Summary of “PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel”

Jeff Brill (jhbrill), Max Liu (yinghal), Melina O'Dell (melodell)

Problem and Motivation

As the size of LLMs increases, the training process becomes more difficult because of memory and network communication limitations. Data parallelism, a naive method for parallelizing LLM training, can be effective for smaller models, but is not feasible as the model size grows. Other methods of parallelism exist (like pipeline and tensor parallelism), and motivation for this work is to discover and evaluate further methods of parallelism. This paper looks at PyTorch Fully Sharded Data Parallel (FSDP) [1] to increase the efficiency of LLM training.

Related Works

There is previous work related to data parallelism, but they suffer from memory constraints, because they need all parameters, gradients, and optimizer states to be stored in the memory of one GPU [3]. Additionally, the Zero Redundancy Optimizer (ZeRO) paper proposes methods of memory conservation by reducing redundancy in data-parallel training [4]. It is mostly equivalent to FSDP.

Solution Overview

The paper introduces FSDP, a data parallelism method for scaling large models that can't fit on a single GPU by sharding dense parameters. FSDP uses deferred initialization to allocate model parameters on a simulated device before loading them onto the GPU. It initializes one FSDP unit on the CPU, shards it, and moves to the next. Model parameters and optimizer states are loaded and freed from GPUs as needed during forward or backward stages, allowing model creation without tensor storage while ensuring correct parameter initialization.

FSDP uses various sharding techniques, with the sharding factor indicating the number of ranks over which parameters are sharded. Full sharding, where the sharding factor equals the number of devices, reduces memory use but increases communication overhead. Hybrid sharding, where the factor is between 1 and the number of devices, combines sharding and replication. Parameters are organized into a large flat structure distributed evenly across ranks for efficient communication. In hybrid sharding, a single reduce-scatter occurs within the sharded groups, followed by an all-reduce within each replicated group to sync gradients, avoiding additional all-gathering. This method leverages data center locality and communication bandwidth for faster training and reduced GPU memory overhead.

FSDP's communication protocol has two parts. In the forward stage, the same input is applied to the sharded parameters. The FSDP unit is sharded, using all-gather to produce tensors as outputs, and the forward pass is computed before freeing peer shards to save memory. The backward stage runs all-gather again for the forward output, as each node has the same weight

but different gradients. Backward computation is then performed, followed by reduce-scatter to reduce and shard gradients, leaving each rank with only a shard of parameters and gradients. The performance of FSDP and DDP are similar when evaluating small size models, but DDP encounters an out-of-memory error for larger models. FSDP is able to accommodate ~5x larger models.

Limitations

Despite its advantages, FSDP faces some drawbacks, particularly in its handling of shared parameters and ensuring mathematical equivalence to local training, specifically in regards to optimizer computations. Sharded optimizer computations usually sacrifice communication bandwidth for memory, and network communication comes with overhead. To avoid a FSDP unit to use a shared parameter that has been resharded by a proceeding FSDP unit, the recommendation is that the shared parameters should be derived from the lowest-common-ancestor unit to ensure they are not unsharded by any preceding units. This need for careful construction of FSDP units can add complexity to model integration and training.

Future Research Directions

For FSDP, further research could explore the combination of FSDP with other parallelism paradigms like tensor and pipeline parallelism, to create more comprehensive and scalable distributed training frameworks. Finally, expanding the usability of FSDP to a broader range of hardware configurations, including heterogeneous clusters with varying GPU capabilities, would make this solution more adaptable and widely applicable.

Summary of Class Discussion

Q: Figure 6a in the PyTorch paper shows that there isn't a clear performance difference between full sharding and hybrid sharding across the 3 model size data points. What is the difference?
A: Full sharding has more communication overhead because there are more GPUs communicating with each other. With hybrid sharding, the communication is not as expensive because some of it is cross-node (with duplication). Hybrid sharding is more limited by the number of GPUs because of the duplication, however. The figure does not do a great job representing the difference between the sharding strategies as the models get even larger, but hybrid sharding would perform better as the model size grows.

Q: What is the difference between FSDP and tensor parallelism?
A: Both types of parallelism use sharding, where GPUs only have a partial amount of parameters. With tensor parallelism, all of the GPUs work on the same input, but do not gather intermediate parameters, only outputs. With FSDP, all the GPUs work on different inputs on the same model parameters. The parameters are sharded, so it needs to gather them before computing, but it doesn't need to gather the outputs. It uses "reduce-scatter" to synchronize gradients to get around it.

Summary of “TUTEL: ADAPTIVE MIXTURE-OF-EXPERTS AT SCALE”

Problem and Motivation

Sparsely-gated Mixture-of-Experts (MoE) is being used to scale deep learning models to trillions of parameters with fixed computational cost. It uses multiple "expert" models with specialized sub-tasks. Due to the uncertainty of workload experts, expert parallelism is known to have load imbalance problems, and the workload for each expert changes dynamically over iterations. One current solution is leveraging static parallelism and static pipelining. However, the authors of this paper point out that static parallelism causes too much overhead when switching parallelism while static pipelining fails to select the optimal strategy for dynamic workload. This paper looks at TUTEL (dynamic adaptive Mixture-of-Experts) [2] to increase the efficiency of LLM training.

Related Works

There is existing work related to increasing the efficiency of MoE while remaining in memory constraints, as the gating function and feed-forward network layer used in the training process are variable. Many MoE frameworks set a static upper bound of capacity, and some implement logic to encourage gating functions to balance the workload of experts. However, this may cause tokens to be dropped, hurting the accuracy of the model.

Solution Overview

TUTE optimizes MoE with dynamically adaptive parallelism and pipelining. It uses a single distribution layout that covers all possibly optimal strategies, thus avoiding reformatting input data or weights. A major advantage of TUTEL is its adaptive parallelism switching. The authors analyzed complexity across expert, data, and model parallelism, choosing DP and EP+DP+MP as optimal solutions. They propose a zero-cost switchable parallelism mechanism. For switchable DP, the system uses local tokens and partitioned weight parameters, with all-gather communication during the forward pass and reduce-scatter during the backward pass. EP+DP+MP operates similarly, ensuring mathematical equivalence to switchable DP. GPUs are grouped for model parallelism, repeating local tokens at the start and performing a local sum at the end. The "r" parameter, representing token partitions, can be tuned for model-specific training.

Additionally, TUTEL's adaptive pipelining optimizes both the pipelining degree and All-to-All algorithms. It uses token partitioning for multi-stream pipelining to overlap network communication and computation. To maintain ML feature correctness, not all operations should be partitioned, only the two All-to-Alls and the intermediate expert. During the forward stage, input tokens are split and asynchronously sent to execute the first All-to-All. The outputs proceed to expert computation and then to the second All-to-All. A barrier follows before

merging partitions. The backward pass mirrors this flow, using input gradients and backward computation between the All-to-All layers.

When evaluating the performance of TUTEL, the authors compare the throughput of optimal parallelism/pipelining and study the gain from adaptivity of TUTEL. They evaluate TUTEL in an Azure A100 cluster with 2048 GPUs and show ~5x speedup for a single MoE layer. It is a very promising outcome for the training and inference of deep learning models.

Limitations

As for TUTEL, one limitation of the paper is that all evaluation was done on the authors' MoE version of the SwinV2 computer vision neural network architecture. The robustness of the TUTEL approach would be much better proven if the authors had also included other modalities, such as language modeling, when performing evaluations.

Future Research Directions

For TUTEL, one area of future research could focus on making TUTEL's optimization techniques more robust in the face of rapidly evolving model architectures and training methodologies.

Summary of Class Discussion

Q: Can TUTEL be used on tasks other than computer vision?
A: Yes, TUTEL can be used for any transformer based architecture.

References

1. Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. Proc. VLDB Endow. 16, 12 (August 2023), 3848–3860. https://doi.org/10.14778/3611540.3611569
2. Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. 2023. Tutel: Adaptive Mixture-of-Experts at Scale. In Proceedings of Machine Learning and Systems, Curan, 269–287. Retrieved from https://proceedings.mlsys.org/paper_files/paper/2023/file/5616d34cf8ff73942cfd5aa922842556-Paper-mlsys2023.pdf
3. Li, Shen, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke et al. "Pytorch distributed: Experiences on accelerating data parallel training." arXiv preprint arXiv:2006.15704 (2020).
4. Rajbhandari, Samyam, 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, pp. 1-16. IEEE, 2020.
5. Liu, H., Zaharia, M., & Abbeel, P. (2024). RingAttention with Blockwise Transformers for Near-Infinite Context. The Twelfth International Conference on Learning Representations. https://openreview.net/forum?id=WsRHpHH4s0