Distributed Deep Learning

Lecture 12 for Advanced Deep Learning Systems

Aaron Zhao, Imperial College London, a.zhao@imperial.ac.uk

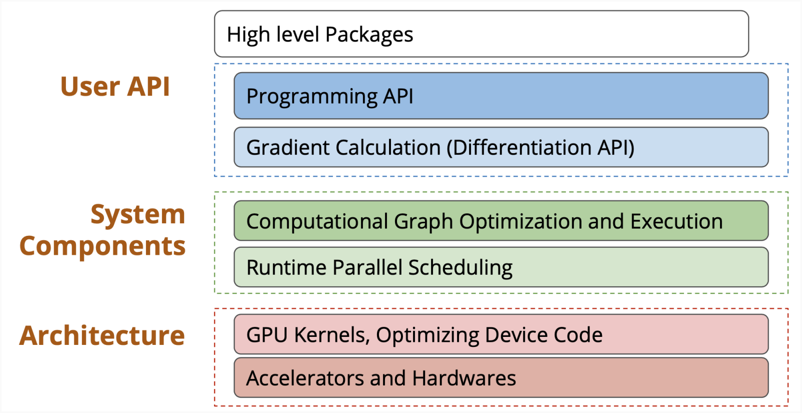
Table of contents

Introduction

Distributed Computing is Scalability

In previous lectures, we mainly looked at single-node computation.

Distributed Computing is Scalability



High level Packages
User API
Programming API
Gradient Calculation (Differentiation API)
System Components
Computational Graph Optimization and Execution
Runtime Parallel Scheduling
Architecture
GPU Kernels, Optimizing Device Code
Accelerators and Hardwares

Distributed Computing is Scalability

- Servers are stored in racks (42U rack).
- Rack servers (typically 3U) are installed in these racks, each server normally has 4-8 GPU cards, interconnected through PCIE.
- Servers in the same rack normally has a faster network (ToR).
- Racks are also connected, but a lot slower in terms of point-to-point latency.

How do we map AI workloads into such systems?

AI Compute Parallelism

Canonical View of ML Parallelism

- Data Parallelism: data is partitioned across distributed workers, but the model is replicated.
- Operator parallelism: partition the computation of a specific operator, such as matmul, along non-batch axes, and compute each part of the operator in parallel across multiple devices.
- Pipeline parallelism: places different groups of ops from the model graph, referred as stages, on different workers

These parallelism can also be mixed.

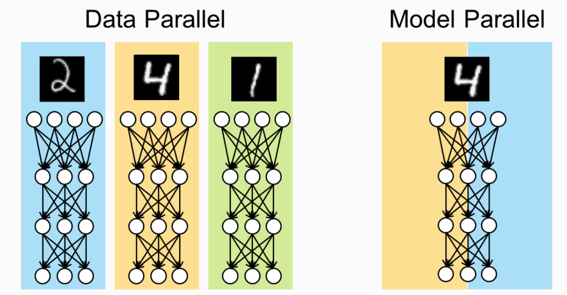
Data Parallelism

We replicate the same model to various workers (N GPUs).

The data is then split into N parts to be deployed on these N GPUs.

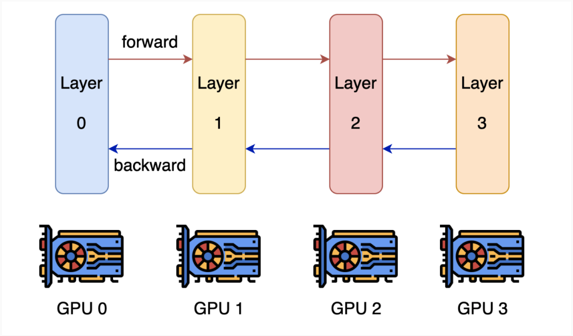
Operator/Model Parallelism

- Different colors represent different GPU devices
- We can separate the model and put different portions of the model to different devices
- Some people say tensor parallelism – this is if you chop inside an operator, and operator/model parallelism normally refers to the fact that you are chopping at the operator granularity.



Pipeline Parallelism

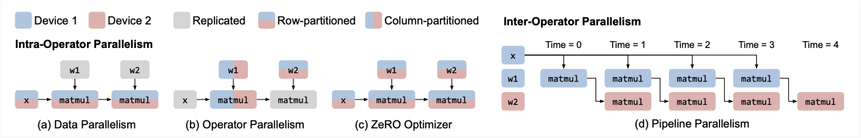
- Very similar to CPU pipelining
- Works both in inference and training



An Alternative View of ML Parallelism

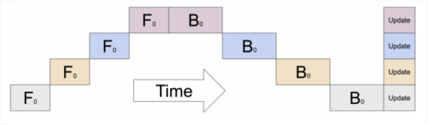
- Intra-operator parallelism: an operator works on multi-dimensional tensors. We can partition the tensor along some dimensions, assign the resulting partitioned computations to multiple devices, and let them execute different portions of the operator at the same time.
- Inter-operator parallelism: we define inter-operator parallelism as the orthogonal class of approaches that do not perform operator partitioning, but instead, assign different operators of the graph to execute on distributed devices.

ML Parallelism

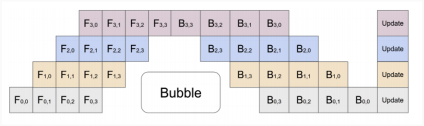


ML Parallelism

Model parallelism



Model and Pipeline parallelism



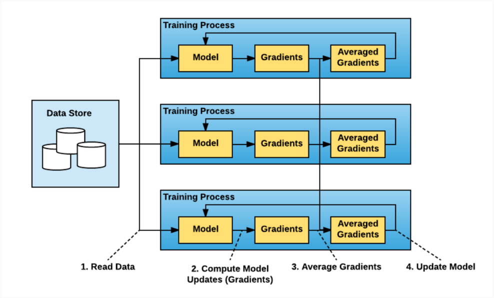
$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$
Bubble
Update

Distributed Training

Parameter Server

It follows the data parallel approach.

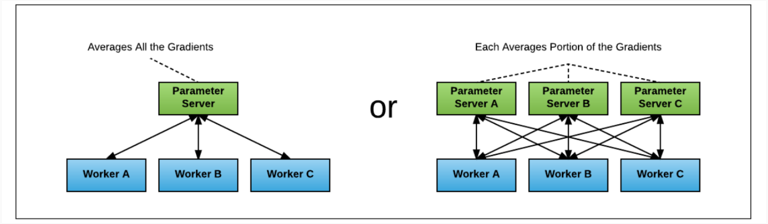
In training, we split the data, and compute update at each local replica, we then need to aggregate the gradients and then propagate back the updated weights.



Parameter Server

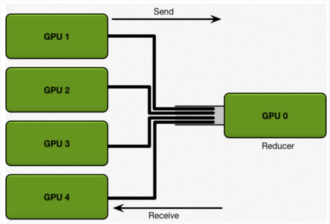
We need to sync the gradients and updated weights.

We gather the gradients in parameter server or parameter servers.



The Ring All Reduce communication pattern

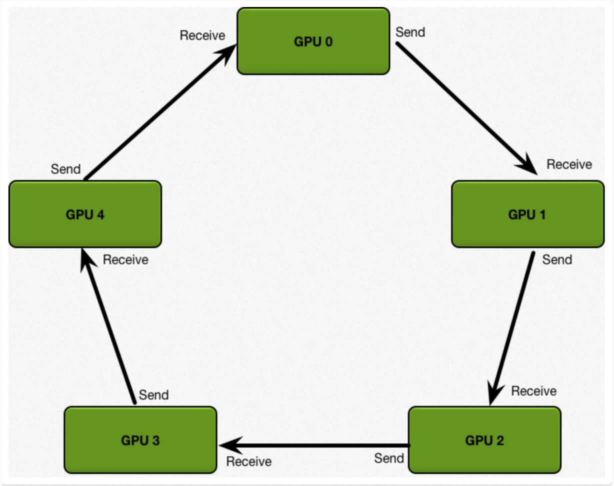
We need to sync the gradients and updated weights.



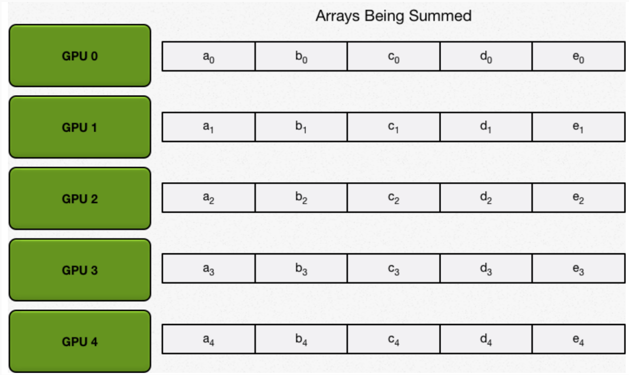
The most obvious communication pattern is to allow a crossbar connection, however, this is not very scalable.

The Ring All Reduce communication pattern

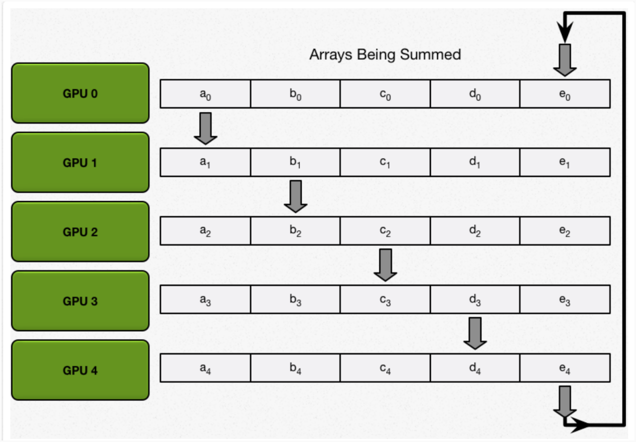
With the following physical connections



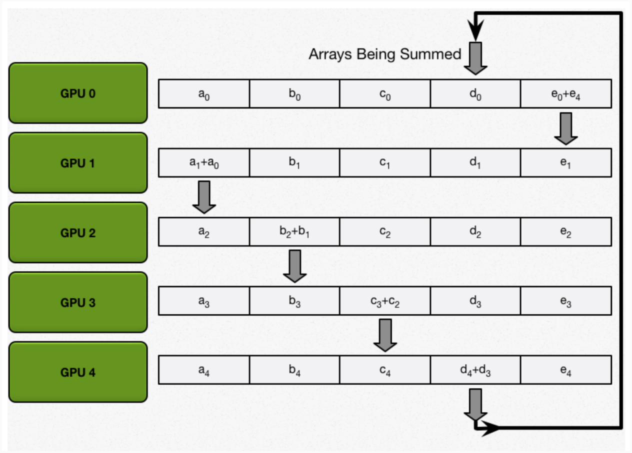
The Ring All Reduce communication pattern - ReduceScatter


Arrays Being Summed
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
a_0 b_0 c_0 d_0 e_0
a_1 b_1 c_1 d_1 e_1
a_2 b_2 c_2 d_2 e_2
a_3 b_3 c_3 d_3 e_3
a_4 b_4 c_4 d_4 e_4

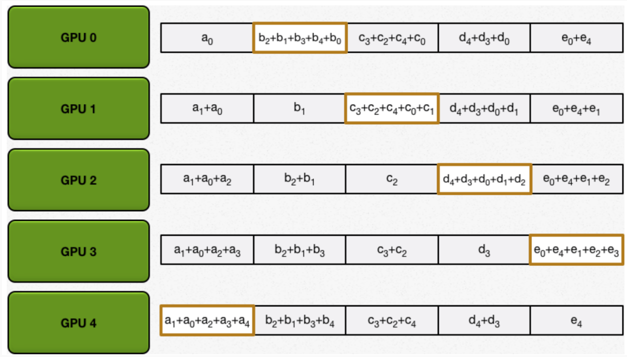
The Ring All Reduce communication pattern - ReduceScatter


Arrays Being Summed
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
a_0 b_0 c_0 d_0 e_0
a_1 b_1 c_1 d_1 e_1
a_2 b_2 c_2 d_2 e_2
a_3 b_3 c_3 d_3 e_3
a_4 b_4 c_4 d_4 e_4

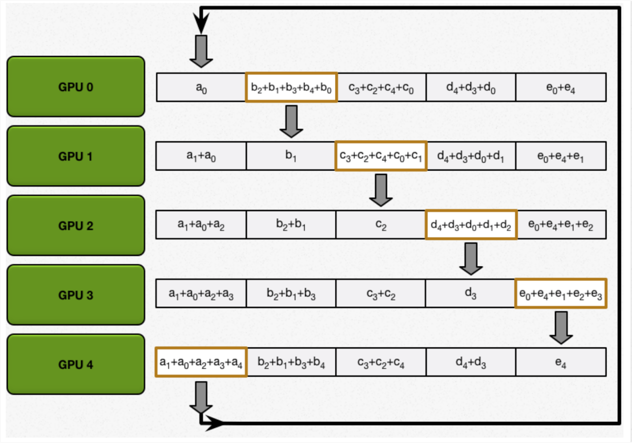
The Ring All Reduce communication pattern - ReduceScatter


Arrays Being Summed
GPU 0
a_0 | b_0 | c_0 | d_0 | e_0+e_4
GPU 1
a_1+a_0 | b_1 | c_1 | d_1 | e_1
GPU 2
a_2 | b_2+b_1 | c_2 | d_2 | e_2
GPU 3
a_3 | b_3 | c_3+c_2 | d_3 | e_3
GPU 4
a_4 | b_4 | c_4 | d_4+d_3 | e_4

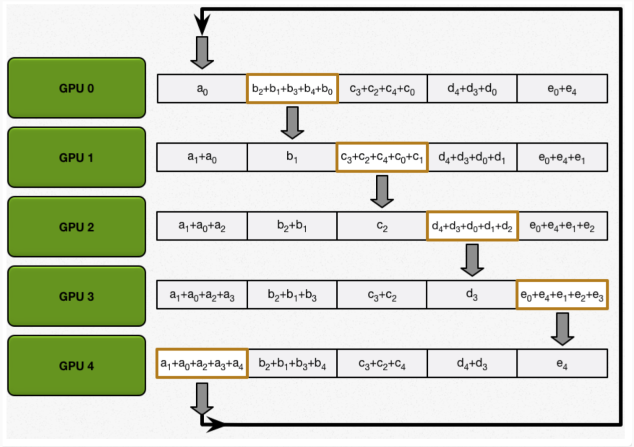
The Ring All Reduce communication pattern - ReduceScatter



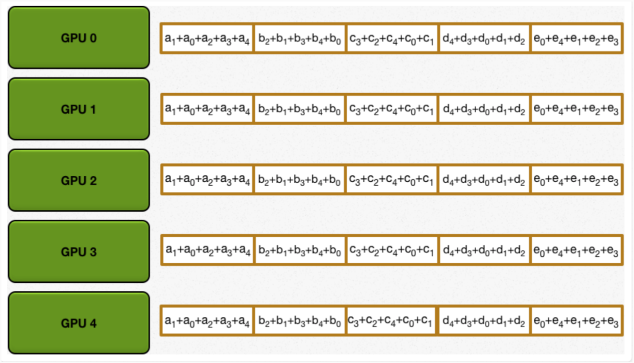
The Ring All Reduce communication pattern - AllGather



The Ring All Reduce communication pattern - AllGather



The Ring All Reduce communication pattern - AllGather



For Normal Users

Normally you do not need to worry. Typically, you don't need to be concerned about the distributed training strategy, as PyTorch, NCCL, and MPI collectively manage most of it for you.

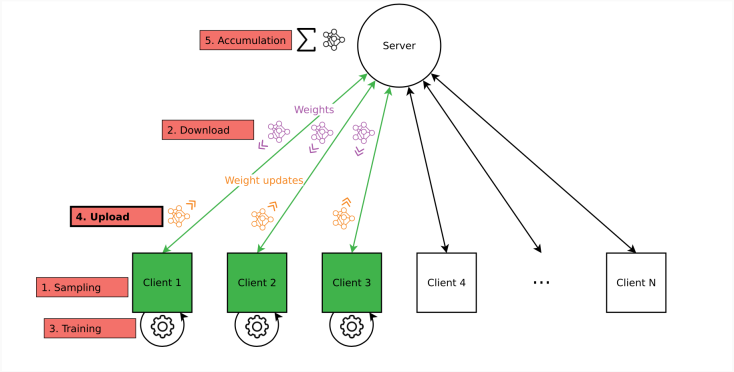
However, you must consider the distributed training strategy if you are using more than 8 GPUs, which exceeds the capacity of a single node.

Federated Learning

The Concept

Let's do not let the user data leave their phone (tricky!).

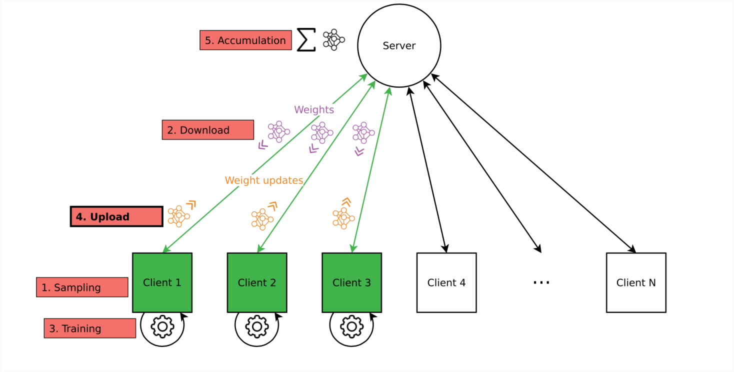
Phase 1 and 2: The server selects a subset of clients, with each algorithm employing a distinct sampling strategy. These clients update their local weights from the server.



The Concept

Let's do not let the user data leave their phone (tricky!).

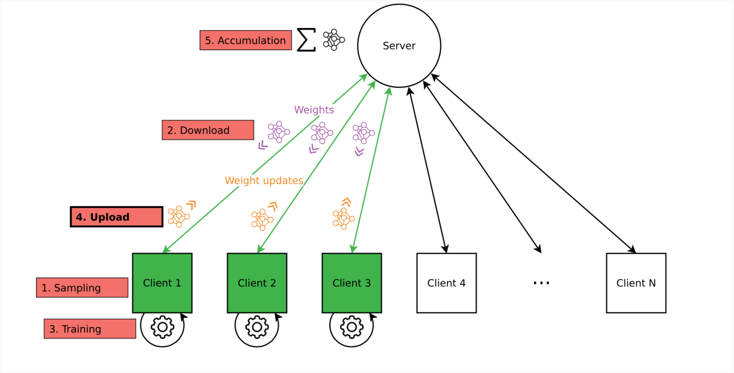
Phase 3: each client perform training on a model replica locally, using its own data. This training is always running concurrently, no matter a device is chosen or not. The server also normally gives a time constraint on this training, if a client did not finish its training in time, it is ignored (this is known as stragglers).



The Concept

Let's do not let the user data leave their phone (tricky!).

Phase 4 and 5: Clients now transfer back weight updates across E epochs back to the server The server accumulate these updates.



Pros and Cons

Biggest advantage: No “data” leaves the user device!

Other advantages

- Some level of privacy protections
- Give people an opportunity to do differential privacy

Disadvantages

- Can take very long to train since effectively there is a subsampling.
- If server is not honest, bad things can happen.

Summary

- Different parallelisms (eg. data, model, pipeline)
- Distributed Training, concepts on parameter servers and Ring All Reduce.
- Other learning paradigms: federated learning.