Hi!

CS 744: PYTORCH

Shivaram Venkataraman

Fall 2020

ADMINISTRIVIA

Assignment 2 out! Due 10/5 (Monday next week)

Bid on topics, submit group (1 sentences) – ~~Oct 5~~ Oct 1

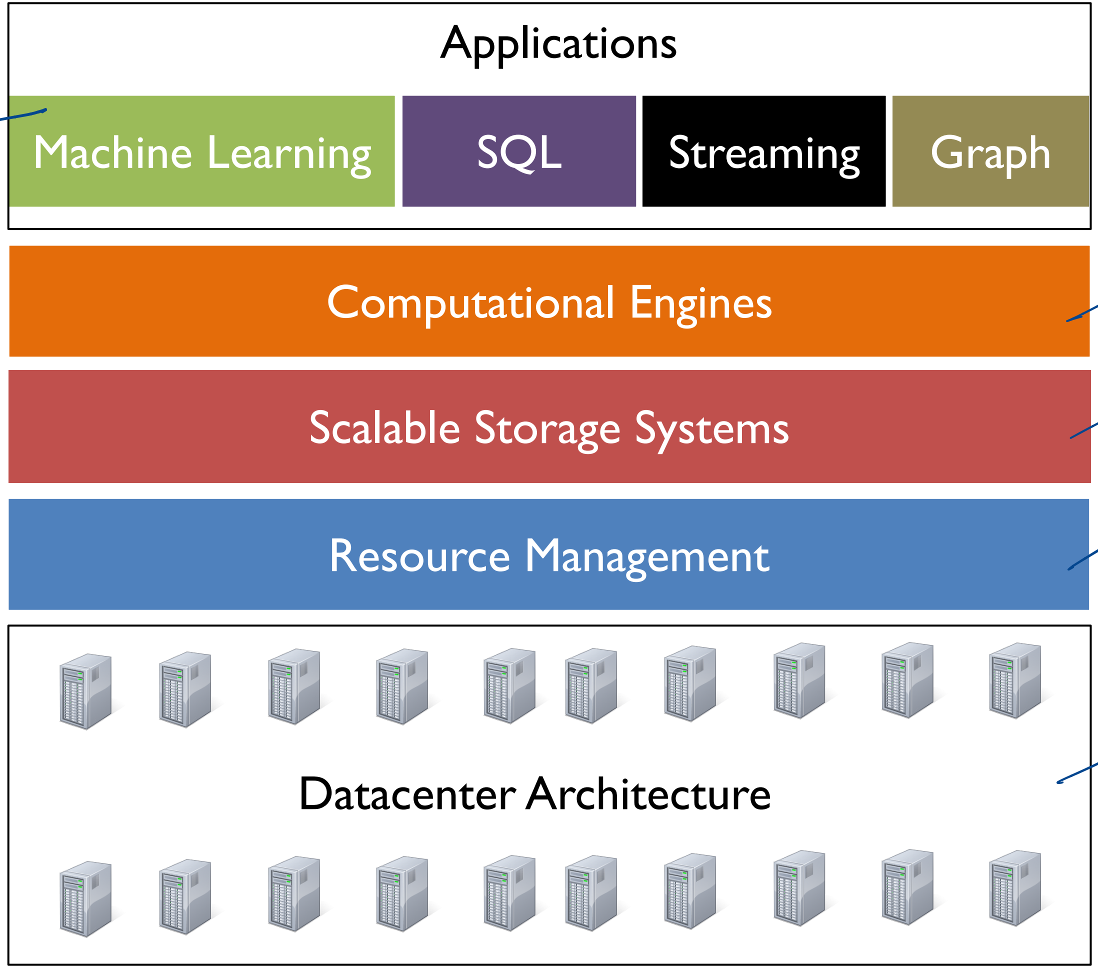
Piazza

Project Proposal (2 pages) – Oct 16

Introduction

Related Work

Timeline (with eval plan)



~ 2-3 weeks

Spark, MapReduce

GFS

Mesos
DRF

EMPIRICAL RISK MINIMIZATION

Supervised learning

↳ Given training data → and labels

Fit a model

$$\min_{w \in \mathbb{R}^d} \sum_{i=1}^{N} \boxed{f}(w, z_i) + P(w)$$

Function

Data (Examples)

Model

Regularization

DEEP LEARNING

→ Pytorch

weights of this layer

FC: 84 = [84 dim]

[input] * = [ ]

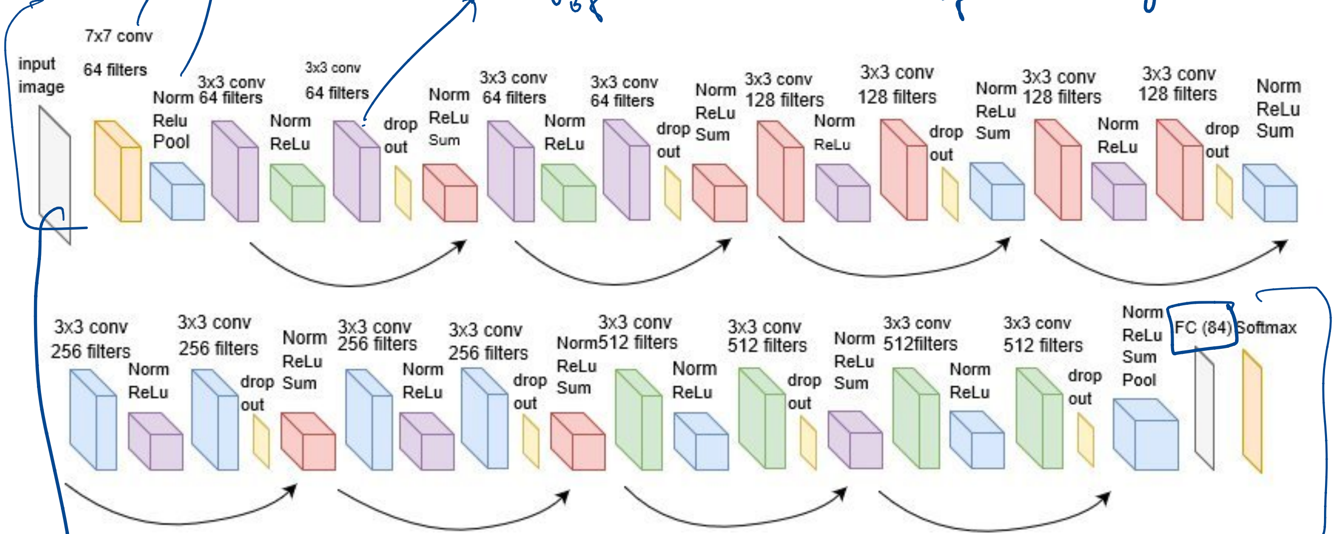
ResNet18

conv

pooling

layers of the model

composition of all layers



- Convolution
- ReLU
- MaxPool
- Fully Connected
- SoftMax

3 d matrix

width × height × 3

RGB image

layer = function(W_i, input)

weights or parameters of this layer

STOCHASTIC GRADIENT DESCENT

Good fit for Spark !?

many iterations !!

iterative computation

$$w^{(k+1)} = w^{(k)} - \underbrace{\alpha_k}_{\text{learning rate}} \underbrace{\nabla f(w^{(k)})}_{\text{gradient}}$$

update step

Initialize w

For many iterations:

did it in parallel

Loss = Forward pass(model) → $\| f(w, \text{input}) - b \|_2$ square loss

Gradient = backward(model) (Chain rule

Update model → model is shared → how do we parallelize

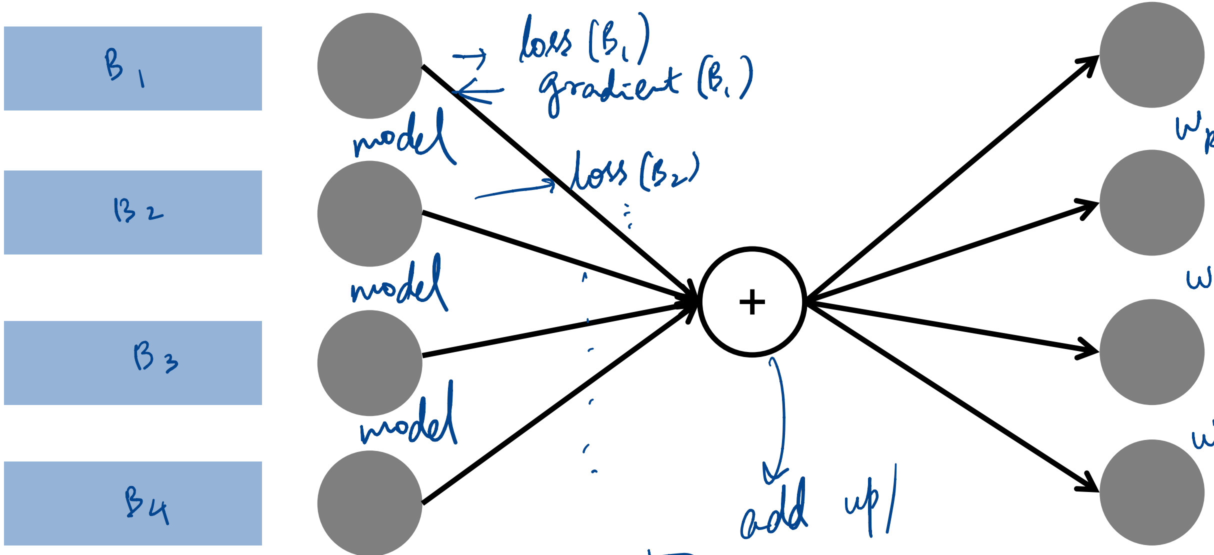
End

every iteration depends on previous

DATA PARALLEL MODEL TRAINING

Parallelize one iteration

model → replicate



64
machine 64
64
64

→ loss (B_1)
← gradient (B_1)

model
loss (B_2)
model
model

w_{k+1} → next iteration input

w_{k+1}
w_{k+1}
w_{k+1}

B = data points
= 256

forward pass

f (model, B)
f(model, B_1)
f (model, B_2)

add up/
average

update step that
takes all grads into account

X = n data points

Full gradient descent = All n,

$\boxed{\text{Mini batch grad descent = B}}$

Incremental grad des cent = 1

COLLECTIVE COMMUNICATION → MPI

g_1, g_2, g_3, g_4 → reduce: $g_1+g_2+g_3+g_4$

broadcast agg grads

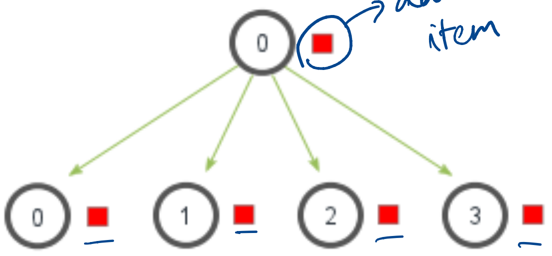
Either broadcast agg grad or broadcast the model w_{k+1}

send, recv ≡ RPC
P2P primitives

Broadcast, Scatter

Gather, Reduce

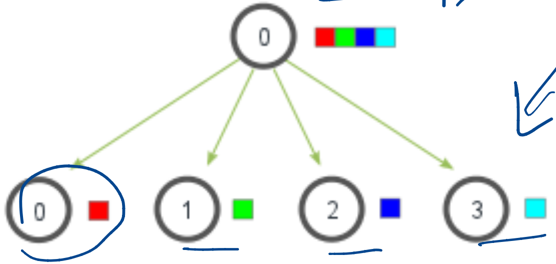
MPI_Bcast



data item

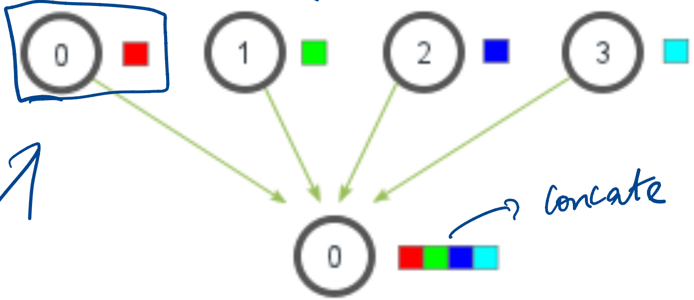
MPI_Scatter

vector [size 4]



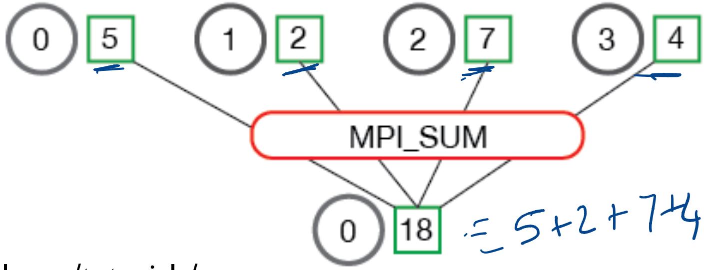
opposite

MPI_Gather (data, item, root)



concate

MPI_Reduce



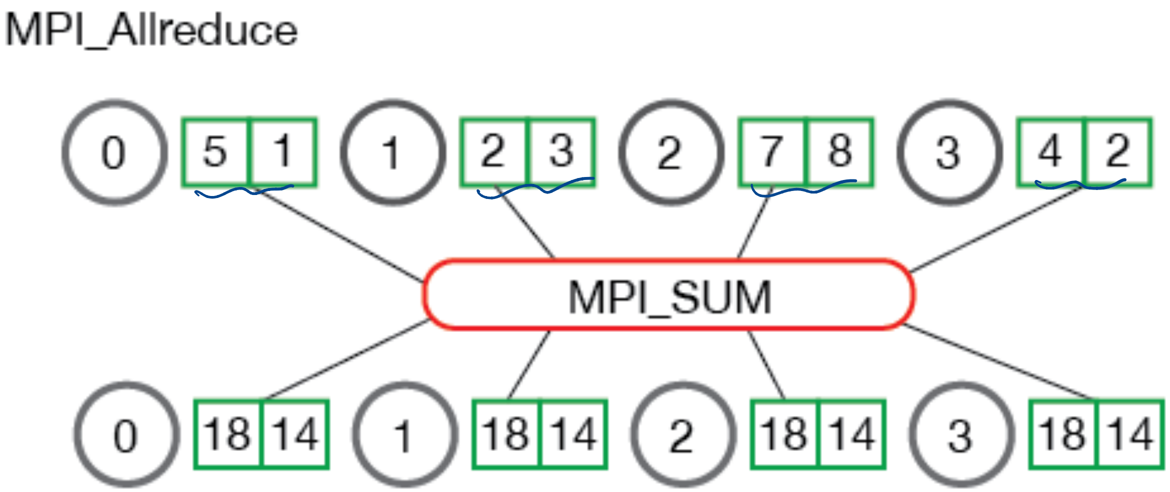
[5, 2, 7, 4]

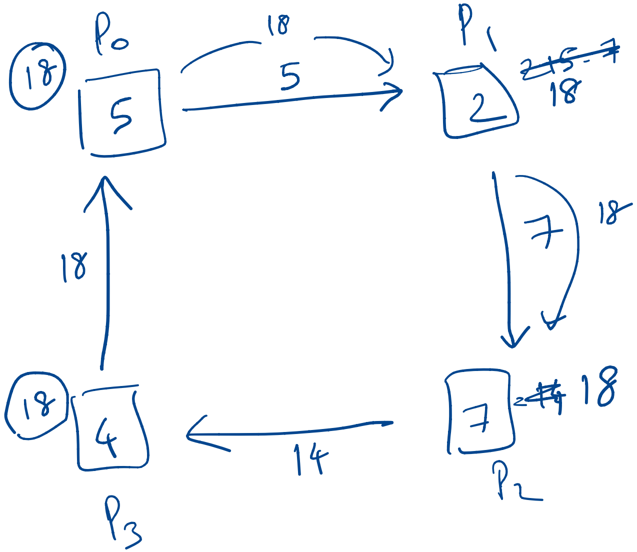
$= 5+2+7+4$

From https://mpitutorial.com/tutorials/

ALL REDUCE

Ring All Reduce





ends

From https://mpitutorial.com/tutorials/

DISTRIBUTED DATA PARALLEL API

```
# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

only line of code change

local model

- Non-intrusive

- Hooks to do optimizations in background

GRADIENT BUCKETING

60 M parameter

Why do we need gradient bucketing?

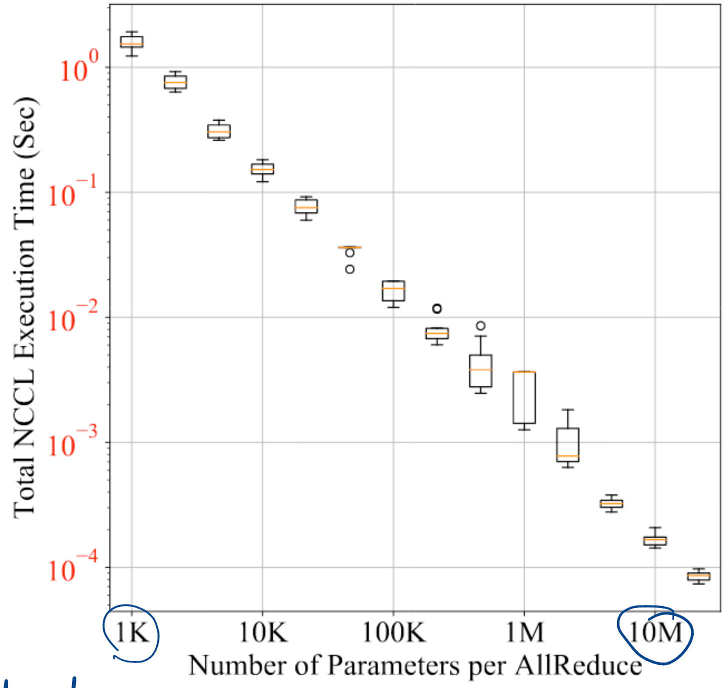
↳ Small tensor sizes lead to greater time for AllReduce

Every AllReduce = (latency cost fixed overhead) + bw cost how much data

→ Why not one big bucket

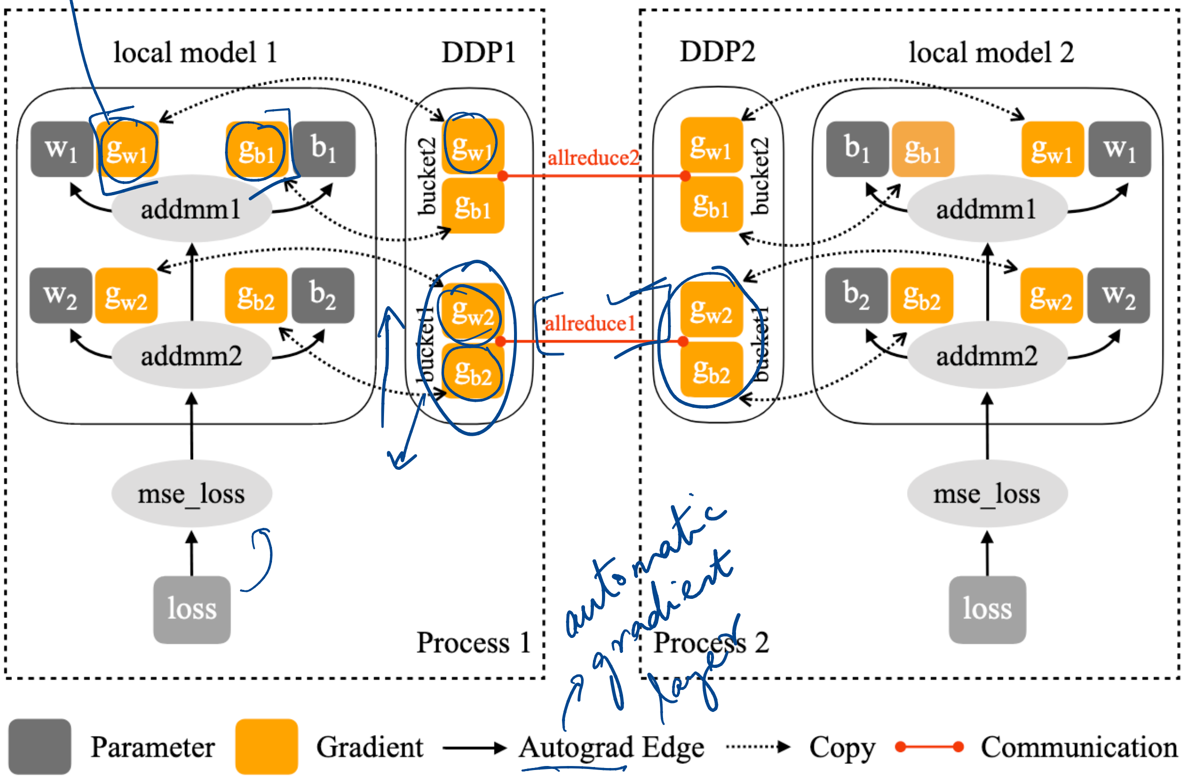
= Wait for all gradients to be ready

= Cannot overlap backward, AllReduce



GRADIENT BUCKETING + ALL REDUCE

parameter ≡ layers



As buckets become ready, we start All Reduce on them → async

In background, the gradient comp continues

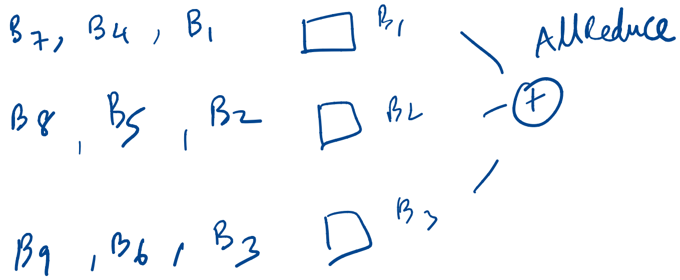
Buckets are defined by size ≡ 25MB

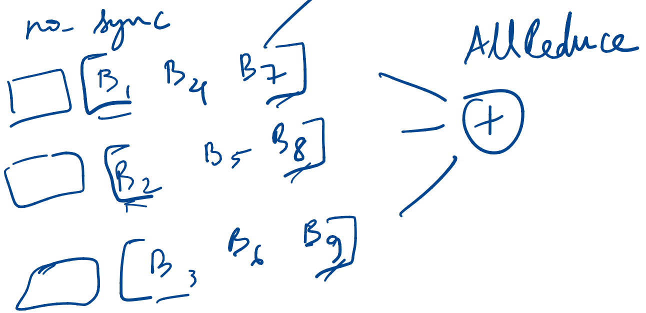
GRADIENT ACCUMULATION

```python
ddp = DistributedDataParallel(net)
with ddp.no_sync():
  for inp, exp in zip(inputs, expected_outputs):
    # no synchronization, accumulate grads
    loss_fn(ddp(inp), exp).backward()
# synchronize grads
loss_fn(ddp(another_inp), another_exp).backward()
opt.step()
```

extra parameter

device can only hold smaller subset in memory

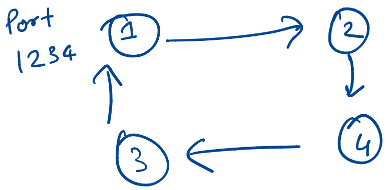




IMPLEMENTATION

AllReduce2 Port 1040

AllReduce1 Port 1234 kicked off

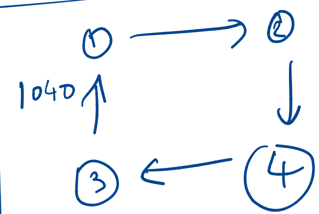


Bucket_cap_mb ≡ Parameter that is tunable

small → overhead

large → no overlap

~ middle ≈ 25 MB



Parameter-to-bucket mapping

↳ layer 2 20MB → bucket 1

layer 3 5MB

layer 4 → bucket 2 → 25MB

layer → bucket 3 → filled up 25MB

backward pass

Round-robin ProcessGroups

gradient

↳ math function

GPUs ≈ on a batch data

CPUs ≈

BREAKDOWN

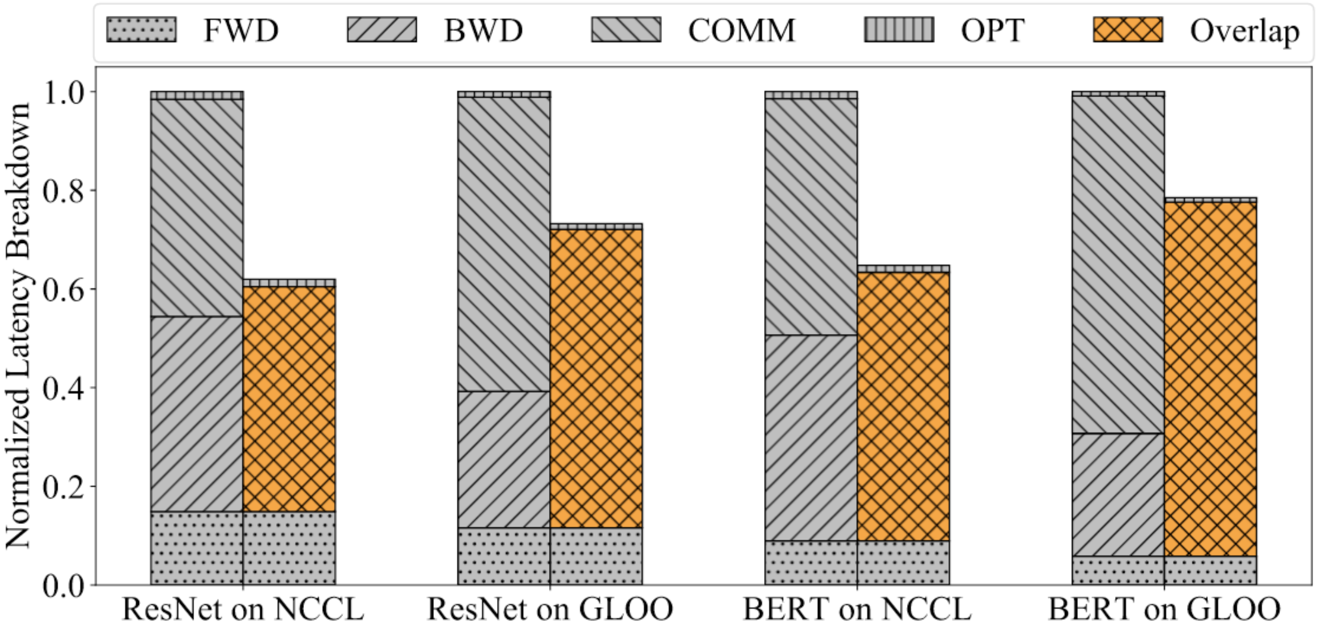


Figure 6: Per Iteration Latency Breakdown

SUMMARY

Pytorch: Framework for deep learning

DistributedDataParallel API

Gradient bucketing, AllReduce

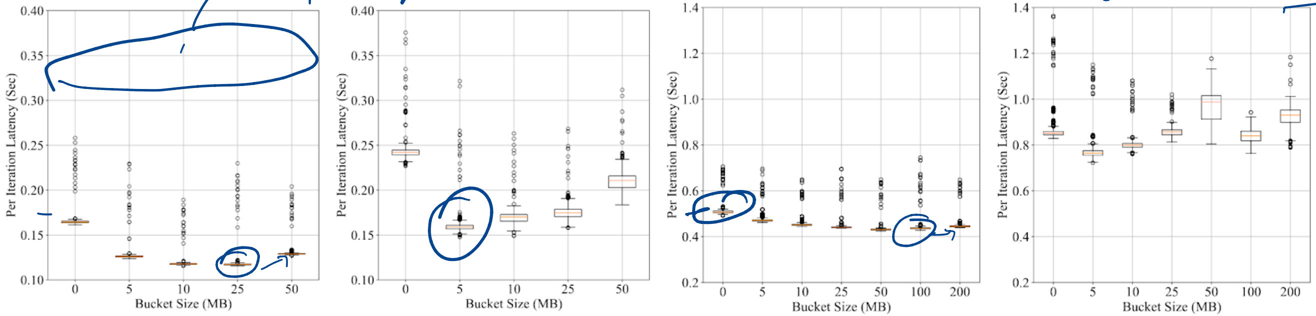
Overlap computation and communication

DISCUSSION

https://forms.gle/6xhVBNBhdzsJ6gBE6

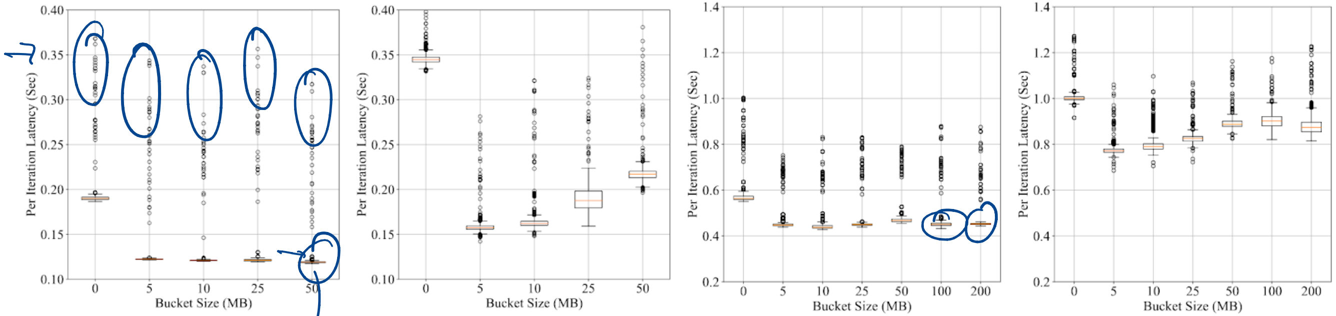
tail latency's better with 16 GPUs

BERT latencies are higher, And larger buckets work well for BERT



(a) ResNet50 on NCCL (b) ResNet50 on Gloo (c) BERT on NCCL (d) BERT on Gloo

Figure 7: Per Iteration Latency vs Bucket Size on 16 GPUs



(a) ResNet50 on NCCL (b) ResNet50 on Gloo (c) BERT on NCCL (d) BERT on Gloo

Figure 8: Per Iteration Latency vs Bucket Size on 32 GPUs

optimum

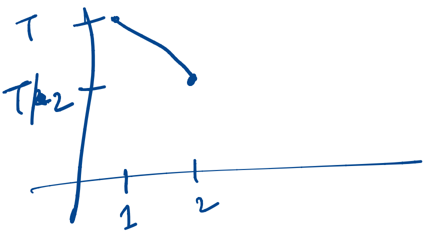
- Time for 32 GPUs ≃ time for 16 GPUs scales well!
- Optimal bucket size depends on Gloo or NCCL
- NCCL is more performant & less variance

Scales well !?

Strong scaling

$B = 256$

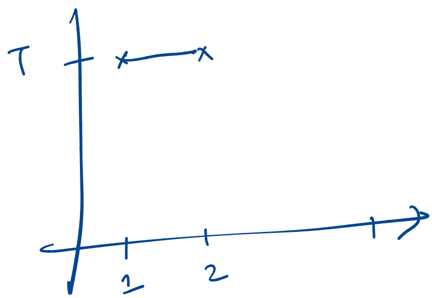
increase num GPUs



This paper

Weak scaling

$B_i = 64$, increase num GPUs



What could be some challenges in implementing similar optimizations for AllReduce in Apache Spark?

spark = "larger workloads" ?

Each worker node on spark had dataset

↳ spark needs to shuffle operation

|||

more expensive than NCCL Ring reduce

O ∇g_1

O ∇g_2

O ∇g_3

O ∇g_4

Tree Reduce using shuffles

Overlap compute / communication

Task completes → shuffle

Not all tasks active at same time

NEXT STEPS

Next class: PipeDream

Assignment 2 is due soon!

Project Proposal

Groups by Oct ~~5~~ 7

2 pager by Oct 16

Alloc bucket

layer0

layer[...]

copy

25MB

User program

scatter

Process Group API

scatter

NCCL

Gloo

P_1 link rank0

P_2

??

[∇g_1 ∇g_2 ∇g_3]

which link

network monitoring

MB/s

time

extra signal

bitmaps & other info !?

input = batches

layer 1 ∇g_1

layer 2 [∇g_2] comm

layer 3 [∇g_3] comm

loss

Reduce → brings to single machine

5

2 + 14 bcast

7