Enhancing Stability for Large Models Training in Constrained Bandwidth Networks

Yun Dai [1] **Tejas Dharamsi** [1] **Byron Hsu** [2] **Tao Song** [1] **Hamed Firooz** [1]

Abstract

Training extremely large language models with billions of parameters is a computationally intensive task that pushes the limits of current data-parallel training systems. While techniques like ZeRO++ (Wang et al., 2024) have enabled efficient distributed training of such giant models on inexpensive low-bandwidth clusters, they can suffer from convergence issues due to potential race conditions in the hierarchical partitioning (hpZ) scheme employed to reduce cross-machine communication. In this work, we first show how these race conditions cause instability when training models with billions of parameters. We then propose a modification to the partitioning algorithm that addresses these convergence challenges while maintaining competitive training efficiency. Empirical evaluation on training the multi-billion parameters Falcon Models and LLama-2 models demonstrates the updated algorithm's ability to achieve reliable convergence on these massive models, where stock ZeRO++ *hpZ* fails to converge. The updated algorithm enables robust training of larger models with 98% throughput and model training speed improvement without sacrificing the quality of convergence.

1. Introduction

The ambition to create more expansive and capable AI models has catalyzed a continual push to achieve greater scale in machine learning model training frameworks. Transformer-based model architectures like GPT (Achiam et al., 2023), Falcon (Almazrouei et al., 2023), Mistral (Jiang et al., 2023) LLaMA (Touvron et al., 2023) and many other have surpassed billions of parameters, enabling superior performance on a wide range of language understanding and generation tasks. The remarkable capabilities unlocked by these massive models have spurred surging interest from both industry and academia to explore the limits of scale for large AI models. However, the immense computational requirements for training these giant neural networks stretch the limits of modern hardware and distributed training frameworks. Crucially, the pursuit of ever-larger models risks exacerbating disparities, as the specialized acceleration hardware required remains cost-prohibitive and inaccessible for many. Democratizing access to train massive models efficiently on modest, commodity hardware is vital to ensure equitable AI development globally.

Data parallelism, where a model's parameters are partitioned across multiple accelerator devices (e.g. CPUs, GPUs, TPUs) during training, has been a key enabling technique for training large deep neural network models. Classic data-parallel implementations like BytePS (Jiang et al., 2020), Horovod (Sergeev & Balso, 2018) and PyTorch DDP (Li et al., 2020) rely on replicating the full model across all devices, leading to substantial memory overheads that impose a hard limit on maximum model size.

Megatron-LM (Shoeybi et al., 2020) from NVIDIA attempted to scale by leveraging model parallelism in addition to data parallelism. However, this approach required specialized high-bandwidth interconnects like NVLink and InfiniBand that are not widely accessible to most developers and researchers.

In contrast, methods like ZeRO (Rajbhandari et al., 2020) took a different approach - eliminating redundant parameter copies across data-parallel devices through intelligent parameter partitioning schemes. ZeRO-Offload (Ren et al., 2021) further optimized memory usage by offloading activations and optimizer states to CPU memory during the respective forward and backward passes. While more memory-efficient, these ZeRO techniques still relied heavily on frequent inter-node communication for collective operations.

A key bottleneck faced by both Megatron-LM and ZeRO was the relatively low inter-node communication bandwidth compared to the intra-node bandwidth within a single multi-

[1]Foundational AI Technologies, LinkedIn Inc. [2]AI Platforms, LinkedIn Inc. Correspondence to: Yun Dai <yudai@linkedin.com>, Tejas Dharamsi <tdharamsi@linkedin.com>, Byron Hsu <byhsu@linkedin.com>, Tao Song <tsong@linkedin.com>, Hamed Firooz <hfirooz@linkedin.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

accelerator system. As these approaches scaled out to larger node counts for training trillion-parameter models, the inter-node communication overheads became pronounced. This communication bottleneck imposed severe scaling limits when using commodity multi-node clusters without ultra high-speed networking fabrics (Liang et al., 2024).

To enable large scale training in network constraint environment, ZeRO++ algorithm (Wang et al., 2024) introduced a set of efficient parallelization strategy leveraging Quantized Weight Communication for ZeRO (*qwZ*), Quantized Gradient Communication for ZeRO (*qgZ*), Hierarchical Weight Partition for ZeRO (*hpZ*) hierarchical partitioning to minimize cross device communication volume. ZeRO++ has been pivotal, enabling training of unprecedentedly massive models like GPT-3 (Brown et al., 2020) with 175B parameters and Turing NLG 530B (Smith et al., 2022) on relatively small GPU clusters.

Despite its scalability claims, the hierarchical partitioning algorithm in the original ZeRO++ implementation can encounter convergence issues on extremely large models like Falcon (40B) and Meta's LLaMA (70B) during full-parameter fine-tuning, depending on the hardware and fabric setup. These issues lead to unreliable and unstable training runs. The convergence failures stem from a subtle race condition between the asynchronous parameter partitioning operation and collective communication primitives, such as `AllGather`, in ZeRO++'s algorithm. Incorrect ordering can lead to corrupted parameter values being communicated across devices, causing training instability and divergence. Addressing these convergence pitfalls is crucial to fully unleash the potential of giant language model training in a reliable and robust manner over commodity hardware.

In this work, we perform an in-depth analysis of the root causes behind ZeRO++ *hpZ*'s convergence failures on publicly available large language models like Falcon and Llama. We identify the key synchronization bugs in ZeRO++'s hierarchical partitioning scheme that trigger these failures. Based on our findings, we propose a simple yet effective modification to ZeRO++ that introduces explicit CUDA synchronization points to ensure all parameter partitioning completes correctly before any collective communication over the partitioned data. Updated algorithm restores reliable convergence for training giant transformer models using ZeRO++ without impacting its highly coveted computational efficiency and scalability advantages.

2. Background

As we push to hundreds of billions parameters models, the memory becomes a burden for training. The ZeRO3 algorithm shards model parameters, gradients, and optimizer states to significantly reduce the memory footprint. However, this reduction comes at the cost of increased communication overhead. Specifically, the algorithm requires `AllGather` operations on the weights during both the forward pass to compute activations and the backward pass to compute gradients, followed by reduce-scatter operations to distribute the gradients across accelerators.

This results in a high dependence on the communication bandwidth within the cluster. Recent research (Liang et al., 2024) (Li et al., 2019) (Ren et al., 2019) has shown that the latency for inter-node GPU communication is typically higher compared to intra-node GPU communication. This can be attributed to the differences in interconnect technologies such as NVLink NVSwitch (NVIDIA, 2024b) for intra-node and Infiniband (NVIDIA, 2024a) for inter-node communication. As a result, the training process often encounters bottlenecks due to slower inter-node communication, making the inter-node network more likely to become the system bottleneck. Improving inter-node network speed can lead to significant performance gains for multi-GPU applications.

To address these bottlenecks, several approaches have been proposed to maximize intra-node communication while minimizing inter-node communication. For example, PyTorch's hybrid shard FSDP (Zhao et al., 2023) performs ZeRO3 only within a node and uses `AllReduce` to synchronize gradients across nodes, similar to the Distributed Data Parallel approach. Another approach, MiCS (Zhang et al., 2022), creates subgroups of GPUs with high intra-group bandwidth, limiting `AllGather` operations to within these subgroups and using AllReduce for inter-group communication to minimize the inter-subgroup communication overhead.

ZeRO++ (Wang et al., 2024) extends these ideas with the introduction of hierarchical partitioning *hpZ*, quantized gradient *qgZ*, and quantized weight *qwZ* schemes. The *hpZ* scheme ensures a full copy of the model within a node and uses `AllReduce` to synchronize gradients across nodes, effectively reducing inter-node communication. The *qgZ* scheme quantizes gradients before `ReduceScatter` operations, and the qwZ scheme quantizes weights before `AllGather` operations, both of which further optimize communication efficiency.

3. Algorithm

ZeRO++ *hpZ* reduces communication overhead by eliminating cross-node `AllGather` communication during the backward pass, with an extra cost of memory. After the forward pass of a layer is done, instead of re-spreading its full weights across all GPUs, ZeRO++ *hpZ* partitions the weights into a secondary copy which is replicated on each node. `AllGather` in the backward pass thus operates on the secondary copy and will only involve intra-node com-

munication, which has multiple factors higher bandwidth than inter-node. Figure 1 provides an illustration of a full training step with *qgZ* and *hpZ* enabled.

Algorithm 1 ZeRO++ *hpZ* with prefetch

Require:
$worldSize : P$
$secondaryWorldSize : P'$
$model : \mathcal{M}\& = \{L_1, L_2, \ldots, L_N\}$

function PREFETCHALLGATHER
  for $L_k \in$ next modules to be executed **do**
    if is_forward(L_k) **then**
      Enqueue AllGather(L_k, P)
    else
      repeat
        wait
      until MemcpyD2D on $L_{k,second}$ finishes
      Enqueue AllGather(L_k, P')
    end if
  end for
end function
while model not converged **do**
  ▷ Forward pass
  for $i = 1, 2, ..., N$ **do**
    Ensure AllGather(L_i, P) finished
    PrefetchAllGather()
    L_i.forward()
    $L_{i,second} \leftarrow empty(\frac{|L_i|}{P'})$
    Copy to $L_{i,second}$ ▷ Async MemcpyD2D
  end for

  ▷ Backward pass
  for $i = N, N-1, ..., 1$ **do**
    Ensure AllGather(L_i, P') finished
    PrefetchAllGather()
    L_i.backward()
    repartition(P)
    ▷ Replaced with INT4 AllToAll if with qgZ
    ReduceScatter(∇L_i, P)
  end for
  optimizer.step()
end while

During partition, the secondary copy tensor gets allocated as `torch.empty` of the following shape

$$\|L_{i,second}\| \leftarrow \frac{N}{secondaryWorldSize} \quad (1)$$

where N is the total number of elements in the weights and $secondaryWorldSize$ is typically equal to the number of GPUs per node. The weights corresponding to the local rank then get copied from the full parameter tensor to the secondary copy.

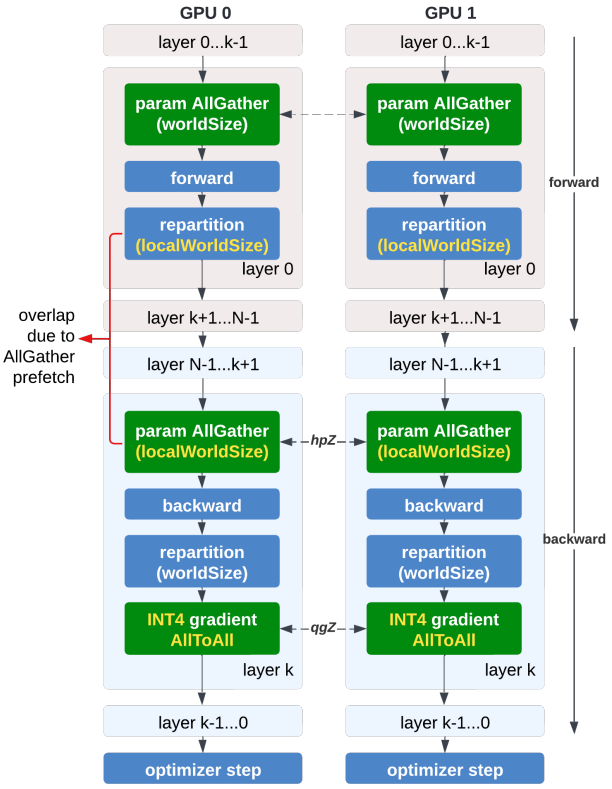

Figure 1. An end-to-end training step on a model with N layers with ZeRO++ qgZ and hpZ. Forward and backward pass on layer k is expanded. With ZeRO3 prefetch, the consequent `AllGather` kernel for the backward pass can be immediately enqueued while post-forward repartition is still in progress.

However, since the Memcpy is from and to tensors both allocated on GPU, hence an asynchronous D2D (device-to-device) copy, there is no guarantee that the secondary copy is settled when the following `AllGather` kernel on it is launched.

With prefetch in ZeRO, which enqueues `AllGather` kernels for following modules beforehand instead of as late as when the backward pass actually happens, a race condition may happen: at the time when a module is still being partitioned into the secondary copy, the `AllGather` kernel for the backward pass on it can be immediately enqueued and launched. This results in `AllGather` aggregating on arbitrarily initialized tensor values, leading to model instability during training and often results in Not-a-Number (NaN) in aggregated parameter values and hence the observed loss.

The race condition is mainly the result of full asynchronization between two operators, Memcpy and `AllGather`. To avoid such a race condition, we add a CUDA synchronization operator between D2D Memcpy and `AllGather`.

The sync operator first makes sure that the Memcpy is finished and upon success of Memcpy D2D, it enqueues the `AllGather` and therefore removes the race condition between the two. The modified algorithm is illustrated in Algorithm 1.

In next section, we show that due to the asynchronous property of Memcopy, there is no guarantee that the secondary copy is settled before the `AllGather`, and as a result, the model performance is not predictable. This is particularly evident for large models trained on commodity bandwidth-limited networks.

4. Experimentation

4.1. Experimentation setup

All experiments are conducted on NVIDIA-A100 GPUs. Eight GPUs compose one GPU node, where GPUs within a node are connected using NVIDIA NVLINK with 600 GB/s bandwidth. To simulate commodity low bandwidth connections, we use 1× 12.5Gbps Ethernet NIC per node.

We leverage two families of off-the-shelf publicly available large language models for our experiments with different sizes: 1) Llama-2 (Touvron et al., 2023), 2) Falcon (Almazrouei et al., 2023). We perform full parameter fine-tuning using the MMLU dataset (Hendrycks et al., 2020) on these models.

Here we define training as unstable if the loss during training diverges to NaN or the training loss does not decrease with the same hyperparameters. We measure the throughput of the training by determining how many tokens are processed in one second given a full GPU node, a.k.a tokens/s/node.

4.2. Divergence Analysis

To demonstrate model training instability, we train various off-the-shelf LLMs with and without stock ZeRO++ *hpZ* and compare the stability with modified *hpZ* outlined in Algorithm 1. As demonstrated in Table 1, for many publicly available large models, the training is unstable with the original ZeRO++ hierarchical partitioning.

Model	without *hpZ*	with *hpZ*	modified *hpZ*
Llama-2-7B	✓	×	✓
Llama-2-13B	✓	×	✓
Llama-2-70B	✓	×	✓
Falcon-40B	✓	×	✓

Table 1. Stock ZeRO++ causes instability in model training due to the race condition between AllGather and Memcopy.

The modified algorithm adds a synchronization operation to guarantee the healthiness of `AllGather`. This impacts the throughput of model training. To measure such impact, we fine-tune multiple off-the-shelf models on the MMLU dataset and measure the throughput of the training.

Model	*qgZ* only	*qgZ* + *hpZ*	*qgZ* + modified *hpZ*
Llama-2-7B	2880	5065.4	5013 (+74%)
Llama-2-13B	2101	2510.5	2521.4 (+20%)
Llama-2-70B	358	488.1	456.9 (+27%)
Falcon-40B	444.5	890	881.9 (+98%)

Table 2. Modified *hpZ* has a small impact on training throughput compared to stock ZeRO++. *qgZ* is enabled for all runs as control. The numbers in paranthesis shows the speed up as compares to baseline of *qgZ* only

As shown in Table 2, the async operation between `AllGather` and memory copy in ZeRO++ sometimes results in lower training throughput compared to stock *hpZ*, but as explained in Table 1, it comes with instability and divergence costs. The modified hierarchical partitioning algorithms have up to 98% higher throughput compared to when *hpZ* is not enabled while keeping the training stability unchanged.

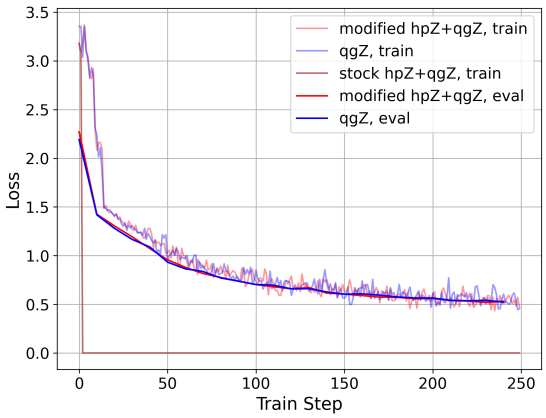

Figure 2. Validation loss convergence per optimization step without *hpZ* and with modified *hpZ* from Algorithm 1, tested on Llama-2-7b for MMLU dataset. Green curve demonstrates convergence issue in training without fix.

Despite having better training throughput, the modified hpZ has no impact on model optimization performance and the training is stable. Figure 2 shows the validation loss during training for MMLU without *hpZ* and with modified *hpZ* for Llama-2-7B. As one can see, there is almost no difference in validation loss convergence per optimization step while from Table 2 the modified *hpZ* converges faster in wall-clock time.

5. Conclusion

In this work we identified and addressed a key convergence issue that affects the training of large language models using the ZeRO++ algorithm on commodity hardware with limited network bandwidth. We analyzed the root cause of the instability, which stems from a race condition between the asynchronous parameter partitioning and collective communication operations in ZeRO++'s hierarchical partitioning scheme. We resolve this bottleneck by introducing explicit CUDA synchronization. This ensures parameter partitioning completes correctly before any collective communication over the partitioned data occurs. Our empirical evaluation demonstrated that the updated algorithm restores reliable convergence when training giant transformer models like the 40B parameter Falcon and 70B parameter LLama-2 on the challenging MMLU dataset.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., Debbah, M., Étienne Goffinet, Hesslow, D., Launay, J., Malartic, Q., Mazzotta, D., Noune, B., Pannier, B., and Penedo, G. The falcon series of open language models, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Jiang, Y., Zhu, Y., Lan, C., Yi, B., Cui, Y., and Guo, C. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 463–479. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/jiang.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Li, A., Song, S. L., Chen, J., Li, J., Liu, X., Tallent, N., and Barker, K. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect, 2019.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., and Chintala, S. Pytorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, aug 2020. ISSN 2150-8097. doi: 10.14778/3415478.3415530. URL https://doi.org/10.14778/3415478.3415530.

Liang, F., Zhang, Z., Lu, H., Leung, V. C. M., Guo, Y., and Hu, X. Communication-efficient large-scale distributed deep learning: A comprehensive survey, 2024.

NVIDIA. Infiniband switching. https://www.nvidia.com/en-us/networking/infiniband-switching/, 2024a.

NVIDIA. Nvlink high-speed interconnect. https://www.nvidia.com/en-us/data-center/nvlink/, 2024b.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021.

Ren, Y., Yoo, S., and Hoisie, A. Performance analysis of deep learning workloads on leading-edge systems, 2019.

Sergeev, A. and Balso, M. D. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

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

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wang, G., Qin, H., Jacobs, S. A., Wu, X., Holmes, C., Yao, Z., Rajbhandari, S., Ruwase, O., Yan, F., Yang, L., and He, Y. ZeRO++: Extremely efficient collective communication for large model training. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=gx2BT0a9MQ.

Zhang, Z., Zheng, S., Wang, Y., Chiu, J., Karypis, G., Chilimbi, T., Li, M., and Jin, X. Mics: Near-linear scaling for training gigantic model on public cloud, 2022.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., Desmaison, A., Balioglu, C., Damania, P., Nguyen, B., Chauhan, G., Hao, Y., Mathews, A., and Li, S. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023.