Optimizing Large-scale Deep Learning by Minimizing Resource Contention for Data Processing

Sarunya Pumma,1 Danielle Buono,2 Fabio Checconi,1 Xinyu Que,2 and Wu-chun Feng1
1Virginia Tech; {sarunya, wfeng}@vt.edu 2IBM T.J. Watson; {dubono, fchecco, xque}@us.ibm.com

Introduction & Motivation

Large-scale Deep Learning

Generating high-accuracy DL models requires large amounts of data to train large deep neural networks (DNN). Supercomputers can speed up the training process from months to minutes via high-order method based machine learning similar to human brain. Large training data is needed to build a model for recognizing complex patterns, Goal: near-human-level perception

What is Deep Learning (DL)?

High-order method based machine learning similar to human brain

Quick & Easy Way to Scale Deep Learning

Using the state-of-art DL software

Load Imbalance in Data Processing of Deep Learning

- **Data processing** of deep learning includes multiple components (as shown in the diagram below):
  - In the current state-of-the-art deep learning frameworks, these components are normally running asynchronously with each other
  - Without coordination, these data processing components compete with each other for resources (e.g., CPU cycles, memory bandwidth, network bandwidth, and I/O engine) causing **load imbalance** in the parallel training environment
  - Such load imbalance can have a significant impact on the scalability of the training especially in large scale

Scalability Analysis & Horovod Background Thread Investigation

Deep Learning frameworks:

- TensorFlow
- PyTorch
- Caffe
- Chainer

Data processing in our DL environment:

- Neural network training (on GPUs)
- Communication (on CPUs)
- Model checking (on CPUs)
- Model compression (on CPUs)
- Data prefetching and packing

HVD (Horovod):

- **Horovod** is a communication plugin for DL frameworks, e.g., TensorFlow
- Internally relies on MPI/NDVIA NCLL/IBM DDL for performance
- Provides nonlocking asynchronous operations
- DL framework’s graph scheduling threads execute Horovod’s background thread to queue tensor transfer requests in Horovod’s request queue
- Horovod’s background thread manages the process to handle tensor transfer request’s progress and completion

Horovod Tensor Ordering

- The background thread manages direct tensor transfer requests based on the order that requests are put into the request queue
- Once requests are within the same background thread, they might not be transferred at the same time
- As it uses collective APIs (MPI/NCLL/IBM DDL), it has to do the extra tensor ordering before issuing tensor transfer to prevent deadlock

Horovod Tensor Ordering Caching

- Horovod attempts to avoid tensor ordering by storing tensor in a response cache for future use
- Communication between background threads are still needed because they do not know which tensors are ready to be transferred

Introduction & Motivation

**Supercomputers can speed up the training process from months to minutes via high-order method based machine learning similar to human brain. Large training data is needed to build a model for recognizing complex patterns, Goal: near-human-level perception.**

**Load Imbalance in Data Processing of Deep Learning**

- **Data processing** of deep learning includes multiple components (as shown in the diagram below):
  - In the current state-of-the-art deep learning frameworks, these components are normally running asynchronously with each other
  - Without coordination, these data processing components compete with each other for resources (e.g., CPU cycles, memory bandwidth, network bandwidth, and I/O engine) causing **load imbalance** in the parallel training environment
  - Such load imbalance can have a significant impact on the scalability of the training especially in large scale

**Scalability Analysis & Horovod Background Thread Investigation**

**Deep Learning frameworks:**

- TensorFlow
- PyTorch
- Caffe
- Chainer

**Data processing in our DL environment:**

- Neural network training (on GPUs)
- Communication (on CPUs)
- Model checking (on CPUs)
- Model compression (on CPUs)
- Data prefetching and packing

**HVD (Horovod):**

- **Horovod** is a communication plugin for DL frameworks, e.g., TensorFlow
- Internally relies on MPI/NDVIA NCLL/IBM DDL for performance
- Provides nonlocking asynchronous operations
- DL framework’s graph scheduling threads execute Horovod’s background thread to queue tensor transfer requests in Horovod’s request queue
- Horovod’s background thread manages the process to handle tensor transfer request’s progress and completion

**Horovod Tensor Ordering**

- The background thread manages direct tensor transfer requests based on the order that requests are put into the request queue
- Once requests are within the same background thread, they might not be transferred at the same time
- As it uses collective APIs (MPI/NCLL/IBM DDL), it has to do the extra tensor ordering before issuing tensor transfer to prevent deadlock

**Horovod Tensor Ordering Caching**

- Horovod attempts to avoid tensor ordering by storing tensor in a response cache for future use
- Communication between background threads are still needed because they do not know which tensors are ready to be transferred