Systems for Scalable ML - DNN Training

Guest Lecturer: Reyna Abhyankar (UCSD CSE)
Outline

• DNN basics

• Distributed DNN training
  • Data parallelism
  • Model parallelism (including PipeDream and Alpa)
Lecture 13
Distributed Training

Song Han
songhan@mit.edu
Neural networks

- Neural networks are made up of nodes or units, connected by links.
- Each link has an associated weight and activation level.
- Each node has an input function (typically summing over weighted inputs), an activation function, and an output.
Neural Network

\[ h_\theta(x) \]

Layer 1
(Input Layer)

Layer 2
(Hidden Layer)

Layer 3
(Output Layer)
Neural Network Training

- Feedforward and back propagation gradient descent


source: https://www.programmersought.com/article/45312377380/
Batch Gradient Descent

train data: $s_1, s_2, s_3, s_4, \ldots, s_n$

iteration 1

compute gradients

update weights

source: https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch
Stochastic Gradient Descent

source: https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch
Minibatch-Based SGD

source: https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch
Models are getting larger and larger

Better model always comes with higher computational cost (vision)

Figures from Once-for-all project page.
Models are getting larger and larger
Better model always comes with higher computational cost (NLP)

NLP’s Moore’s Law: Every year model size increases by 10x

NLP model size and computation are increasing **exponentially**

Models are getting larger and larger

Large Models Take Longer Time to Train

<table>
<thead>
<tr>
<th>Models</th>
<th>#Params (M)</th>
<th>Training Time (GPU Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>26</td>
<td>31</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>45</td>
<td>44</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>108</td>
<td>84</td>
</tr>
<tr>
<td>Turing-NLG 17B</td>
<td>17,000</td>
<td>TBA</td>
</tr>
<tr>
<td>GPT-3 175B</td>
<td>175,000</td>
<td>3,100,000</td>
</tr>
</tbody>
</table>

Measured on Nvidia A100

If without distributed training, a single GPU would take **335 years** to finish GPT-3!
Distributed Training is Necessary

- Developers / Researchers’ time are more valuable than hardware.
- If a training takes 10 GPU days
  - Parallelize with distributed training
  - 1024 GPUs can finish in 14 minutes (ideally)!
- The develop and research cycle will be greatly boosted

Let’s see a use case of distributed training!
Parallelism in Distributed Training

- Data Parallelism
- Model Parallelism
- Compare the Advantages and Disadvantages of Two Parallelism
Introduction to Distributed Training

Data Parallelism

ML Model

Training Dataset

Data Parallelism

GPU 1

GPU 2

... 

GPU N
Introduction to Distributed Training

Data Parallelism

ML Model

Split the data

Training Dataset

GPU 1

GPU 2

...  

GPU N
Introduction to Distributed Training

Data Parallelism

ML Model

Training Dataset

Same model across devices

GPU 1

GPU 2

GPU N
Dive into Data Parallelism

Scaling Distributed Machine Learning with the Parameter Server

Parameter Server
The central controller of the whole training process

Worker nodes
The hardware accelerators and dataset storage.

Two different roles in framework:

- **Parameter Server**: receive gradients from workers and send back the aggregated results
- **Workers**: compute gradients using splitted dataset and send to parameter server

Scaling Distributed Machine Learning with the Parameter Server. Mu Li et al. 2014

Figure credits from: Deep Gradient Compression. Lin et al. 2018
Problems with Parameter Server

When number of workers is small 😊

When number of workers is large 😞

The bandwidth requirement of parameter server grows linearly w.r.t number of workers.
Distributed Communication

Point-to-Point: Send and Recv

Send: n0 -> n3

Recv: n0 -> n3

Point-to-point communication: transfer data from one process to another

- Send & Receive are the most common distributed communication schemes.
- Implemented in Socket / MPI / Gloo / NCCL
Distributed Communication

Naive All-Reduce Implementation - Parallel Reduce

Pseudocode:

```
Parallel for i:=0 to N:
    Allreduce(work[i])
```

Time: $O(1) \leftarrow \text{improved}$

Bandwidth: $O(N^2) \leftarrow \text{worse}$

Perform **ALL** reduce operations simultaneously.
Distributed Communication

Recursive Halving All Reduce

Step 1 - Each node exchanges with neighbors with offset 1

Step 2 - Each node exchanges with neighbors with offset 2

Step 3 - Each node exchanges with neighbors with offset 4

For N workers, AllReduce finish in $\log(N)$ steps.

### All Reduce Implementations Comparison

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time</th>
<th>Peak Node Bandwidth</th>
<th>Total Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Server</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>All-Reduce - Sequential</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>All-Reduce - Ring</td>
<td>O(N)</td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td>All-Reduce - Parallel</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(N^2)</td>
</tr>
<tr>
<td>All-Reduce - Recursive Halving</td>
<td>O(lgN)</td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>

AllReduce with proper implementations reduce the peak bandwidth from O(N) to O(1) with little time overhead.
Parallelism in Distributed Training

- Data Parallelism
- Model Parallelism
- Compare the Advantages and Disadvantages of Two Parallelism
Data Parallelism Cannot Train Large Models

Though model parallelism has better device utilization, if train a super-large model (e.g., GPT-3)

Even the best GPU CANNOT fit the model into memory!
Introduction to Distributed Training

Model Parallelism

Figures credit from CMU 15-849 [Jia 2022]
Model Parallelism Designed for Large Model Training

In order to fit training into hardware, instead of splitting the data, model parallelism split the model

\[
\text{350GB / 8 cards} = 43.75G < 80G
\]

With model parallelism, large ML models can be placed and trained on GPUs.
Introduction to Distributed Training

Model Parallelism

Training Dataset

Single copy of data

GPU 1

GPU 2

GPU N

ML Model

Figures credit from CMU 15-849 [Jia 2022]
Introduction to Distributed Training

Model Parallelism

Training Dataset

ML Model

Split the model

GPU 1

GPU 2

…

GPU N

Figures credit from CMU 15-849 [Jia 2022]
Introduction to Distributed Training

Model Parallelism

Figures credit from CMU 15-849 [Jia 2022]
Introduction to Distributed Training

Model Parallelism

Figures credit from CMU 15-849 [Jia 2022]
Model Parallelism Workflow

Naive Implementation

(a). Training data flow

(b). Training timeline

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

(a). Training data flow

(b). Training timeline

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.
Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

(a). Training data flow

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

(a). Training data flow

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

(b). Training timeline

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

(a). Training data flow

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

But accelerators are significantly under-utilized during model parallelism!

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Naive Model Parallelism Suffers from Utilization

Only one device is computing at a time and others are waiting for it.

Theoretical utilization: 25% (low!)

Usual data parallelism utilization: ~75%

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Pipeline Parallelism

Gpipe: Easy Scaling with Micro-Batch Pipeline Parallelism

- Split a single batch to micro batches
  - [16, 10, 512] ->
    - [4, 10, 512]
    - [4, 10, 512]
    - [4, 10, 512]

- Motivation: model parameters are not changed during computation within a batch, thus we can pipeline computation and communication

(a). Naive model parallelism

(b). Pipeline parallelism

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Pipeline Parallelism

Micro-batch improves the device utilization

(a). Naive model parallelism

Utilization: (25%)

The more chunks (#num of micro batches), the higher of device utilization.

(b). Pipeline parallelism

Utilization: 57% (2.5x improvement)

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
GPipe Pipeline Bubbles

- GPipe synchronizes weights at each mini batch
- Pro: maintain the original maths => potentially more efficient SGD
- Con: pipeline bubble => inefficient GPU use

Figure 3: GPipe’s inter-batch parallelism approach. Frequent pipeline flushes lead to increased idle time.

Image from Narayanan eval. “PipeDream: Generalized Pipeline Parallelism for DNN Training” [SOSP 2019]
PipeDream: 1F1B Scheduling

- Workers alternate between forward and backward passes on minibatches
- Gradients used to update model immediately
- Storing multiple weight versions so that the backward and forward of the same minibatch
- Ensuring the same minibatch uses the same weight version across workers for forward and backward

Pros: no pipeline bubble, very efficient GPU utilization

Cons: need to store multiple versions of weights (and intermediate results), not the original maths

Image from Narayanan eval. “PipeDream: Generalized Pipeline Parallelism for DNN Training” [SOSP 2019]
PipeDream-2BW

- Only saving two versions of weights
- Gradient coalescing

Image from Narayanan eval. “Memory-Efficient Pipeline-Parallel DNN Training” [PMLR 2021]
Parallelism in Distributed Training

• Data Parallelism
• Model Parallelism

• Compare the Advantages and Disadvantages of Two Parallelism
Comparison between two parallelism

Data Parallelism:
- Split the data
- Same model across devices
- Easy to parallelize, high utilization
- N copies of model

Model Parallelism:
- Split the model
- Move activations through devices
- Hard to parallelize, load balancing issue
- Single copy of model

Figures credit from CMU 15-849 [Jia 2022]
Alpa

Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning

Lianmin Zheng\textsuperscript{1,*}, Zhuohan Li\textsuperscript{1,*}, Hao Zhang\textsuperscript{1,*}, Yonghao Zhuang\textsuperscript{4}, Zhifeng Chen\textsuperscript{3}, Yanping Huang\textsuperscript{3}, Yida Wang\textsuperscript{2}, Yuanzhong Xu\textsuperscript{3}, Danyang Zhuo\textsuperscript{6}, Eric P. Xing\textsuperscript{5}, Joseph E. Gonzalez\textsuperscript{1}, Ion Stoica\textsuperscript{1}

\textsuperscript{1}UC Berkeley \hspace{1em} \textsuperscript{2}Amazon Web Services \hspace{1em} \textsuperscript{3}Google \hspace{1em} \textsuperscript{4}Shanghai Jiao Tong University \hspace{1em} \textsuperscript{5}MBZUAI \& Carnegie Mellon University \hspace{1em} \textsuperscript{6}Duke University

*Equal contribution
What are System Challenges?

1. What if the input dataset is very large?

😊 Easy.
Use data parallelism: partition input data and replicate the model

2. What if the model is very large?

😢 Hard !!

Challenge: How to partition a computational graph?
Partition Computational Graphs

Strategy 1: Inter-operator Parallelism

Strategy 2: Intra-operator Parallelism

Trade-off

<table>
<thead>
<tr>
<th></th>
<th>Inter-operator Parallelism</th>
<th>Intra-operator Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Less</td>
<td>More</td>
</tr>
<tr>
<td>Device Idle Time</td>
<td>More</td>
<td>Less</td>
</tr>
</tbody>
</table>
Partition Computational Graphs

Multiple intra-op strategies for a single node
- Row-partitioned
- Column-partitioned
- Replicated

Pipeline the execution for inter-op parallelism

Combine Intra-op and Inter-op

Training Throughput of an MoE Model
Network Topology

Challenge: How to handle heterogeneous network topology?
A unified **compiler** that **automatically** finds and executes the best **Inter-op** and **Intra-op** parallelism for **large** deep learning models.
Alpa’s Main Contributions

- Two-level hierarchical space of parallelism techniques.
- Effective optimization algorithms at each level.
- Efficient compiler and runtime system implementation.
Computational Graph

Whole Search Space

Alpa Hierarchical Space

Inter-op Parallelism

Intra-op Parallelism
Alpa Compiler: Hierarchical Optimization

 Computational Graph

 Dynamic Programming

 Inter-op Pass

 Integer Linear Programming

 Intra-op Pass

 Cost Estimation

 Run-time Orchestration
Inter-op Pass

Computational Graph

\[
x \xrightarrow{conv} relu \xrightarrow{conv} add \xrightarrow{avgpool} matmul \xrightarrow{relu} matmul \xrightarrow{softmax}
\]

\[k_1, k_2, w_1, w_2\]
Inter-op Pass

Graph Partitioning

Stage 1
- k1
  - x
    - conv
    - relu

Stage 2
- k2
  - conv
  - add
  - avgpool

Stage 3
- w1
  - matmul
  - relu
  - matmul

Stage 4
- w2
  - softmax

or

Stage 1
- k1
  - x
    - conv
    - relu

Stage 2
- k2
  - conv
  - add
  - avgpool

Stage 3
- w1
  - matmul
  - relu
  - matmul

Stage 4
- w2
  - softmax

or

...
Inter-op Pass

Partitioned Computational Graph

Device Assignment
Inter-op Pass

Stage 1:
- k1
- x → conv → relu → conv → k2

Stage 2:
- avgpool
- matmul
- relu

Stage 3:
- w1
- matmul

Stage 4:
- w2
- softmax

Submesh Choice 1

Submesh Choice 2
Inter-op Pass

\[ T^* = \min_{s_1, \ldots, s_S} \left\{ \sum_{i=1}^{S} t_i + (B-1) \cdot \max_{1 \leq i \leq S} \{t_i\} \right\} \]

Solved together by Dynamic Programming

More details on the DP algorithm can be found in the paper.
Intra-op Pass

Integer Linear Programming Formulation

Decision vector
Parallel strategies of each operator

Minimize Computation cost + Communication cost

More details on the ILP algorithm can be found in the paper.
Compilation Time Optimization

Communication-aware operator clustering in ILP & DP

Early stopping in DP

Distributed Compilation

**Alpa Compilation Time:** < 40 min for the largest experiment.

- Can be further reduced by at least 50% with search space pruning.
Evaluation: Comparing with Previous Works

**GPT (up to 39B)**

![Graph showing throughput vs. number of GPUs for GPT (up to 39B)].

- **Best Manual System**
- **Alpa**

Match specialized manual systems.

**GShard MoE (up to 70B)**

![Graph showing throughput vs. number of GPUs for GShard MoE (up to 70B)].

- **Best Manual System**
- **Alpa**

Outperform the manual baseline by up to 8x.

**Wide-ResNet (up to 13B)**

![Graph showing throughput vs. number of GPUs for Wide-ResNet (up to 13B)].

- **Another Auto System**
- **Alpa**

Generalize to models without manual plans.

*Weak scaling results where the model size grow with #GPUs.*

*Evaluated on 8 AWS EC2 p3.16xlarge nodes with 8 16GB V100s each (64 GPUs in total).*
Case Study: Wide-ResNet Partition on 16 GPUs.
@alpa.parallelize: automatic model-parallel training

- Hierarchical view: inter-op and intra-op
- Match or outperform specialized systems
- Generalizes to new models