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)

lambda: B.data = A.data + 1

Push the Operation to Engine

engine.push(
  Exec Function,
  read = [v1],
  mutate= [v2])
Recap: Example Scheduling

A = 2

B = A + 1

D = A * B
Data Parallelism

- Train replicated version of model in each machine
- Synchronize the gradient
How to do Synchronization over Network

This Lecture

fullc-forward w1
fullc-forward w2
softmax-forward
calc-backward
calc-backward
calc-backward
softmax-backward
log-loss

data
g1

g2

data

sync g1 update w1
sync g2 update w2

label

w1

w2
Distributed Gradient Aggregation, Local Update

Many replicas of the same graph run in parallel

\[
\text{G1} = \text{sum}(g1 \text{ over replicas})
\]

\[
w1 \leftarrow lr \times \text{G1}
\]
Allreduce: Collective Reduction

**Interface**

result = allreduce(float buffer[size])

**Running Example**

Machine 1

```python
comm = communicator.create()
a = [1, 2, 3]
b = comm.allreduce(a, op=sum)
assert b == [2, 2, 4]
```

Machine 2

```python
comm = communicator.create()
a = [1, 0, 1]
b = comm.allreduce(a, op=sum)
assert b == [2, 2, 4]
```
Use Allreduce for Data Parallel Training

grad = gradient(net, w)

for epoch, data in enumerate(dataset):
    g = net.run(grad, in=data)
    gsum = comm.allreduce(g, op=sum)
    w -= lr * gsum / 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 Diagram]

1

2

3

2

6

2

3

1

1

3
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
- 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!

\[
A = 2 \\
B = \text{comm.allreduce}(A) \\
D = A * B
\]

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

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

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

Many replicas of the same graph run in parallel

```
softmax-forward
```

```
log-loss
```

```
data
```

```
fullc-forward
```

```
w1
```

```
w2
```

```
fullc-backward
```

```
g1
```

```
g2
```

```
softmax-backward
```

```
fullc-backward
```

```
label
```

```
log-loss
```

Update result on remote server and send updated results back

```
w1 -= lr * sum(g1 over replicas)
```

Parameter Server
Parameter Server Abstraction

**Interface**

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

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

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

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

\[
\text{ps.push(weight_index, g)}
\]

\[
\text{w} = \text{ps.pull(weight_index)}
\]
PS Data Consistency: BSP

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

$t = 0$
- pull weight

$t = 1$
- push gradient (delay = 0)

$t = 2$
- push gradient (delay = 1)

$t = 1$
- pull weight

$t = 2$
- push gradient
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?
Defining a Receive Function

```
def event.on_data_received():
    # Notify engine receive complete
    cb();
```

Asynchronous function that takes a callback from engine

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

Use the callback to notify engine that data receive is finished.
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?

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