

Distributed Training with Model Parallel

Mengzhou Xia, Alex Wettig

11/09/2023

Questions

1. What do we care about with distributed training and performance? What happens under the hood with FSDP?
2. What hardware setup do I need for different distributed training strategies? What are the caveats?
3. What are the various efficient finetuning optimizations? What are the tradeoffs?
4. What open-source codebases can I use right now? What are the pros and cons?

Basics of distributed training

- LLM training involves large model (10B+) and dataset (1T+ tokens) sizes.
- Maximize **throughput** for efficient training (tokens/s)
- LLMs demand substantial GPU vRAM for model weights and optimizer states.
 - a. Weights: $N * 2$ bytes
 - b. Gradients: $N * 2$ bytes
 - c. Adam optimizer states: $N * 12$ bytes (copies of parameters, momentum and variance)

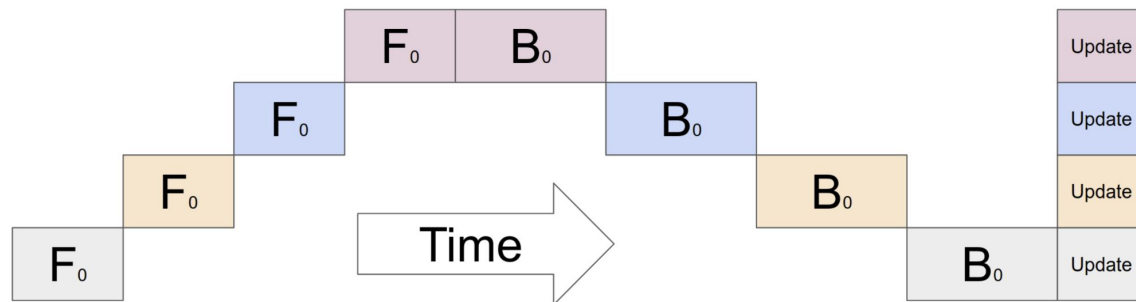
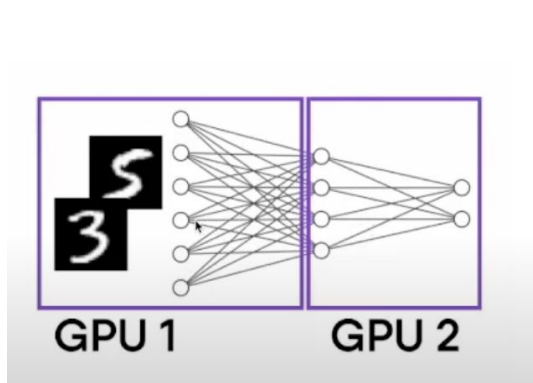
e.g., Falcon 40B * (2 + 2 + 12) = 720GB, 9 A100 at least

Naive Model Parallelism (MP)

- Vertically slices the model.
- Different layers on different GPUs. Example: 12-layer model on 3 GPUs.

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

- Naive MP: waiting for the previous GPUs to process the data



lots of idleness!

FSDP - Fully-Sharded Data Parallel

- FSDP Unit - Vertically splitting (layers)
- Sharding - Horizontally splitting
 - Store FSDP parameters on FlatParameter
 - Split FlatParameter on multiple processes

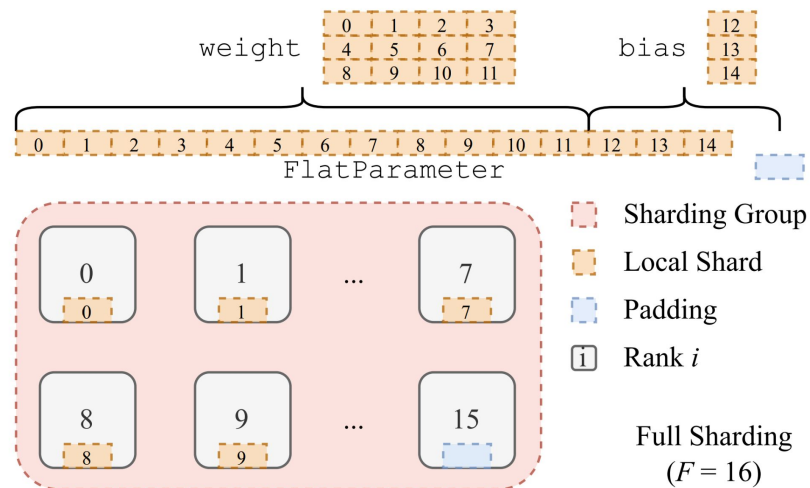
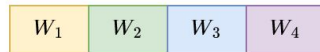


Figure 3: Full Sharding Across 16 GPUs

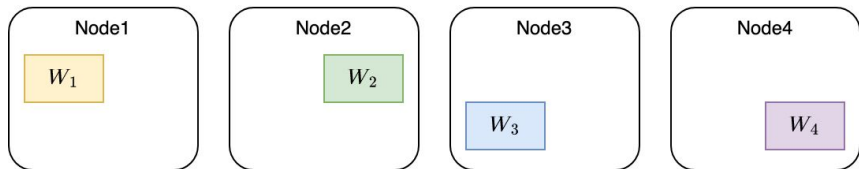
FSDP - Fully-Sharded Data Parallel

- All-Gather per FSDP-unit
 - Before forward
 - Before backward
- You can do this asynchronously across layers
- No activation are exchanged!

A FSDP-Unit

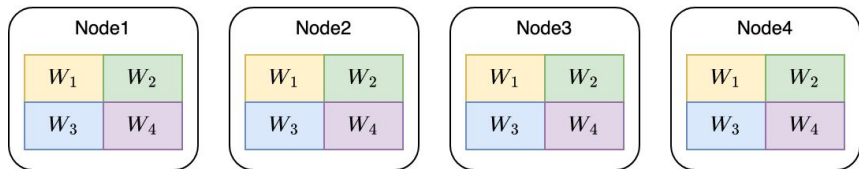


Sharding

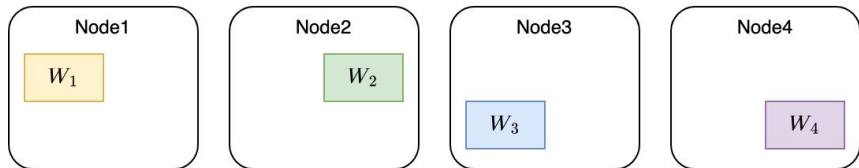


All-Gather

[Before both forward / Backward]



Free Peer Shards After Usage



How can you use FSDP?

- Supported by transformers!
- By adding fsdp config to the TrainingArguments
- Simplest setup:
 - `-- fsdp`
 - `-- full_shard auto_wrap`
- More setups:
 - [Torch FSDP interface](#)
 - [Huggingface Accelerator setup](#)
 - [Pytorch lightning](#)

Efficient Fine-tuning

- Mixed Precision (BF16, FP16), supported by Huggingface
- Parameter-efficient finetuning, supported by [PEFT](#)
 - Only finetune additional parameters, eventually merged into the main model
 - Saves the memory for optimization states for the freezing parameters
- Flash Attention: fast, memory-efficient and exact!
 - Supported by huggingface on Llama and Falcon, through `use_flash_attention=True` to `AutoModel`
- Gradient Checkpointing
 - reduce memory consumption by only retaining a subset of intermediate activations, and recomputing the rest as needed, slows down by 20%
- Quantization
 - Post training quantization: [LLM.int8\(\)](#), [GPTQ](#)
 - `Load_in_8bit` supported by huggingface's `from_pretrained` interface
 - Quantization-aware training: [QLoRA](#)