Accelerating Large Language Model Training with Hybrid GPU-based Compression

Lang Xu
The Ohio State University
Columbus, Ohio
xu.3304@osu.edu

Quentin Anthony
The Ohio State University
Columbus, Ohio
anthony.301@osu.edu

Qinghua Zhou
The Ohio State University
Columbus, Ohio
zhou.2595@osu.edu

Nawras Alnaasan
The Ohio State University
Columbus, Ohio
alnaasan.1@osu.edu

Radha Gulhane
The Ohio State University
Columbus, Ohio
gulhane.2@osu.edu

Aamir Shafi
The Ohio State University
Columbus, Ohio
shafi.16@osu.edu

Hari Subramoni
The Ohio State University
Columbus, Ohio
subramoni.1@osu.edu

Dhabaleswar K. (DK) Panda
The Ohio State University
Columbus, Ohio
panda.2@osu.edu

Abstract—**Data Parallelism (DP), Tensor Parallelism (TP), and Pipeline Parallelism (PP) are the three strategies widely adopted to enable fast and efficient Large Language Model (LLM) training. However, these approaches rely on data-intensive communication routines to collect, aggregate, and re-distribute gradients, activations, and other important model information, which pose significant overhead. Co-designed with GPU-based compression libraries, MPI libraries have been proven to reduce message size significantly, and leverage interconnect bandwidth, thus increasing training efficiency while maintaining acceptable accuracy.**

In this work, we investigate the efficacy of compression-assisted MPI collectives under the context of distributed LLM training using 3D parallelism and ZeRO optimizations. We scaled up to 192 V100 GPUs on the Lassen supercomputer. First, we enabled a naïve compression scheme across all collectives and observed a 22.5% increase in TFLOPS per GPU and a 23.6% increase in samples per second for GPT-NeoX-20B training. Nonetheless, such a strategy ignores the sparsity discrepancy among messages communicated in each parallelism degree, thus introducing more errors and causing degradation in training loss. Therefore, we incorporated hybrid compression settings toward each parallel dimension and adjusted the compression intensity accordingly. Given their low-rank structure [1], we apply aggressive compression on gradients when performing DP All-reduce. We adopt milder compression to preserve precision while communicating activations, optimizer states, and model parameters in TP and PP. Using the adjusted hybrid compression scheme, we demonstrate a 17.3% increase in TFLOPS per GPU and a 12.7% increase in samples per second while reaching baseline loss convergence. *

Index Terms—**All-reduce, Large Language Model, Compression, GPU-Aware MPI, Deep Learning, Distributed Training**

I. INTRODUCTION

In recent years, an abundance of Large Language Models (LLM) emerged with impressive abilities in downstream Natural Language Processing (NLP) tasks involving machine translation, dialogue systems, text generation, and so on. Some spotlights include LLaMA [2], GPT-4 [3] and GPT-NeoX-20B [4]. However, to ensure exceptional performance, these models often scale up to billions of parameters, thus requiring increased data size and computation by multiple orders of magnitude. Since the introduction of scaling laws of LLMs [5], model size has been growing from 100 million (BERT [6]) to 500 billion (Megatron-Turing NLG [7]). As a result, one Graphic Processing Unit (GPU) cannot fit a model and its input data anymore, making it necessary to scale out to more workers.

High-Performance Computing (HPC) systems are designed and engineered to support sizeable scientific research and deep learning workloads. These HPC systems typically consist of thousands of nodes equipped with two to four advanced GPUs that maximize floating point operations per second (FLOPS), making them ideal for large-scale data-intensive distributed pre-training of LLMs. Inter- and Intra-node communication play a significant role in accelerating parallel applications. The Message Passing Interface (MPI) supports a variety of highly-optimized communication routines and has been a favored parallel programming model deployed on HPC systems [8]. With the advances in GPUDirect technology [9], GPU-aware MPI libraries [10]–[12] vastly accelerate GPU data transfer and leverage interconnect bandwidth. MPI also serves as a popular communication back-end for distributed machine learning jobs [13]–[15].

Training massive language models requires meticulous arrangement of memory resources and parallelism strategies. Two well-known solutions to this problem are 3D parallelism [7] and the Zero Redundancy Optimizer (ZeRO) [16]. Here, 3D parallelism refers to Data Parallelism (DP) [17], Pipeline Parallelism (PP) [18], [19] and Tensor Parallelism (TP) [20], which are often implemented to support extremely parallel execution of pre-training missions of LLMs across hundreds of GPUs. Also, tensor parallelism and pipeline parallelism are generally categorized under model parallelism. Data parallelism generally aims to partition one input data batch into mini-batches and distribute them to each GPU. This method enables parallel input data processing but requires a data-intensive All-reduce to aggregate the gradients at the end of

backward propagation before updating each model replica. Pipeline parallelism further divides the model layers among workers and performs corresponding forward and backward computations on the micro-batches through a pipeline manner. This process involves mainly point-to-point activation passing from one device to another. Tensor parallelism focuses on splitting tensor computation among workers but also requires All-gathering and All-reducing tensors from all the devices. ZeRO reduces GPU memory footprint by partitioning model states among GPUs, further eliminates replications, and uses gather-based routine to reconstruct model states [16].

A. Motivation

Applying 3D parallelism with ZeRO for DL model training imposes data-intensive collective communications within and across nodes. Given the limited interconnect bandwidth between GPU nodes, such large data transfer leads to drastic overhead. Profiling conducted by previous works [21] has also addressed that different parallelism strategy requires huge portions of communication and heterogeneous collective operations (see Figure 1). Consequently, optimizing point-to-point and collective communication schemes became critical to mitigate such bottlenecks. Previous research has demonstrated that when co-designed with GPU-based compression libraries like ZFP [22] and MPC [23], GPU-aware MPI collectives were able to leverage interconnect bandwidth and stage outstanding throughput benefits [24]–[26].

Conducting GPU-based compression techniques on data buffers drastically decreases the message size being communicated. However, representing high-precision data using lower precision always results in accuracy degradation, which is often observed within Deep Learning workloads. Prior researchers mainly experiment with small deep learning models to pick the most suitable compression rate for lossy ZFP library [26]. Yet, these models contain much fewer parameter numbers than LLMs, emitting smaller message sizes during transfer. **One key motivator of this work is to use compression-assisted MPI collectives to accelerate large language model training.**

B. Challenges

This section addresses the following challenges:

C1) The first challenge was determining the different model information communicated in parallelism stages. Pipeline parallelism, tensor parallelism, and data parallelism usually pose collective communications on various training parameters. Depending on the specific implementation, tensor parallelism [20] usually calls All-reduce and All-gather on activations during forward passes and gradients during backward passes. Pipeline parallelism [18], [19] typically features point-to-point operations to pass activations and gradients from one stage to another. Finally, data parallelism requires All-reduce on the gradients of each data parallel rank and re-distributing the aggregated global gradients back to each model replica to perform subsequent updates [15],

[27]. To overcome this challenge, we present a thorough illustration of the communication routines involved in 3D parallelism as well as in ZeRO stage 1 (Figure 2 4). We further explain this demonstration in Section III

C2) The second challenge is that compressing messages communicated during training depends on the parallelism stages and requires prudent design. We observed that naïvely applying compression scheme to all parallelism stages introduced outstanding training speedup, but at the same time led to degradation in training quality (Section IV-C). Current studies have shown that model parallelism has inherently different characteristics than data parallelism. In data parallelism, gradients communicated among different ranks are observed to be low-rank or sparse. Yet, in model parallelism where communicating activations is the bottleneck, activations are analyzed as dense [1]. Therefore, applying the same compression intensity to both gradients and activations may lead to a loss of accuracy. Given these differences, compression intensity may vary to achieve a balanced solution that maximizes both throughput benefits and model training performance.

C3) The third challenge we tackle in this paper originates from the fact that some model information, specifically gradients, is processed in both model-parallel and data-parallel stages. For example, both data parallelism and tensor parallelism feature the communication of gradients in their executions. When designing compression schemes, we should only apply aggressive compression to these gradients once, preventing over-extracting and hurting meaningful information, thus destroying accuracy. However, we cannot use lossless or high-precision methods towards these gradients since most negligible values will not be adequately extracted. Consequently, we need to avoid over-compression of gradients and maintain compression intensity. This challenge is being solved by applying different compression intensities towards different parallelism communication paths.

C. Solutions

In this paper, we propose *MZHybrid* and *ZHybrid*, two hybrid compression schemes that utilize GPU-based compression on LLM training data, thus expediting the training process while maintaining acceptable model performance. We adopted MPI collectives co-designed with lossless MPC and lossy ZFP libraries to reduce the amount of data movement and leverage inter-node bandwidth. We analyzed different communication scenarios under 3D parallelism and ZeRO stage 1 optimization and designed an appropriate compression approach that considers the sparsity distinction between messages in the training process.

D. Contributions

This paper contributes in the following manners:

1) We experimented naïve compression schemes for both MPC and ZFP on modern large-scale HPC systems. We

(a) Proportion of computation to communication for distributed DL training

(b) Breakdown of individual communication operations for distributed DL training
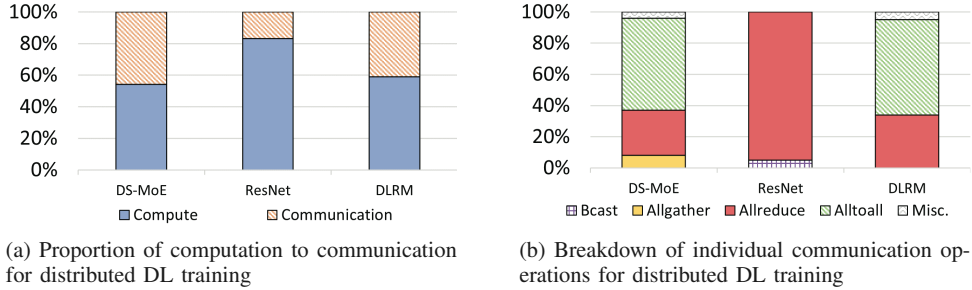
Fig. 1: Communication profiling conducted in MCR-DL [21] for data-parallel and hybrid-parallel DL models.

reported up to a 23.6% increase in training samples per second and up to a 22.5% increase in TFLOPS per GPU over non-compressed collective communications. We analyzed such schemes' benefits and shortcomings, leading to further design choices (Section IV-C).

2) We proposed and designed *MZHybrid*, a hybrid compression scheme that utilizes lossless MPC for model-parallel communications and lossy ZFP rate for data-parallel communications. (Section III-A) We also proposed and designed *ZHybrid*, a hybrid compression scheme that utilizes different ZFP compression rates for communicating model-parallel and data-parallel messages. (Section III-B).

3) We evaluated *MZHybrid* performance on modern large-scale HPC systems. We reported up to 4.4% increase in training samples per second and up to 5.3% increase in TFLOPS per GPU compared to non-compressed collective solutions. We demonstrated a significant improvement in model quality compared to the naïve ZFP scheme (Section IV-E). We also evaluated *ZHybrid* performance on modern large-scale HPC systems. We reported up to 17.3% increase in training samples per second as well as up to 12.7% increase in TFLOPS per GPU compared to non-compressed collective solutions. We also showed noticeable enhancement in model quality compared to naïve ZFP scheme (Section IV-F).

4) To the best of our knowledge, this is the first work that utilizes MPI collectives co-designed with GPU-based compression libraries to accelerate both model parallelism, data parallelism, and ZeRO communication (Section V).

II. BACKGROUND

A. GPU-based Compression libraries and MPI

The recent advancements in GPU technology, such as enhanced memory bandwidth, elevated core counts, and superior computation capabilities, have been a major factor in the adoption of GPU-based compression libraries. MPC [23] is a lossless compression technique that leverages the identification of similarity between consecutive floating-point numbers to compress the data and achieve a high compression ratio. ZFP [22], [28] is the state-of-the-art GPU-based compression library that supports high throughput read and write

random access. These compression libraries have also been co-designed with MPI collective communication to achieve high-performance, on-the-fly message compression for modern, dense GPU clusters [24]. Collective-level optimizations often avoid superfluous compression operations and utilization of GPU-kernels [25], [26], [29].

B. 3D Parallelism and ZeRO

This section covers backgrounds on 3D parallelism, which combines data-parallel, pipeline-parallel, and tensor-parallel. We will also discuss relevant works on ZeRO.

1) Data Parallelism (DP): Data Parallelism [17] is a distributed DL technique that distributes training data across multiple GPUs with model replicas to perform parallel training steps. Data Parallelism can significantly improve throughput compared to single-node training, as evidenced by relevant applications [15], [27], [30]. However, due to memory restrictions, DP has limitations with large, dense neural networks and high-resolution images.

2) Model Parallelism (MP): Model parallelism overcomes the limitation of DP by splitting the model into layers and distributing it on different devices. Common approaches to achieve model parallelism include Pipeline Parallelism (PP) [18], [19] and Tensor Parallelism (TP) [20]. PP uses inter-layer model parallelism with micro-batches executed in a pipeline, while TP employs sub-tensor splitting for parallel processing across multiple workers, optimizing tensor operations.

Higher-degree parallelism strategies emerged to further scale and harness the benefits of individual distributed models. MT-NLG [7] uses 3D parallelism, a combination of DP, PP, and TP, to train a billion-parameter model, facilitating larger model sizes and improved training capabilities.

When analyzing the communication pattern for DP and MP, we observe that DP involves less frequent All-reduce operations on larger data sizes. At the same time, MP requires more frequent point-to-point operations on smaller data.

3) ZeRO: ZeRO (Zero Redundancy Optimizer) [16] provides a memory optimization technique and overcomes the memory limitation of DP and the scalability issue of MP. This is achieved by partitioning optimizer states, gradients, and model parameters across GPUs to eliminate redundant storage and GPU memory consumption. Efforts have also been made to offload certain training parameters to CPU and NVMe memory [31] and optimize communication overhead [32].

For example, ZeRO stage 1 typically involves All-gather and Reduce-Scatter operations to collect and aggregate optimizer states.

C. Compression in Data Parallelism and Model Parallelism

Data parallelism incurs communication overhead due to the All-reduce operation required for gradient aggregation. Various works have proposed gradient compression techniques for DP to address the overhead and improve training speed while maintaining accuracy [33], [34]. These works are based on the sense that most of the gradients communicated in DP are sparse, which means these data structures have most of their elements concentrated in a few dimensions. In contrast, the remaining dimensions contain negligible values. A comprehensive study [1] has performed a low-rank analysis on the activation data in MP and concluded the opposite findings: activations in MP are dense, leading to its significance in preserving accuracy.

III. DESIGN

This section first details communication routines and messages in typical 3D parallelism and ZeRO stage 1 scenarios. Next, we will introduce the *MZHybrid* scheme and the *ZHybrid* scheme.

Figure 2 illustrates a typical 3D parallelism setting split across eight global workers. This setting contains two DP ranks, two PP stages within each DP rank, and two TP degrees within each PP stage. We examine communication calls in each parallelism dimension in the following paragraphs. In Figure 2, the global batch is split into two mini-batches, and each goes into a DP rank. Each DP rank will produce its own local gradients after a forward and a backward pass on the model replicas. Then, the root rank will issue an All-reduce call on these local gradients and re-distribute the global aggregated gradients to each DP rank before updating the model replica parameters. The All-reduce call is graphed between DP ranks in the middle of Figure 2. Typically, this All-reduce is conducted upon sparse gradients. However, as model size increases, such collective communication gradually becomes a bottleneck, given the increase in message size. Next, we split the model replica into two PP stages within each DP rank. Each PP stage will include a subset of the network layers. For example, in Figure 2, we assume a model with eight layers, and we split them across two pipeline stages, with pipeline stage 0 having network layers 0-3 and pipeline stage 1 having network layers 4-7. Pipeline stages mainly call point-to-point communication routines like MPI_Send & MPI_Recv to communicate gradients and activations with each other. These are also drawn between pipeline stages in Figure 2.

Next, we consider Tensor Parallelism. TP focuses on parallelizing matrix computations among workers. In this scenario, we parallelize computation workloads across two GPUs. For specific implementation, we refer to Megatron-LM [20]. In their approach, they split GEMM operations in MLP blocks, Self-Attention blocks, and the output embedding layer among

GPUs. This method requires two All-reduce primitives for a forward and a backward pass in a single transformer layer. For the output embedding layer, an All-reduce aggregates the different portions of the input embedding, and an All-gather will act to obtain GEMM outputs. Please refer to Figure 3 for details. In this parallelism dimension, the main communication primitives involved are All-reduce and All-gather—these primitives aggregate activations in forward passes and gradients in backward passes. We also illustrate these on the sides of pipeline stage blocks in Figure 2.

MPI Collectives	DP	PP	TP	ZeRO stage 1
All-reduce	✓	✗	✓	✗
All-gather	✗	✗	✓	✓
Reduce-Scatter	✗	✗	✗	✓
Point-to-point	✗	✓	✗	✗

TABLE I: MPI Collective communication in each parallelism stage.

Next, we detail about ZeRO stage 1 communications. ZeRO stage 1 optimizes GPU memory consumption by partitioning optimizer states over several workers. Optimizers like Adam require considerable per-parameter information and can acquire twice as much memory as a complete model. ZeRO stage 1 only keeps a fraction of optimizer states on each GPU. When the optimizer needs to update its state, the primary worker will call an All-gather to gather all the fractions on other workers. After the optimizer completes its updates on all the current global optimizer states, ZeRO will perform a Reduce-Scatter to redistribute all the optimizer states back to the original device. We exemplify a ZeRO stage 1 scenario with optimizer states split across 4 GPUs in Figure 4. Finally, in Table I, we listed all the MPI collective communication per parallelism stage.

A. MZHybrid: MPC for MP + ZFP for DP

MZHybrid	MPI Collectives	Compression Schemes
DP	All-reduce	ZFP
PP	Point-to-point	MPC
TP	All-reduce	MPC
	All-gather	MPC
ZeRO stage 1	All-gather	MPC
	Reduce-Scatter	MPC

TABLE II: MZHybrid: Compression scheme specified for each collective communication

In this section, we introduce the first hybrid compression scheme: *MZHybrid*. *MZHybrid* uses lossless MPC scheme for MP communication and lossy ZFP scheme for DP communication.

We provide illustrations using *MZHybrid* under the typical 3D parallelism scenario in Figure 5. We enforce lossless MPC for All-reduces in TP, point-to-point sends&recvs in PP, and lossy ZFP for All-reduce between DP ranks. For ZeRO stage 1 under *MZHybrid*, we enforce lossless MPC for All-gathers and Reduce-Scatters. Given that activations
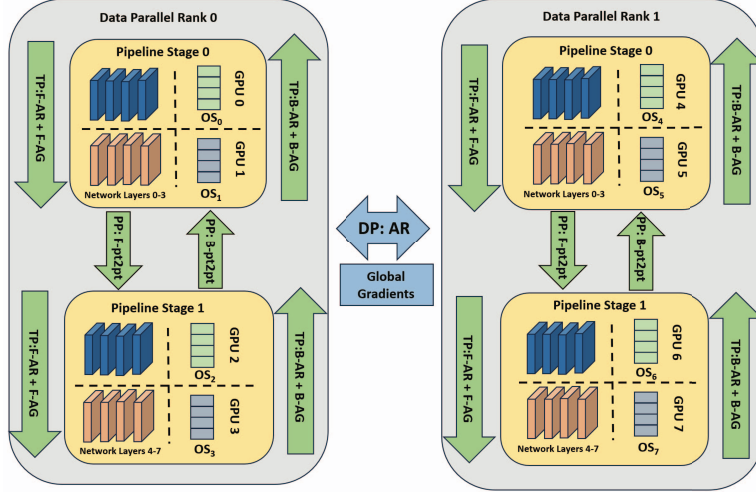
Fig. 2: A base 3D parallelism setting with a total of 8 GPUs. This setting contains two data parallel ranks, two pipeline stages and two tensor parallel degree. Collective operations for each parallelism degree are detailed. Optimizer states are also split across 8 workers. *DP*: Data Parallelism, *PP*: Pipeline Parallelism, *TP*: Tensor Parallelism, *OS*: optimizer state shards with device index as subscript. *AR*: All-reduce operations. *AG*: All-gather operations. F and B prefix indicates forward passes and backward passes respectively.
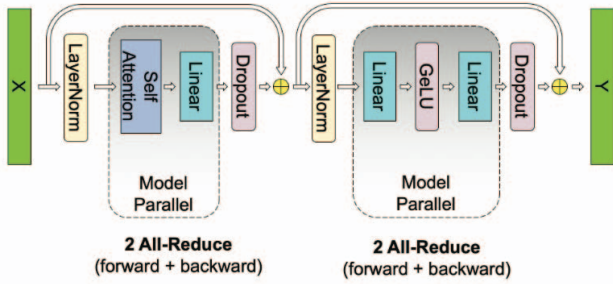
Fig. 3: Megatron-LM: communication primitives involved in a single transformer layer [20].
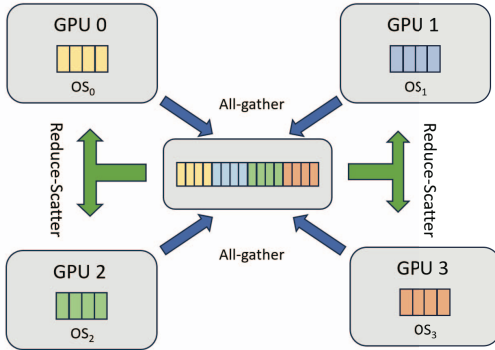
Fig. 4: ZeRO: communication primitives involved in ZeRO stage 1 over 4 GPUs. In our experiments, ZeRO stage 1 is integrated into 3D parallelism, here we use a separate graph to demonstrate communication routines for clarity. OS: optimizer state partitions with GPU number as index.

communicated in MP are mostly dense (contrary to gradients) [1], we want to preserve precision on these data to maintain model training performance. For communicating large and mostly sparse gradients between DP ranks, we apply an aggressive lossy ZFP scheme. It is worth mentioning that in MP settings, communication on gradients also exists during backward passes. To avoid over-compressing gradients, we apply MPC schemes to those gradients. We use Table II to specify our compression scheme choice for each collective involved for *MZHybrid*. We also experimented with different rates of ZFP under *MZHybrid*, please refer to IV-E.

B. ZHybrid: high-rate-ZFP for MP + low-rate-ZFP for DP

ZHybrid	MPI Collectives	Compression Schemes
DP	All-reduce	low-rate ZFP
PP	Point-to-point	high-rate ZFP
TP	All-reduce	high-rate ZFP
	All-gather	high-rate ZFP
ZeRO stage 1	All-gather	high-rate ZFP
	Reduce-Scatter	high-rate ZFP

TABLE III: ZHybrid: Compression scheme specified for each collective communication

This section presents the second hybrid compression scheme: *ZHybrid*. *ZHybrid* adopts a lossy ZFP scheme for all communication, including 3D parallelism stages and ZeRO stage 1. However, for different parallelism ranks, we apply different rates of ZFP. Since high-rate ZFP are better at preserving accuracy [28], we apply them towards MP units and optimizer state operations in ZeRO. For DP, We apply low-rate ZFP to large and sparse gradient reduction to eliminate negligible values. For experiments with different rates of ZFP under *ZHybrid*, please refer to IV-F. We use Table III to
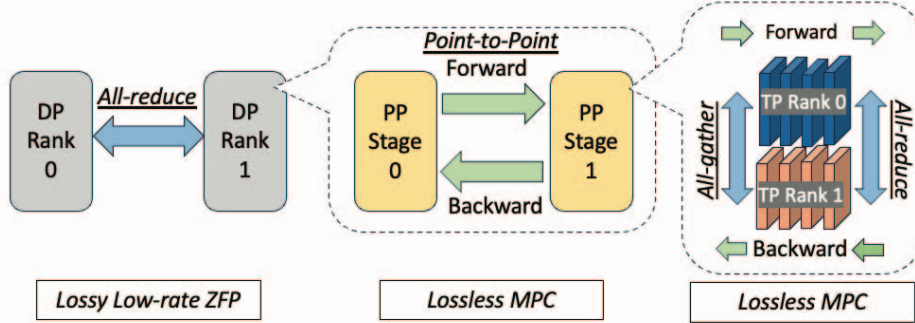
Fig. 5: *MZHybrid*: 3D Parallelism communication settings. Collective operations as well as compression schemes for each parallelism degree are detailed. We force lossless MPC for TP and PP while lossy ZFP for DP.

CPU	IBM Power9 44 Cores/Node
Memory	256 GB
GPU	4 NVIDIA Tesla V100 (64 GB Memory) with NVLINK
Interconnect	Mellanox IB EDR (100 Gb/s)

TABLE IV: Lassen cluster configuration

specify our compression scheme choice for each collective involved for *ZHybrid*.

IV. EVALUATION

In this section, we evaluate different hybrid compression schemes for LLM training in terms of training throughput (TFLOPS) and loss. We conducted experiments on the Lassen supercomputer hosted at Lawrence Livermore National Laboratory [35]. The cluster comprises 792 GPU nodes, each with four 16 GB memory NVIDIA Tesla V100 GPUs and two 44-core IBM Power 9 CPUs. Inter-node connection is established through Mellanox Infiniband EDR with a bandwidth of 100Gb/s, and the 4 GPUs on each node are split into two pairs connected via NVLINK (Table IV). Experiments on more advanced GPU architecture (A100, H100) would be beneficial, yet we believe that the core findings and narrative would largely remain the same.

A. Software Libraries

We invoked compression-assisted reduce-scatter-allgather All-reduce based on compression-assisted reduce-scatter and all-gather implemented on top of MVAPICH2-GDR 2.3.7 [11] for all training experiments. We chose the GPT-NeoX library given its open-sourced documentation and implementation of LLM parallel training procedures. This library was implemented based on Megatron-LM [20] and DeepSpeed [36] to support 3D parallelism. Furthermore, it has also been augmented with various novel optimizations, including ZeRO [16], etc. We compiled PyTorch v1.13.1 and the latest DeepSpeed from source with GPU-aware MPI support.

B. Training Configuration

The inter-node interconnect is InfiniBand-EDR-100Gb/s, and the 4 GPUs on each node are split into two pairs connected

via NVLINK. We agree that experiments on more advanced GPU architecture would be beneficial, but the fundamental storyline would be unchanged.

We selected the largest language model checkpointed in the GPT-NeoX library - GPT-NeoX-20B. Due to a lack of resources to train the original foundation model, it was necessary to conduct fine-tuning using a more constrained dataset. Specifically, the model was fine-tuned on 'Books3', a subset of the 'Pile' dataset developed by EleutherAI [37]. We set up the model using the same hyperparameters in the original paper [4]. We trained the model for 4000 steps. We changed the parallelism settings to match the Lassen GPU node configuration. We set the pipeline parallelism degree to be 6 across nodes and model parallelism degree to be 4 within nodes. This makes up a base training environment among 24 GPUs for one model replica. We use a gradient accumulation step of 1, a training micro batch size per GPU of 4, and scaled training up to 192 V100 GPUs. For optimizer settings, we adopted the Adam optimizer with beta values of 0.9 and 0.95 as well as an epsilon of 1.0e-8. We also enabled ZeRO optimizer to distribute optimizer states across devices to reduce memory consumption.

C. Naïve ZFP Compression Scheme

We first forced naïve lossy ZFP compression schemes to all parallelism. We conclude our results of testing naïve ZFP(rate:8) and ZFP (rate:16) by reporting training throughput in two aspects: samples per second and TFLOPS per V100 GPU. We also documented test loss on Books3 for model performance validation. In Figure 7a, when compared to default MVAPICH2-GDR implementations, conducting ZFP compression techniques showcased a **23.6%** increase in total samples per second for rate:8 and a **15.4%** respectively for rate:16 on 192 V100 GPUs. In Figure 7b, we observed a **22.5%** increase in TFLOPS per V100 GPU for rate:8 and a **11.14%** increase in that for rate:16 on 192 V100 GPUs. The throughput benefit difference between ZFP rate:8 and rate:16 is expected since, with a lower rate, we are extracting out more information, thus leading to more bandwidth being freed and better throughput. We continued to evaluate test loss on the trained model; we discern some degradation in the loss curves in Figure 7c. Compared to baseline showing a steep
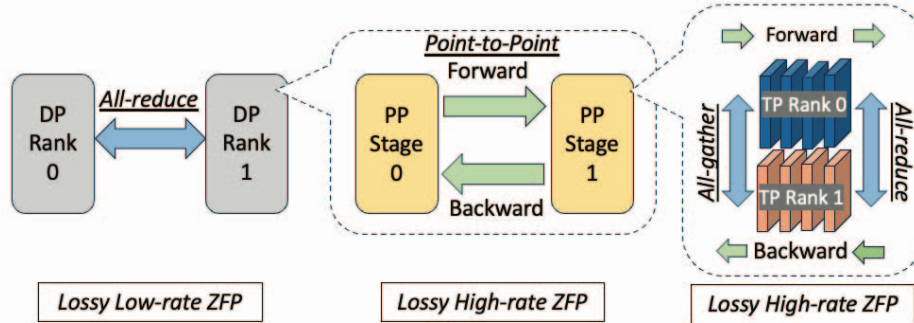
Fig. 6: *ZHybrid*: 3D Parallelism communication settings. Collective operations as well as compression schemes for each parallelism degree are detailed. We force high-rate ZFP for TP and PP while low-rate ZFP for DP.

decrease in test loss, naïvely compressing all messages using ZFP produces flatter loss curves and, eventually, larger loss values. It is also worth noting that a larger ZFP compression rate yields loss curves and values closer to baseline since less model information is being compressed during message passing.

D. Naïve MPC Compression Scheme

Secondly, we switched to using pure lossless MPC compression. We similarly applied MPC to all parallelism in Figure 8a and Figure 8b, and we report that using MPC didn't show significant throughput benefits. Nevertheless, in Figure 8c, we observed that when comparing to baseline loss curves, MPC elicited a better ability to match desired model performance and hardly produced any degradation. This observation is anticipated since MPC compression is a lossless scheme and proficient in preserving model data precision.

E. MZHybrid

In this section, we evaluate the *MZHybrid* compression scheme—applying lossless MPC in MP communication and lossy ZFP in DP communication. We recorded results for ZFP rate:8 and ZFP rate:16. In Figure 9a, we saw that using ZFP rate:8 together with MPC, training samples per sec showed a **4.4%** increase on 192 GPUs when compared to MVAPICH2-GDR baseline. For TFLOPS per GPU, we demonstrated in Figure 9b a **5.3%** raise when training across 192 GPUs. When it comes to model performance, we plotted loss curves with MZHybrid against naïve ZFP approach. Figure 9c staged that *MZHybrid* significantly reduces loss values when compared to naïve schemes for both ZFP rate:8 and ZFP rate:16. We anticipate such observations, although MPC struggles at providing throughput benefits for large message size, the speed-up provided by ZFP offsets such shortcoming. At the same time, the loss curve showed that incorporating MPC significantly improves model performance.

F. ZHybrid

We continued experimenting with *ZHybrid*: using different ZFP compression rates for MP and DP stages. We conducted experiments on two cases: one is ZFP(rate:24) for MP and ZFP(rate:8) for DP, and the second one is ZFP(rate:16) for MP

and ZFP(rate:8) for DP. While applying ZFP(rate:16) for MP, we observed a **20.4%** increase in training samples per second and a **20.6%** increase in TFLOPS per GPU. When considering ZFP(rate:24) for MP, we also see a **17.3%** increase in training samples per second and a **12.7%** increase in TFLOPS per GPU (Figure 10a, 10b). Then we compared *ZHybrid* against naïve ZFP in Figure 10c in terms of test loss and observed lower final loss values, which translates to better model quality. Increasing the ZFP rate for MP communication improves model performance as expected.

Both evaluated *ZHybrid* cases(rate:24 MP & rate:16 MP) staged lower loss values over naïve ZFP solution. Also, the moving average of the loss is higher for *ZHybrid*(rate:16 MP), which conforms with our expectation that lower ZFP rates lead to higher overall loss landscape (Figure 13).

G. Discussions

We compare our hybrid compression approach with NCCL [38], a collective communication library highly optimized for NVIDIA GPUs and networking. Our approach *ZHybrid*(rate:16-MP, rate:8-DP) exhibits up to 7.6% increase in samples per second and 12.9% increase in TFLOPS per GPU on 192 V100 GPUs. While higher ZFP rate (rate:24-MP, rate:8-DP) results in less performance gain, we still achieved up to 4.9% increase in samples per second and 5.5% increase in TFLOPS per GPU on the same scale. The reason for this is that as we scale up, inter-node bandwidth begins to saturate. Compression-assisted MPI collectives are capable to reduce message size during transfer and mitigate communication stress in this scenario, resulting in better GPU compute utilization. Compared to *ZHybrid*, *MZHybrid* provided more benefits on loss convergence rather than training throughput due to the overhead of lossless compression. The two proposed hybrid schemes benefit us in either training efficiency or quality; specific choices depend on the end-user's preference and metrics.
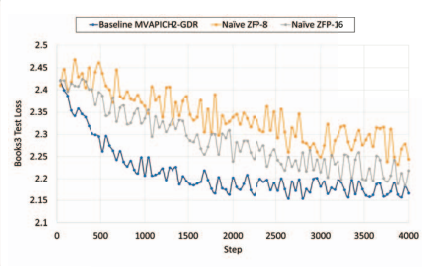
The key takeaway we addressed is that higher ZFP compression rates(i.e., less aggressive compression)lead to loss closer to baseline than low ZFP rates, which matches intuition. There is no correct answer for a "proper" rate since this depends on the use case, and this paper seeks to quantify the ZFP rate
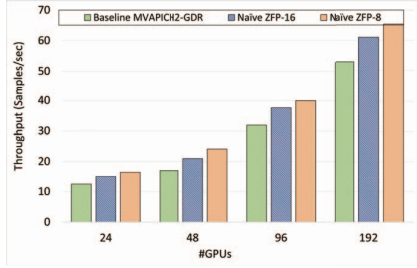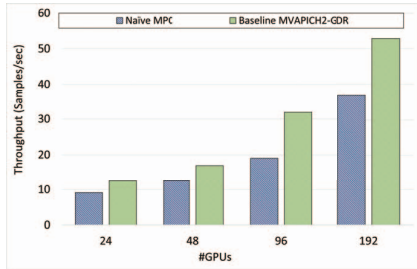
(a) Naïve ZFP: Training samples per second
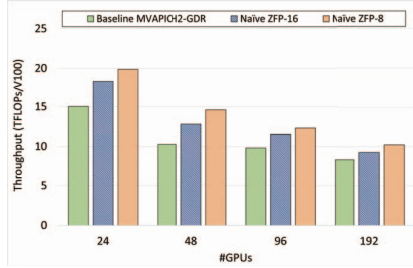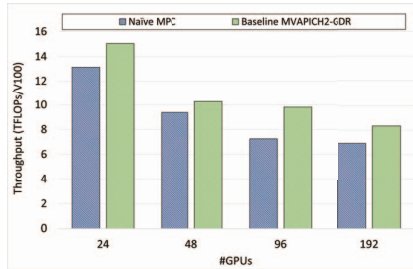
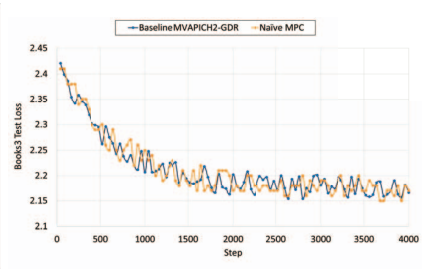(b) Naïve ZFP: TFLOPS per GPU

(c) Naïve ZFP: Books3 test loss

Fig. 7: Naïve ZFP Compression Scheme
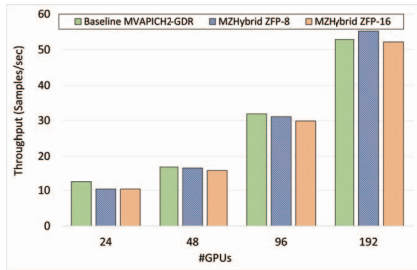

(a) Naïve MPC: Training samples per second
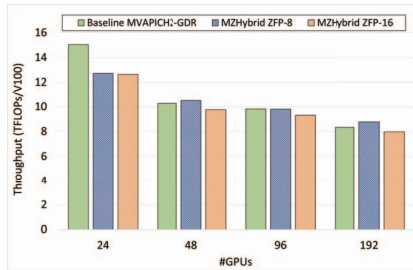
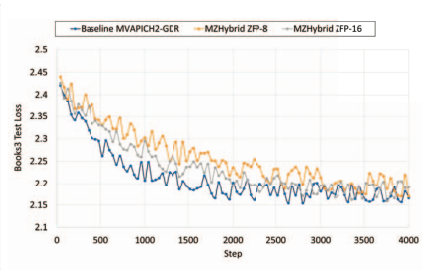(b) Naïve MPC: TFLOPS per GPU

(c) Naïve MPC: Books3 test loss

Fig. 8: Naïve MPC Compression Scheme
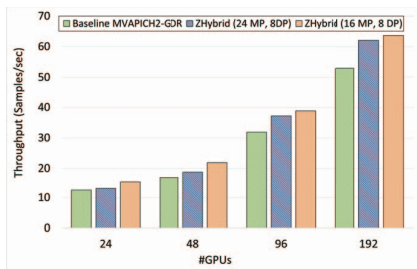

(a) MZHybrid: Training samples per second
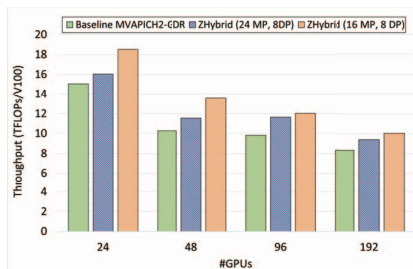
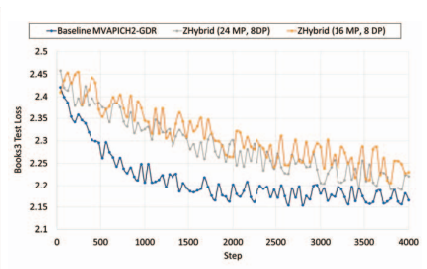(b) MZHybrid: TFLOPS per GPU

(c) MZHybrid: Books3 test loss

Fig. 9: MZHybrid Compression Scheme

(a) ZHybrid: Training samples per second

(b) ZHybrid: TFLOPS per GPU

(c) ZHybrid: Books3 test loss
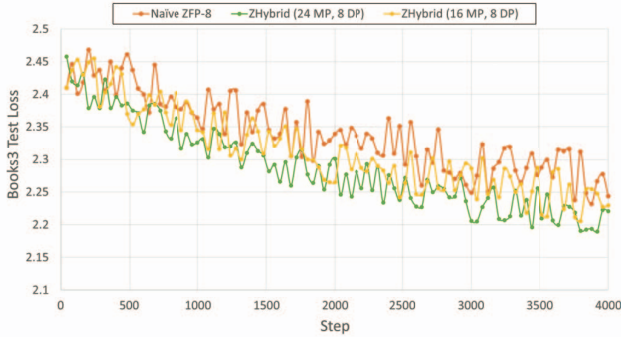
Fig. 10: ZHybrid Compression Scheme

Fig. 11: Comparison between *ZHybrid* (rate:24 MP & rate:16 MP) and naïve ZFP solution in test loss.
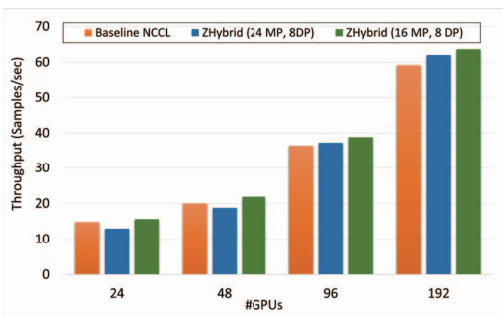

Fig. 12: Comparison between *ZHybrid* and NCCL in samples per sec.

tradeoff so that this "proper" rate can be selected. The broad guidelines we find are to choose a model-parallel(TP+PP) compression rate slightly smaller than an FP-32 precision but large enough to provide a loss increase that our application can tolerate. At this stage, there is no detailed analysis for every rate, and all that is left to future work.

V. RELATED WORK

Several studies have been done on compressing gradients to reduce training time [33], [34]. Furthermore, C. C. Chen
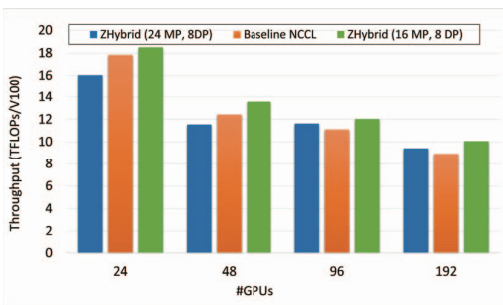

Fig. 13: Comparison between *ZHybrid* and NCCL in TFLOPS per GPU.

et al. [39] provided a hybrid communication compression method, which chooses the best compression method for every gradient to maintain model accuracy while reducing training time. Several distributed Deep Learning framework has also been integrated with mixed-precision training [40].

These studies primarily focus on gradient compression for DP, with limited work dedicated to compression techniques in MP. Current efforts have also been attributed to mostly compressing gradients during training but hardly on activations. GPU-based compression has also been co-designed with various MPI collectives to speed up distributed DL training under a certain degree of parallelism, such as PyTorch-FSDP [26].

Similar hybrid communication schemes has also been featured in works like MCR-DL [21]. MCR-DL supports mix-and-matching communication backends across MPI and NCCL [38] for all point-to-point and collective operations. However, this work is not based on MCR-DL. The implementation invokes compression-assisted MPI collectives directly on the PyTorch level. The only reason MCR-DL is mentioned is to refer to its communication profiling insights [21].

VI. CONCLUSION

In this paper, we propose two hybrid compression schemes, namely *MZHybrid* and *ZHybrid*, to leverage LLM training efficiency with adequate model performance. These two designs consider the fundamental differences among parallelism strategies (DP, PP, and TP) and the sparsity in the message communicated in these schemes. *MZHybrid* applied lossless MPC towards model-parallel communication and lossy ZFP towards data-parallel communication. *ZHybrid* forces different ZFP rates for different parallelism stages. For model-parallel, we chose high-rate ZFP to preserve the precision of dataflow occurring within a model. For data parallel, we adopt low-rate ZFP to reduce gradient size communicated across models.

The proposed design *MZHybrid* demonstrates up to 4.4% increase in training samples per second and 5.3% increase in TFLOPS per GPU compared to non-compression approaches with significant improvement in training quality over naïve compression. The proposed design *ZHybrid* demonstrates up to 20.4% increase in training samples per second and 20.6% increase in TFLOPS per GPU compared to non-compression approaches and still poses noticeable improvement in model performance over naïve compression—the two hybrid schemes emphasized on training throughput benefits or model quality.

We believe our approach can generalize to other use-cases which exhibit heavy collective communication and are saturating interconnect bandwidth.

In future work, we plan to co-design more MPI collective operations with GPU-based compression libraries to accelerate more scientific applications and Deep Learning workloads. Choices include All-reduce and other communication routines. Also, we expect that more advanced compression techniques and libraries can be incorporated such as cuSZ [41] and block-based quantization [42].

References

[1] S. Bian, D. Li, H. Wang, E. P. Xing, and S. Venkataraman, "Does compressing activations help model parallel training?" 2023.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[3] OpenAI, "Gpt-4 technical report," 2023.

[4] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang, M. Pieler, U. S. Prashanth, S. Purohit, L. Reynolds, J. Tow, B. Wang, and S. Weinbach, "Gpt-neox-20b: An open-source autoregressive language model," 2022.

[5] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[7] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," 2022.

[8] "MPICH: High-Performance Portable MPI," http://www.mpich.org. Accessed: March 6, 2024.

[9] NVIDIA, "NVIDIA GPUDirect," https://developer.nvidia.com/gpudirect, 2011, Accessed: March 6, 2024.

[10] Open MPI, "Open MPI: Open Source High Performance Computing," https://www.open-mpi.org/, 2004, Accessed: March 6, 2024.

[11] Network-Based Computing Laboratory, "MVAPICH: MPI over Infini-Band, Omni-Path, Ethernet/iWARP, and RoCE," http://mvapich.cse.ohio-state.edu/.

[12] IBM, "IBM Spectrum MPI: Accelerating high-performance application parallelization," https://www.ibm.com/us-en/marketplace/spectrum-mpi, 2018, Accessed: March 6, 2024.

[13] A. Jain, A. A. Awan, A. M. Aljuhani, J. M. Hashmi, Q. G. Anthony, H. Subramoni, D. K. Panda, R. Machiraju, and A. Parwani, "Gems: Gpu-enabled memory-aware model-parallelism system for distributed dnn training," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[14] A. Jain, A. Shafi, Q. Anthony, P. Kousha, H. Subramoni, and D. K. Panda, "Hy-fi: Hybrid five-dimensional parallel dnn training on high-performance gpu clusters," in *High Performance Computing: 37th International Conference, ISC High Performance 2022, Hamburg, Germany, May 29 – June 2, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 109–130. [Online]. Available: https://doi.org/10.1007/978-3-031-07312-0_6

[15] Q. Anthony, L. Xu, H. Subramoni, and D. Panda, "Scaling single-image super-resolution training on modern hpc clusters: Early experiences," 06 2021, pp. 923–932.

[16] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimization towards training A trillion parameter models," *CoRR*, vol. abs/1910.02054, 2019. [Online]. Available: http://arxiv.org/abs/1910.02054

[17] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, no. 4, aug 2019. [Online]. Available: https://doi-org.proxy.lib.ohio-state.edu/10.1145/3320060

[18] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," 2019.

[19] Z. Li, S. Zhuang, S. Guo, D. Zhuo, H. Zhang, D. Song, and I. Stoica, "Terapipe: Token-level pipeline parallelism for training large-scale language models," 2021.

[20] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020.

[21] Q. Anthony, A. Awan, J. Rasley, Y. He, A. Shafi, M. Abduljabbar, H. Subramoni, and D. Panda, "Mcr-dl: Mix-and-match communication runtime for deep learning," May 2023.

[22] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, 08 2014.

[23] A. Yang, H. Mukka, F. Hesaaraki, and M. Burtscher, "MPC: A Massively Parallel Compression Algorithm for Scientific Data," in *IEEE Cluster Conference*, September 2015.

[24] 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.

[25] Q. Zhou, Q. Anthony, A. Shafi, H. Subramoni, and D. K. D. Panda, "Accelerating broadcast communication with gpu compression for deep learning workloads," in *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2022, pp. 22–31.

[26] Q. Zhou, Q. Anthony, L. Xu, A. Shafi, M. Abduljabbar, H. Subramoni, and D. Panda, "Accelerating distributed deep learning training with compression assisted allgather and reduce-scatter communication," May 2023.

[27] Q. Anthony, A. A. Awan, A. Jain, H. Subramoni, and D. K. D. Panda, "Efficient training of semantic image segmentation on summit using horovod and mvapich2-gdr," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 1015–1023.

[28] J. Diffenderfer, A. Fox, J. Hittinger, G. Sanders, and P. Lindstrom, "Error analysis of zfp compression for floating-point data," *SIAM Journal on Scientific Computing*, vol. 41, pp. A1867–A1898, 02 2019.

[29] Q. Zhou, P. Kousha, Q. Anthony, K. Shafie Khorassani, A. Shafi, H. Subramoni, and D. K. Panda, "Accelerating mpi all-to-all communication with online compression on modern gpu clusters," in *High Performance Computing*, A.-L. Varbanescu, A. Bhatele, P. Luszczek, and B. Marc, Eds. Cham: Springer International Publishing, 2022, pp. 3–25.

[30] Q. Anthony, L. Xu, A. Shafi, H. Subramoni, and D. Panda, "Scamp: Scalable meta-parallelism for deep learning search," May 2023.

[31] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning," arXiv, April 2021. [Online]. Available: https://www.microsoft.com/en-us/research/publication/zero-infinity-breaking-the-gpu-memory-wall-for-extreme-scale-deep-learning/

[32] G. Wang, H. Qin, S. A. Jacobs, C. Holmes, S. Rajbhandari, O. Ruwase, F. Yan, L. Yang, and Y. He, "Zero++: Extremely efficient collective communication for giant model training," 2023.

[33] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," 2018.

[34] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2020.

[35] "Lassen - Livermore Computing center - Specifications," https://hpc.llnl.gov/hardware/platforms/lassen.

[36] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[37] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," 2020.

[38] NVIDIA, "NCCL2," https://developer.nvidia.com/nccl, 2017, Accessed: March 6, 2024.

[39] C.-C. Chen, Y.-M. Chou, and J. Chou, "Phy: A performance-driven hybrid communication compression method for distributed training," *Journal of Parallel and Distributed Computing*, vol. 180, p. 104719, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731523000898

[40] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2018.

[41] 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

[42] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," 2022.