ZeRO++: Extremely Efficient Collective Communication for Giant Model Training

Guanhua Wang*, Heyang Qin*, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari
Olatunji Ruwase, Feng Yan[1], Lei Yang[2], Yuxiong He
Microsoft
{ guanhuawang, heyangqin, samjacobs, connorholmes, samyamr, olruwase, yuxhe } @microsoft.com

ABSTRACT

Zero Redundancy Optimizer (ZeRO) has been used to train a wide range of large language models on massive GPUs clusters due to its ease of use, efficiency, and good scalability. However, when training on low-bandwidth clusters, or at scale which forces batch size per GPU to be small, ZeRO's effective throughput is limited because of high communication volume from gathering weights in forward pass, backward pass, and averaging gradients. This paper introduces three communication volume reduction techniques, which we collectively refer to as ZeRO++, targeting each of the communication collectives in ZeRO. First is block-quantization based all-gather. Second is data remapping that trades-off communication for more memory. Third is a novel all-to-all based quantized gradient averaging paradigm as replacement of reduce-scatter collective, which preserves accuracy despite communicating low precision data. Collectively, ZeRO++ reduces communication volume of ZeRO by 4x, enabling up to 2.16x better throughput at 384 GPU scale.

KEYWORDS

Large model training, High performance computing, Deep learning

1 EXTENDED INTRODUCTION

Deep learning (DL) models have been applied successfully in many different domains such as image/video analysis, natural language processing, speech recognition, etc. Over years, the quality, functionality, and coverage of these models have continued to improve. Model size has been a key factor in this improvement. There is a strong correlation of model size with accuracy and improved functionality, and as result, the model size has grown dramatically in recent years. For example, parameter size grows from 100 million to 500+ billion from BERT [9] to Megatron-Turing NLG [33].
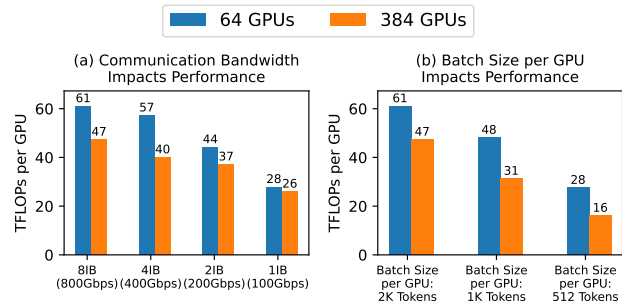
Figure 1: Large scale training throughput are constrained by network bandwidth and batch size per GPU

With the increase in model size, the memory and compute requirements for training have increased significantly beyond the capability of a single accelerator (e.g., a GPU). Training massive models requires efficiently using aggregated computing power and memory across hundreds or even thousands of GPU devices. There are two popular approaches to this, namely 3D parallelism [22, 36] and Zero Redundancy Optimizer (ZeRO) [29].

3D parallelism combines data parallelism [2, 6], pipeline parallelism [13, 14, 21] and tensor parallelism [32] to distribute model training workloads across hundreds of GPUs. This approach can achieve excellent per-GPU computing and memory efficiency. However, a major downside here is the system and user complexity. It puts the burden of refactoring the single GPU code to work for 3D parallelism on data scientists and AI practitioners, which is nontrivial and often cumbersome.

In contrast, ZeRO offers an alternative that requires no model code refactoring. ZeRO is a memory efficient variation of data parallelism [2, 6] where model states are partitioned across all the GPUs, instead of being replicated, and reconstructed using gather based communication collectives on-the-fly during training. This allows ZeRO to effectively leverage the aggregate GPU memory across machines, at the expense of minimal communication overhead compared to standard data parallel training (2M vs 3M for model size of M) [29], while still achieving excellent throughput scalability [30].

1.1 Limitations of ZeRO

Ease of use of ZeRO combined with its ability to scale efficiently across hundreds to thousands of GPUs, has resulted in its wide adoption. However, there are two critical scenarios where efficiency of ZeRO can be limited due to communication overhead: i) clusters with low-bandwidth, and ii) at very small batch sizes per GPU.

Guanhua Wang*, Heyang Qin*, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari and Olatunji Ruwase, Feng Yan[1], Lei Yang[2], Yuxiong He

On one hand, clusters with low-bandwidth is common in majority of cloud computing environments. Although high performance nodes like DGX boxes [10, 11] are equipped with high-bandwidth NVLink [25] and NVSwitch [26] as intra-node interconnects, cross-node links are often less than 100Gbps ethernet which makes it the communication bottleneck. As shown in Figure 1(a), the per GPU throughput on low bandwidth clusters is only half of that with high-bandwidth clusters.

On the other hand, even on high-bandwidth clusters, when running on thousands of GPUs, the batch size per GPU is limited by the maximum global batch size that can be used during the training without sacrificing convergence efficiency [2, 17, 39]. In other words, as global batch size cannot be increased indefinitely without slowing down model convergence, training on thousands of GPUs forces the batch size per GPU to be very small, which reduces the compute-to-communication ratio and thus creates a communication bottleneck. As shown in Figure 1(b), the per GPU throughput is heavily impacted by small batch size per GPU, which is a result of communication bottleneck.

However, rare efforts have been made to optimize end-to-end communication efficiency for ZeRO. There are many previous work on reducing communication overhead in distributed model training, such as 1-bit LAMB [18], 1-bit Adam [35] and other error compensation compression techniques for gradient averaging [1, 12, 31, 34]. However, none of them can work with ZeRO as they all assume model state replication, while model states are partitioned in ZeRO. We start from scratch and provide an end-to-end system for reducing all communication overhead in ZeRO training.

1.2 ZeRO++

In this paper, we present a novel system of communication optimizations collectively called ZeRO++ that offers dramatic communication volume reduction for ZeRO. Below we discuss the main communication overheads in ZeRO, followed by three different communication optimizations in ZeRO++ that address them.

Assume the model size as M. During the forward pass, ZeRO [29] conducts an all-gather operation to collect all the parameters (M) needed to train for all model layers. In the backward pass, ZeRO re-collects parameters (M) with all-gather first, then each GPU can compute local gradients. After that, ZeRO operates reduce-scatter function to aggregate and redistribute gradients (M) across accelerators. In total, ZeRO has a total communication volume of $3M$, spreads evenly across 2 all-gather and 1 reduce-scatter.

To reduce these communication overheads, ZeRO++ has three sets of communication optimizations, targeting each of the above mentioned three communication collectives respectively:

Quantized Weight Communication for ZeRO (qwZ) First, in order to reduce parameter communication volume during forward all-gather, we adopt quantization on weights to shrink down each model parameter from FP16 (2 bytes) to INT8 (1 byte) data type before communicating, thus reducing the communication volume by half. However, naively conducting quantization on weights may lose model training accuracy. In order to preserve decent model training precision, we adopt block-based quantization [8], which conducts independent quantization on each subset of model parameters. There is no existing implementation for high performance

block-based quantization. Thus, we implement highly optimized quantization CUDA kernels from scratch.

Hierarchical Weight Partition for ZeRO (hpZ) Second, to reduce communication overhead of all-gather on weights during backward, we trade GPU memory for communication. More specifically, instead of spreading whole model weights across all the machines, we maintain a full model copy within each machine. At the expense of higher memory overhead, this allows us to replace the expensive cross-machine all-gather on weights with intra-machine all-gather, which is substantially faster due to much higher intra-machine communication bandwidth.

Quantized Gradient Communication for ZeRO (qgZ) Third, reducing communication cost of gradients using reduce-scatter is even more challenging. Directly applying quantization to reduce communication volume is infeasible. The main issue is, even by incorporating block-based quantization to reduce-scatter operation, it will still significantly hurt model training accuracy. The key reason behind is quantization will decrease value precision. And reduction on low-precision values will accumulate and amplify the errors. Therefore, we propose a novel and much more efficient gradient communication paradigm as a general replacement of reduce-scatter collective, where the gradients are compressed using block-based INT4 quantization during the communication to reduce the communication volume, but the full precision is recovered before the reduction operator to preserve training accuracy. We call this *qgZ*, and is designed to i) overcome significant accuracy loss that would result from low-precision reduction if we were to simply implement reduce-scatter in INT4/INT8, and ii) avoid accuracy degradation and significant latency overhead of a long sequence of quantization and dequantization steps needed by a ring [23] or tree [5, 37] based reduce-scatter (e.g., left of Figure 5), even if we did the reductions in full-precision. Furthermore, *qgZ* leverages the hierarchical nature of modern GPU clusters, where intra-node bandwidth is significantly higher than inter-node, to first reduce gradients within a node before doing cross-node reduction to minimize inter-node communication volume, resulting in 2/4x communication volume reduction (INT8/4) compared to FP16 reduce-scatter. We further reduce end-to-end latency of *qgZ* by pipelining intra-node and inter-node communication and conducting CUDA kernel fusion.

Communication Volume Reduction By incorporating all three components above, we reduce the cross-node communication volume from $3M$ down to $0.75M$. More specifically, for forward all-gather operation on model weights, by applying INT8 quantization, we reduce the communication size from M to $0.5M$. During backward all-gather on weights, with our secondary copy of model parameters, we reduce the communication size from M to 0. By replacing backward fp16 reduce-scatter on gradients to our novel all-to-all based INT4 reduce-scatter, we reduce cross-node communication from M to $0.25M$. Thus, in total, we reduce $3M$ communication to $0.75M$.

Evaluation We implemented ZeRO++ and performed extensive evaluation demonstrating three key results: i) scalability of GPT-3 like models on up to 384 GPUs achieving over 45% of sustained peak throughput, ii) consistent speedup of up to 2.4x over ZeRO [29] baseline across models ranging from 10-138B parameters, and iii) comparing with baseline in 4x higher bandwidth cluster, ZeRO++

achieves similar throughput in low-bandwidth setting. In addition, we show the impact of each of the three optimizations in ZeRO++ and how they compose together. Furthermore, we also show the impact of our optimized kernel implementations on end-to-end system throughput. Finally, we conduct convergence evaluation indicating that ZeRO++ has negligible impact on model convergence and maintains similar model training accuracy as ZeRO baseline.

The main contributions of this paper are as follows:

- Blocked quantized weights (qwZ) reduces communication volume of all-gather of weights by 50%.
- Hierarchical partitioning of model weights (hpZ) completely eliminates inter-node all-gather communication in backward propagation.
- Novel, all-to-all quantized gradient reduction collective (qgZ) reduces gradient communication by 75% comparing with reduce-scatter.
- Optimized Integration of each of the above techniques into existing ZeRO implementation, that enables communication and computation overlapping, and leverages custom high performance CUDA kernels for quantization, dequantization, as well as operator fusion (section 4). Our implementation translates the 4x communication volume reduction of ZeRO++ into real throughput improvement.
- Extensive experiments shows that i) over 45% of sustained peak throughput even at small batch sizes, ii) up to 2.4x end-to-end system improvement over ZeRO, and iii) achieving similar throughput in low-bandwidth cluster compared to baseline in high-bandwidth cluster. In addition, we present performance breakdown and analysis of diffrent components of ZeRO++.Our end-to-end training shows that ZeRO++ does not affect model convergence.
- ZeRO++ is open-sourced and released as part of https://github.com/microsoft/DeepSpeed

2 BACKGROUND AND RELATED WORK

2.1 Data, Model and 3D parallelism

Data parallelism (DP), pipeline parallelism (PP), and tensor parallelism (TP) are three forms of parallelism used to train large models across multi-GPU clusters. [6, 15, 20, 22] DP is commonly used when model size fits within a single GPU memory. In DP, each GPU holds a full copy of model weights and trains on separate input data. MP is orthogonal to DP, and is often used in cases where model size cannot fit into a single GPU's memory. Instead of splitting input data, model parallelism partitions a full model into pieces and assigns each model piece onto a GPU. There are mainly two approaches for model parallelism: i) pipeline parallelism (PP) and ii) tensor parallelism (TP). PP [14, 15, 20] splits models vertically, creating sequential stages consisting of a contiguous subset of layers. While there is sequential dependency between stages for an input micro-batch, the stages can be executed in parallel across micro-batches. In contrast, TP [22] splits each layer across multiple GPUs, where each GPU works on a different part of the layer for the same input.

3D parallelism [33, 36] refers to combination of DP, PP, and TP, and is capable of achieving excellent throughput and scalability, and has been used to train a wide range of large language models [4,

19, 22, 28]. Despite being highly efficient, 3D parallelism is severely limited by the fact that it requires complete rewrite of model and training pipeline to make them compatible with 3D parallelism [33].

Algorithm 1: ZeRO algorithm

Input	: $model, worldSize$
Output	: $model$

1 **while** $model$ not $converged$ **do**
2   $all\_gather\_Parameters(worldSize)$;
3   $model.forward()$;
4   $partition(worldSize)$;
5   $all\_gather\_Parameters(worldSize)$;
6   $model.backward()$;
7   $partition(worldSize)$;
8   $reduce\_scatter\_Gradients(worldSize)$;
9   $optimizer.step()$;
10 **end while**
11 **Return:** $model$

2.2 ZeRO Optimizer

ZeRO is a memory-optimized solution for data parallel training. ZeRO partitions and distributes all model states (i.e., parameters, gradients, optimizer states) among GPUs in use and recollects model states only when the layer needs to be computed. There are three different stages for using ZeRO to optimize on-device memory usage. In ZeRO stage 1 (ZeRO-1), only optimizer states are split and spread across all GPUs in use. ZeRO stage 2 (ZeRO-2) partitions both optimizer states and gradients, where ZeRO stage 3 (ZeRO-3) splits all three components of model states as parameters, gradients, and optimizer states.

ZeRO-3 is the most memory efficient solution for model training at large scale, but at the cost of more collective communications. Algorithm 1 illustrates the high-level pseudocode for ZeRO-3. During model training, ZeRO-3 lazy-schedules the fetching of parameters until the computation needs to happen on a particular layer. Before forward propagation, ZeRO launches an all-gather to collect the full model weights and then computes the forward pass (line 2-3) of Algorithm 1. Then ZeRO empties the all-gather weights buffer after forward computation completes (line 4). During backward, ZeRO re-collects all model weights again via a second all-gather (line 5) to calculate gradients (line 6). Once gradients are calculated on each GPU, ZeRO empties weights buffer again (line 7) and conducts a reduce-scatter operation to do gradient averaging and re-distribution (line 8). Model states and parameters are updated in optimizer step (line 9). In a nutshell, to minimize the on-device memory footprint using ZeRO-3, three collective communication operations are issued at each training iteration, which include 2 all-gather on weights and 1 reduce-scatter on gradients.

2.3 Communication Reduction Techniques

Quantization: Quantization is often used to reduce memory footprint, and data movement volume by using low precision to represent data [7, 8]. However, the loss of information from representing high precision data with lower precision often comes with accuracy degradation. Many related work focus on improving

quantization accuracy. The fundamental challenge of quantization accuracy lies in the vast difference in number ranges and granularity between high precision and low precision data (Eg. FP32/16 vs. INT8). Some related work [41] propose to filter the outliers in data to mitigate the gap in numerical ranges. Yet their accuracy hinges on the quality of outlier filtering and it brings extra filtering overhead. Dettmers et al. [8] proposes to use block based quantization on optimizer states to improve the quantization accuracy yet it requires changes to the model structure thus limits its usability.

Gradient Compression: Starting from 1-bit SGD of error-compensation compression [31], gradient compression has been pushed to an extreme direction of using just a single bit. To deal with non-linear gradient-based optimizers like Adam or Lamb, 1-bit quantization algorithms like 1-bit Adam [35] and 1-bit Lamb [18] are proposed, which achieve extreme efficient gradient communication in distributed training. However, 1-bit Adam/LAMB cannot be directly applicable to ZeRO-3. The main reason is 1-bit Adam/Lamb assumes each GPU has the full view of optimizer states (OS) for the model, but ZeRO-3 splits it across all the GPUs in use. Therefore, it is infeasible to directly apply existing gradient compression techniques at ZeRO-3 and we need to design our own.

ZeRO Communication Reduction: To reduce expensive cross-node communication, recent optimization on ZeRO-3, such as MiCS [40], trades on-device memory for communication. In MiCS, the GPU cluster is divided into sub-groups, and model states are partitioned within a sub-group but replicated across sub-groups. By keeping the sub-group size small, MiCS can either leverage high bandwidth intra-node interconnect, or use hierarchical communication to lower the communication volume. hpZ in ZeRO++ adopts a similar approach of trading memory for less communication. The key difference is that hpZ only do secondary partition on weights, while keeping all other model states partitioned across all GPUs. This allows hpZ to achieve significant communication reduction without the massive memory overhead of MiCS.

3 DESIGN

In this section, we elaborate on the design of our three key optimizations in ZeRO++ introduced in Section 1 for reducing the communication overhead of ZeRO: i) Quantized Weight Communication for ZeRO (qwZ), ii) Hierarchical Partitioning for ZeRO (hpZ), and iii) Quantized Gradient communication for ZeRO (qgZ). After that, we discuss the end-to-end impact of these optimizations to reduce to total communication volume of ZeRO.

3.1 Quantized Weight Communication for ZeRO (qwZ)

As discussed in Section 2.2, ZeRO partitions the model weights across all the ranks (i.e., GPUs) and fetches the FP16 weights layer-by-layer right before they are needed in computation via all-gather for the forward and backward of each training iteration. To reduce the communication overhead of forward all-gather on weights, qwZ, quantizes FP16 weights to INT8 right during the all-gather, and dequantizes them back to FP16 on the receiver side, and then conducts layer computation.
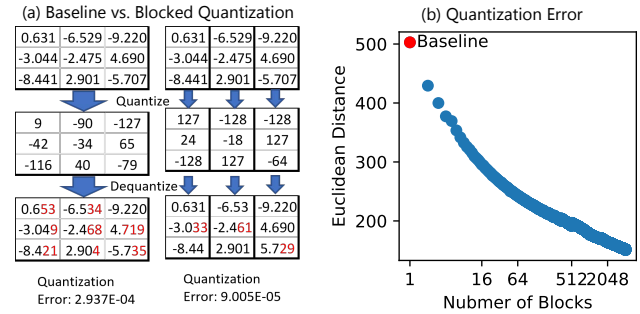

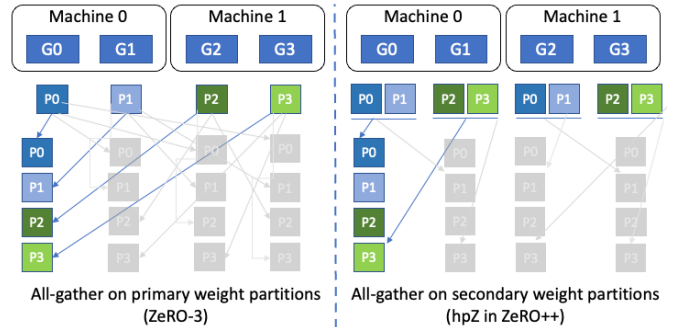Figure 2: Illustration & example of block based quantization vs. baseline

Figure 3: hpZ removes cross node traffic in backward all-gather by holding secondary weight partitions in on-device memory.

While this reduces the communication volume of the all-gather by 2x, doing so naively results in two major issues: i) the lowering of precision results in significant accuracy degradation during training as discussed in 2.3 , and ii) the quantization and dequantization overhead negates any throughput gain from communication volume reduction. We discuss the optimized implementation of qwZ to minimize the quantization and dequantization overhead in Section 4. Here, we primarily focus on design choices to mitigate accuracy degradation.

qwZ uses blocked based quantization to improve the quantization accuracy. As illustrated in Figure 2, each weight tensor is divided into smaller chunks, and converted into INT8 by symmetric quantization, using an independent quantization scaling coefficient. By keeping the quantization granularity small, we significantly mitigate the gap in number ranges and granularity.

We show an example of the quantization error of performing block based quantization vs. the non-blocked quantization baseline in Figure 2(a). Fig. 2(b) shows a case study of weights quantization on BERT model, where block based quantization reduces the quantization error by 3x. More in-depth convergence evaluations are shown in Sec. 5.

3.2 Hierarchical Partitioning for ZeRO (hpZ)

ZeRO-3 partitions all its model states across all its ranks, resulting in communication collectives that span all the GPUs. With hpZ, we notice that it is possible to have different partitioning for different model states, limiting the communication collectives to a subset of the GPUs. Given that on modern GPU clusters, intra-node communication bandwidth is significantly higher than inter-node communication bandwidth, this presents opportunities to reduce the inter-node communication.

More specifically, in hpZ, we eliminate the inter-node all-gather during the backward pass by holding secondary FP16 weights partition within each node. We do this by creating a hierarchical partitioning strategy consisting of two partitions: first, all model states are partitioned globally across all devices as in ZeRO-3, which we call primary partition. Second, a secondary copy of FP16 parameters is partitioned at the sub-global level (e,.g., compute node, see figure 3), which we call secondary partition. This secondary copy of FP16 parameters is replicated across multiple secondary partitions.

Consider a 64-node cluster, each node with 8 GPUs. Model weights are partitioned in two stages: i) across all 512 GPUs that we call primary partition, and ii) the same weights are also partitioned within a compute node across 8 GPUs, that we call secondary partition. In this example, for the secondary partition, each compute node in the cluster holds a full replica of FP16 weights partitioned among the 8 GPUs within the node, and there are 64 of such replicas in total.

3.2.1 A training iteration with hpZ. During the forward pass of a training iteration, we all-gather weights based on the primary partition across all GPUs. However, once the weights are consumed during the forward pass, they are partitioned based on the secondary partition. Given the temporal consistency of model parameters between forward and backward passes, when the weights are needed again during the backward pass, we all-gather weights based on this secondary group. Note that when the secondary partitioning is set to be a compute node, this avoids any inter-node communication for this all-gather. Finally, at the end of the iteration, during the optimizer step, all the model states, as well as the primary copy of the fp16 parameter are updated based on the primary partition. hpZ makes two changes to baseline ZeRO pseudocode in Algorithm 1: i) in line 4, parameter partitioning is based on *secondary group size*, ii) parameter all-gather preceding backward pass in line 5 is also based on *secondary group size*.

Our design of hpZ is flexible to support any *secondary group size*. The group size controls how many ranks (i.e., GPUs) are in the secondary partition. It is also a measure of memory-communication trade-off of hpZ. Simply put, by default, hpZ secondary partition is node-based (recall intra-node bandwidth is multiple factors of inter-node bandwidth for current and future hardware configurations) but can be extended to support multiple compute nodes as needed.

3.2.2 Memory Usage Analysis. By design, hpZ trades memory for communication efficiency. It is important to analyze this tradeoff. Recall that standard data parallel DNN (DP) replicates model parameters across data parallel ranks, ZeRO-3 on the other hand partitions parameter across data parallel ranks. A midway approach is model
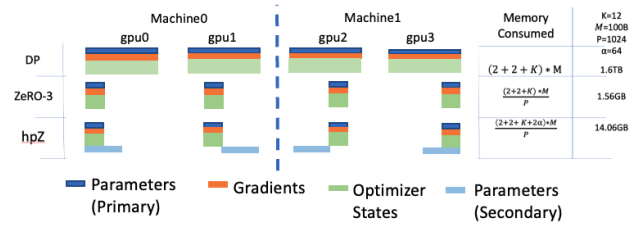

Figure 4: Per-device memory consumption analysis of standard data parallel (DP), ZeRO stage 3 (ZeRO-3) and proposed hierarchical partitioning of ZeRO parameters (hpZ). K denotes the memory multiplier of optimizer states, M represents the number of trainable parameters, P is the data parallel group size or world size, and α is the number of secondary groups or ratio of world size to the number of ranks in the secondary group. A typical real world scenario example is provided in the last column. We assume a model size of 100B trained on 1024 V100 GPU DGX cluster (64 compute nodes, 16 GPUs per node).

parameters partitioned across a subset of devices as long as model parameters fit.

Figure 4 provides a concrete memory usage estimate of a typical large language model of size of 100B parameters, with primary group size of 1024 GPUs and secondary group size of 16 GPUs (e.g., DGX-2 V100 node). As shown in Figure 4, with our proposed method, hpZ consumes $8.9x$ more memory than ZeRO-3, our approach is still $114x$ less memory requirement than standard DP. This marginal increase in memory usage is compensated for by efficient intra-node communication schedule. By eliminating or reducing inter-node communication for backward pass, hpZ reduces the end-to-end communication of ZeRO by $1.5x$, while still supporting model training with hundreds of billions of parameters.

3.3 Quantized Gradients Communication for ZeRO (qgZ)

In this section, we propose a novel quantized reduce-scatter algorithm called qgZ based on all-to-all collectives that enables a 4x communication volume reduction of gradient reduce-scatter by replacing FP16 with INT4 quantized data, while overcoming precision loss challenges described in Section 1, as well as numerous system challenges that we will outline in this section.

qgZ leverages all-to-all collectives to implement quantized reduce-scatter which includes three major components: 1) all-to-all-based implementation of quantized gradient reduce-scatter, 2) reducing communication volume with hierarchical collectives, 3) tensor slice reordering for correct gradient placement. We talk about each of them step-by-step.

3.3.1 All-to-all based implementation. A naive approach towards quantized reduce-scatter, while avoiding precision loss due to reduction is to apply quantization and dequantization to a ring-based reduce-scatter directly as shown on the left of Figure 5. We can inject quantization and dequantization on each GPU. Once a GPU
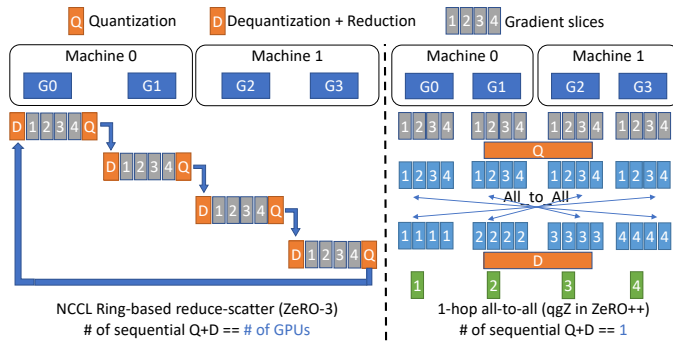
Figure 5: Comparison between ZeRO-3 ring-based reduce-scatter and qgZ 1-hop all-to-all.
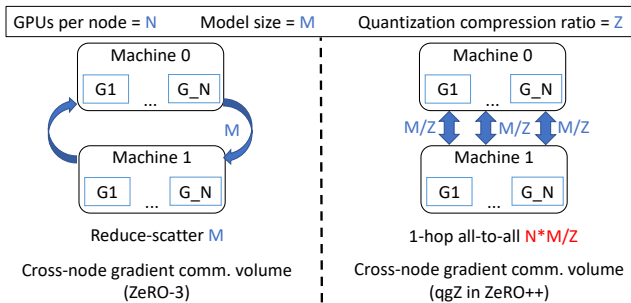

Figure 6: Communication volume comparison between ZeRO-3 reduce-scatter and qgZ 1-hop all-to-all.

Figure 7: qgZ apply hierarchy all-to-all to reduce cross node traffic.

receives gradients from its predecessor, we dequantize it to recover full precision and conduct a local reduction. Next we can quantize local reduction output and pass quantized data to its successor. To finish the whole reduce-scatter, the number of sequential quantization and dequantization kernels is equal to the number of GPUs (i.e., n) in use.

Thus, applying quantization and dequantization on existing ring based reduce-scatter collective will lead to high communication latency and low value precision due to multiple sequential quantization and dequantization steps. Although recent tree-based collective like Blink[38] could reduce the number of sequential kernels from n to log(n), the long latency and low precision issue is not completely resolved.

To overcome this, we completely abandon existing ring-based reduce-scatter approach and incorporate 1-hop all-to-all collective for our gradient communication. As shown on the right of Figure 5, we first apply quantization on a given tensor, then we conduct all-to-all communication among all the GPUs. After all-to-all, we apply another dequantization to recover the data precision and then reduce on high-precision values to get the final gradient reduction output. By replacing ring-based solution with our all-to-all collective, we reduce the number of sequential quantization+dequantization kernel from the number of GPUs to 1. Thus, we solve the long latency and low precision issues when applying quantization in reduce-scatter for supercomputing scenarios like DGX boxes connected in fat-tree topology.
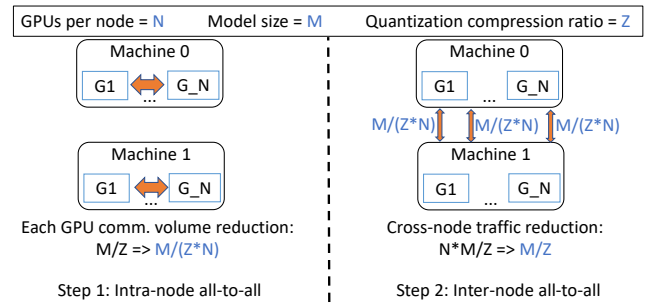
3.3.2 Reducing inter-node communication volume. Although replacing reduce-scatter with all-to-all achieves single-shot quantization and dequantization, it introduces a new problem; the inter-node communication volume increases instead of decreasing despite the quantization of data. We elaborate on this in Figure 6.

Here we assume model size of M, GPU per node is N, gradient compression ratio as Z. Reduce-scatter, reduces the data during transmission over the ring, thus the total amount of data for cross-node communication is M. However, when using our 1-hop all-to-all approach, even though the data are compressed before communication (i.e., M/Z), each GPU needs to send out M/Z amount of data to GPUs on the other nodes. Therefore, each machine will generate $N*M/Z$ amount of cross-node communication data, which is much bigger than reduce-scatter communication volume.

To address this, we do a hierarchical 2-hop all-to-all instead of 1-hop: a) first intra-node all-to-all and b) followed by inter-node all-to-all, which is shown as Figure 7. First, with high-bandwidth links among GPUs inside a machine, we conduct intra-node all-to-all on quantized data, then dequantize data and reduce on dequantized data. After intra-node quantization, all-to-all, dequantization, and reduction, we reduce the data size per GPU from M/Z to $M/(Z*N)$. After intra-node all-to-all is completed, we conduct the inter-node all-to-all communication, which is similar to 1-hop all-to-all we described above. Given that now each GPU only needs to send out $M/(Z*N)$ data, the communication volume per machine is now $M/(Z*N)*N = M/Z$. By adopting this hierarchical all-to-all communication as 2-hop approach, we resolve the communication volume blow-up issue in our 1-hop scheme perfectly. Note that even though the total communication volume is doubled (one intra-node, the other inter-node), intra-node communication introduces negligible overhead given NVLink/NVswitch high bandwidth, and cross-node traffic has been significantly reduced, which is the major bottleneck in gradient communication.

3.3.3 Tensor slice reordering for correct data placement. With the 2-hop all-to-all, the inter-node communication volume is as expected, however, this introduces a gradient misplacement issue. We describe this issue using a 2x2 example, where we have 2 machines and each machine has 2 GPUs. As shown in Figure 8, the correct final gradient placement is shown as green boxes in the figure, where GPU 0 holds final gradient partition 1, GPU 1 holds gradient partition 2, so on and so forth.
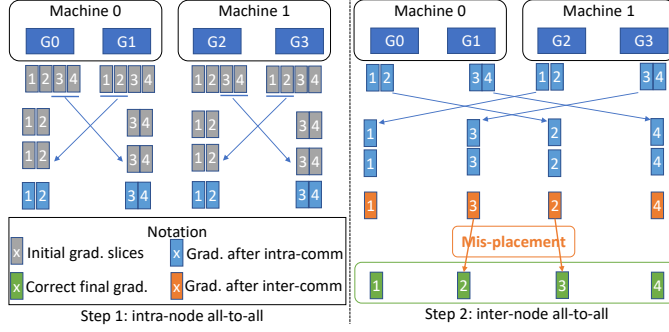
Figure 8: Gradient partition misplacement when applying hierarchical all-to-all in qgZ.

Figure 9: Tensor slices reordering to correct gradient misplacement in qgZ.

Our 2-step all-to-all communication works as follows, first we divide all gradients on each GPU into 4 chunks, then conduct our intra-node all-to-all. After intra-node all-to-all finishes, GPU0 (i.e., G0) holds partial aggregated gradient partition 1,2 whereas G1 holds gradient partition 3,4. Same thing happens on G2 and G3. Since G1 does not have gradient partition 2 (which is supposed to be held by G1) while G2 does not have gradient partition 3, after inter-node all-to-all, there is gradient misplacement issue on both G1 and G2.

We address this with tensor slice reordering. As shown in Figure 9, before intra-node all-to-all begin, we first swap the tensor slice order of slice 2 and 3, which is shown as orange arrows. Then after intra-node all-to-all is completed, G1 now has gradient 2 while G2 has gradient 3. Therefore, after the inter-node all-to-all, all GPUs get the correct gradient placement. Mathematically, given X GPUs per node and Y nodes in total, each GPU will hold X*Y gradient slices initially. Our tensor slice reordering works as follows:

$$before : [0, 1, 2, 3, 4, ...YX - 3, YX - 2, YX - 1] \quad (1)$$

$$after : [0, X, 2X, ...(Y - 1)X, 1, X + 1, (Y - 1)X + 1, ...YX - 1] \quad (2)$$

Based on Equation 1 and 2, we can map each original tensor slice position (i.e., Equation 1) to new tensor slice position (i.e., Equation 2) on each GPU to correct final gradient misplacement issue.

In summary, by solving above three challenges step-by-step, we design a novel gradient communication and reduction protocol, which can be a more communication efficient and generalized replacement of reduce-scatter collective. We discuss some of the optimization and implementation details for our approach in Sec. 4.

3.4 ZeRO++ Communication Volume Analysis

Table 1 illustrates theoretical communication volume comparison between ZeRO-3 and ZeRO++. We assume the model size of M. As described in Section 2, during ZeRO-3 there are 3 collective calls: all-gather on weights in forward pass, then all-gather on weights in backward pass and last is reduce-scatter on gradients in the backward. And each collective communicates M volume of data.

With ZeRO-3, in total we need to communicate 3M data per each training iteration. Given that intra-node communication is fast with NVLink and NVSwitch, we ignore intra-node communication and focus on cross-node traffic only. For all-gather in the forward pass, by incorporating our quantized weights communication, we reduce
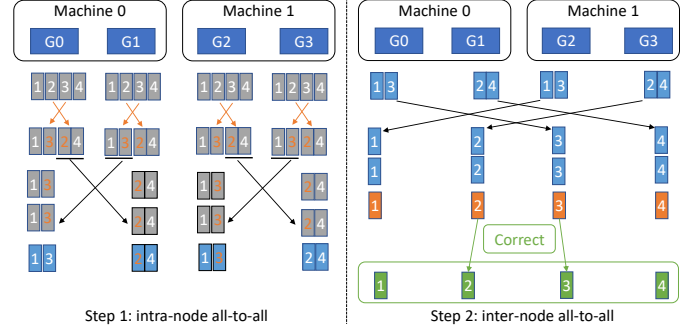
Comm. Volume	forward all-gather	backward all-gather	backward reduce-scatter
ZeRO-3	M	M	M
ZeRO++	0.5M	0	0.25M

Table 1: Communication volume comparison between ZeRO-3 and ZeRO++.

communication volume from M to 0.5M. During the all-gather in the backward pass, by holding secondary weights partition within each node, we completely removed cross-node traffic. For reduce-scatter in the backward pass, by replacing reduce-scatter with our novel quantized gradient communication protocol, we reduce cross-node traffic from M to 0.25M. Therefore, compared with ZeRO-3, ZeRO++ reduces communication volume from 3M down to 0.75M for each training iteration.

4 OPTIMIZED IMPLEMENTATION

In this section, we discuss two key optimizations that enable ZeRO++ to fully realize the potential of 4x communication volume reduction to improve throughput without getting limited by implementation overheads: i) overlapping different communication and compute streams, when doing so enables better resource utilization, and ii) optimized CUDA kernels for quantization, dequantization, and tensor slice reordering operators, and kernel fusion across these operators when appropriate to minimize the memory traffic overhead. Below we discuss the two lines of optimization in detail.

4.1 Overlap Compute and Communication

To reduce end-to-end communication time, we overlap quantization computation with communication for all-gathering of weights in both forward and backward passes. For the hierarchical all-to-all based reduce-scatter implementation of gradients, we overlap the intra-node communication with inter-node communication.

4.1.1 Communication-computation overlapping on weights. For all-gather on weights, we enable communication-computation overlap using two key features : i) we track the execution order of model layers to get the sequence they will be fetched. ii) we guarantee asynchronous quantization execution. Specifically, the call to the
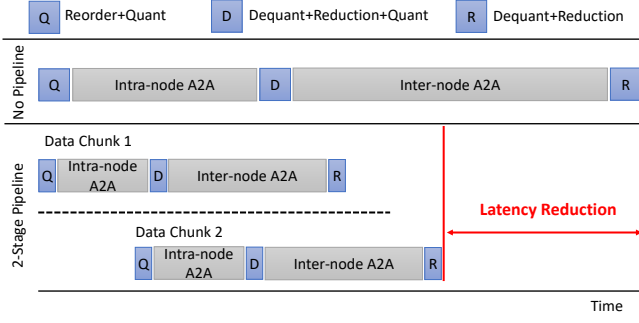
Figure 10: Pipelining and overlapping intra-node communication with inter-node communication in *qgZ*.

Algorithm 2: Generalized tensor slice reordering (*qgZ*)

Constants : *stages*, *nodeSize*, *nodes*
Input   : *partitionID*
Output   : *mappedPartitionID*

1 $totalDevices \leftarrow nodeSize * nodes$;
2 $stageID \leftarrow partitionID \% totalDevices$;
3 $chunkID \leftarrow \frac{partitionID}{totalDevices}$;
4 $pipelineOffset \leftarrow stageID * totalDevices$;
5 $chunkOffset \leftarrow \frac{stageID}{nodeSize}$;
6 $chunkBase \leftarrow (chunkID \% nodeSize) * nodes$;
7 **Return:** $pipelineOffset + chunkBase + chunkOffset$;

---

quantization kernel is non-blocking and we further avoid operations that involve explicit/implicit CUDA synchronization (e.g. tensor concatenation), making the quantization a non-blocking operation that can be launched asynchronously.

With this two features, as ZeRO fetch parameters for each layer, the communication of the current layer and the quantization of the next layer can be launched at the same time on different CUDA streams. When the quantized data are needed for the next layer, ZeRO++ synchronizes the quantization stream to make sure the quantized data are ready. This approach hides the quantization cost of the next layer under the communication time span of the current layer which hides the quantization overhead.

4.1.2 Hierarchical Collectives for Gradient Communication. As discussed in Sec. 3.3.2, our all-to-all based gradient communication is broken into two stages: first intra-node communication followed by inter-node communication. The inter-node communication depends on the results of the intra-node communication, therefore, with a naive implementation, inter-nodes links are idle during intra-node communication and vice versa. To reduce latency by leveraging both inter-node and intra-node links in parallel, we chunk our input gradient tensor and pipeline transfer between intra-node communication and inter-node communication. As shown in Figure 10, compared with "no pipeline" case on the top, simply adopting a "2-stage pipeline" transfer achieves the amount of end-to-end latency reduction shown as the red arrow-line in Figure 10. By overlapping intra-node and inter-node communication, the end-to-end latency of gradient communication is significantly reduced.

Doing this pipeline correctly has implications on our tensor slice reordering process. The more pipeline stages we have, the more fine-grained tensor slices are needed for reordering. Therefore, we also propose a generalized tensor slices reordering scheme as algorithm 2, which covers both w/ and w/o pipelining data transfer cases. Here stages refer to the number of pipeline stages we have, nodeSize is the number of GPUs per node and nodes is the number of nodes.

Next, we discuss how we optimize our CUDA kernels to further reduce all quantization related overhead.

4.2 CUDA Kernels

As existing quantization implementations are unable to capture the combination of data mapping and high throughput necessary to minimize kernel overhead, we implement and optimize custom CUDA kernels to implement these primitives. In particular, these kernels aim to (1) saturate device memory bandwidth and (2) minimize the total traffic via fusion.

Maximizing Bandwidth Utilization: A core quantization and dequantization library of composable operators was developed as the foundation for ZeRO++. The core primitives leverage efficient vectorized memory accesses at the maximum granularity a given GPU architecture supports. In order to satisfy the alignment requiments these instructions have, model state is partitioned such that quantization granularities will be 16B aligned. Additionally, we leverage instruction level parallelism to overlap multiple memory transactions with each other. In practice, the combination of vectorized accesses and instruction level parallelism enables the quantization library to achieve full GPU memory bandwidth utilization.

Minimizing Total Traffic: Multiple techniques are used to reduce the total memory traffic for quantization kernels. First, the size of each quantization block is tuned so as to express sufficient parallelism to schedule across a GPU's streaming multiprocessors and cache values not quantized yet in the register file while calculating the quantization scale and offset for the block. Second, we fuse tensor reshaping and quantization into the same kernel to avoid redundantly loading data from global memory. For example, the tensor slice reordering (i.e., orange arrow-lines in Figure 9) is realized within a fused quantization and remapping kernel.This fused kernel achieves the same level of performance as a single quantization kernel working with contiguous data. Finally, we fuse sequential dequantization, reduction, and quantization operations into single kernel implementation, which reduces total memory traffic by 9x in *qgZ*.

5 EVALUATION

In this section, we perform three sets of evaluations for ZeRO++. First, we perform end-to-end evaluations showing : i) scalability evaluation on up to 384 GPUs , ii) speedup over state-of-the-art (SOTA) baseline across models ranging from 10-138B parameters, and iii) throughput comparisons for cluster setting with varied cross-node bandwidth. Second, we perform throughput analysis
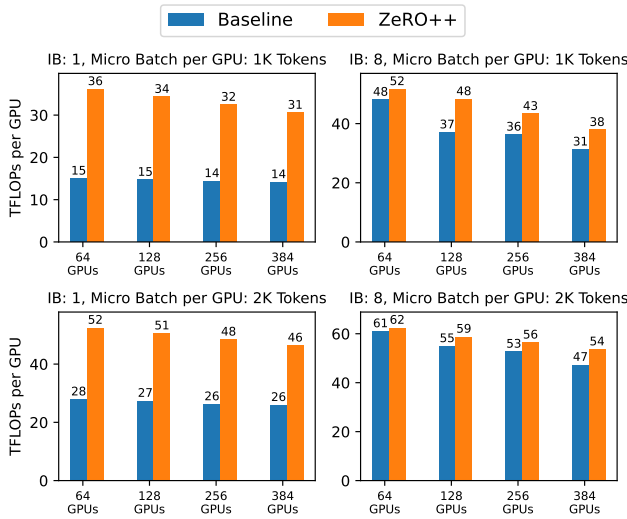
Figure 11: Scalability on up to 384 GPUs of 18B model with different numbers of InfiniBand connections and tokens per GPU

Table 2: End-to-end speedup of ZeRO++ on 384 GPUs with different model sizes

Model Size	Tokens per GPU	1 IB Connection			8 IB Connections		
		Baseline TFLOPs	ZeRO++ TFLOPs	Speedup	Baseline TFLOPs	ZeRO++ TFLOPs	Speedup
138B	2K	19.96	37.90	1.90x	47.55	55.30	1.16x
138B	1K	11.25	21.81	1.94x	34.19	44.38	1.30x
91B	2K	19.99	38.06	1.90x	47.74	56.26	1.18x
91B	1K	11.27	21.93	1.95x	34.49	44.36	1.29x
49B	2K	20.06	38.08	1.90x	48.05	56.24	1.17x
49B	1K	11.27	21.95	1.95x	34.54	44.46	1.29x
18B	2K	25.98	46.40	1.79x	47.31	53.65	1.13x
18B	1K	14.15	30.57	2.16x	31.27	37.87	1.21x

Figure 12: ZeRO++ achieving high bandwidth cluster performance with significantly lower bandwidth

and breakdown, evaluating the impact of different components of ZeRO++, as well as the impacts of our kernel optimizations on end-to-end throughput. Finally, we show convergence evaluation indicating that ZeRO++ doesn't harm model convergence and maintains similar model training accuracy.

5.1 Methodology

Hardware: 24 NVIDIA DGX-2 nodes where each with 16 V100 SXM3 32 GB GPUs [11]. The nodes are connected by InfiniBand (IB) with NVIDIA SHARP support [16], achieving total inter-node bandwidth of over 800 Gbps. To evaluate ZeRO++ in clusters under different network environments, we show the performance of ZeRO++ running with different cross-node bandwidth by enabling from 1 to 8 IB connections (i.e., 100 Gbps to 800 Gbps).

Baseline: We use ZeRO-3 as the baseline given its ease-to-use for training giant models at large scale. To evaluate the performance of our optimized kernels, we also implemented ZeRO++ with PyTorch quantization[27] and non-fused kernels as baselines for our ablation study.

Model Configurations: We use GPT-style transformer models for evaluation. Given Megatron-Turing-NLG [33] training 530B model on 2K GPUs using 2K tokens per GPU (i.e., micro batch size), we evaluate ZeRO++ with the same 2k tokens per GPU setting. We also evaluate on 1K tokens per GPU to test ZeRO++ with more extreme scale scenario. The number of layers and hidden sizes are adjusted to have models of different sizes. Please refer to the appendix and our open-sourced evaluation scripts for hyperparameters and other training details.

5.2 E2E System Evaluations

We evaluate ZeRO++ end-to-end performance here. One key metric we use here is the percentage of *peak performance*, which is shown

as equation 3.

$$peak\_performance = achieved\_TFLOPs/max\_TFLOPs \quad (3)$$

Given that we use V100 GPU, its *max_TFLOPs* is 120 TFLOPs [24] for mixed precision computation. Thus, our reported *peak performance* refers to the percentage number of *achieved_TFLOPs*/120.

5.2.1 Scalability upto 384 GPUs. In Figure 11, we present ZeRO++ scalability evaluation from 64 to 384 GPUs with 18B model on both low (1 IB) and high (8 IB) bandwidth clusters. On low bandwidth cluster, ZeRO++ achieves 30% and 38.3% of peak performance (120 TFLOPs) even at 384 GPUs for 1K and 2K batch sizes, which is much higher compared to 12.5% and 21.6% as baseline peak performance. This presents up to **2.4x** better throughput. On high bandwidth cluster, despite having significantly more bandwidth, ZeRO++ still enables up to 1.29x better throughput, and can achieve up 45% of sustained peak throughput at 384 GPUs. ZeRO++ significantly speed up large scale training for low bandwidth clusters while archiving decent speedup even on high bandwidth clusters.

5.2.2 Throughput for different model sizes. Table 2 compares training throughput for models of 18B-138B on 384 GPUs between ZeRO++ and baseline on both low and high bandwidth clusters. On low bandwidth cluster, ZeRO++ consistently achieves over 31.5% and 18.1% peak performance for 2K and 1K batch sizes on all models. Compared with the baseline peak performance of 16.6% and 9.3%, the speedup is up to **2.16x**. On high bandwidth cluster, ZeRO++ peak performances are 44.7% and 31.5%, which is 1.3x over the baseline peak performance of 31.5% and 26.0%. ZeRO++ is robust and offers consistent speedup across different model and batch sizes as well as across clusters with different network bandwidths.

Guanhua Wang*, Heyang Qin*, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari and Olatunji Ruwase, Feng Yan[1], Lei Yang[2], Yuxiong He

Table 3: End-to-end performance when using ZeRO++ w.\wo. optimized kernels

	Optimized Quantization Kernel	Optimized Fusion Kernel	TFLOPs
Baseline	N/A	N/A	15
ZeRO++	No	No	19.73
ZeRO++	No	Yes	21.6
ZeRO++	Yes	No	31.40
ZeRO++	Yes	Yes	**36.16**

5.2.3 Democratization for large scale training. Figure 12 compares the throughput of ZeRO++ on a low cross-node bandwidth (200 Gbps as 2 IB) cluster with the baseline running on 800 Gbps high-bandwidth (8 IB) cluster. For small model of 18B, ZeRO++ achieves a higher peak performance of 41.6% compared with baseline peak performance of 39.1% despite running with 4x lower cross-node bandwidth. For large model of 138B, ZeRO++ and baseline achieve the same peak performance of 40%, but baseline runs at 4x higher cross-node bandwidth. This evaluation shows that ZeRO++ makes large scale training more accessible by significantly decreasing the minimum cross-node bandwidth requirement for efficient training. Furthermore, it demonstrates that optimized ZeRO++ implementation effectively translates the 4x communication reduction of ZeRO++ into real end-to-end system throughput gain.
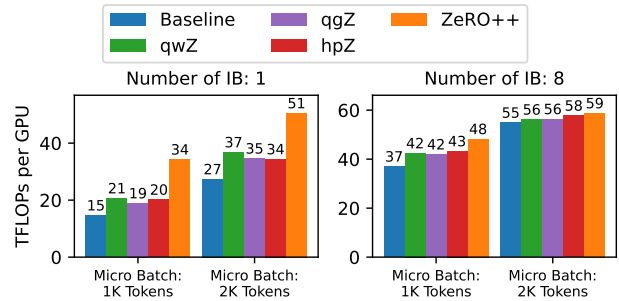
5.3 Throughput Breakdown and Analysis

5.3.1 Impact of Individual Techniques. In Figure 13, we show the individual and combined impact of qwZ, hpZ, and qgZ, on the throughput of 18B model on 128 GPUs. On low bandwidth clusters, each of these techniques enables a speedup ranging from 1.3-1.4x compared with baseline, while achieving an aggregated speedup of up to 2.26x. Note that our TFLOPs throughput is calculated from wall-clock time measurement, ZeRO++ aggregated throughput gain is not equivalent to sum of qgZ, qwZ, hpZ gain. We can validate the theoretical speedup with composition of our techniques by accumulating the speedup multiplicatively: $1.4 * 1.26 * 1.3 = 2.29$, which is very near to what we achieved as 2.26x.

For high bandwidth clusters, the individual speedup ranges between 1.13-1.16x, for a combined speedup of up to 1.3x. The figure demonstrates that each of these techniques has a similar impact towards throughput improvement and they compose effectively to produce a much larger aggregated speedup.

5.3.2 Impact of Kernel Optimizations. Here, we evaluate the impact of our optimized kernels on ZeRO++ throughput using a 18B model running on 64 GPUs.
Quantization Kernel: As shown in Table 3, compared with the baseline that uses PyTorch quantization [27], our optimize quantization kernels can achieve up to 1.67x speedup in terms of end-to-end throughput. Also, the baseline implementation suffers performance degradation as group number increases which means the throughput gap will be larger when used with larger models.

Figure 13: Throughput of 18B models on128 GPUs with ZeRO++, qwZ, qgZ, hpZ, and baseline on different numbers of InfiniBand connections

Table 4: hpZ vs MiCS evaluation on a 4 node cluster (16 V100 GPUs per node)

Model Size	Token Size	ZeRO TFLOPs	hpZ TFLOPs	MiCS TFLOPs
7.5B	1K	36.99	38.39	38.96
7.5B	2K	53.3	54.4	52.72
18B	1K	51.47	52.42	OOM
18B	2K	60.94	61.44	OOM

Kernel Fusion: As described in Section 4.2, kernel fusion is one of our key optimizations to improve memory throughput when executing sequences of CUDA kernels. Our fusion includes 1) tensor-reorder and quantization fusion 2) intra-node dequant, intra-node reduction and inter-node quant fusion. As shown in Table 3, we achieve up to 1.15x speedup on the end-to-end throughput.

5.3.3 Comparing hpZ with MICS. As previously discussed in Section 2, closely related to hierarchical weight partition for ZeRO (*hpZ*) is *MiCS*[40]. Key difference of the two methods is what data are replicated in secondary group; model weights are replicated in *hpZ*, entire model states are replicated in *MiCS*. Table 4 shows per-GPU throughput of both methods for different model and token size configurations. The table also shows that given a secondary partition size of a single node (16 V100 GPUs), *hpZ* can support 18 billion parameter model where as *MiCS* reports out-of-memory (OOM) at this scale.

5.4 Model convergence analysis

Next we evaluate ZeRO++'s impact on model convergence by training GPT-350M model with 30B tokens on the pile dataset [3] using ZeRO++, ZeRO++ with basic (non-blocked) quantization, and ZeRO-3 as baseline. All hyperparameters are kept the same between baseline training and ZeRO++ trainings to ensure fair comparison. The convergence is measured by the validation LM loss.

As shown in Figure 14, we present end-to-end training trace. The training with basic (non-blocked) quantization diverged at the beginning so there is no visible data, on the contrary, ZeRO++
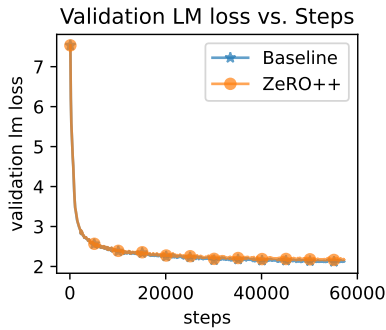
Figure 14: Training convergence for GPT-350M on 30B tokens

Table 5: Validation loss at the end of training (GPT 350M / 30B tokens)

	Evaluation LM loss
Baseline	2.121762
ZeRO++ (hpZ&qwZ&qgZ on)	2.165584
ZeRO++ (hpZ&qwZ on; qgZ on for first 50%)	2.134013
ZeRO++ (hpZ&qwZ on; qgZ off)	2.121653

is closely aligned with the baseline, and also confirms our previous analysis of better quantization accuracy by using block based quantization.

We further extended the convergence evaluation by comparing the final evaluation loss at the end of training. As shown in Table 5, even with all three optimizations on, the final evaluation loss is only off by 1%. We further merged this convergence gap by using a straightforward interleaving schedule where the hierarchical partitioning and quantized weights are turned on throughout the training and the quantized gradient is only turned on for the first 50% of training. For a more extended case, we also evaluate hierarchical partitioning and quantized weights alone. The results suggest our convergence is identical to the baseline in this case.

6 CONCLUSION

This paper present ZeRO++, an efficient collective communication solution for giant model training using ZeRO stage-3. We optimize both model weights and gradients communication in forward and backward pass of each training iteration. To reduce communication volume of model weights in forward propagation, we adopt block-based quantization and data pre-fetching. To remove cross-node communication of weights during backward pass, we hold secondary model partition on each node to trade memory for communication. To minimize gradient communication during backward propagation, we design and implement a novel all-to-all based gradient quantization and reduction scheme. By incorporating all the

three optimizations above, we improve system throughput up to 2.16x in large scale model training using 384 V100 GPUs. We envision ZeRO++ as the next generation of easy-to-use framework for training giant models at trillion-level model scale.

7 AUTHORSHIP AND MAJOR CREDIT ATTRIBUTION

- **Guanhua Wang:** design and implementation of qgZ, code integration, high performance quantization kernels design and implementation, solving all CUDA kernel conflicts in code merging, majority of paper writing.
- **Heyang Qin:** design and implementation of qwZ, code integration/resolve conflicts in code merging, experimental design and evaluation, in depth convergence study.
- **Sam Ade Jacobs:** design and implementation of hpZ, code integration/resolve conflicts in code merging.
- **Connor Holmes:** design and implementation of high performance quantization kernels.
- **Samyam Rajbhandari:** chief architect
- **Olatunji Ruwase:** technical support
- **Yuxiong He:** team lead

REFERENCES

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. *Advances in neural information processing systems* 30 (2017).

[2] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–43.

[3] Stella Biderman, Kieran Bicheno, and Leo Gao. 2022. Datasheet for the Pile. arXiv:2201.07311 [cs.CL]

[4] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. (2022).

[5] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert Van De Geijn. 2007. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience* 19, 13 (2007), 1749–1783.

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. 2012. Large scale distributed deep networks. *Advances in neural information processing systems* 25 (2012).

[7] Tim Dettmers. 2015. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561* (2015).

[8] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit Optimizers via Block-wise Quantization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022.* OpenReview.net. https://openreview.net/forum?id=shpkpVXzo3h

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] dgx1 2017. NVIDIA DGX-1. https://www.nvidia.com/en-us/data-center/dgx-1/.

[11] dgx2 2018. NVIDIA DGX-2. https://www.nvidia.com/en-us/data-center/dgx-2/.

[12] Nikoli Dryden, Tim Moon, Sam Ade Jacobs, and Brian Van Essen. Communication Quantization for Data-Parallel Training of Deep Neural Networks. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments* (Salt Lake City, Utah) *(MLHPC '16)*. IEEE Press, 1–8.

[13] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[14] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems* 32 (2019).

[15] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. 2018. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. *ArXiv* abs/1811.06965 (2018).

[16] Infiniband Sharp white paper 2021. NVIDIA InfiniBand Adaptive Routing Technology. https://nvdam.widen.net/s/whmszwfrbt/infiniband-white-paper-adaptive-routing-1846350.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[18] Conglong Li, Ammar Ahmad Awan, Hanlin Tang, Samyam Rajbhandari, and Yuxiong He. 2021. 1-bit LAMB: Communication Efficient Large-Scale Large-Batch Training with LAMB's Convergence Speed. *CoRR* abs/2104.06069 (2021). arXiv:2104.06069 https://arxiv.org/abs/2104.06069

[19] Microsoft. 2020. Turing-NLG: A 17-billion-parameter language model by Microsoft. https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/.

[20] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Granger, Phil Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *ACM Symposium on Operating Systems Principles (SOSP 2019)*.

[21] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*. PMLR, 7937–7947.

[22] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. https://doi.org/10.1145/3458817.3476209

[23] N NVIDIA. 2017. NVIDIA Collective Communications Library (NCCL).

[24] Nvidia V100 datasheet 2017. NVIDIA TESLA V100 GPU ACCELERATOR. https://www.penguinsolutions.com/computing/wp-content/uploads/2019/

[25] NVLink 2017. NVIDIA NVLINK. http://www.nvidia.com/object/nvlink.html.

[26] NVSwitch 2018. NVIDIA NVSWITCH. http://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf.

[27] Quantization - PyTorch documentation 2023. Quantization - PyTorch documentation. https://pytorch.org/docs/stable/quantization.html.

[28] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[29] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[30] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*.

[31] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth annual conference of the international speech communication association*.

[32] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[33] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model. *arXiv preprint arXiv:2201.11990* (2022).

[34] Nikko Ström. 2015. Scalable distributed DNN training using commodity GPU cloud computing. (2015).

[35] Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, and Yuxiong He. 2021. 1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed. *CoRR* abs/2102.02888 (2021). arXiv:2102.02888 https://arxiv.org/abs/2102.02888

[36] DeepSpeed Team and Rangan Majumder. 2020. DeepSpeed: Extreme-scale model training for everyone. https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/.

[37] 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.

[38] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil R. Devanur, and Ion Stoica. 2020. Blink: Fast and Generic Collectives for Distributed ML. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org. https://proceedings.mlsys.org/book/299.pdf

[39] Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. 2019. Reducing BERT Pre-Training Time from 3 Days to 76 Minutes. *CoRR* abs/1904.00962 (2019). arXiv:1904.00962 http://arxiv.org/abs/1904.00962

[40] Zhen Zhang, Shuai Zheng, Yida Wang, Justin Chiu, George Karypis, Trishul Chilimbi, Mu Li, and Xin Jin. 2022. MiCS: Near-linear Scaling for Training Gigantic Model on Public Cloud. arXiv:2205.00119 [cs.DC]

[41] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Christopher De Sa, and Zhiru Zhang. 2019. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 7543–7552. http://proceedings.mlr.press/v97/zhao19c.html