PipeDream: Generalized Pipeline Parallelism for DNN Training

Deepak Narayanan§, Aaron Harlap†, Amar Phanishayee★, Vivek Seshadri★, Nikhil R. Devanur★, Gregory R. Ganger†, Phillip B. Gibbons†, Matei Zaharia§

★ Microsoft Research † Carnegie Mellon University § Stanford University
Deep Neural Networks have empowered state of the art results across a range of applications...

- **Image Classification**
- **Machine Translation**
- **Speech-to-Text**
- **Game Playing**

Hello, my name is Deepak
...but first need to be trained!

$W$ optimized using standard iterative optimization procedures

$W = W - \eta \cdot \nabla W$

$x_i = \text{tiger}$

$y_i = \text{tiger}$

$\hat{y}_i = \text{lion}$
Background: DNN Training

Model training time- and compute- intensive!

\[ W = W - \eta \cdot \nabla W \]

\( y_i = \text{tiger} \)

\( \text{prediction} \)

\( \text{activations} \)

\( \text{loss}(y_i, \hat{y}_i) \)

\( \nabla W \)

\( \text{gradients} \)

\( \text{Weight parameters} \ W \)
Parallelizing DNN Training: Data Parallelism

Gradient aggregation using AllReduce

\[ \nabla W = \nabla W^1 + \nabla W^2 + \cdots + \nabla W^n \]

Despite many performance optimizations, communication overhead high!

8xV100s with NVLink (AWS)
PyTorch + NCCL 2.4
Parallelizing DNN training: Model Parallelism

Single version of weights split over workers

Activations and gradients sent between workers using peer-to-peer communication

Low hardware efficiency
PipeDream: Pipeline-Parallel Training

We propose **pipeline parallelism**, a combination of data and model parallelism with pipelining.

Pipeline-parallel training up to **5.3x faster** than data parallelism without sacrificing on final accuracy of the model.
Pipelining in DNN Training != Traditional Pipelining

• How should the operators in a DNN model be partitioned into pipeline stages?
  • Each operator has a **different computation time**
  • Activations and gradients need to be **communicated** across stages

• How should forward and backward passes of different inputs be scheduled?
  • Training is **bidirectional**
  • Forward pass followed by backward pass to compute gradients

• How should weight and activation versions be managed?
  • Backward pass operators depend on **internal state** ($W$, activations)
Outline

• Background and Motivation

• **Challenges for effective pipeline-parallel training**
  • Partitioning and load balancing operators across workers
  • Scheduling of forward and backward passes of different inputs
  • Managing weights and activation versions for effective learning

• Evaluation
How do we assign operators to pipeline stages?

- Desiderata #1: $t_1, t_2, t_3$ as close to each other as possible
  - Compute resources seldom idle $\rightarrow$ better hardware efficiency

- Desiderata #2: $t_{1\rightarrow2}^{\text{comm}}$ and $t_{2\rightarrow3}^{\text{comm}}$ minimized
  - Less communication $\rightarrow$ better hardware efficiency
How do we assign operators to pipeline stages?

Compute time = 2
Throughput = \((1 / 2) \times 2 = 1\)

Compute time = 1
Throughput = 1

Better load balancing across stages

Data-parallel communication small

Replication of stages helps load balance computation and reduce communication between workers

For some operators,
\[ \sum_i W_i < 2a_{\text{int}} \]
Example PipeDream configuration

Stages can have different replication factors
PipeDream Profiler and Optimizer

Input DNN \[\rightarrow\] Profiler \[\rightarrow\] Computational graph with profile \[\downarrow\] Optimizer \\
Deployment constraints such as number of accelerators, memory and interconnect characteristics

Determines a partitioning of operators amongst workers, while also deciding replication factors

Generalizes along many axes
- Hardware topologies
- Model structures
- Memory capacities of workers

See paper for details of algorithm!
Outline

• Background and Motivation

• **Challenges for effective pipeline-parallel training**
  • Partitioning and load balancing operators across workers
  • **Scheduling of forward and backward passes of different inputs**
  • Managing weights and activation versions for effective learning

• Evaluation
1F1B Scheduling

Workers alternate between forward and backward passes
- Workers always utilized
- Gradients used to update model immediately

To support stage replication, need to modify this mechanism slightly – see paper for details!
Outline

- Background and Motivation

- **Challenges for effective pipeline-parallel training**
  - Partitioning and load balancing operators across workers
  - Scheduling of forward and backward passes of different inputs
  - **Managing weights and activation versions for effective learning**

- Evaluation
Naïve pipelining leads to weight version mismatches

Naïve pipelining leads to **mismatch in weight versions**

\[
\begin{align*}
x_n & \quad \rightarrow \quad W_n & \quad \rightarrow & \quad y_n \\
W_{n+1} & \quad \vdots \\
\nabla x_n & \quad \leftarrow \quad W_{n+p} & \quad \leftarrow & \quad \nabla y_n
\end{align*}
\]

Forward pass

Backward pass

**Input n sees updates in backward pass not seen in the forward pass, leading to incorrect gradients**
1F1B Scheduling + Weight Stashing

Naive pipelining leads to mismatch in weight versions

Store multiple <weight, activation> versions
• Ensures same weight versions used in both forward and backward pass

\[ x_n \rightarrow W_n \rightarrow y_n \quad \text{Forward pass} \]

\[ W_{n+1} \]

\[ \vdots \]

\[ \nabla x_n \leftarrow W_n \leftarrow \nabla y_n \quad \text{Backward pass} \]

• Worst case memory footprint similar to data parallelism \( = n \cdot (|W| + |A|) / n \)
Outline

• Background and Motivation

• Challenges for effective pipeline-parallel training

• Evaluation
  • Setup
  • Comparison to Data Parallelism on Time-to-Accuracy
  • Communication Overhead of Pipeline Parallelism
  • Comparison to Model Parallelism and Hybrid Parallelism on Throughput
  • PipeDream’s Memory Footprint
Evaluation Setup

• Integrated PipeDream with PyTorch in ~3000 lines of Python code

• Integrated with PyTorch’s communication library
  • NCCL backend for Data Parallelism baselines
  • Gloo backend for PipeDream

• Experiments run on three different server types
  • Cluster A: 4xV100 GPUs, PCIe intra-server, and 10 Gbps inter-server (Azure)
  • Cluster B: 8xV100 GPUs, NVLink intra-server, and 25 Gbps inter-server (AWS)
  • Cluster C: 1xTitan X, and 40 Gbps inter-server (private)
PipeDream > Data Parallelism (DP) end-to-end

(a) Cluster-A.

(b) Cluster-B.

5.28x faster

2.46x faster
## PipeDream vs. Data Parallelism on Time-to-Accuracy

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Dataset</th>
<th>Accuracy Threshold</th>
<th># Servers × # GPUs per server (Cluster)</th>
<th>PipeDream Config</th>
<th>Speedup over DP</th>
<th>Epoch time</th>
<th>TTA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image Classification</strong></td>
<td>VGG-16 [48]</td>
<td>ImageNet [44]</td>
<td>68% top-1</td>
<td>4x4 (A) 2x8 (B)</td>
<td>15-1</td>
<td>5.28×</td>
<td>2.98×</td>
<td>5.28×</td>
</tr>
<tr>
<td></td>
<td>ResNet-50 [26]</td>
<td>ImageNet [44]</td>
<td>75.9% top-1</td>
<td>4x4 (A) 2x8 (B)</td>
<td>16</td>
<td>1×</td>
<td>1×</td>
<td>1×</td>
</tr>
<tr>
<td></td>
<td>AlexNet [37]</td>
<td>Synthetic Data</td>
<td>N/A</td>
<td>4x4 (A) 2x8 (B)</td>
<td>15-1</td>
<td>4.92×</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Translation</strong></td>
<td>GNMT-16 [55]</td>
<td>WMT16 EN-De</td>
<td>21.8 BLEU</td>
<td>1x4 (A) Straight</td>
<td></td>
<td>1.46×</td>
<td>2.2×</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4x4 (A) Straight</td>
<td></td>
<td>2.34×</td>
<td>2.92×</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2x8 (B) Straight</td>
<td></td>
<td>3.14×</td>
<td>3.14×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GNMT-8 [55]</td>
<td>WMT16 EN-De</td>
<td>21.8 BLEU</td>
<td>1x4 (A) Straight</td>
<td></td>
<td>1.5×</td>
<td>1.5×</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3x4 (A) Straight</td>
<td></td>
<td>2.95×</td>
<td>2.95×</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2x8 (B) Straight</td>
<td></td>
<td>1×</td>
<td>1×</td>
<td></td>
</tr>
<tr>
<td><strong>Language Model</strong></td>
<td>AWD LM [40]</td>
<td>Penn Treebank [41]</td>
<td>98 perplexity</td>
<td>1x4 (A) Straight</td>
<td></td>
<td>4.25×</td>
<td>4.25×</td>
<td></td>
</tr>
<tr>
<td><strong>Video Captioning</strong></td>
<td>S2VT [54]</td>
<td>MSVD [11]</td>
<td>0.294 METEOR</td>
<td>4x1 (C) 2-1-1</td>
<td></td>
<td>3.01×</td>
<td>3.01×</td>
<td></td>
</tr>
</tbody>
</table>
## PipeDream vs. Data Parallelism on Time-to-Accuracy

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Dataset</th>
<th>Accuracy Threshold</th>
<th># Servers × # GPUs per server (Cluster)</th>
<th>PipeDream Config</th>
<th>Speedup over DP</th>
<th>Epoch time</th>
<th>TTA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VGG-16 [48]</td>
<td>ImageNet [44]</td>
<td>68% top-1</td>
<td>4x4 (A) 2x8 (B)</td>
<td>15-1</td>
<td>5.28×</td>
<td>2.98×</td>
<td>2.46×</td>
</tr>
<tr>
<td></td>
<td>GNM-8 [55]</td>
<td>WMT16 EN-DE</td>
<td>21.8 BLEU</td>
<td>1x4 (A) 3x4 (A) 2x8 (B)</td>
<td>Straight</td>
<td>1.5×</td>
<td>2.95×</td>
<td>2.95×</td>
</tr>
<tr>
<td></td>
<td>AWD LM [40]</td>
<td>Penn Treebank [41]</td>
<td>98 perplexity</td>
<td>1x4 (A)</td>
<td>Straight</td>
<td>4.25×</td>
<td>4.25×</td>
<td>1×</td>
</tr>
<tr>
<td></td>
<td>S2VT [54]</td>
<td>MSVD [11]</td>
<td>0.294 METEOR</td>
<td>4x1 (C) 2-1-1</td>
<td></td>
<td>3.01×</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

_experiments on 4 different tasks: image classification, translation, language modeling, video captioning_
PipeDream vs. Data Parallelism on Time-to-Accuracy

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Dataset</th>
<th>Accuracy Threshold</th>
<th># Servers × # GPUs per server (Cluster)</th>
<th>PipeDream Config</th>
<th>Speedup over DP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VGG-16 [48]</td>
<td>ImageNet [44]</td>
<td>68% top-1</td>
<td>4x4 (A)</td>
<td>15-1</td>
<td>5.28× 5.28×</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2x8 (B)</td>
<td></td>
<td>2.98× 2.46×</td>
</tr>
<tr>
<td>Translation</td>
<td>GNMT-16 [55]</td>
<td>WMT16 EN-De</td>
<td>21.8 BLEU</td>
<td>4x4 (A)</td>
<td>Straight</td>
<td>1× 1×</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2x8 (B)</td>
<td></td>
<td>1× 1×</td>
</tr>
<tr>
<td></td>
<td>GNMT-8 [55]</td>
<td>WMT16 EN-De</td>
<td>21.8 BLEU</td>
<td>1x4 (A)</td>
<td>Straight</td>
<td>4.92× N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3x4 (A)</td>
<td></td>
<td>2.04× N/A</td>
</tr>
<tr>
<td>Language Model</td>
<td>AWD LM [40]</td>
<td>Penn Treebank [41]</td>
<td>98 perplexity</td>
<td>1x4 (A)</td>
<td>Straight</td>
<td>1.46× 2.2x</td>
</tr>
<tr>
<td>Video Captioning</td>
<td>S2VT [54]</td>
<td>MSVD [11]</td>
<td>0.294 METEOR</td>
<td>4x1 (C)</td>
<td></td>
<td>2.34× 2.92×</td>
</tr>
</tbody>
</table>

With the same number of GPUs, PipeDream up to 5.3x faster than Data Parallelism
## PipeDream vs. Data Parallelism on Time-to-Accuracy

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Dataset</th>
<th>Accuracy Threshold</th>
<th># Servers × # GPUs per server (Cluster)</th>
<th>PipeDream Config</th>
<th>Speedup over DP</th>
<th>Epoch time</th>
<th>TTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16 [48]</td>
<td>ImageNet [44]</td>
<td>68% top-1</td>
<td>4x4 (A) 2x8 (B)</td>
<td>15-1</td>
<td>5.28× 2.98×</td>
<td>N/A</td>
<td>4.92×</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>1× 1×</td>
<td></td>
<td>2.04×</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15-1</td>
<td>1.46× 2.34×</td>
<td></td>
<td>3.14×</td>
<td>3.14×</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Straight</td>
<td>1.5× 2.95×</td>
<td></td>
<td>1×</td>
<td>1×</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Straight</td>
<td>4.25× 3.01×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Straight</td>
<td>4.25× 3.01×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language Model</td>
<td>AWD LM [40]</td>
<td>Penn Treebank [41]</td>
<td>98 perplexity</td>
<td>1x4 (A)</td>
<td>1.5× 2.95×</td>
<td></td>
<td>1×</td>
<td>1×</td>
</tr>
<tr>
<td>Video Captioning</td>
<td>S2VT [54]</td>
<td>MSVD [11]</td>
<td>0.294 METEOR</td>
<td>4x1 (C)</td>
<td>2-1-1</td>
<td></td>
<td>3.01×</td>
<td></td>
</tr>
</tbody>
</table>

Optimizer recommends a number of different configurations like 15-1, Straight, and a fully data-parallel setup.
PipeDream reduces communication overhead

For many models, intermediate activations and gradients order of magnitude smaller than communication with Data Parallelism (DP)
Conclusion

• Model and data parallelism often suffer from high communication overhead and low resource utilization for certain models and deployments

• PipeDream shows pipelining can be used to accelerate DNN training

• Pipelining, when combined with data and model parallelism in a principled way, achieves end-to-end speedups of up to 5.3x

Code available at https://github.com/msr-fiddle/pipedream

https://cs.stanford.edu/~deepakn/