Member-only story

LLM Training — Fully Sharded Data Parallel (FSDP): An Efficient Distributed Training Technique in PyTorch

Don Moon Follow 5 min read · May 31, 2024

15

Fully Sharded Data Parallel (FSDP) is an open-source distributed training technique provided by PyTorch. While Data Parallelism (DP) with no model sharding is typically the go-to method when a model fits within the memory of a single GPU, FSDP becomes an effective alternative for training models that exceed the memory capacity of a single GPU.

In this blog post, we will first summarize how FSDP works, followed by an evaluation of its efficiency in terms of GPU utilization. Our focus is on understanding the core functionality of FSDP. For more detailed information, I recommend referring to the original paper.

How FSDP Works?

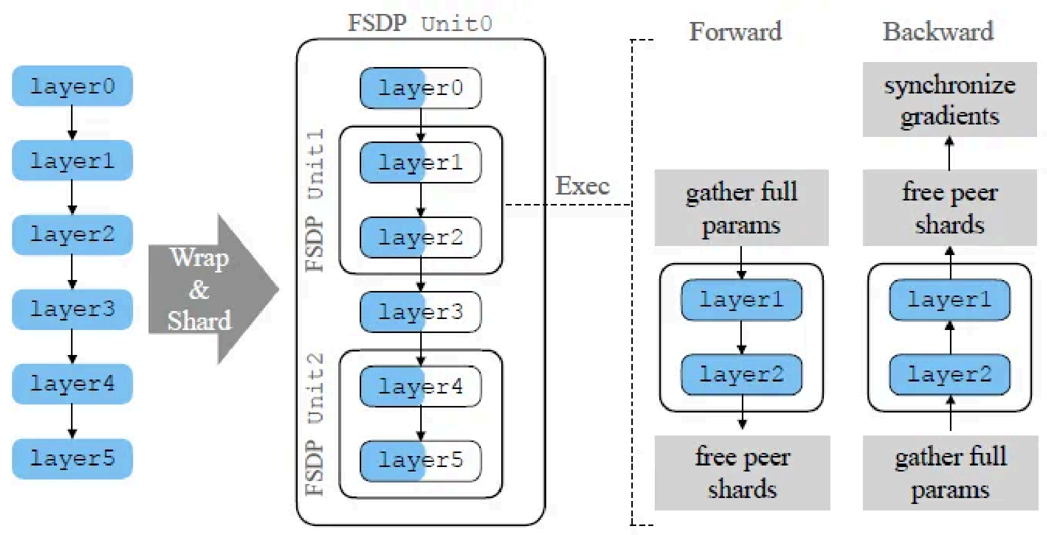
The figure below illustrates the workflow of FSDP on two GPUs using a simple six-layer model, divided into three segments: [layer0, layer3], [layer1, layer2], and [layer4, layer5]. Each segment is referred to as a FSDP unit, whose parameters are sharded across the two GPUs.

> *Note: This example can be easily extended to systems with more GPUs, where each FSDP unit's parameters are sharded across all available GPUs in the training*

system.

Before starting forward computations on [layer1, layer2] (FSDP unit 1), each GPU collects the required unsharded parameters from other GPUs via All-Gather collective communication. These parameters enable the local processing of layers 1 and 2 within each GPU, with each GPU processing its own batch in parallel, similar to normal Data Parallelism. After processing each FSDP unit, FSDP releases the temporarily gathered shards to minimize memory usage. Consequently, during the forward phase, FSDP fully materializes only one FSDP unit at a time while keeping the others sharded, significantly reducing the memory footprint at the cost of increased communication among GPUs.

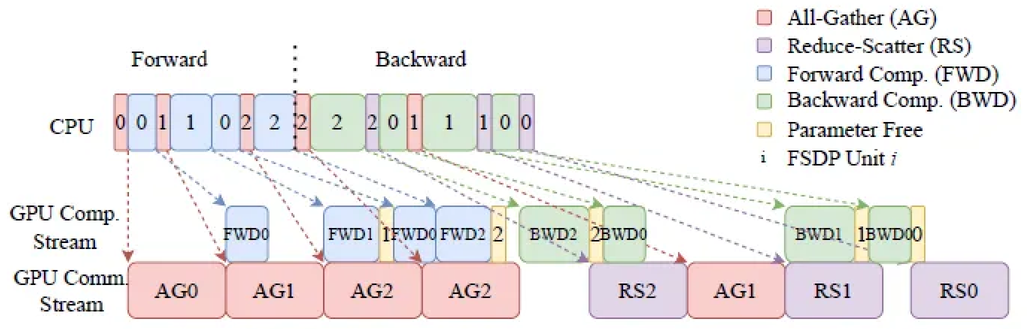
The same principle applies during backward computations: FSDP unit 1 gathers the parameters for layers 1 and 2 before computations and releases them afterward. Once gradients computations are complete, it triggers a ReduceScatter collective communication to shard the reduced gradients over the GPUs. Ultimately, each GPU retains only a shard of the parameters along with the corresponding gradients and optimizer states, reducing memory capacity required per GPU throughout the model's training process.



FSDP running on 2 GPUs [1]

FSDP employs various optimizations for training efficiency. The figure below illustrate one training step for the aforementioned example. The CPU launches compute and communication streams trying to overlap computations with communications as much as possible. For example, during forward and backward, CPU initiates prefetching of parameters for the next FSDP units while starting computations for the current FSDP unit.

- During forward, All-Gather for FSDP unit1 (AG1) is launched in parallel with FWD computations for FSDP unit0 (FWD0) so that FWD computations for FSDP unit1 (FWD1) can be started as soon as possible.
- During backward, Reduce-Satter of the result of BWD2 is run in parallel with BWD0, and All-Gather for FSDP unit1 (AG1) is launched before BWD1 compute stream.



Computation and communication stream during forward and backward

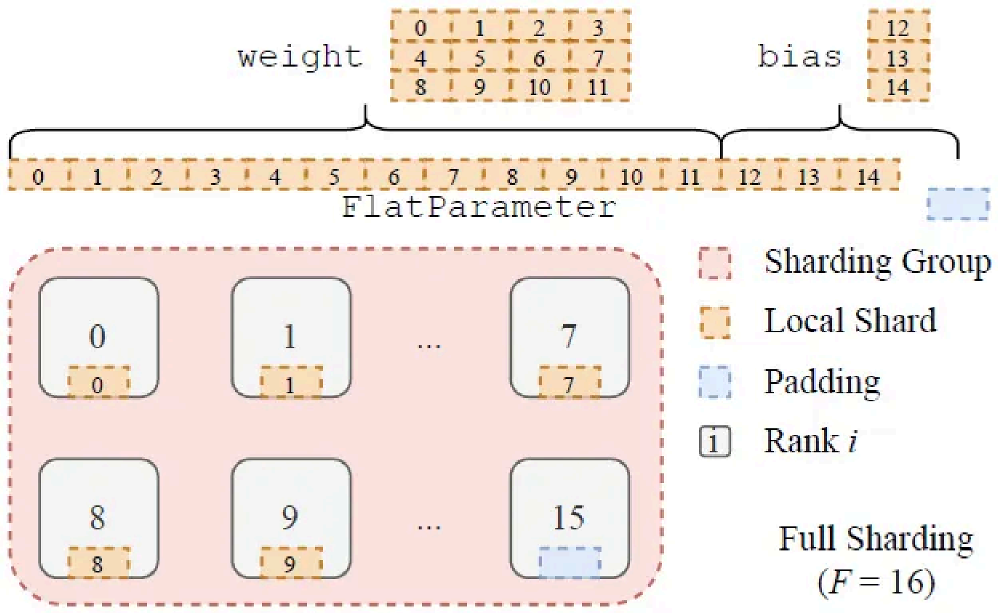
Sharding Strategy

FSDP provides two sharding strategies: ***Full Sharding*** and ***Hybrid Sharding***. Full Sharding would be optimal for a small cluster of GPUs, all connected via high-speed interconnects such as NVLink, which can handle the heavy communication load required for All-Gather or Reduce-Scatter operations during forward and backward computations. This requirement often exceeds the capabilities of slower networks, such as Ethernet. Hybrid Sharding, in contrast, is tailored for a larger cluster of GPUs that incorporate a combination of both high-speed NVLink and slower Ethernet (or PCIe) connections.

Full Sharding

FSDP improves communication efficiency by grouping all parameters within an FSDP unit into a large, flat structure called a FlatParameter. This structure is created by concatenating flattened parameters and padding the result to make its size divisible by the sharding factor (F). The FlatParameter is then divided into F equal parts with each part assigned to a different GPU.

For instance, consider sharding a neural network layer with a dimension of 4x3 across 16 GPUs using one FSDP unit, as depicted below. Here, each GPU manages just one component of the FlatParameter. The last GPUs in the sequence handles any excess padding. Both the FlatParameter and its corresponding gradients and optimizer states share the same dimensions.



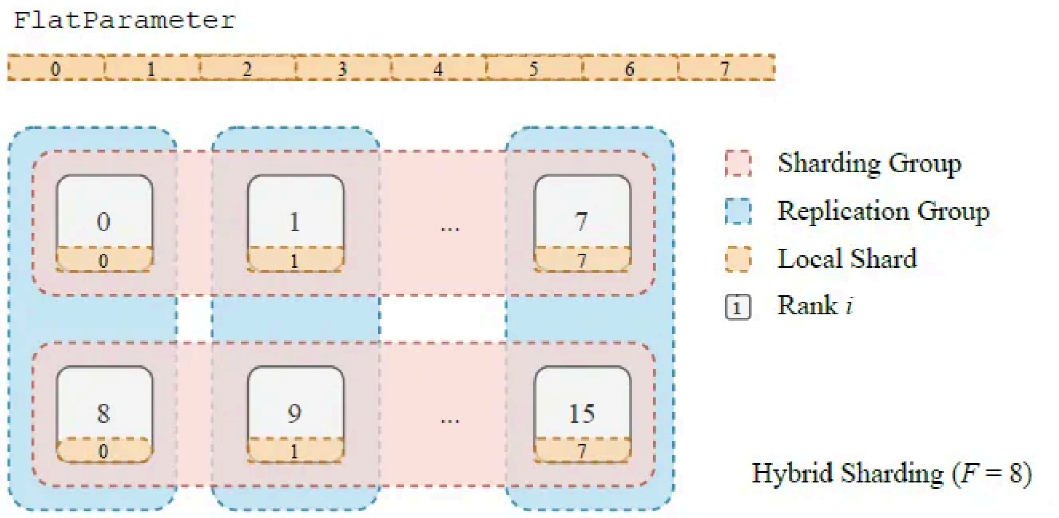
Full Sharding Across 16 GPUs [1]

Hybrid Sharding

Hybrid Sharding involves a sharding factor (F) that is greater than 1 but less than the total number of GPUs. This method merges aspects of both sharding and model replication. For example, with 16 GPUs and a sharding factor of 8, the GPUs are divided into two groups of eight, each operating under a full sharding protocol. Additionally, these two model replica groups communicate with each other to update their gradients through an All-Reduce operation, just like Data Parallelism with no model sharding.

Datacenters often employ a fat-tree network topology with over-subscription, creating opportunities to maximize local communication and minimize cross-host traffic. Hybrid sharding emerges as an effective strategy to align device mesh with datacenter layouts to leverage this locality. For example, in a cluster with *W* accelerators divided evenly into G hosts, setting *F*=*W*/*G* confines collective operations like AllGather and ReduceScatter to the same host, enhancing communication efficiency. This configuration not only reduces cross-host traffic significantly but also improves performance by minimizing delays from stragglers and reducing network congestion.

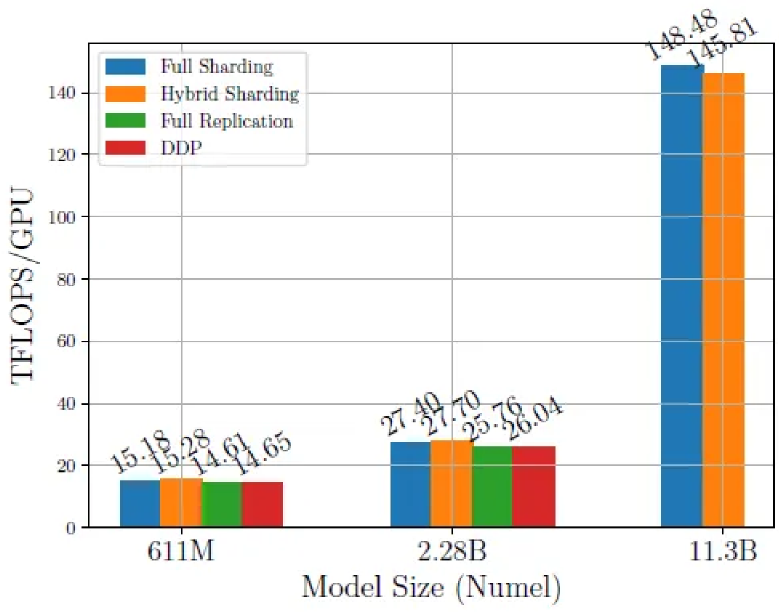
Additionally, hybrid sharding addresses specific needs of medium-sized models, which are too large for full replication without exceeding memory limits but underutilize hardware resources when fully sharded. By adjusting the sharding factor (*F*), hybrid sharding offers a flexible balance between memory usage and computational throughput, optimizing resource utilization and performance.



Hybrid Sharding on 16 GPUs [1]

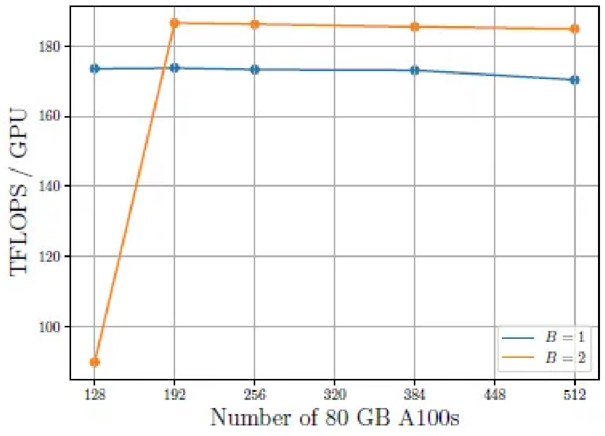
Linear Performance Scaling

FSDP, including both Full Sharding and Hybrid Sharding, scales effectively from small to large models while maintaining per-GPU performance comparable to DDP, as shown in the figure below.



Scaling of T5 models from 611M to 11.3B

FSDP also ensures near-linear training performance scaling with respect to the number of GPUs. Given that the A100's peak performance is 312 TFLOPs, GPU utilization remains around 55 to 60% even with up to 512 GPUs, implying that we can reduce training time proportionally as we increase the number of GPUs.



Training GPT-175B using FSDP with batch size 1 and 2 [1]

Wrap-up

FSDP is an open-source distributed training technique available through the PyTorch framework. It effectively overlaps communication overheads with computations, achieving near-linear performance scaling with the number of GPUs when training large models.

References

[1] PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel, arXiv:2304.11277, Sep 2023

[2] NVIDIA Collective Communications Library (NCCL), https://developer.nvidia.com/nccl

Distributed Training   Data Parallelism   Llm Training