Designing Efficient Pipelined Communication Schemes using Compression in MPI Libraries

Bharath Ramesh, Qinghua Zhou, Aamir Shafi, Mustafa Abduljabbar, Hari Subramoni, Dhabaleswar K. Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, USA
{ramesh.113, zhou.2595, shafi.16, abduljabbar.1, subramoni.1, panda.2}@osu.edu

***Abstract*—The emergence of trillion-parameter models in AI, and the deployment of dense Graphics Processing Unit (GPU) systems with high-bandwidth inter-GPU and network interconnects underscores the need to design efficient architecture-aware large message communication operations. GPU-based on-the-fly compression communication designs help reduce the amount of data transferred across processes, thereby improving large message communication performance. In this paper, we first analyze bottlenecks in state-of-the-art on-the-fly compression-based MPI implementations for blocking as well as non-blocking point-to-point communication operations. We then propose efficient point-to-point designs that improve upon state-of-the-art implementations through fine-grained overlap of copy, compression and communication operations. We demonstrate the efficacy of our proposed designs by comparing against state-of-the-art communication runtimes using micro-benchmarks and candidate communication patterns. Our proposed designs deliver 28.7% improvements in latency, 49.7% in bandwidth, and 36% in bi-directional bandwidth using micro-benchmarks, and up to 16.5% improvements for 3D stencil-based communication patterns over state-of-the-art designs.**

***Index Terms*—HPC, Infiniband, MPI, RDMA, Compression, GPU**

I. Introduction

The advent of Graphics processing units (GPUs) has enabled applications to perform a wide variety of compute intensive tasks at a much faster rate than CPUs. Owing to their massive compute capabilities, High Performance Computing (HPC) clusters such as the #4 supercomputer, named Summit, on the Top500 list [1] have employed multiple GPUs per node spanning thousands of nodes. These clusters employ high-bandwidth inter-node interconnects such as Infiniband [2] and inter-GPU interconnects such as NVIDIA NVLink [3] to facilitate large volumes of distributed communication between GPUs in the system along with low latency. The Message Passing Interface (MPI) is the de-facto standard for distributed communication on HPC clusters, providing APIs for point-to-point as well as collective communication operations. The trend towards building supercomputers with GPUs and high performance interconnects is only expected to expand with the move towards exascale. The onus of utilizing different interconnects and compute elements in super-computing systems while achieving the lowest possible communication latency between processes falls on MPI libraries.

Researchers in the past [4], [5], [6] have optimized GPU to GPU data transfers using a mix of efficient software-level protocols and hardware features such as GPUdirect RDMA, NVLink load/stores, etc. Even state-of-the-art schemes that don't use compression often saturate network links. This motivated the need to reduce the amount of data communicated across these links using compression. Observing the high overhead of compression libraries at runtime for communication operations, authors in [7] proposed an on-the-fly compression framework to reduce volume of communication across links and improve large message data transfer latency in MPI. The limitation of their approach was primarily the blocking nature of compression and data transfer operations. This problem is exacerbated when the time for compression increases, since the network largely remains under-utilized.

This paper revisits the challenge of designing on-the-fly compression aware MPI libraries and proposes new designs to more effectively overlap compute as well as communication operations involved in transferring messages between GPUs using compression in MPI. The overall idea is to start the transfer of inter-process messages as soon as possible and in an architecture aware fashion. To do this, we identify bottlenecks at different layers in state-of-the-art on-the-fly compression implementations and enhance them by overlapping compression, copy and communication operations. We take ZFP [8] as the candidate compression library, and NVIDIA V100s as candidate GPUs, but our designs are compression library/GPU architecture agnostic.

II. Contributions

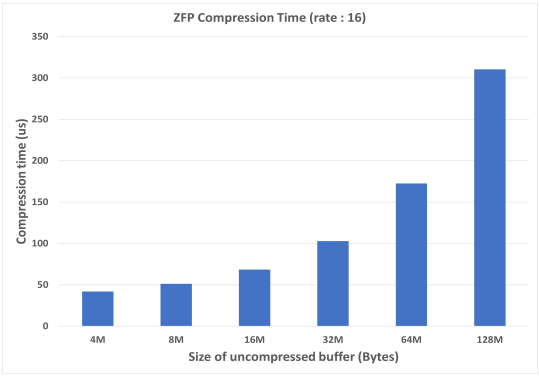
This paper makes the following key contributions:

1) Analyze and highlight bottlenecks in state-of-the-art MPI implementations that use compression for large message GPU-based point-to-point messages. C
2) Identify potential areas of overlap for MPI point-to-point transfers using compression.
3) Propose point-to-point designs for GPU buffers in MPI that can effectively pipeline compression and message transfer.

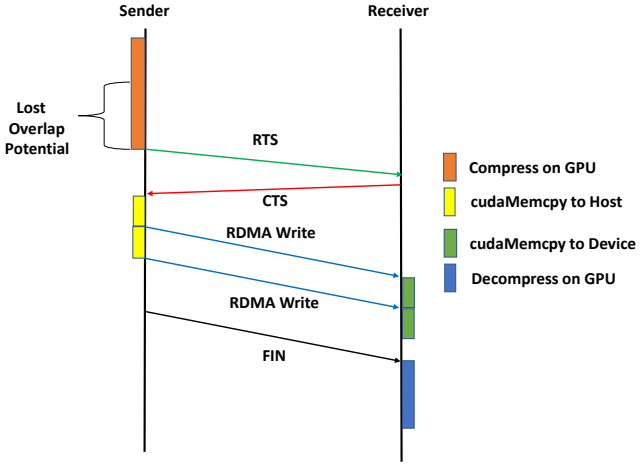
*This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, #2112606, and XRAC grant #NCR-130002

4) Compare proposed designs with existing baseline compression-based implementations using micro-benchmarks and 3-D stencil communication patterns. Our proposed designs outperform state-of-the-art with **28.7%** improvements in latency, **49.7%** in bandwidth, and 36% in bi-directional bandwidth at the micro-benchmark level, and up to **16.5%** improvements for 3D stencil-based communication patterns.

III. Motivation



(a) Time Taken for compression operations using ZFP, with a fixed-rate of 16 (compression ratio = two). The graph shows a steep increase in compression time for larger messages.



(b) State-of-the-art compression design timeline, showing interactions between the sender and receiver processes. As shown in the figure, the sender process waits to send an RTS message until the compression operation completes, which leads to lost overlap potential and under-utilization of the network.

Figure 1: Figures demonstrating compression bottlenecks in state-of-the-art designs.

We motivate the need for improved designs by highlighting compression bottlenecks in existing state-of-the-art implementations. Figure 1(b) shows the general flow of transferring a message between two processes (a sender and a receiver) using existing RDMA-based PUT protocols with host-staging and compression. The sender first uses the GPU to compress the data to be sent. The sender then sends a ready to send (RTS) packet (containing metadata) over the network indicating readiness for sending data to the receiver. The receiver replies back with a clear to send (CTS) to complete the handshake and enable the sender to start RDMA operations. The sender side then initiates asynchronous memory copies to the host, so that RDMA operations can be performed between CPUs. Once the memory copy is complete, the host process issues an RDMA write operation to remote memory resident on the receiver's RAM. The problem with this approach is that the RDMA operation is initiated only after compression is complete, which results in sub-optimal utilization of the underlying network since the time for compression is relatively high for larger messages (shown in figure 1(a)). This brings us to the challenge of designing a transfer scheme for large messages that minimizes the impact of compression, increases potential for overlap, and improves the latency of point-to-point communication operations.

IV. Design and Implementation

A. Pipelined Inter-node point-to-point design (Proposed-pipelined-ZFP)

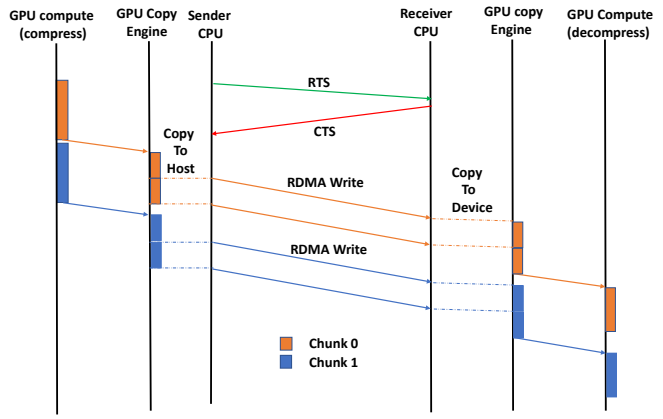


Figure 2: Data flow for the proposed pipelined point-to-point design. Orange/blue indicate different chunks of the buffer to be exchanged. The two chunks operate independently and the compute engines on the GPU could potentially overlap compression of both chunks.

1) Overview: MPI libraries traditionally use two methods for transferring data between NVIDIA GPUs depending on the system characteristics/message size - either through GPUDirect RDMA [9] or by staging through the host [4]. Figure 2 shows the high-level flow of the proposed pipelined design, in cases where the preferred transfer method is staging to the host. Note that cases involving GPUDirect RDMA would follow the same flow, and would only avoid the data staging operations shown in the figure. The overall idea of the design is to divide the buffer into a specified number of chunks, and perform compression/decompression and RDMA operations by overlapping compute, copy and RDMA operations. The Figure shows two colors — orange and blue, each denoting a specific chunk of the message. At first, the sender process initiates asynchronous compression operations for both chunks and immediately sends an RTS message to the destination process. The receiver replies back with a CTS containing information

required for remote RDMA operations to be performed by the sender. On completion of any compression operation, the sender initiates a memory copy to host, followed by an RDMA write. The receiver side mirrors this step by copying from host to device and then decompressing buffers as soon as all the data for a chunk is received. The memory copies to host/device are chunked to pipeline memory copies and RDMA operations. Chunking compression operations facilitates overlapping compression and memory copy operations (and hence RDMA operations). Each point-to-point operation is associated with a request object, used to track details on a per send/recv basis. Specific details are described in subsequent sections.

2) Maintaining buffer pools: We maintain two kinds of memory pools — one on the GPU and another on the host. The GPU memory pool is used as a staging area for compression and decompression operations. The host memory pool is used as a staging area for RDMA transfers between CPUs, and is useful in cases where using GPUDirect RDMA results in bottlenecks [4]. The size of buffers in the pool is equal to the maximum message size supported for data transfer. If transfers that exceed the buffer size are performed, they are chunked so that the maximum amount of data transfer handled at any time is the size of the buffers in the pool. All pools are initialized in MPI_Init and are hence a one-time cost. All buffers in the pool are registered using ibv_reg_mr, so that the HCA can perform data transfers using them. A registration cache is maintained to amortize the cost of registration operations in the critical path. All temporary buffers used for compression/data staging are obtained from the two memory pools described here.

3) Finding ideal chunk sizes: The number of chunks used can directly impact the latency of communication operations when pipelining. We empirically determine this number based on timing information of prior runs of the same buffer/message size combination. Based on our observations, splitting buffers into two/four chunks yield the best results in most cases, though this could potentially change with other architectures/systems.

4) InfiniBand transport considerations: We only consider using RC transports, due to guarantees provided with respect to ordering of messages and ease of implementation. All RDMA writes described in our designs are actually RDMA write with immediate operations (with request object pointers as immediate data), which generate a notification of completion on the receiver side and hence reduce the need for sending many finish (FIN) messages. However, notification of completion does not guarantee that the data is resident in the receiver's memory. To handle this, we wait for another message to arrive on the same Queue Pair (QP) before processing any RDMA write completion event. Due to this, our design only ends up using one FIN message per QP (for the last chunk sent in the point-to-point operation).

V. Experimental Evaluation

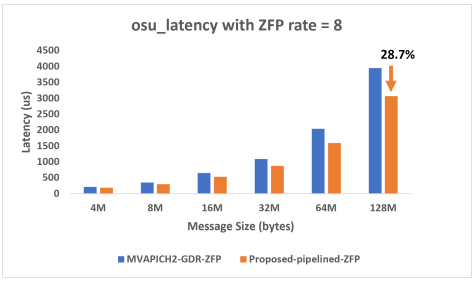
In this section, we describe experiments we conducted to evaluate our proposed pipelined designs (Proposed-pipelined-ZFP). We use ZFP as the underlying compression library. We only compare our proposed design against MVAPICH2-GDR 2.3.7 with ZFP support (referred to as MVAPICH2-GDR-ZFP in the graphs), which is the only publicly available production MPI library with support for compression. Since compression based schemes are known to outperform implementations that do not use compression [7], we omit results comparing against other popular production MPI libraries. All numbers reported are an average of five runs, to minimize effects of system noise showing up in the final results.

A. Experimental setup

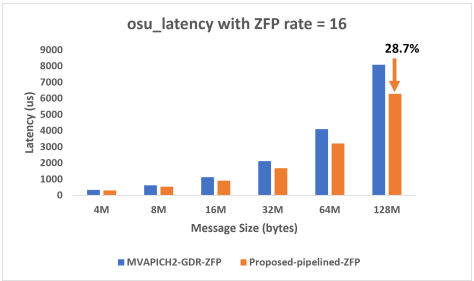
The cluster we use for our experiments comprises of dual socket IBM POWER9 processors, with two NVIDIA V100 GPUs per socket connected via NVLinks. We use MVAPICH2-GDR 2.3.7 with support for compression as the baseline for our comparisons, using parameters as proposed by authors in [7]. The cluster runs RHEL 7.9 as the operating system, with MLNX_OFED_LINUX_4.7 on a v4.14 ppc64le kernel.

B. Micro-Benchmark Evaluation

In this section, we explain our observations on running the OSU micro-benchmarks suite [10] v5.9. All experiments are run with the default number of iterations set by the benchmark. We demonstrate the efficacy of our proposed designs on latency, uni-directional bandwidth as well as bi-directional bandwidth benchmarks using different window sizes. All micro-benchmark evaluations below are for inter-node operations.



(a) ZFP Rate 8



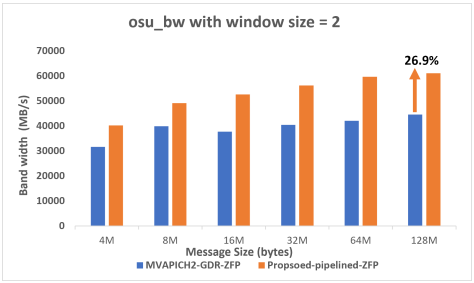
(b) ZFP Rate 16

Figure 3: Comparison of proposed pipelined implementation against MVAPICH2-GDR-ZFP with different compression rates using osu_latency for message sizes ranging from 4M to 128M bytes. Lower is better.

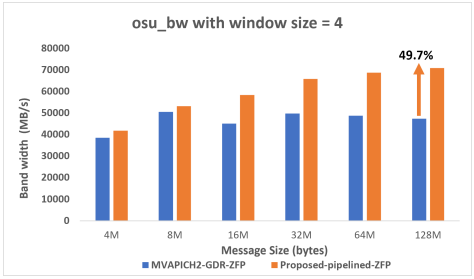
*1) **osu_latency**:* Figure 3 shows results on running comparisons using osu_latency. For a compression rate of 8

(indicating a compression ratio of 128/32 = 4), we observe up to 28.7% improvements over MVAPICH2-GDR-ZFP for a message size of 128MB. Our proposed designs only show nominal improvements for relatively smaller messages because the cost of running multiple chunked compression operations far exceeds the cost of compressing the entire buffer. The improvements are more pronounced at larger message sizes as larger messages involve high compression costs which result in better bandwidth utilization when using chunked compression schemes. The trend remains largely the same with a rate of 16, where we reduce the latency by 28.7% over MVAPICH2-GDR-ZFP.

*2) **osu_bw**:* We run bandwidth benchmarks using two different window sizes. The window size represents the total number of concurrent data transfers that are initiated by a process. Figure 4 shows the comparison between our proposed designs and MVAPICH2-GDR-ZFP. We observe up to 26.9% improvements for 128M messages with a window size of two, and up to 49.7% improvements for a window size of four. This demonstrates that pipelining compression significantly improves the utilization of the network, as network operations are initiated a lot sooner than sequential compress and send schemes.



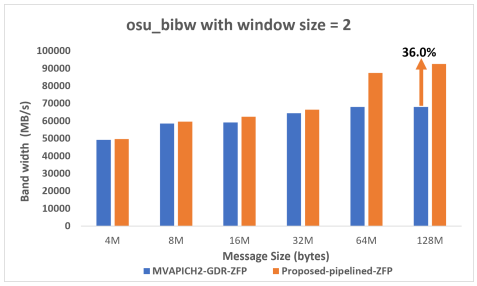
(a) Window size = 2



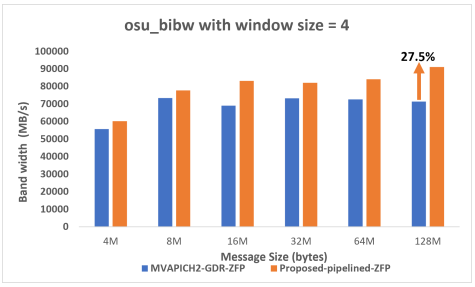
(b) Window size = 4

Figure 4: Comparison of proposed pipelined implementation against MVAPICH2-GDR-ZFP with a compression rate of 8 using osu_bw for message sizes ranging from 4M to 128M bytes. Higher is better.

*3) **osu_bibw**:* Our evaluation of bi-directional bandwidth is done using the same window sizes used in the osu_bw evaluation. The bi-bandwidth benchmark emulates communication patterns where two processes send and receive multiple messages at the same time. As shown in Figure 5, our proposed designs improve bi-directional bandwidth by up to 36% for a window size of two, and up to 27.5% for a window size of four.



(a) Window size = 2



(b) Window size = 4

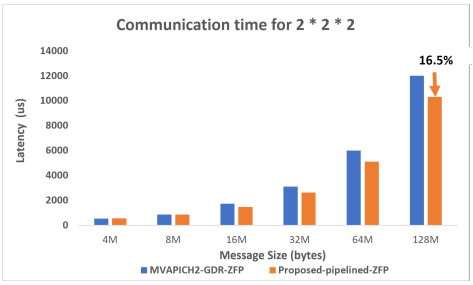
Figure 5: Comparison of proposed pipelined implementation against MVAPICH2-GDR-ZFP with a compression rate of 8 using osu_bibw for message sizes ranging from 4M to 128M bytes. Higher is better.

C. 3D stencil communication evaluation

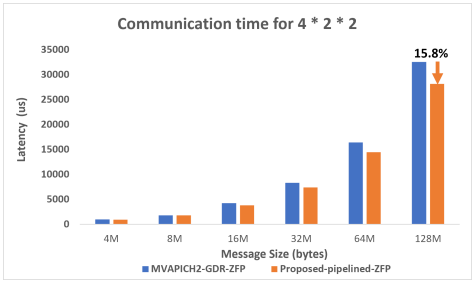
3D stencil is a near-neighbor communication pattern that involves exchange of data in three directions - x, y and z. The pattern often involves concurrent non-blocking point-to-point operations, and is used widely in a variety of applications. In this evaluation, we run 3D stencil experiments for data buffers resident on the GPU. Figure 6 shows the latency of running a 3D stencil communication pattern for two process grid sizes. The 2 * 2 * 2 grid is run using 8 GPUs, with one GPU per node and the 4 * 2 * 2 grid is run on 16 GPUs, with two GPUs per node. The trends for 3D stencil follow a similar pattern as seen in the micro-benchmark evaluations. As shown in Figure 6(a) and 6(b), we observe up to 16.5% over MVAPICH2-GDR-ZFP for a message size of 128MB.

VI. Related Work

Compression technologies have been utilized to accelerate the MPI libraries and HPC applications. In [11], Ke et al. integrated CPU-based compression algorithms in an MPI library to compress the messages. In the CoMPI library [12], multiple compression algorithms were integrated into an MPI library to accelerate various operations and achieved benefits for applications running on CPUs. In [13], Filgueira et al. developed a Dynamic-CoMPI library that can adaptively select the best compression algorithm under a specific communication environment. The performance of compression algorithms has been pushed forward with advanced hardware accelerators such as GPUs. Some GPU-based compression algorithms GFC [14], MPC [15] have outperformed the CPU-based algorithms. Recently, NVIDIA released the lossless GPU-based compression



(a) Grid size = 2 * 2 * 2



(b) Grid size = 4 * 2 * 2

Figure 6: 3D stencil communication latency for on 8 nodes, for 8 GPUs and 16 GPUs respectively. The grid sizes map to the layout of how processes are arranged in 3D.

library nvCOMP [16] for HPC applications. To achieve a higher compression ratio, lossy compression algorithms such as SZ [17] and ZFP [8] have been developed and applied to various scientific applications. A recent dual-quantization scheme of SZ was proposed [18] based on advanced NVIDIA GPU architectures. The study [19] showed high compression throughput and good error-bounded accuracy at scale. To solve the bottleneck of existing network bandwidth and further improve the performance of transferring large GPU data, an on-the-fly compression framework [7] was proposed to integrate the GPU-based compression algorithms MPC [15] and ZFP [8] into MPI library. This on-the-fly compression is an initiative work along this direction. However, there are limitations in their point-to-point based compression as analyzed in this paper and we proposed designs to overcome those limitations.

VII. Conclusion and Future Work

The rapid adoption of GPU systems in large scale supercomputers motivate the need to optimize communication operations for distributed applications. Interconnect links are often saturated due to the disparity between the volume of data exchanged and available bandwidth. Compressing large messages is known to be an effective approach in reducing the volume of data exchanged. In this paper, we analyzed bottlenecks of state-of-the-art compression based point-to-point schemes in MPI libraries and proposed designs to more efficiently pipeline compression, communication and copy operations. Our proposed designs show up to 28.7% improvements in latency, 49.7% in bandwidth and 36% in bi-directional bandwidth over state-of-the-art designs. As future work, we plan to evaluate our designs at larger scales and more complex application communication patterns.

VIII. Acknowledgements

We would like to thank LLNL for providing resources.

References

[1] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, "TOP 500 Supercomputer Sites," http://www.top500.org.
[2] InfiniBand Trade Association, http://www.infinibandta.com.
[3] "NVIDIA NVLink and NVSwitch," https://www.nvidia.com/en-us/data-center/nvlink/.
[4] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, "Efficient Inter-node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters With NVIDIA GPUs," in *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013, pp. 80–89.
[5] K. Hamidouche, A. Venkatesh, A. A. Awan, H. Subramoni, C. H. Chu, and D. K. Panda, "Exploiting GPUDirect RDMA in Designing High Performance OpenSHMEM for NVIDIA GPU Clusters," in *2015 IEEE International Conference on Cluster Computing*, Sept 2015, pp. 78–87.
[6] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. Panda, *NV-Group: Link-Efficient Reduction for Distributed Deep Learning on Modern Dense GPU Systems*. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3392717.3392771
[7] Q. Zhou, C. Chu, N. S. Kumar, P. Kousha, S. M. Ghazimirsaeed, H. Subramoni, and D. K. Panda, "Designing high-performance mpi libraries with on-the-fly compression for modern gpu clusters*," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 444–453.
[8] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, 08 2014.
[9] NVIDIA, "NVIDIA GPUDirect," https://developer.nvidia.com/gpudirect, 2011, Accessed: September 24, 2023.
[10] OSU Micro-benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/.
[11] Jian Ke, M. Burtscher, and E. Speight, "Runtime Compression of MPI Messanes to Improve the Performance and Scalability of Parallel Applications," in *SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004, pp. 59–59.
[12] R. Filgueira, D. Singh, A. Calderón, and J. Carretero, "CoMPI: Enhancing MPI Based Applications Performance and Scalability Using Run-Time Compression," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, Sep. 2009, pp. 207–218.
[13] R. Filgueira, J. Carretero, D. Singh, A. Calderón, and A. Núñez, "Dynamic-CoMPI: Dynamic optimization techniques for MPI parallel applications," *The Journal of Supercomputing*, vol. 59, pp. 361–391, 04 2012.
[14] M. A. O'Neil and M. Burtscher, "Floating-Point Data Compression at 75 Gb/s on a GPU," in *Fourth Workshop on General Purpose Processing on Graphics Processing Units*, March 2011.
[15] A. Yang, H. Mukka, F. Hesaaraki, and M. Burtscher, "MPC: A Massively Parallel Compression Algorithm for Scientific Data," in *IEEE Cluster Conference*, September 2015.
[16] NVIDIA, "nvCOMP," https://github.com/NVIDIA/nvcomp, 2020, Accessed: September 24, 2023.
[17] S. Di and F. Cappello, "Fast Error-bounded Lossy HPC Data Compression with SZ," in *International Parallel and Distributed Processing Symposium(IPDPS)*, 2016.
[18] J. Tian, S. Di, K. Zhao, C. Rivera, M. H. Fulp, R. Underwood, S. Jin, X. Liang, J. Calhoun, D. Tao, and F. Cappello, "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*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3–15. [Online]. Available: https://doi.org/10.1145/3410463.3414624
[19] S. Jin, P. Grosset, C. M. Biwer, J. Pulido, J. Tian, D. Tao, and J. P. Ahrens, "Understanding GPU-Based Lossy Compression for Extreme-Scale Cosmological Simulations," *ArXiv*, vol. abs/2004.00224, 2020.