Lecture 11: Distributed Training and Communication Protocols
Where are we

**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
Where are we

- Programming API
- Gradient Calculation (Differentiation API)
- Computational Graph Optimization and Execution
- Runtime Parallel Scheduling / Networks
- GPU Kernels, Optimizing Device Code
- Accelerators and Hardwares
Recap: Parallel Scheduling Engine

The Tagged Data

Pack Reference to Related Things into Execution Function (via Closure)

Push the Operation to Engine

A.data
B.data

\[ \text{lambda: } B.data = A.data + 1 \]

\[
\text{engine.push(}
\begin{array}{c}
\text{Exec Function} \\
\text{read} = [v1], \\
\text{mutate} = [v2]
\end{array}
\text{)}
\]
Recap: Example Scheduling

\[
A = 2 \quad \Rightarrow \quad \text{engine.push(lambda: A.data=2, read=[], mutate=[A.var])}
\]

\[
B = A + 1 \quad \Rightarrow \quad \text{engine.push(lambda: B.data=A.data+1, read=[A.var], mutate=[B.var])}
\]

\[
D = A \times B \quad \Rightarrow \quad \text{engine.push(lambda: D.data=A.data \times B.data, read=[A.var, B.var], mutate=[D.var])}
\]
Data Parallelism

- Train replicated version of model in each machine
- Synchronize the gradient
How to do Synchronization over Network
Distributed Gradient Aggregation, Local Update

Many replicas of the same graph run in parallel

\[ G_1 = \text{sum}(g_1 \text{ over replicas}) \]

\[ w_1 \leftarrow lr \times G_1 \]
Allreduce: Collective Reduction

**Interface**

\[
\text{result} = \text{allreduce(float buffer[size])}
\]

**Running Example**

Machine 1

\[
\begin{align*}
\text{comm} &= \text{communicator.create()} \\
\text{a} &= [1, 2, 3] \\
\text{b} &= \text{comm.allreduce(a, op=\text{sum})} \\
\text{assert b} &= [2, 2, 4]
\end{align*}
\]

Machine 2

\[
\begin{align*}
\text{comm} &= \text{communicator.create()} \\
\text{a} &= [1, 0, 1] \\
\text{b} &= \text{comm.allreduce(a, op=\text{sum})} \\
\text{assert b} &= [2, 2, 4]
\end{align*}
\]
Use Allreduce for Data Parallel Training

\[
\text{grad} = \text{gradient(net, w)}
\]

\[
\text{for epoch, data in enumerate(dataset):}
\quad g = \text{net.run(\text{grad, in=}data)}
\quad \text{gsum} = \text{comm.allreduce(g, op=}\text{sum) }
\]

\[
\text{w} -= \text{lr} * \text{gsum} / \text{num_workers}
\]
Common Connection Topologies

All-to-all: (plugged to same switch)

Ring (NVLink)

Tree-Shape
Discussion: 3min

- How to Implement Allreduce over Network
- What is impact of network topology on this
Tree Shape Reduction

- Logically form a reduction tree between nodes
- Aggregate to root then broadcast
Tree Shape Reduction
Tree Shape Reduction
Tree Shape Reduction

Question: What is Time Complexity of Tree Shape Reduction
Ring based Reduction

- Form a logical ring between nodes
- Streaming aggregation
Ring based Reduction
Ring based Reduction
Ring based Reduction
Ring based Reduction
Ring based Reduction

Each node have correctly reduced result of one segment!
This is called **reduce_scatter**
Ring based Reduction: Allgather phase
Ring based Reduction: Allgather phase
Ring based Reduction: Allgather phase
Ring based Reduction: Allgather phase

Question: What is Time Complexity of Ring based Reduction
Allreduce Libraries

- MPI offers efficient CPU allreduce
- dmlc/rabit: fault tolerant variant
- facebookincubator/gloo
- Parameter Hub: from UW

- NCCL: Nvidia’ efficient multiGPU collective
GPUDirect and RMDA

From Nvidia
NCCL: Nvidia’s Efficient Multi-GPU Collective

- Uses unified GPU direct memory accessing
- Each GPU launch a working kernel, cooperate with each other to do ring based reduction
- A single C++ kernel implements intra GPU synchronization and Reduction
Discussion: 4min

- What are advantages and limitations of Allreduce
- How to integrate allreduce with dependency scheduler?
Schedule Allreduce Asynchronously

Make use of mutation semantics!

\[ A = 2 \quad \rightarrow \quad \text{engine.push(} \]
\[ \quad \lambda: \text{A.data}=2, \]
\[ \quad \text{read}=[]], \text{mutate}=[\text{A.var}]\)]

\[ B = \text{comm.allreduce}(A) \quad \rightarrow \quad \text{engine.push(} \]
\[ \quad \lambda: \text{B.data}=\text{allreduce}(A.\text{data}), \]
\[ \quad \text{read}=[\text{A.var}], \text{mutate}=[\text{B.var}, \text{comm.var}]\))\]

\[ D = A \times B \quad \rightarrow \quad \text{engine.push(} \]
\[ \quad \lambda: D.\text{data}=A.\text{data} \times B.\text{data}, \]
\[ \quad \text{read}=[A.\text{var}, B.\text{var}], \text{mutate}=[D.\text{var}]\))\]
Distributed Gradient Aggregation, Remote Update

Many replicas of the same graph run in parallel

Update result on remote server and send updated results back

w1 -= lr * sum(g1 over replicas)
Parameter Server Abstraction

**Interface**

```python
ps.push(index, gradient)

ps.pull(index)
```
PS Interface for Data Parallel Training

\[
\text{grad} = \text{gradient}(\text{net}, \text{w})
\]

for epoch, data in enumerate(\text{dataset}):
    \[g = \text{net.run}(\text{grad}, \text{in} = \text{data})\]

\[\text{ps.push(weight\_index, g)}\]
\[\text{w} = \text{ps.pull(weight\_index)}\]
PS Data Consistency: BSP

- “Synchronized”
  - Gradient aggregated over all workers
  - All workers receive the same parameters
  - Give same result as single batch update
  - Brings challenges to synchronization
PS Consistency: Asynchronous
The Cost of PS Model: All to All Pattern

- Each worker talks to all servers
- Shard the parameters over different servers
- What is the time complexity of communication?
Integrate Schedule with Networking using Events

Asynchronous function that takes a callback from engine

```
lambda cb: ps.receive(A.data)
```

Receive Function

Engine

Use the callback to notify engine that data receive is finished

```
def event.on_data_received():
    # notify engine receive complete
    cb();
```
Model Parallel Training

- Map parts of workload to different devices
- Require special dependency patterns (wave style)
  - e.g. LSTM
Question: How to Write Model Parallel Program?

```python
for i in range(num_layers):
    for t in range(num_time_stamp):
        out, state = layer[i].forward(data[i][t], state)
        data[i+1][t] = out.copyto(device[i])
```

Scheduler tracks these dependencies
Discussion: What’s Special about Communication

Requirements
● Track dependency correctly
● Resolve resource contention and allocation
● Some special requirement on channel
  ○ Allreduce: ordered call

Most of them are simplified by a scheduler