Programming Assignment 2 - Distributed LLM Fine Tuning

Assignment Overview

Goal: Distributed Fine-Tuning of LLaMA on 2 GPUs

Dataset
Same dataset as Programming Assignment 1.

Pretrained Model
- LLaMA 3B model
- On the cloud burst compute file system: /scratch/BDML25SP/

Key Focus
We will be focusing on distributed training
- Data Parallelism
- Tensor Parallelism
- Pipeline Parallelism

The goal of the assignment is to implement these techniques on 2 GPUs and achieve high training efficiency (time per epoch).

Deliverables
1. A report documenting:
    a. Distributed training techniques used.
    b. Training performance (time per epoch) and evaluation results.
    c. Step by step guide on how to run the training code.

2. Code access on HPC

Evaluation
Compute the perplexity metric on the remaining 10% of the dataset. The assignment will be evaluated primarily on the basis of how time efficient the fine-tuning code is, and the final perplexity score will not hold as much weight.

Data Parallelism

Data parallelism involves **replicating the model on both GPUs** and **splitting the training data** across them. We have covered this paradigm in class in the paper Pytorch Distributed (https://arxiv.org/pdf/2006.15704).

Example code:

```
import torch
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP

dist.init_process_group("nccl", rank=rank, world_size=2)

model = LLaMAModel().cuda(rank)
model = DDP(model, device_ids=[rank])
```

Tensor Parallelism

Splits weight matrices of large layers (like Transformer blocks) **across multiple GPUs**. Each GPU holds **only part of the model's layers**. We covered this in the Tofu paper (https://arxiv.org/pdf/1807.08887).

Example code:

```
import torch
import torch.nn as nn
import torch.distributed as dist
from torch.distributed.tensor.parallel import parallelize_module

# Initialize distributed environment
dist.init_process_group("nccl")

model = LLaMAModel().cuda(rank)
# Apply tensor parallelism (split layers across 2 GPUs)
parallelize_module(model, parallel_mode="column", devices=[0, 1])

```

Pipeline Parallelism

Pipeline parallelism **assigns different layers** of the model to different GPUs and processes micro-batches sequentially. We have covered two systems of this type in GPipe (https://arxiv.org/pdf/1811.06965) and PipeDream (https://arxiv.org/pdf/1806.03377).

Example Code:

```python
import torch
import torch.nn as nn
from torch.distributed.pipeline.sync import Pipe

model = LLaMAModel().cuda(rank)
# Define layer partitions (e.g., 3 layers per GPU)
model = Pipe(model, balance=[3, 3], devices=[0, 1])

```

Evaluation

Measure time per epoch as the primary metric. Other parts are the same as in Programming Assignment 1.