209AS Lab3

Due date: Jun 12th

Ver 1.0

Introduction

This lab will focus on modeling distributed inference. More specifically, you will calculate the total inference latency and communication traffic for a 4-layer CNN distributed across a 4-node machine, utilizing tensor and pipeline parallelism.

Background

Distributed Inference

Distributed inference involves distributing a neural network model across multiple compute nodes to handle inference tasks. This approach can significantly reduce the memory requirement of each node or increase the overall throughput. It is particularly useful for handling large models or large volumes of input data that cannot be processed efficiently on a single machine.

For this exercise, we will focus on tensor and pipeline parallelism, which are two parallelization techniques that can reduce the memory requirement during inference. These two methods are especially useful for today's LLMs where the models are often way larger than the DRAM size of a single GPU/compute node.

Tensor Parallelism

Tensor parallelism is a technique where the computation of tensors (multidimensional arrays used in neural networks) is divided across different processors or nodes. This type of parallelism involves splitting the model's layers or the data tensors themselves, allowing different parts of a computation to be executed simultaneously on different hardware units. The main goal is to speed up processing by utilizing the computational power of multiple nodes concurrently. In this exercise, tensor parallelism means to split the computation of each layer across all nodes.

Pipeline Parallelism

Pipeline parallelism involves dividing the model into different segments or stages, each of which is processed on a separate node. Data flows sequentially from one stage (node) to the next, much like an assembly line in a factory. Each node processes a different part of the model, and as soon as it finishes processing one batch of data, it passes it to the next stage while simultaneously working on the next batch. This method aims to maximize hardware utilization by continuously feeding data through the model's pipeline, reducing idle times and improving throughput. In this exercise, pipeline parallelism means each node will process a unique convolution layer.

Problem setup

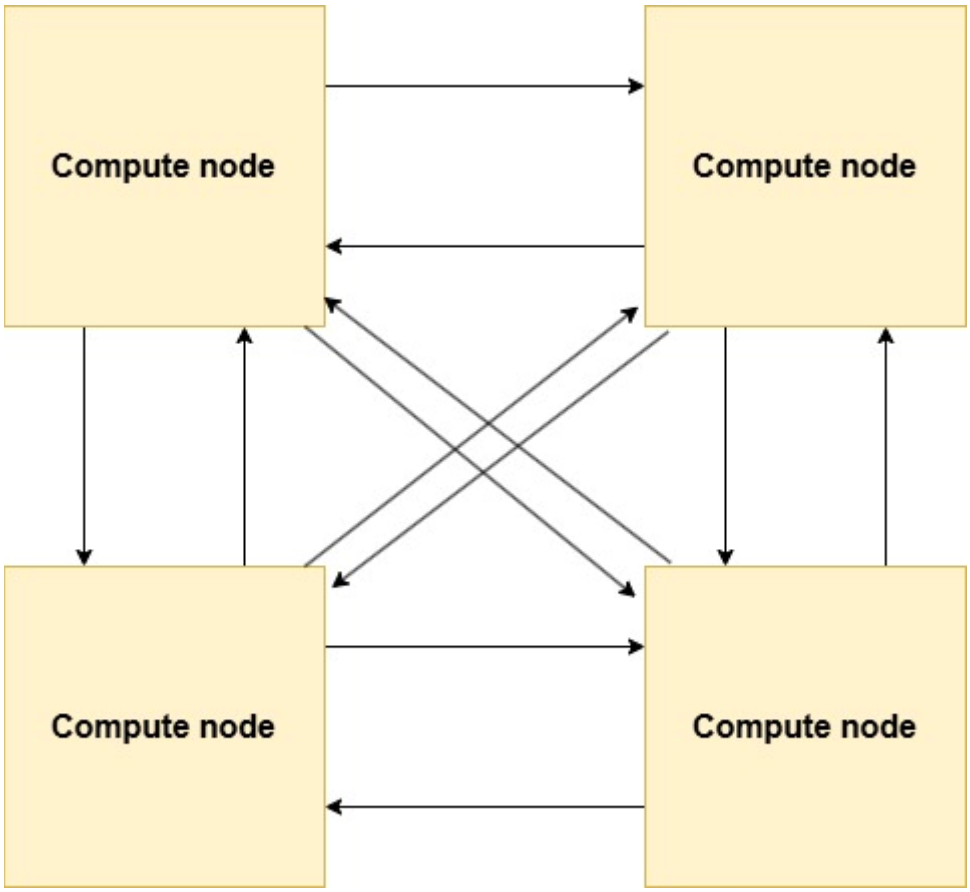
The model

While usually distributed inference are applied for large models such as LLMs, to reduce the complexity, we will use a relatively small CNN model which I think you should be pretty familiar with at this point.

The model that will be used in this exercise is provided in the table below, which contains 4 layers. Initial inputs have dimension 3x32x32, and the activation dimension is reduced by 2 on each x/y dimension because of convolution. **FP16 (16-bit per weight)** will be used as the precision for this model.

Layer ID	Weight Dimension (# filter, # in_channel, fx, fy)	Input activation dimension
1	128,3,3,3	32,32
2	128,128,3,3	30,30
3	128,128,3,3	28,28
4	128,128,3,3	26,26

The hardware



The hardware for this exercise is a multi-node compute system with 4 compute nodes. You can imagine each compute node is similar to the design of Project 1 part 2.

Each compute node contains a 512 KB on-chip SRAM, and there is **no** DRAM in the entire system, which means all the weights and activations need to be stored in the SRAM of the nodes.

All-to-all network is adopted for this hardware, where each compute node is connected to all other three through dedicated links as shown in the diagram. So each node has three output links and three input links.

In this problem, you will treat each compute node as a black box, you don't need to know nor model the details within it. The compute throughput will be provided and you can use that for latency calculations.

System specifications

Compute throughput per node: 50 GOPS (giga operations per second)

Communication bandwidth per link: 2 GB/s (Note there are 12 links in total)

Notes and assumptions

- You can calculate the latency of executing a given number of operations (remember 1 MAC = 2 OPS) with a single node simply by dividing it with the compute throughput per node provided.
- For tensor parallelism, the model is distributed in a way that for each layer, all the filters are evenly distributed across the nodes.
- For pipeline parallelism, each layer is stored and computed in a separate node.
- For both cases, the final network output needs to be stored in a single node, but it can be any node as long as the output is complete.
- You can imagine the initial inputs are already in the SRAM.
- You can ignore basically all overhead except for computation and communication between the nodes.

Objectives

1. **For a single input, calculate the total latency as well as the total communication traffic (in bytes) between nodes for both parallelization methods**. The total latency is defined as the time needed to process the entire neural network for this input. The total communication traffic is defined as the accumulated traffic of all links throughout the execution. How do the results compare between the two methods?
2. Repeat the latency and total network traffic calculation for a total of 32 inputs, processed one by one. Here latency means total time required to fully compute all 32 inputs.
3. Report-only (no codes needed): Do you know why we have to distribute the inference across the nodes in this problem? Can we do data parallelism for the 32 inputs case (which means each node has full weights and processes a subset of total inputs)?
4. Report-only (no codes needed): what could happen to the total communication latency for the tensor and pipeline parallelism if the network structure is changed to ring topology (4 links total)? You don't need to provide simulation results, some qualitative analysis is good enough.

Hints:

- For tensor parallelism, since filters are evenly distributed across the nodes for every layer, each node only has a partial output after executing a layer.
- For pipeline parallelism, each node processes a single layer, and layers have dependencies, so…

Deliverables

Code: Submit a single python script named as lab3.py, which should print out the results for both questions like shown below:

```
------------------Q1 single input------------------
Total latency of tensor parallelism is xxx seconds
Total network traffic of tensor parallelism is xxx bytes
Total latency of pipeline parallelism is xxx seconds
Total network traffic of pipeline parallelism is xxx bytes
------------------Q2 32 inputs------------------
Total latency of tensor parallelism is xxx seconds
Total network traffic of tensor parallelism is xxx bytes
Total latency of pipeline parallelism is xxx seconds
Total network traffic of pipeline parallelism is xxx bytes
```

The code should still be packed into a tarball before submission, named as UID_Lastname_Firstname_Lab3_pinXXXX.tar.gz

Submit the tarball to '/w/class.1/ee/ee209w/ee209wt2/submission/project3/'.

Please follow the submission instructions from Lab 1 for full instructions and examples.

Report: Submit a 4–6 page report in the form of slides to gradescope, summarizing your approach, **results**, analysis, and answers.