micropp

Syllabus

Monolith

- **Concept:** Establish a baseline on a single CPU.
- **Lab:** Just `nn.Sequential` with 16 layers and a simple training loop.

Motivation for PP

- **The Memory Wall:** Why models don't fit on one GPU.
  - **Model Size:** 10 Billion Parameters (10B).
  - **Precision:** FP32 (4 bytes per parameter).
  - **VRAM:** 40 GB.
  - **Hardware:** NVIDIA RTX 4090 (24 GB VRAM).
- **Solution:** Model Partitioning (slicing `nn.Sequential`).

Manual

Cut the `nn.Sequential` into two pieces: `part1` and `part2`.

- **The Exercise:** Try to train it by manually passing the output of `part1` into `part2`.
- **The Lesson:** You have to manage the "hand-off" of the activations; autograd handles the gradients for us.

Distributed Basics

- **Concept:** Rank, World Size, and Process Group.
  - **The Process Group:** Imagine a conference call. Before anyone can talk, they must dial in. `init_process_group` is dialing in.
  - **World Size:** The total number of people on the call (e.g., 4 GPUs).
  - **Rank:** Your unique ID badge (0, 1, 2, 3).
  - **Rank 0** is the "Boss" (usually handles logging, saving checkpoints, and data loading).
- **Concept:** What torchrun does on a CPU.
  - **Process Isolation:** `torchrun` spawns completely separate Python interpreter instances. Each has its own memory space and its own "Global Interpreter Lock" (GIL).
  - **True Parallelism:** Because these are separate processes (not threads), the OS schedules them across different physical CPU cores.
  - **The Network Bridge:** When you call `dist.send`, the data leaves Process A and travels through a local networking socket (the "Gloo" backend) to reach Process B.
- **Lab:** Spawn 2 processes on GPU (or CPU) and ping-pong a tensor.

Pipeline Parallelism

Naive

- **Concept:** Stop-and-wait execution.
- **Lab:** Implement the Naive Schedule. Measure utilization using nvidia-smi if cuda is available.

GPipe

- **Concept:** Changing the loop from "Batch" to "Chunks."
- **Lab:** GPipe (Fill -> Drain).

1F1B

- **Concept:** Interleaving Chunks.
- **Lab:** 1F1B (Steady State).

Library

1. `comms.py` (The Glue)

- **Initialization:** A wrapper around `dist.init_process_group()`.
- **Topology:** `get_next_rank()` and `get_prev_rank()` based on rank.
- **P2P:** `send_tensor(tensor, dest)` and `recv_tensor(shape, src)`.

2. `model.py` (The Subject)

- **ShardedMLP:** A class `ShardedMLP(nn.Module)` with 16 `Linear` layers.

3. `schedule.py` (The Engine)

- **Micro-batcher:** A utility to `split` a batch tensor into chunks.
- **The Orchestrator:** A loop that manages the “Clock Cycle”.