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

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

Abstract

While the Zero Redundancy Optimizer (ZeRO) excels in training large-scale models, it struggles to achieve good throughput in environments with limited bandwidth or small batches where communication becomes a major bottleneck. Inspired by the principles of fine-grained quantization in machine learning algorithms, we designed ZeRO++, an optimizer robust to quantization effects that allows for significant communication volume reduction using low-precision quantization techniques. ZeRO++ composes of three communication volume reduction techniques (low-precision all-gather, data remapping, and low-precision gradient averaging) to significantly reduce the communication volume up to 4x that enables up to 2.16x better throughput at 384 GPU scale. Our results also show ZeRO++ can speedup the RLHF by 3.3x compared to vanilla ZeRO. To verify the convergence of ZeRO++, we test up to 13B model for pretraining with 8/6-bits all gather and up to 30B model for finetuning with 4-bit or 2-bit all gather, and demonstrate on-par accuracy as original ZeRO (aka standard training). As a byproduct, the model trained with ZeRO++ is weight-quantized, which can be directly used for inference without post-training quantization or quantization-aware training.

1 Introduction

The size of deep learning (DL) models has increased from 100 million to over 500+ billion parameters, ranging from BERT [8] to Megatron-Turing NLG [25]. 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 these massive models requires the efficient aggregation of computing power and memory across hundreds or even thousands of GPU devices. There are two popular approaches to alleviate this, namely 3D parallelism [15, 27] and Zero Redundancy Optimizer (ZeRO) [21].

Compared to 3D parallelism, ZeRO is more easy to use without model code refactoring. ZeRO is a memory efficient variation of data parallelism [1, 4] 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 mini-

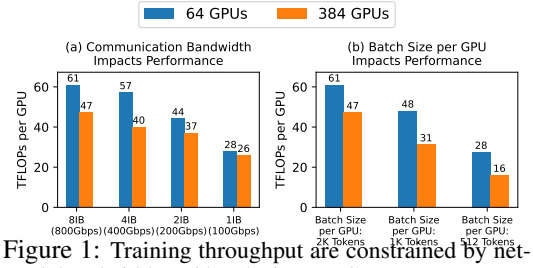

Figure 1: Training throughput are constrained by network bandwidth and batch size per GPU.

*Equal Contribution. Code has been released as a part of https://github.com/microsoft/DeepSpeed, FY is from University of Houston, LY is from University of Nevada-Reno

37th Conference on Neural Information Processing Systems (NeurIPS 2023). MLsys Workshop.

mal communication overhead (50%) compared
to standard data parallel training (2M vs 3M for model size of M) [21], while still achieving excellent throughput scalability [22].

However, the communication overhead of ZeRO can limit throughput in two important scenarios (1) low-bandwidth cluster and (2) low-compute training scenario, e.g., the batch size per GPU is small. We demonstrate these two cases in Figure 1. As can be seen, as the bandwidth of the cluster becomes smaller and/or the training batch size decreases, the training efficiency of ZeRO significantly reduces. To overcome the communication overhead of ZeRO, we present a novel system of communication optimizations collectively called ZeRO++, for which the contribution can be summarized:

Quantized Weight Communication for ZeRO (qwZ). First, in order to reduce parameter communication volume during forward all-gather (which is M for ZeRO), we adopt quantization on weights to shrink down each model parameter from FP16 to lower-precision data type (e.g., 8/6-bits) before communicating. To preserve decent model training precision, we adopt block-based quantization [6, 30], 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. Another great byproduct of qwZ is that it automatically converts the model weight from full-precision (16-bits) to lower precision for efficient inference without any inference quantization methods. Particularly, as compared to the current popular post-training quantization methods [30, 9], qwZ can push the weight-precision to 2-bits for finetuned model without significant accuracy drop.

Hierarchical Weight Partition for ZeRO (hpZ). Second, to reduce the communication overhead of all-gather on weights during backward (which is M for ZeRO), 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 node. 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. hqZ reduces the all-gather for backward from volume M to 0.

Quantized Gradient Communication for ZeRO (qgZ). Third, reducing the communication cost of gradients using reduce-scatter (which is M for ZeRO) is even more challenging. Directly applying quantization to reduce communication volume is infeasible due to the training accuracy drop. To better preserve the accuracy, we propose a novel hierarchical gradient scatter-reduction schedule, where an inter-node reduction gradient reduction is first applied and then a quantized gradient communication (e.g., 8/4-bits) is used resulting in 2/4x communication reduction. To achieve the best outcome, we incorporate pipelining intra-node and inter-node communication and conducting CUDA kernel fusion.

The details of background and deign are given in Appendix A and Appendix B. By incorporating all three components above, we reduce the cross-node communication volume by 4x from $3M$ down to less than $0.75M$ ($<$0.5M for forward, 0 for backward, and 0.25M for gradient reduce-scatter). We extensively test ZeRO++'s system performance, and show (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 across models ranging from 10-138B parameters, (iii) comparing with baseline in 4x higher bandwidth cluster, ZeRO++ achieve better throughput in low-bandwidth setting, (iv) 3.3x better throughput for RLHF training and generation.

To verify the convergence of ZeRO++, we tested it on both pretraining and fine-tuning. Our results show that ZeRO++ can (1) achieve on-par training loss as ZeRO for up to 13B GPT-3 models on pretraining with both 8-bits and 6-bits qwZ, and (2) realize similar (slightly worse) model quality as ZeRO for OPT-30B finetuning with 4-bits (2-bits) qwZ. The 2-bits qwZ trained model can be directly served for inference, which is significantly better than current post-training quantization methods (PTQ) [9] while ZeRO++ even saves training cost and PTQ cost.

2 Evaluation

This section evaluates ZeRO++ in three areas. First, it shows end-to-end throughput scalability and speedup over baseline for standard and RLHF training across different models, model sizes, hardware configurations and cluster settings, demonstrating consistent speedup (up to 3.3x) across the board. Second, it shows convergence properties of ZeRO++ for both pre-training and fine-tuning demonstrating its robustness and tolerance to extreme quantization all the way down to 2-bits.

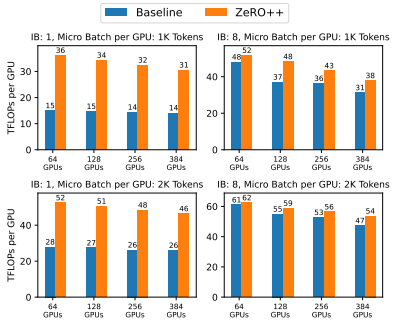

Figure 2: Scalability on up to 384 GPUs of 18B model with different numbers of InfiniBand connections and tokens per GPU.

Table 1: End-to-end speedup of ZeRO++ on V100 and A100 GPUs.

(a) 384 V100 GPUs

Model Size	Tokens per GPU	1 IB Connection Baseline TFLOPs	1 IB Connection ZeRO++ TFLOPs	1 IB Connection Speedup	8 IB Connections Baseline TFLOPs	8 IB Connections ZeRO++ TFLOPs	8 IB Connections Speedup
138B	2K	19.96	37.90	90%	47.55	55.30	16%
138B	1K	11.25	21.81	94%	34.19	44.38	30%
91B	2K	19.99	38.06	90%	47.74	56.26	18%
91B	1K	11.27	21.93	95%	34.49	44.36	29%
49B	2K	20.06	38.08	90%	48.05	56.24	17%
49B	1K	11.27	21.95	95%	34.54	44.46	29%
18B	2K	25.98	46.40	79%	47.31	53.65	13%
18B	1K	14.15	30.57	116%	31.27	37.87	21%

(b) 32 A100 GPUs

Model Size	Tokens per GPU	1 IB Connection Baseline TFLOPs	1 IB Connection ZeRO++ TFLOPs	1 IB Connection Speedup	4 IB Connections Baseline TFLOPs	4 IB Connections ZeRO++ TFLOPs	4 IB Connections Speedup
18B	2K	64.99	111.66	71.8%	134.65	134.94	0.2%
18B	1K	34.52	65.2	88.8%	85.16	93.12	9.3%

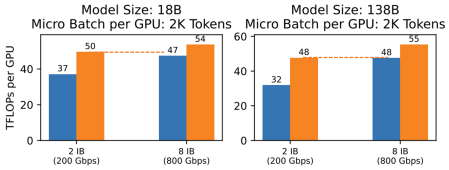

Figure 3: ZeRO++ achieves high bandwidth cluster performance with much lower bandwidth.

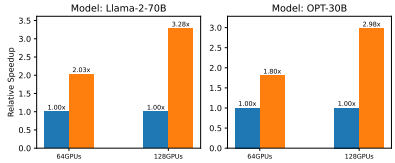

Figure 4: ZeRO++ achieves up to 3.3x better throughput for RLHF training.

Third, it shows ablation studies demonstrating the impact of each component of ZeRO++ and the effectiveness of our kernel optimizations. See Appendix C for detailed hardware and configuration.

2.1 E2E System Evaluations

We evaluate ZeRO++ end-to-end performance and present an ablation study here. A key metric is the percentage of *peak performance*: peak_performance = achieved_TFLOPs/max_TFLOPs. E.g. when we use V100 GPU, its max_TFLOPs is 120 TFLOPs [18] for mixed precision computation. Thus, our reported *peak performance* refers to the percentage number of achieved_TFLOPs/120.

Scalability upto 384 GPUs In Figure 2, 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.

Throughput for different model sizes and GPU architectures Table 1(a) 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%. Table 1(b) illustrates the performance of ZeRO++ on 32 A100 GPUs, demonstrating that ZeRO++ surpasses the baseline even on a smaller scale cluster. Specifically, in low-bandwidth clusters, ZeRO++ exceeds the baseline by 88.8%, and in high-bandwidth clusters, it achieves a 9.3% improvement over the baseline. ZeRO++ exhibits robustness an uniform speedup across varying model and batch sizes, and differing network bandwidths and GPU architectures.

Democratization for large scale training Figure 3 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 a 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++

Table 2: Perplexity after fine-tuning different models with various quantization bits in ZeRO++.

	OPT-1.3B	OPT-13B	LLaMA-30B
FP16 (Baseline)	1.804	1.698	1.490
ZeRO++ 6-bits	1.809	1.705	1.500
ZeRO++ 4-bits	1.830	1.705	1.494
ZeRO++ 2-bits	2.218	1.809	1.544

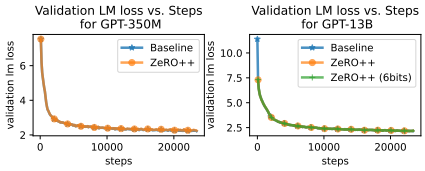

Figure 5: Convergence curves: Pretraining GPT-350M and GPT-13B on the Pile dataset.

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.

Speedup for RLHF Reinforcement Learning from Human Feedback (RLHF) is a unique and commonly employed scenario in LLM training. In Figure 4, we present a comparison between ZeRO++ and the baseline for OPT-30B and LLaMA-2-70B, demonstrating that ZeRO++ can achieve up to 3.3x and 2.97x improved throughput forLLaMA-2-70B and OPT-30B respectively. The throughput is measured at step3 of RLHF training where there are both training and generation. It's noteworthy that the results were obtained with 8 InfiniBand connections on V100 GPUs, and we anticipate an even larger gap in lower bandwidth clusters. These findings underscore the versatility and efficacy of ZeRO++ across diverse training scenarios and various model families.

2.2 Model Convergence Analysis

This section assesses ZeRO++'s impact on the convergence of large models during both pretraining and fine-tuning stages.

Pretraining. We analyze the pretraining of GPT-350M and GPT-13B models on the Pile dataset [2], employing ZeRO++ with non-blocked quantization, ZeRO++ (with blocked quantization), and ZeRO-3 as the baseline. To maintain fairness, all hyperparameters remain consistent between baseline and ZeRO++ trainings. Convergence is measured by the validation LM loss. Figure 5 displays the comprehensive pretraining trace. Training with 8-bit non-blocked quantization diverged initially, rendering no visible data. Conversely, ZeRO++ with 8-bit blocked quantization aligns closely with the baseline, reinforcing our prior analysis that block-based quantization achieves superior quantization accuracy. Furthermore, ZeRO++ convergence remains closely aligned with the baseline even when using 6-bit quantization.

Fine-tuning. We fine-tuned the pre-trained models: OPT-1.3B/-13B [31] and LLaMA-30B [28], with ZeRO++ using FP6, INT4, and INT2 precision for qwZ and INT-8 for qgZ on the high-quality open-source instruction datasets: Dahoas/rm-static, Dahoas/full-hh-rlhf, Dahoas/synthetic-instruct-gptj-pairwise, and yitingxie/rlhf-reward-datasets. We kept all training hyperparameters same across all setups. The evaluation relies on the metric – perplexity (the lower, the better). Table 2 reveals the robustness of ZeRO++, as well as its ability to create low-precision models during fine-tuning without requiring post-quantization. Notice, the validation perplexity of ZeRO++ varies only slightly from the baseline, with a mere 0.27%-0.41% deviation, even when quantized to 4-bits.

3 Conclusion

This paper presents ZeRO++, an efficient collective communication solution for large model training using ZeRO stage-3. We optimize both model weights and gradients communication in the 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 a novel all-to-all based gradient quantization and reduction scheme.

By incorporating all the three optimizations above, we improve system scalability and throughput for pre-training, fine-tuning and RLHF training with speedup of up to 3.3x. Furthermore, ZeRO++ can automatically quantize the parameters to ultra-low precision making the resulting model inference ready without post-training quantization. We envision ZeRO++ as the next generation of easy-to-use framework for training large models at scale.

References

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

[2] Stella Biderman, Kieran Bicheno, and Leo Gao. Datasheet for the pile, 2022.

[3] 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. GPT-NeoX-20B: An open-source autoregressive language model. 2022.

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

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

[6] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 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, 2022.

[7] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

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

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[10] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *ArXiv*, abs/1811.06965, 2018.

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

[12] Conglong Li, Ammar Ahmad Awan, Hanlin Tang, Samyam Rajbhandari, and Yuxiong He. 1-bit LAMB: communication efficient large-scale large-batch training with lamb's convergence speed. *CoRR*, abs/2104.06069, 2021.

[13] Microsoft. 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/`, 2020.

[14] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Granger, Phil Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *ACM Symposium on Operating Systems Principles (SOSP 2019)*, October 2019.

[15] 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. 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*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[16] NVIDIA DGX-2. `https://www.nvidia.com/en-us/data-center/dgx-2/`, 2018.

[17] Nvidia infiniband adaptive routing technology. https://nvdam.widen.net/s/whmszwfrbt/infiniband-white-paper-adaptive-routing-1846350, 2021.

[18] Nvidia tesla v100 gpu accelerator. https://www.penguinsolutions.com/computing/wp-content/uploads/2019/03/penguin-computing-tesla-v100-ds.pdf, 2017.

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

[20] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

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

[22] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 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, 2021.

[23] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 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*, 2014.

[24] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Q-BERT: hessian based ultra low precision quantization of BERT. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8815–8821. AAAI Press, 2020.

[25] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[26] Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, and Yuxiong He. 1-bit adam: Communication efficient large-scale training with adam's convergence speed. *CoRR*, abs/2102.02888, 2021.

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

[28] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[29] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil R. Devanur, and Ion Stoica. Blink: Fast and generic collectives for distributed ML. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020.

[30] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183, 2022.

[31] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[32] Zhen Zhang, Shuai Zheng, Yida Wang, Justin Chiu, George Karypis, Trishul Chilimbi, Mu Li, and Xin Jin. Mics: Near-linear scaling for training gigantic model on public cloud, 2022.

[33] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Christopher De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7543–7552. PMLR, 2019.

Appendix

A Background and Related Work

This section only describes the core background needed for understanding the techniques discussed in this paper. A broader discussion of the related work is provided in Appendix D .

A.1 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, and ZeRO stage 3 (ZeRO-3) splits all three components of model states as parameters, gradients, and optimizer states.

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
```

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 lazily 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 the optimizer step (line 9).

In summary, to minimize the on-device memory footprint using ZeRO-3, at each iteration there are three collective communication operations: two all-gather on weights and one reduce-scatter on gradients.

A.2 Communication Reduction Techniques

Quantization is often used to reduce memory footprint, and data movement volume by using low precision to represent data [5, 6]. However, the loss of information from representing high precision data with lower precision often comes with accuracy degradation. Related work aims to enhance quantization accuracy by addressing the challenges associated with differences in number ranges and granularity between high and low precision data. Some related work [33] 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. [6] propose to use block based quantization on optimizer states to improve the quantization accuracy. However, it requires modifications to the model structure, thereby limiting its usability.

Gradient Compression techniques, including 1-bit SGD, 1-bit Adam, and 1-bit Lamb, have been developed to optimize gradient communication in distributed training by utilizing minimal bit representation [23, 26, 12]. However, direct application of these methods to ZeRO-3 is infeasible as they assume a full view of optimizer states (OS) across GPUs, which is not the case in ZeRO-3.

ZeRO Communication Reduction. Recent optimizations on ZeRO-3, like MiCS [32], divide the GPU cluster into sub-groups and leverage high bandwidth intra-node interconnect, or hierarchical communication to minimize communication volume. hpZ in ZeRO++ adopts a similar approach yet it performs only secondary partitioning 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.

B 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). We further discuss the end-to-end impact of these optimizations to reduce to total communication volume of ZeRO in Appendix E.4.

B.1 Quantized Weight Communication for ZeRO (qwZ)

As discussed in Section A.1, 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 lower precision right during the all-gather, and dequantizes them back to FP16 on the receiver side, and then conducts layer computation.

While this reduces the communication volume of the all-gather by more than 2x, doing so naively results in two major issues: i) the lowering of precision results in significant accuracy degradation during training as discussed in Section A.2 , and ii) the quantization and dequantization overhead negates any throughput gain from communication volume reduction.

To improve quantization accuracy we use blocksize based quantization [30, 24] (further details in Appendix E.1). To mitigate the quantization and dequantization overhead, we develop custom optimized implementation of qwZ (further details in Appendix F)

Additional Benefits of qwZ

i) *Automatic Quantization* qwZ automates parameter quantization during training, allowing higher compression (to 2 bits) with minimal accuracy loss, thereby obviating the need for post training quantization [9, 30] (see Section 2 for more details),

ii) *Memory footprint reduction* For untrainable/frozen weights (e.g., in LoRA or multimodal training), qwZ alleviates the necessity of persistent FP16 storage, which reduces memory footprint. This reduction can allow for a) training with larger batch sizes for better throughput, b) inference or training with fewer resources without running out of memory, and c) speed up inference by reducing the amount of parameter data that needs to be read from memory.

In fact, given the automatic quantization and memory footprint reduction, qwZ has a similar effect for fine-tuning as QLoRA [7], in terms of weight quantization and memory requirement reduction, but in addition ZeRO++ with qwZ can support significantly larger model sizes than QLoRA due to the weight partitioning from ZeRO.

B.2 Hierarchical Partitioning for ZeRO (hpZ)

With 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 6), 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.

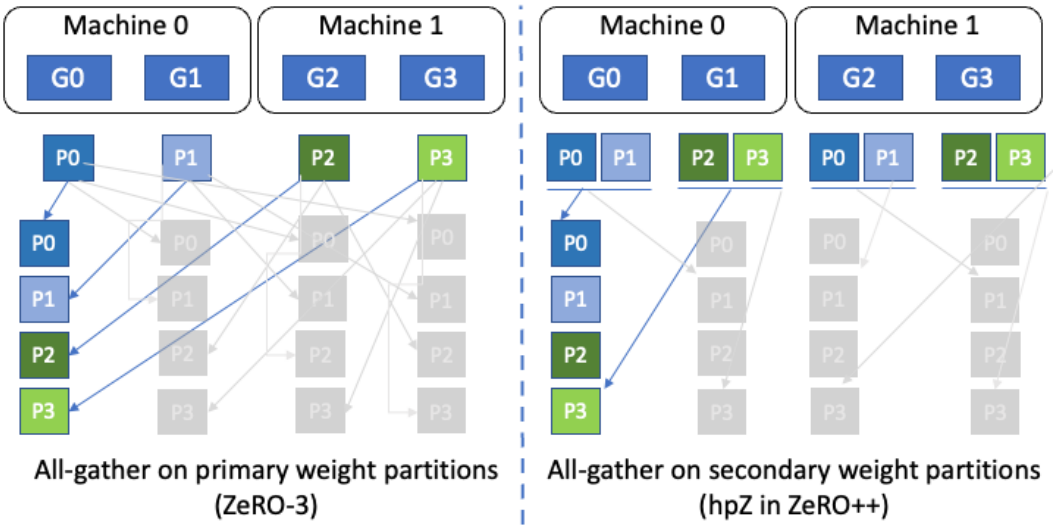

Figure 6: hpZ removes cross node traffic in backward all-gather by holding secondary weight partitions in on-device memory.

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 have been used during the forward pass, these weights are then 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. When the secondary partitioning is set to be inside 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, and the primary copy of the fp16 parameters 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 can flexibly 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 the memory-communication trade-off of hpZ discussed in Appendix E.2.

B.3 Quantized Gradients Communication for ZeRO (qgZ)

qgZ is a novel quantized reduce-scatter algorithm based on all-to-all collectives that enables a 4x communication volume reduction of gradient reduce-scatter by replacing FP16 with INT4. qgZ has three components: 1) all-to-all-based quantized gradient reduce-scatter, 2) reducing communication volume with hierarchical collectives (details in Appendix E.3), 3) tensor slice reordering for correct gradient placement.

B.3.1 All-to-all based implementation

A naive way to quantized reduce-scatter without precision loss is to apply quantization and dequantization to a ring-based reduce-scatter directly as shown on the left of Figure 7. We inject quantization and dequantization on each GPU. Once a GPU receives gradients from its predecessor, we dequantize it to full precision and conduct a local reduction. After that we quantize local reduction output and pass it to its successor. To complete the whole reduce-scatter, the number of sequential quantization and dequantization kernels is equal to the number of GPUs in use. Thus, applying quantization 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 collectives like Blink[29] could reduce the number of sequential kernels from n to log(n), the long latency and low precision issue is not completely resolved.

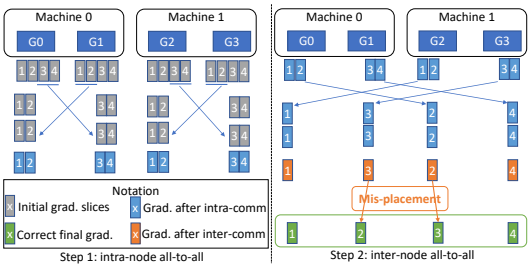

Figure 8: Gradient partition misplacement when applying hierarchical all-to-all in qgZ.

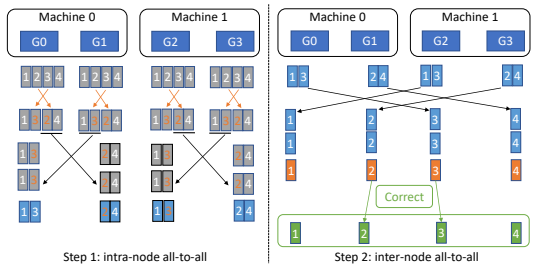

Figure 9: Tensor slices reordering to correct gradient misplacement in qgZ.

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 7, 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 kernels from the number of GPUs to 1. We further reduce cross node communication volume by incorporating hierarchical collectives named *2-hop all-to-all*, which is detailed in Appendix E.3.

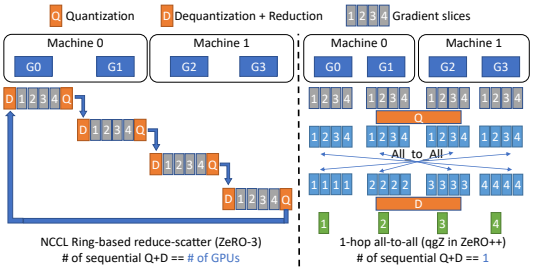

Figure 7: Comparison between ZeRO-3 ring-based reduce-scatter and qgZ 1-hop all-to-all.

B.3.2 Tensor slice reordering for correct data placement

With our *2-hop all-to-all*, 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.

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 partitions 1,2 whereas G1 holds gradient partitions 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. In Figure 9, before intra-node all-to-all, we first swap 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 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 holds X*Y gradient slices initially. 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 Eq. 1 and 2, we can map each original tensor slice position (i.e., Eq. 1) to new tensor slice position (i.e., Eq. 2) on each GPU to correct final gradient misplacement issue.

B.4 Optimized Implementation

To optimize ZeRO++, we implemented key optimizations: i) overlapping compute streams and, ii) optimizing CUDA kernels. Overlapping involves concurrent execution of quantization compu-

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**

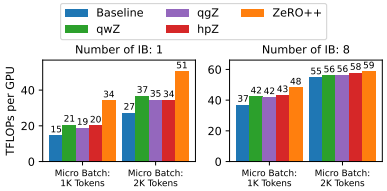

Figure 10: Throughput of 18B models on128 GPUs with ZeRO++, qwZ, qgZ, hpZ, and baseline on various InfiniBand connections.

tation and communication during weight all-gathering, leveraging asynchronous quantization and execution order tracking. The CUDA kernels were optimized for quantization, dequantization, and tensor slice reordering to fully utilize device memory bandwidth and minimize memory traffic by 9x in qgZ. Additionally, a hierarchical approach in gradient communication and a generalized tensor slice reordering scheme are introduced to minimize latency and address the needs of different pipeline stages. These optimizations allow ZeRO++ to efficiently leverage 4x communication volume reduction, enhancing its throughput. Please refer to the Appendix F for implementation details and communication analysis.

C Additional Details of Evaluations

C.1 Methodology

Hardware: 24 nodes where each with 16 V100 SXM3 32 GB GPUs [16] . The nodes are connected by InfiniBand (IB) with [17] support, 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 bandwidths 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 [19] and non-fused kernels as baselines for our ablation study.

Model Configurations: We use transformer models for evaluation including GPT, OPT, andLLaMA for various model sizes. Given Megatron-Turing-NLG [25] 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.

C.2 Throughput Breakdown and Analysis

Impact of Individual Techniques. In Figure 10, 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 with a larger aggregated speedup.

Impact of Kernel Optimizations. We evaluate our optimized kernels on ZeRO++ throughput using an 18B model running on 64 GPUs.

Quantization Kernel: As shown in Table 3, compared with the baseline that uses [19], our optimized quantization kernels can achieve up to 1.67x speedup in terms of end-to-end throughput. Also, the baseline implementation suffers performance degradation as the group number increases which means the throughput gap will be larger when used with larger models.

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

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

Kernel Fusion: As described in Appendix F.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.

C.3 Comparing hpZ with MICS

As previously discussed in Appendix A, closely related to hierarchical weight partition for ZeRO (hpZ) is $MiCS$[32]. The key difference between the two methods is what data are replicated in the secondary group; only model weights are replicated in hpZ, while entire model states are replicated in $MiCS$. Table 4 shows the 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 whereas $MiCS$ reports out-of-memory (OOM) at this scale.

D Further Background and Related Work

D.1 Data, Tensor 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. [4, 15, 14, 10] 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 [11, 14, 10] 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 [15] splits each layer across multiple GPUs, where each GPU works on a different part of the layer for the same input.

3D parallelism [25, 27] refers to combination of Data Parallelism , Pipeline Parallelism , and Tensor Parallelism [4, 15, 14, 10], and is capable of achieving excellent throughput and scalability, and has been used to train a wide range of large language models [13, 15, 20, 3]. 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 [25].

E Design Details

E.1 Block-size Based Quantization

As illustrated in Figure 11, 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 11(a). Figure 11(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 Section 2.

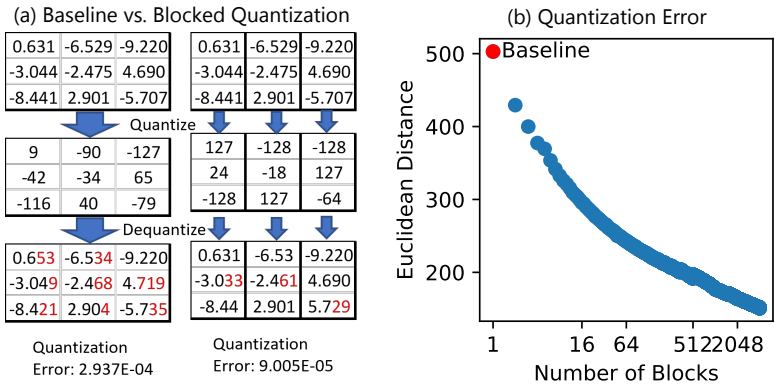

Figure 11: Illustration & example of block based quantization vs. baseline

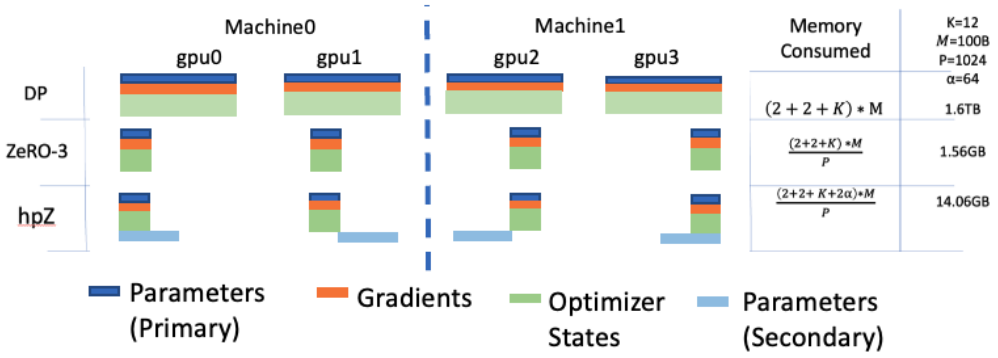

Figure 12: Per-device memory consumption analysis of standard data parallel (DP), ZeRO stage 3 (ZeRO-3) and proposed hierarchical partitioning of ZeRO parameters (hpZ). K denotes the memory multiplier of optimizer states, M represents the number of trainable parameters, P is the data parallel group size 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).

E.2 Memory Usage Analysis of hpZ

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

Figure 12 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 12, 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 supporting model training with hundreds of billions of parameters.

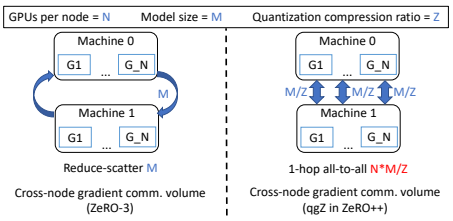

Figure 13: Compare the communication volume of ZeRO-3 reduce-scatter with the qgZ 1-hop all-to-all.

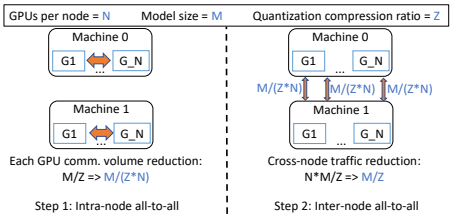

Figure 14: qgZ apply hierarchy all-to-all to reduce cross node traffic.

E.3 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 13.

Given that intra-machine often have high bandwidth interconnects (e.g., NVLink, NVSwitch), cross-machine communication links are often the bottleneck. Given this, we analysis cross node communication volumes and ignore intra-node communication volumes.

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 14. 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.

E.4 ZeRO++ Communication Volume Analysis

Comm. Volume	forward all-gather	backward all-gather	backward reduce-scatter
ZeRO-3	M	M	M
ZeRO++	0.5M	0	0.25M

Table 5: Communication volume comparison between ZeRO-3 and ZeRO++.

Table 5 illustrates theoretical communication volume comparison between ZeRO-3 and ZeRO++. We assume the model size of M. As described in Section A, 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

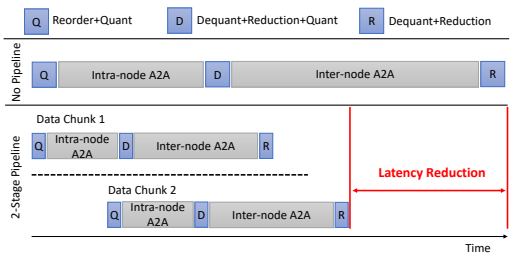

Figure 15: Pipelining and overlapping intra-node communication with inter-node communication in qgZ.

focus on cross-node traffic only. For all-gather in the forward pass, by incorporating our quantized weights communication, we reduce 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.

F 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.

F.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.

F.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 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.

F.1.2 Hierarchical Collectives for Gradient Communication

As discussed in Section E.3, 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 15, 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 15. By overlapping intra-node and inter-node communication, the end-to-end latency of gradient communication is significantly reduced.

Algorithm 2: Generalized tensor slice reordering (qgZ)

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

$totalDevices \leftarrow nodeSize * nodes$;
$stageID \leftarrow partitionID \,\%\, stages$;
$chunkID \leftarrow \frac{partitionID}{stages}$;
$pipelineOffset \leftarrow stageID * totalDevices$;
$chunkOffset \leftarrow \frac{chunkID}{nodeSize}$;
$chunkBase \leftarrow (chunkID \,\%\, nodeSize) * nodes$;
Return: $pipelineOffset + chunkBase + chunkOffset$;

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.

F.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 requirements 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.