Distributed Training

So we want to train large language models

Main question: Can we do it simply by writing a pytorch module called LargeLanguageModel and then do

loss = loss_fn(LargeLanguageModel(input), label)

And then

loss.backward()?

Difficulty of training large language models

- Main challenge: Memory consumption
- Memory consumption comes from two folds:
  - Storing the models weights in memory (usually in FP16)
  - Storing the optimizer states (the momentums) in memory (usually in FP32, for accuracy purpose)
- For a 7B model, storing the model's weights in FP16 requires 14G of memory.
- Storing the optimizer states in FP32 requires 56G memory
  - In total 70G memory used! Recall that H100 GPU only has 80G memory

Memory is the biggest concern

- We can not train models larger than 7B in a single GPU…
  - Can not be stored in memory
- We need some other way to scale up the model.
- Distributed Training (on multiple GPUs)

Sharded Optimization

- Spirit: Store things across multiple GPUs.
  - Store Model’s weights on different GPUs
  - Store Optimizer states on different GPUs.

Optimizer Sharding

- For a 7B model, storing the model's weights in FP16 requires 14G of memory.
- Storing the optimizer states in FP32 requires 56G memory
  - In total 70G memory used! Recall that H100 GPU only has 80G memory
- Storing the optimizer states requires a lot of memory!!!
- Sharding: We store optimizer states in different GPUs.

Sharding Optimizer States

- Suppose we have m GPUs.
- Suppose the optimizer state is w
- We partition w into m same size parts w = (w1, w2, ..., wm)
- We store wi on the i-th GPU.

Fully Sharding

- Instead of Sharding only optimizer states, we can also shard the model weights:
- We split the model weights W into m parts, and store each part on each GPU.

Tensor Parallel (Model Parallel)

- Sharding versus Tensor Parallel:
- For sharding, we aggregate model weights across GPUs and then we still do forward/backward in each single GPU.
- Tensor Parallel: Forward/Backward are Done across different GPUs.
- RowParallelLinearLayer
- ColumParallelLinearLayer

Tensor Parallel

- Parallel Linear Layer:

- f(A x), we want to distributedly compute forward Ax and the backward A^T y

- We store A across different GPUs, we distribute x/y to those GPUs in order to compute the forward/backward.

Pipeline Parallel

- We convert the model into a nn.Sequential module (a sequential of layers)
- We store each layer on each GPU.

- Forward/backward pass is run through each layers