PyTorch Distributed: Experiences on Accelerating Data Parallel Training

Asher Noel and Jeremy Hsu
CS 265: Big Data Systems
Machine Learning is everywhere!
The world with more accurate models
Bigger Data Sets

+  

Larger Models

=  

More Accurate Models
We live in the Big Data era
Larger/Deeper Models

Regular Neural Network

Deep Neural Network
But training large models is computationally expensive!
How long did training GPT-3 take?

1969 Fastest Supercomputer (CDC 6600): 3 Billion Years

One NVIDIA Tesla V100 GPU: 355 Years

1,024 NVIDIA A100 GPU’s: 34 days
($3,440,640 on AWS p4 at $8.19/A100/hr)
What does this look like in reality?
how to implement an LSM-tree in C++
Your search - **how to implement an LSM-tree in C++** - matched old documents, but the search recommendation model is currently re-training.

Suggestions:

- Talk a walk.
- Try again next year.
- Try searching on Bing.
Your search - how to implement an LSM-tree in C++ - matched old documents, but the search recommendation model is currently re-training.

Suggestions:
- Talk a walk.
- Try again next week.
- Try searching on Bing.
Bigger Data Sets

+ 

Larger Models

= 

Distributed/parallel training required!
How to Parallelize

Model Parallelism

Data Parallelism

Machine 4

Machine 2

Machine 3

Machine 1

Machine 1

Machine 2

Machine 3

Machine 4
How to Parallelize

Model Parallelism

Data Parallelism

PyTorch Distributed
### Non-Distributed Training Iteration:

<table>
<thead>
<tr>
<th>Forward pass</th>
<th>Backward pass</th>
</tr>
</thead>
</table>

### Distributed Data Parallel (DDP) Training Iteration:

<table>
<thead>
<tr>
<th>Forward pass</th>
<th>Backward pass</th>
</tr>
</thead>
</table>
| Communication (Typically *AllReduce*)
|              |
|              |              |
|              |              |
At Meta:

“Within Facebook, a workload study in May 2020 showed that 60% of production GPU hours were spent on the PyTorch distributed data parallel package”
Challenges and Complications
Complication 1

Theory: $A \rightarrow B$

Practice:

- What topology?
- What version?
- Which GPU is down?
Complication 2

*PyTorch API on Outside:*

*PyTorch API on Inside:*

*More knobs!"*
Complication 3
PyTorch Distributed Optimizations on DDP

1. Gradient Bucketing
2. Overlapping Communication with Computation
3. Gradient Accumulation
PyTorch Distributed Optimizations

1. Gradient Bucketing
2. Overlapping Communication with Computation
3. Gradient Accumulation
Gradient Bucketing

local model 1

\[ w_1, g_{w1}, b_1 \]

\[ w_2, g_{w2}, b_2 \]

DDP1

\[ g_{w1}, g_{b1} \]

\[ g_{b1} \]

AllReduce

bucket2

DDP2

\[ g_{w1}, g_{b2} \]

\[ g_{b2} \]

AllReduce

bucket1

local model 2

\[ b_1, g_{b1}, g_{w1}, w_1 \]

\[ b_2, g_{b2}, g_{w2}, w_2 \]

mse_loss

loss
Gradient Bucketing
PyTorch Distributed Optimizations

1. Gradient Bucketing
2. Overlapping Communication with Computation
3. Gradient Accumulation
Without overlap:

| Forward pass | Backward pass | Communication |

With overlap:

| Forward pass | Backward pass | Communication |
PyTorch Distributed Optimizations

1. Gradient Bucketing
2. Overlapping Communication with Computation
3. Gradient Accumulation
### Without gradient accumulation:

<table>
<thead>
<tr>
<th>Batch i</th>
<th>Update grad</th>
<th>Batch i+1</th>
<th>Update grad</th>
<th>Batch i+2</th>
<th>Update grad</th>
<th>Batch i+3</th>
<th>Update grad</th>
</tr>
</thead>
</table>

### With gradient accumulation:

| Batch i | Batch i+1 | Batch i+2 | Batch i+3 | Update grad |
Experiments and Evaluation
Experimental Setup

- Exclusive 32 NVIDIA Tesla V100 GPUs across 4 servers
- Communication backends: NCCL (GPU-optimized) and Gloo (CPU-optimized)
- Large models: ResNet50 (image classification) and BERT (NLP)
Impact of Bucket Size

Larger Buckets Decrease Latency

(a) NCCL
(b) GLOO

(a) ResNet50 on NCCL
(b) ResNet50 on Gloo

“Bucket Size” Tradeoff
Impact of Overlapping Communication

![Graph showing normalized latency breakdown for different communication patterns and models.](image-url)
Impact of Gradient Accumulation

(a) ResNet50 on NCCL
(b) ResNet50 on Gloo
(a) Batch Size = 8
(b) Batch Size = 256
Experimental and Logical Gaps

- Did not test scalability on training BERT
- Needs more discussion on impact of skipping gradient synchronization
- Did not test impact of different network standards (InfiniBand, Ethernet, etc.)
Next Steps for PyTorch Distributed
## Current Limitations

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Synchronous</th>
<th>Asynchronous</th>
<th>Cross-Iteration</th>
<th>Intra-Iteration</th>
<th>Data-Parallel</th>
<th>Model-Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT DDP [9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PT RPC [6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Potential Next Steps

● Adding model parallel and/or asynchronous training implementations?
● Deeper integrations with PyTorch RPC?
● Adaptive parallelism based on model, network, hardware, and data set characteristics?
Appendix
AllReduce

(a) Tree AllReduce
(b) Round-robin AllReduce
(c) Butterfly AllReduce