gh*Z, we now evaluate the performance of *gh*ZCCL-accelerated collective communications on 64 NVIDIA A100

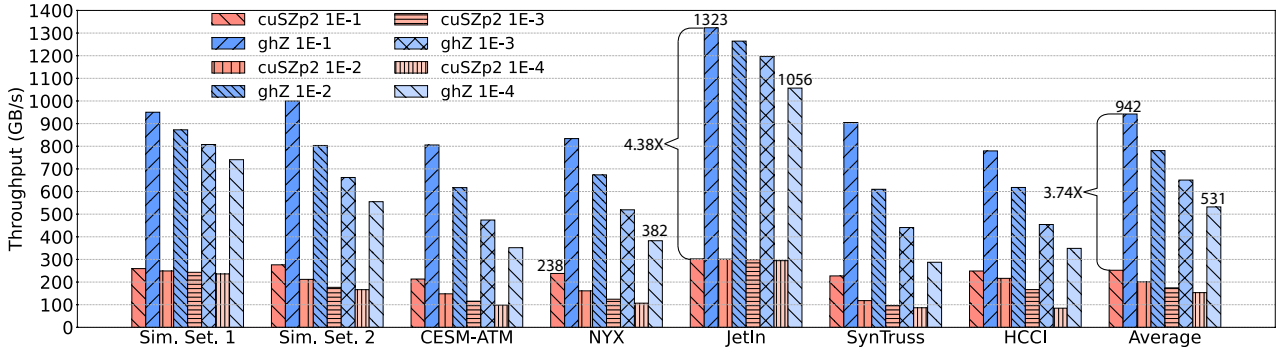


Figure 8: DOC handling throughput of *ghZ* when comparing with cuSZp2.

REL	Sim. Set. 1	Sim. Set. 2	CESM-ATM	NYX	JetIn	SynTruss	HCCI
			Compression Ratio (Original Data Size / Compressed Data Size)				
1E-1	**avg: 115.32** (95.90~127.98)	**avg: 103.94** (92.18~119.28)	**avg: 77.74** (5.12~122.81)	**avg: 93.14** (32.61~127.99)	**avg: 127.94** (127.94~127.94)	**avg: 35.28** (35.28~35.28)	**avg: 90.78** (90.78~90.78)
1E-2	**avg: 100.02** (58.18~127.98)	**avg: 51.84** (21.63~95.45)	**avg: 26.10** (3.46~59.09)	**avg: 64.18** (7.46~127.72)	**avg: 125.49** (125.49~125.49)	**avg: 10.08** (10.08~10.08)	**avg: 47.04** (47.04~47.04)
1E-3	**avg: 86.69** (33.72~127.98)	**avg: 34.26** (8.75~75.80)	**avg: 12.04** (2.55~33.52)	**avg: 35.21** (4.39~124.54)	**avg: 117.16** (117.16~117.16)	**avg: 5.50** (5.50~5.50)	**avg: 22.29** (22.29~22.29)
1E-4	**avg: 77.44** (22.53~127.96)	**avg: 25.83** (5.25~60.87)	**avg: 7.19** (1.98~21.87)	**avg: 19.48** (3.03~88.90)	**avg: 100.31** (100.31~100.31)	**avg: 3.81** (3.81~3.81)	**avg: 7.64** (7.64~7.64)
			Compression Quality (PSNR)				
1E-1	**avg: 52.51** (39.42~91.91)	**avg: 41.09** 37.08~46.46	**avg: 33.00** (24.83~41.64)	**avg: 46.60** (24.82~79.90)	**avg: 66.58** (66.58~66.58)	**avg: 31.96** (31.96~31.96)	**avg: 39.38** (39.38~39.38)
1E-2	**avg: 69.36** (55.71~110.55)	**avg: 54.00** 48.34~61.46	**avg: 48.49** (44.60~54.24)	**avg: 57.77** (44.73~86.75)	**avg: 73.93** (73.93~73.93)	**avg: 46.57** (46.57~46.57)	**avg: 54.62** (54.62~54.62)
1E-3	**avg: 87.16** (73.32~125.46)	**avg: 72.15** (66.12~80.39)	**avg: 67.30** (64.60~73.17)	**avg: 71.65** (63.89~92.98)	**avg: 87.13** (87.13~87.13)	**avg: 65.92** (65.92~65.92)	**avg: 67.98** (67.98~67.98)
1E-4	**avg: 106.27** (92.78~142.67)	**avg: 91.86** (85.83~100.15)	**avg: 86.75** (84.73~92.31)	**avg: 87.52** (84.77~98.25)	**avg: 101.61** (101.61~101.61)	**avg: 85.92** (85.92~85.92)	**avg: 84.45** (84.45~84.45)

Table 4: Compression ratio and quality of *ghZ*: each cell is formatted as "avg: value (min~max)".

GPUs. We compare our approach against three state-of-the-art baselines: (1) the MPI collectives offered by Cray-MPI[17], (2) the collective communications from NCCL [12], and (3) the compression-accelerated collectives from gZCCL[19], as summarized in Table 3.

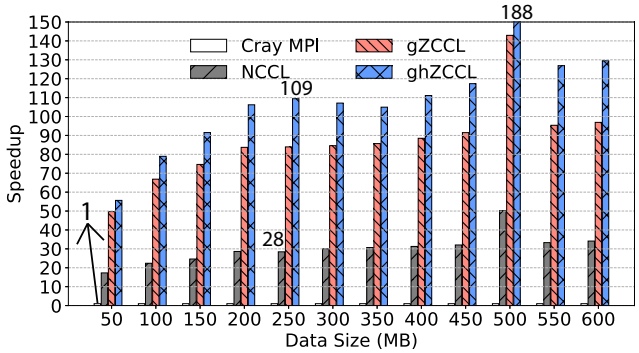


Figure 9: Performance evaluation of *ghZ*CCL-accelerated Reduce against SOTA baselines in different data sizes.

6.3.1 Reduce. In Figure 9, we evaluate *gh*ZCCL against multiple baselines using the Reduce operation, with speedups measured relative to Cray MPI. The results show that *gh*ZCCL consistently outperforms all counterparts across all data sizes. Compared to the second-best solution, gZCCL, *gh*ZCCL achieves a 1.34× speedup at 600 MB. This improvement stems from *gh*ZCCL's ability to significantly reduce DOC-related overheads by co-designing GPU homomorphic compression with collective communications. Furthermore, *gh*ZCCL is up to 3.85× and 188× faster than NCCL and Cray MPI, respectively. The substantial performance gain over NCCL is attributed to *gh*ZCCL's ability to reduce overall communication volume and mitigate network congestion through its ultra-fast homomorphic compression. The improvement is even more pronounced compared to Cray MPI, as its Reduce operation is not fully GPU-centric, leading to significant device-host data transfer and CPU computation overheads.

6.3.2 Allreduce. Figure 10 presents a performance comparison between *gh*ZCCL and other state-of-the-art communication libraries using the Allreduce collective operation. Similar to the observations in Section 6.3.1, *gh*ZCCL consistently outperforms all baselines, achieving up to 8.55× performance improvement over Cray MPI. When compared to NCCL, *gh*ZCCL achieves a 3.21× speedup, primarily due to its significantly improved communication efficiency enabled by lightweight homomorphic compression, which reduces the amount of data transmitted. In contrast, NCCL lacks this capability and must communicate with uncompressed data. Among all the solutions, gZCCL delivers the second-best performance; however, it still underperforms *gh*ZCCL by up to 2.19×. This performance gap arises from gZCCL's substantial decompression and recompression overheads, whereas *gh*ZCCL directly operates on compressed data without decompression. Additionally, we observe that Cray MPI exhibits relatively better performance in Allreduce compared to its performance in Reduce (Section 6.3.1). This is because Cray MPI is specifically optimized to make Allreduce GPU-centric, as it is the most widely used collective operation.

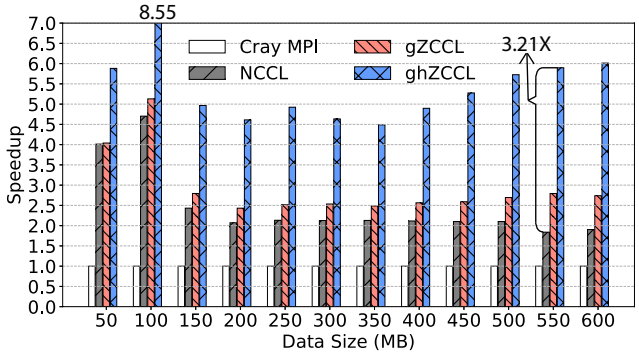
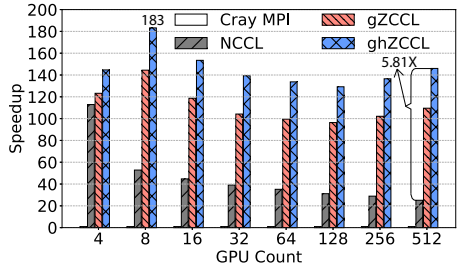


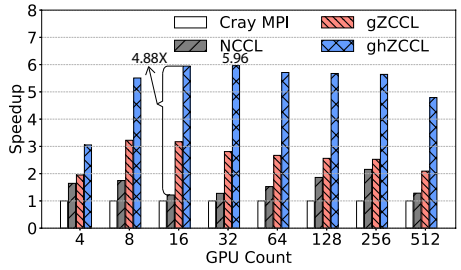
Figure 10: Performance evaluation of *gh*ZCCL-accelerated Allreduce against SOTA baselines in different data sizes.

6.4 Evaluating the Scalability of *gh*ZCCL

To further evaluate the performance of *gh*ZCCL, Figure 11 analyzes its scalability on 512 NVIDIA A100 GPUs. The results demonstrate that *gh*ZCCL maintains strong scalability across varying GPU counts, significantly outperforming baseline solutions. In Subfigure 11a, *gh*ZCCL achieves up to 183× speedup over Cray MPI and 5.81× over NCCL. Additionally, it outperforms gZCCL by up to 1.34×, exhibiting better scalability than the previously best compression-accelerated communication solution. A similar trend is observed in Subfigure 11b, where *gh*ZCCL surpasses gZCCL by 2.29× on 512 GPUs. Moreover, *gh*ZCCL achieves even greater performance improvements over Cray MPI and NCCL, with up to 5.96× and 4.88× speedups, respectively. This superior performance is attributed to *gh*ZCCL's ability to directly operate on and communicate with compressed data, effectively reducing communication overhead and optimizing compression to improve overall runtime efficiency.



(a) Reduce



(b) Allreduce

Figure 11: Scalability evaluation of *gh*ZCCL against SOTA baselines in different GPU counts.

Evaluation Takeaway 2: Evaluated on up to 512 NVIDIA A100 GPUs, *gh*ZCCL significantly outperforms state-of-the-art communication libraries, achieving speedups of up to 2.29×, 5.81×, and 188× compared to gZCCL, NCCL, and Cray MPI, respectively.

6.5 Image Stacking Performance and Accuracy Analysis

In this section, we use the image stacking application to evaluate both the performance and accuracy of the proposed *gh*ZCCL. Image stacking is widely used in various scientific fields, including atmospheric science and geology, to generate high-resolution images by combining multiple individual images. This process involves an Allreduce operation. As highlighted by Gurhem in [15], researchers utilize MPI to merge these individual images into final composite images.

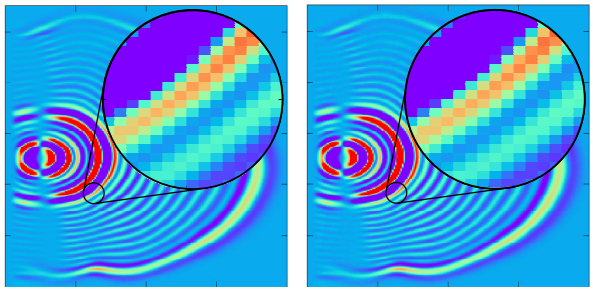
Table 5 shows that *gh*ZCCL significantly outperforms both NCCL and gZCCL, achieving a 2.34× speedup over NCCL, whereas gZCCL only reaches a 1.23× speedup. Since Cray MPI consistently underperforms NCCL, we omit its results to save space. To gain deeper insight into the performance differences, we break down the runtime of gZCCL and *gh*ZCCL. The results reveal that gZCCL's runtime is primarily dominated by the compression and operation time

Methods	Speedups	Compr.+Oper.	Comm.	Others
gZCCL	1.23	82.57%	14.80%	2.63%
***gh*ZCCL**	2.34	69.45%	28.41%	2.14%
NCCL	1	No breakdown because of complexity		

Table 5: Performance analysis of image stacking. The speedups are based on NCCL and the last three columns represent performance breakdown.

(Compr.+Oper.), accounting for 82.57% of the total execution time. This substantial DOC overhead is successfully mitigated in *gh*ZCCL, reducing the proportion to 69.45%—or 36.37% relative to gZCCL's total runtime. These improvements stem from our ultra-fast homomorphic compression-communication co-design, which enables GPUs to directly communicate and compute with compressed data, significantly reducing the overhead associated with traditional DOC workflows.

After demonstrating the high performance of *gh*ZCCL, we further evaluate its numerical accuracy (PSNR and NRMSE) and visual quality. With an absolute error bound of 1E-4, *gh*ZCCL achieves an impressive PSNR of 73.60 and an excellent NRMSE of 2.1E-4. Figure 12 presents a visual comparison of stacking images using *gh*ZCCL and the original uncompressed NCCL method. The comparison reveals no visual differences between the two images, confirming that *gh*ZCCL effectively preserves image quality. This combination of high numerical accuracy and visual quality underscores the effectiveness of *gh*ZCCL in delivering superior performance while maintaining exceptionally high data quality.



(a) NCCL (lossless) **(b) *gh*ZCCL**

Figure 12: Compare the visual quality of *gh*ZCCL with NCCL.

7 CONCLUSION AND FUTURE WORK

In this paper, we introduce *gh*ZCCL, a novel GPU-aware homomorphic compression-accelerated collective communications library that enables direct GPU computation and communication on compressed data. Through evaluations on up to 512 NVIDIA A100 GPUs and 7 application datasets, *gh*ZCCL significantly outperforms state-of-the-art compression and communication libraries: achieving speedups of up to 3.89× over cuSZp2, 2.29× over gZCCL, 5.81× over NCCL, and 188× over Cray MPI. Moving forward, we plan to extend *gh*ZCCL to additional hardware platforms, including AI accelerators (e.g., Groq LPU, SambaNova RDU) and FPGAs, further broadening its impact on both compression and communication across diverse computing architectures.

ACKNOWLEDGMENTS

This research was supported by the U.S. Department of Energy (DOE) Office of Science, Advanced Scientific Computing Research (ASCR), under contracts DE-AC02-06CH11357 and DE-SC0024207. Additional support was provided by the National Science Foundation through grants OAC-2104023 and OAC-2311875. This work also utilized resources of the Argonne Leadership Computing Facility, a DOE Office of Science user facility at Argonne National Laboratory.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

[2] Ahmed M. Abdelmoniem, Ahmed Elzanaty, Mohamed-Slim Alouini, and Marco Canini. 2021. An Efficient Statistical-based Gradient Compression Technique for Distributed Training Systems. arXiv:2101.10761 [cs.LG]

[3] George Almási, Philip Heidelberger, Charles J. Archer, Xavier Martorell, C. Chris Erway, José E. Moreira, B. Steinmacher-Burow, and Yili Zheng. 2005. Optimization of MPI Collective Communication on BlueGene/L Systems. In *Proceedings of the 19th Annual International Conference on Supercomputing*.

[4] Ammar Ahmad Awan, Khaled Hamidouche, Jahanzeb Maqbool Hashmi, and Dhabaleswar K Panda. 2017. S-Caffe: Co-designing MPI runtimes and Caffe for scalable deep learning on modern GPU clusters. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 193–205.

[5] Alan Ayala, Stanimire Tomov, Xi Luo, Hejer Shaeik, Azzam Haidar, George Bosilca, and Jack Dongarra. 2019. Impacts of Multi-GPU MPI collective communications on large FFT computation. In *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*. IEEE, 12–18.

[6] Gaurav Bansal, Ajith Mascarenhas, and Jacqueline H. Chen. 2015. Direct Numerical Simulations of Autoignition in Stratified Dimethyl-Ether (DME)/Air Turbulent Mixtures. *Combustion and Flame* 162 (2015), 688–702. https://doi.org/10.1016/j.combustflame.2014.08.021

[7] M. Bayatpour and M. A. Hashmi. 2018. SALaR: Scalable and Adaptive Designs for Large Message Reduction Collectives. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 1–10. https://doi.org/10.1109/CLUSTER.2018.00009

[8] Jon Calhoun, Franck Cappello, Luke N Olson, Marc Snir, and William D Gropp. 2019. Exploring the feasibility of lossy compression for PDE simulations. *The International Journal of High Performance Computing Applications* 33, 2 (2019), 397–410. https://doi.org/10.1177/1094342018762036

[9] Qiaoling Chen, Qinghao Hu, Guoteng Wang, Yingtong Xiong, Ting Huang, Xun Chen, Yang Gao, Hang Yan, Yonggang Wen, Tianwei Zhang, and Peng Sun. 2024. AMSP: Reducing Communication Overhead of ZeRO for Efficient LLM Training. arXiv:2311.00257 [cs.DC]

[10] Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms, and Kalyan Kumaran. 2018. Characterization of MPI usage on a production supercomputer. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 386–400.

[11] Community Earth System Model (CESM) Atmosphere Model. 2019. http://www.cesm.ucar.edu/models/. Online.

[12] NVIDIA Corp. 2023. NCCL – Optimized primitives for inter-GPU communication. https://github.com/NVIDIA/nccl.

[13] Michael Garland Duane Merrill. 2016. Single-pass Parallel Prefix Scan with Decoupled Look-back. https://research.nvidia.com/sites/default/files/pubs/2016-03_Single-pass-Parallel-Prefix/nvr-2016-002.pdf.

[14] R. W. Grout, A. Gruber, H. Kolla, P.-T. Bremer, J. C. Bennett, A. Gyulassy, and J. H. Chen. 2012. A Direct Numerical Simulation Study of Turbulence and Flame Structure in Transverse Jets Analysed in Jet-Trajectory Based Coordinates. *Journal of Fluid Mechanics* 706 (2012), 351–383. https://doi.org/10.1017/jfm.2012.257

[15] Jérôme Gurhem, Henri Calandra, and Serge G. Petiton. 2021. Parallel and Distributed Task-Based Kirchhoff Seismic Pre-Stack Depth Migration Application. In *2021 20th International Symposium on Parallel and Distributed Computing (ISPDC)*. 65–72. https://doi.org/10.1109/ISPDC52870.2021.9521599

[16] Wenbin He, Hanqi Guo, Tom Peterka, Sheng Di, Franck Cappello, and Han-Wei Shen. 2018. Parallel Partial Reduction for Large-Scale Data Analysis and Visualization. In *2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV)*. 45–55. https://doi.org/10.1109/LDAV.2018.8739165

[17] HPE. [n. d.]. Cray MPI/MPICH. https://cpe.ext.hpe.com/docs/24.03/mpt/mpich/index.html.

[18] Jiajun Huang, Sheng Di, Xiaodong Yu, Yujia Zhai, Jinyang Liu, Yafan Huang, Ken Raffenetti, Hui Zhou, Kai Zhao, Zizhong Chen, Franck Cappello, Yanfei Guo, and Rajeev Thakur. 2024. POSTER: Optimizing Collective Communications with Error-bounded Lossy Compression for GPU Clusters. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '24)*. 454–456. https://doi.org/10.1145/3627535.3638467

[19] Jiajun Huang, Sheng Di, Xiaodong Yu, Yujia Zhai, Jinyang Liu, Yafan Huang, Ken Raffenetti, Hui Zhou, Kai Zhao, Xiaoyi Lu, Zizhong Chen, Franck Cappello, Yanfei Guo, and Rajeev Thakur. 2024. gZCCL: Compression-Accelerated Collective Communication Framework for GPU Clusters. In *Proceedings of the 38th ACM International Conference on Supercomputing* (Kyoto, Japan) *(ICS '24)*. 437–448. https://doi.org/10.1145/3650200.3656636

[20] Jiajun Huang, Sheng Di, Xiaodong Yu, Yujia Zhai, Jinyang Liu, Zizhe Jian, Xin Liang, Kai Zhao, Xiaoyi Lu, Zizhong Chen, Franck Cappello, Yanfei Guo, and Rajeev Thakur. 2024. hZCCL: Accelerating Collective Communication with Co-Designed Homomorphic Compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Atlanta, GA, USA) *(SC '24)*. IEEE Press, Article 104, 15 pages. https://doi.org/10.1109/SC41406.2024.00110

[21] Jiajun Huang, Sheng Di, Xiaodong Yu, Yujia Zhai, Zhaorui Zhang, Jinyang Liu, Xiaoyi Lu, Ken Raffenetti, Hui Zhou, Kai Zhao, Khalid Alharthi, Zizhong Chen, Franck Cappello, Yanfei Guo, and Rajeev Thakur. 2025. ZCCL: Significantly Improving Collective Communication With Error-Bounded Lossy Compression. arXiv:2502.18554 [cs.DC]

[22] Jiajun Huang, Sheng Di, Xiaodong Yu, Yujia Zhai, Zhaorui Zhang, Jinyang Liu, Xiaoyi Lu, Ken Raffenetti, Hui Zhou, Kai Zhao, Zizhong Chen, Franck Cappello, Yanfei Guo, and Rajeev Thakur. 2024. An Optimized Error-controlled MPI Collective Framework Integrated with Lossy Compression. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 752–764. https://doi.org/10.1109/IPDPS57955.2024.00072

[23] Jiajun Huang, Jinyang Liu, Sheng Di, Yujia Zhai, Zizhe Jian, Shixun Wu, Kai Zhao, Zizhong Chen, Yanfei Guo, and Franck Cappello. 2023. Exploring Wavelet Transform Usages for Error-bounded Scientific Data Compression. In *2023 IEEE International Conference on Big Data (BigData)*. 4233–4239. https://doi.org/10.1109/BigData59044.2023.10386386

[24] Yafan Huang, Sheng Di, Guanpeng Li, and Franck Cappello. 2024. cuSZp2: A GPU Lossy Compressor with Extreme Throughput and Optimized Compression Ratio. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*.

[25] Yafan Huang, Sheng Di, Xiaodong Yu, Guanpeng Li, and Franck Cappello. 2023. cuSZp: An Ultra-fast GPU Error-bounded Lossy Compression Framework with Optimized End-to-End Performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[26] Sian Jin, Sheng Di, Xin Liang, Jiannan Tian, Dingwen Tao, and Franck Cappello. 2019. DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19)*. 159–170. https://doi.org/10.1145/3307681.3326608

[27] Sian Jin, Chengming Zhang, Xintong Jiang, Yunhe Feng, Hui Guan, Guanpeng Li, Shuaiwen Leon Song, and Dingwen Tao. 2021. COMET: a novel memory-efficient deep learning training framework by using error-bounded lossy compression. *Proc. VLDB Endow.* 15, 4 (Dec. 2021), 886–899. https://doi.org/10.14778/3503585.3503597

[28] Suha Kayum et al. 2020. GeoDRIVE – A high performance computing flexible platform for seismic applications. *First Break* 38, 2 (2020), 97–100.

[29] Pavol Klacansky, Haichao Miao, Attila Gyulassy, Andrew Townsend, Kyle Champley, Joseph Tringe, Valerio Pascucci, and Peer-Timo Bremer. 2022. Virtual Inspection of Additively Manufactured Parts. In *2022 IEEE 15th Pacific Visualization Symposium (PacificVis)*. 81–90. https://doi.org/10.1109/PacificVis53943.2022.00017

[30] Argonne National Laboratory. 2023. MPICH – A high-performance and widely portable implementation of the MPI-4.0 standard. https://www.mpich.org.

[31] Xin Liang, Sheng Di, Sihuan Li, Dingwen Tao, Bogdan Nicolae, Zizhong Chen, and Franck Cappello. 2019. Significantly improving lossy compression quality based on an optimized hybrid prediction model. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19)*. Association for Computing Machinery, New York, NY, USA, Article 33, 26 pages.

[32] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2018. Deep Gradient Compression: Reducing the communication bandwidth for distributed training. In *The International Conference on Learning Representations*.

[33] Peter Lindstrom. [n. d.]. cuzfp. https://github.com/LLNL/zfp/tree/develop/src/cuda_zfp.

[34] Jinyang Liu, Jiannan Tian, Shixun Wu, Sheng Di, Boyuan Zhang, Robert Underwood, Yafan Huang, Jiajun Huang, Kai Zhao, Guanpeng Li, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2024. cuSZ-i: High-Ratio Scientific Lossy Compression on GPUs with Optimized Multi-Level Interpolation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '24)*. Article 13, 15 pages. https://doi.org/10.1109/SC41406.2024.00019

[35] Gabriel Marcus, Yuantao Ding, Paul Emma, Zhirong Huang, Ji Qiang, Tor Raubenheimer, Marco Venturini, and Lanfa Wang. 2015. High Fidelity Start-to-end Numerical Particle Simulations and Performance Studies for LCLS-II. In *37th International Free Electron Laser Conference*.

TUP007. https://doi.org/10.18429/JACoW-FEL2015-TUP007
[36] NYX simulation. 2019. https://amrex-astro.github.io/Nyx. Online.
[37] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117–124.
[38] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. 2015. Exploration of Lossy Compression for Application-Level Checkpoint/Restart. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 914–922. https://doi.org/10.1109/IPDPS.2015.67
[39] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.
[40] Jiannan Tian, Sheng Di, Kai Zhao, Cody Rivera, Megan Hickman Fulp, Robert Underwood, Sian Jin, Xin Liang, Jon Calhoun, Dingwen Tao, et al. 2020. 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*. 3–15.
[41] TOP500.org. 2024. TOP500 LIST - NOVEMBER 2024. https://top500.org/lists/top500/2024/11/.
[42] Didem Unat, Ilyas Turimbetov, Mohammed Kefah Taha Issa, Doğan Sağbili, Flavio Vella, Daniele De Sensi, and Ismayil Ismayilov. 2024. The Landscape of GPU-Centric Communication. arXiv:2409.09874 [cs.DC]
[43] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: ternary gradients to reduce communication in distributed deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*.
[44] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. 2019. Full-state quantum circuit simulation by using data compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19)*. Article 80, 24 pages. https://doi.org/10.1145/3295500.3356155
[45] Xiaodong Yu, Sheng Di, Kai Zhao, Jiannan Tian, Dingwen Tao, Xin Liang, and Franck Cappello. 2022. Ultrafast Error-Bounded Lossy Compression for Scientific Datasets. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*.
[46] Boyuan Zhang, Jiannan Tian, Sheng Di, Xiaodong Yu, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. 2023. FZ-GPU: A Fast and High-Ratio Lossy Compressor for Scientific Computing Applications on GPUs. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '23)*. 14 pages. https://doi.org/10.1145/3588195.3592994
[47] Kai Zhao, Sheng Di, Maxim Dmitriev, Thierry-Laurent D. Tonellot, Zizhong Chen, and Franck Cappello. 2021. Optimizing Error-Bounded Lossy Compression for Scientific Data by Dynamic Spline Interpolation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 1643–1654. https://doi.org/10.1109/ICDE51399.2021.00145
[48] Q. Zhou, C. Chu, N. S. Kumar, P. Kousha, S. M. Ghazimirsaeed, H. Subramoni, and D. K. Panda. 2021. Designing High-Performance MPI Libraries with On-the-fly Compression for Modern GPU Clusters. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.
[49] Qinghua Zhou, Pouya Kousha, Quentin Anthony, Kawthar Shafie Khorassani, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda. 2022. Accelerating MPI All-to-All Communication With Online Compression On Modern GPU Clusters. In *High Performance Computing: 37th International Conference, ISC High Performance 2022, Proceedings*.