A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters

Yimin Jiang∗†, Yibo Zhu†, Chang Lan‡, Bairen Yi†, Yong Cui∗, Chuanxiong Guo†

*Tsinghua University, †ByteDance, ‡Google
Deep Neural Network (DNN)

**DNN Models**
- ResNet-50 (26M params)
- BERT-Large (387M params)
- GPT-3 (175B params)

**DNN Training**

<table>
<thead>
<tr>
<th>DNN model</th>
<th>Device</th>
<th>Computation time (FP + BP)</th>
<th>Estimated #iters. to converge</th>
<th>Estimated training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>1x Tesla V100</td>
<td>32ms + 64ms (batch size 32)</td>
<td>3.6M</td>
<td>96 hours</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>1x Tesla V100</td>
<td>339ms + 508ms (batch size 35)</td>
<td>8M</td>
<td>78.4 days</td>
</tr>
</tbody>
</table>

Need distributed training to scale out!
Data-parallel DNN Training

Forward Propagation (FP)

Input: $X_1$

$Y_1 = f(X_1)$

$\nabla W_1 = \frac{\partial \text{Loss}(Y_1, \hat{Y}_1)}{\partial W}$

Next Iteration

Update Parameter

$W' = W - f(\nabla W)$

Communication

$\nabla W = \nabla W_1 + \nabla W_2$

Input: $X_2$

$Y_2 = f(X_2)$

$\nabla W_2 = \frac{\partial \text{Loss}(Y_2, \hat{Y}_2)}{\partial W}$

GPU-1

GPU-2

This work covers these two stages
All-reduce and PS

• Architectures based on data parallelism: All-reduce and PS

- All workers are homogeneous
- Use collective communication to exchange the gradients

All-reduce

- Heterogeneous bipartite graph
- GPU workers + CPU servers
- Push gradients + pull parameters

Parameter Server (PS)
Existing Solutions Are Insufficient

VGG-16 performance with 32 GPUs. ByteScheduler is from [SOSP’19].

What are the problems of existing solutions?
P1: Sub-optimal Inter-machine Communication

Existing solutions fail to address the characteristics of heterogeneous clusters.
P2: Sub-optimal Intra-machine Communication

- The NIC bandwidth is close to PCIe’s bandwidth
- Existing solutions cause PCIe contention → NIC not saturated

Need to consider the intra-machine topology carefully
P3: The CPU Bottleneck

How to address the CPU bottleneck?

\[ \nabla W = \nabla W_1 + \nabla W_2 \]

- \( f \): Optimizer function
- \( W' = W - f(\nabla W) \)

- Exceed the limit
- 100Gbps
- e.g., 6-channel DDR4-2666 memory (1024Gbps)
  (Max. # memory access: 1024/100 ≈ 10)

CPU cannot match network rate

# Estimated Memory Access

RMSProp on CPU

Throughput (Gbps)
Our solution: BytePS

Problem 1
Sub-optimal inter-machine communication

An optimal inter-machine communication strategy that is generic and unifies all-reduce and PS

Problem 2
Sub-optimal intra-machine communication

Intra-machine optimization that accelerates communication inside GPU machines with diverse topology

Problem 3
The CPU bottleneck

Summation Service that aggregates gradients on CPUs and moves parameter update to GPUs
Outline

1. Background and Motivation

2. Design and Implementation

3. Evaluation
Opportunity and Design Goal

- **Opportunity:** There are spare CPUs and bandwidth in heterogeneous clusters
- **Design goal:** leverage any spare resources

![Heterogeneous Clusters diagram]

- GPU Machines
- CPU Machines

3 months trace from ByteDance Clusters

- Avg CPU utilization: 20~35%
- 20~45% run non-dist jobs (BW unused)
1. Inter-machine Communication

PS only uses links between GPU-CPU

All-reduce only uses links between GPUs

Best strategy: combine together

Bandwidth not saturated

How to partition the link workload?
Best Partition Strategy

- **x**: % traffic between GPU-CPU
- **y**: % traffic between GPU-GPU

Optimal partition strategy:

\[ x = \frac{2k(n-1)}{n^2 + kn - 2k} \quad \text{and} \quad y = \frac{n(n-k)}{n^2 + kn - 2k} \]

- \( n \): # of GPU machines
- \( k \): # of CPU machines

This strategy unifies PS and all-reduce, and is optimal with any # of CPU machines.

0 < \( k \) < \( n \): better than all-reduce and PS

\( k = 0 \): equal to all-reduce

\( k = n \): equal to PS
2. Intra-machine Communication

Existing solutions (e.g., MPI, NCCL)
Bottleneck link: \( M \times 2(8-1)/8 = 7M/4 \) traffic

Our solution: CPU-assisted Aggregation
Bottleneck link: \( M/4 \times 4 = M \) traffic

- CPU-assisted aggregation outperforms MPI/NCCL by 24% in theory
- Principle: avoid direct copy between GPUs under different PCIe switches
More Details in the Paper

• Solution for NVLink-based machines

• Design principles for different topology

• Optimality analysis

• Discussion about GPU Direct RDMA
3. Address the CPU Bottleneck

Throughput (Gbps)

**CPU-friendly**

- Gradient Summation
- Parameter Update

Heavy for CPU

CPU Summation is faster than network

**RMSProp on CPU**

- CPU
- CPU-MKL
- Network

**CPU is good at summation**

<table>
<thead>
<tr>
<th>Throughput (Gbps)</th>
<th>Summation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FP 16</strong></td>
<td>300</td>
</tr>
<tr>
<td><strong>FP 32</strong></td>
<td></td>
</tr>
</tbody>
</table>

Network bandwidth

\[ f: \text{Optimizer function} \]

\[ W' = W - f(\nabla W) \]
Summation Service can address the CPU bottleneck efficiently
System Architecture

- Intra-machine optimization
- Optimal Inter-machine strategy
- Address the CPU bottleneck
Usage and Deployment

BytePS supports TensorFlow, PyTorch and MXNet

Easy to use: compatible to most widely used APIs
- Horovod-alike APIs
- Native APIs for PyTorch and TensorFlow

BytePS has been deployed in ByteDance for CV/NLP tasks
Outline

1. Background and Motivation

2. Design and Implementation

3. Evaluation
Evaluation Setup

<table>
<thead>
<tr>
<th>Item</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV models</td>
<td>ResNet-50 (TF), VGG-16 (MXNet), UGATIT-GAN (PyTorch)</td>
</tr>
<tr>
<td>NLP models</td>
<td>Transformer (TF), BERT-Large (MXNet), GPT-2 (PyTorch)</td>
</tr>
<tr>
<td>Hardware</td>
<td>Each machine with 8x V100 GPUs (32GB) and a 100GbE NIC</td>
</tr>
<tr>
<td>Network</td>
<td>RoCEv2, full-bisection bandwidth</td>
</tr>
<tr>
<td>Baseline</td>
<td>Horovod, PyTorch DDP, Native-PS of TF and MXNet</td>
</tr>
</tbody>
</table>

- All experiments are performed on production clusters
- All chosen models are representative of production workloads
Inter-machine Communication

BytePS achieves near-optimal communication performance

With more CPU machines, BytePS achieves higher E2E performance

[Traffic microbenchmark] on 8x 1-GPU machines with different # CPU machines

[End-to-end] on 8x 8-GPU machines with different # CPU machines. Models: GPT-2 and UGATIT-GAN
Intra-machine Optimization

For PCIe-only topology, up to 20% gain

For NVLink-based topology, up to 18% gain

[PCIe-only] 8x 8-GPUs

[NVLink-based] 8x 8-GPUs
End-to-end Scalability (up to 256 GPUs)

BytePS outperforms All-reduce and PS by up to 84% and 245%, respectively
Breakdown of Performance Gains

+ Inter-machine: +66%

+ Intra-machine: +22%

+ Summation Service: +18%

VGG-16 (batch size / GPU = 32), 4x 8-GPUs machines, 2x CPU machines
Related Work

• Communication Acceleration
  • Gradient compression: 1-bit, top-k, Adacomp [AAAI’18]
  • Scheduling: ByteScheduler [SOSP’19], P3/TicTac [SysML’19]
  • Pipelined parallelism: PipeDream [SOSP’19]
  • Hierarchical all-reduce: BlueConnect [SysML’19]

• New hardware or architecture for DNN training
  • New AI chips: TPU, Habana → BytePS design is generic and not GPU-specific
  • In-network all-reduce: Infiniband switch ASIC (SHArP)
  • In-network PS: P4 switch [HotNets’17]
  • Rack-scale dedicated servers: PHub [SoCC’18]
Conclusion

BytePS: A unified system for distributed DNN training acceleration
- Optimal inter-machine communication
- Topology-aware intra-machine optimizations
- Address the CPU bottleneck with Summation Service

Deployed in ByteDance for CV and NLP training tasks

Open-sourced at https://github.com/bytedance/byteps
- Support TensorFlow, PyTorch, MXNet
Thank you

Yimin Jiang
jymthu@gmail.com