# Literature Review

Group Members: Tom Ballas, Zhengqi(Drago) Dong, Arpan Jain

Deep Learning is a subset of machine learning that attempts to build neural network models that can learn relevant data representations through a large dataset. Increases in the availability of large datasets and computation power has caused a rise in the popularity of Deep Neural Networks (DNNs), and these models have been applied to many different areas including image, speech, health, and many more. However, with the increase in available data, models must become more complex which increases the memory and computational requirements. This has caused researchers to analyze the possibility of utilizing parallelization strategies within DNNs. The approaches to distributed DNN training include data, model, and hybrid parallelism.

In the first paper [1], the author proposed the Data Parallelism approach that utilizes the optimized communication primitives in Message Passing Interface (MPI) to distribute DNN training across multiple machines/GPUs on TensorFlow. Data parallelism involves replicating a model across multiple devices and then distributing the data across the devices. Figure 1 on the following page illustrates the process of data parallelism. Each device learns the model independently without modifying the standard backpropagation algorithm, and the weights and biases are synchronized/averaged across all devices via allReduced operation. The allReduced operation is provided by many MPI interfaces, e.g., OpenMPI. Since MPI is specialized for Supercomputers communication interface, the overall communication overhead can be greatly reduced compared to the sockets interface.

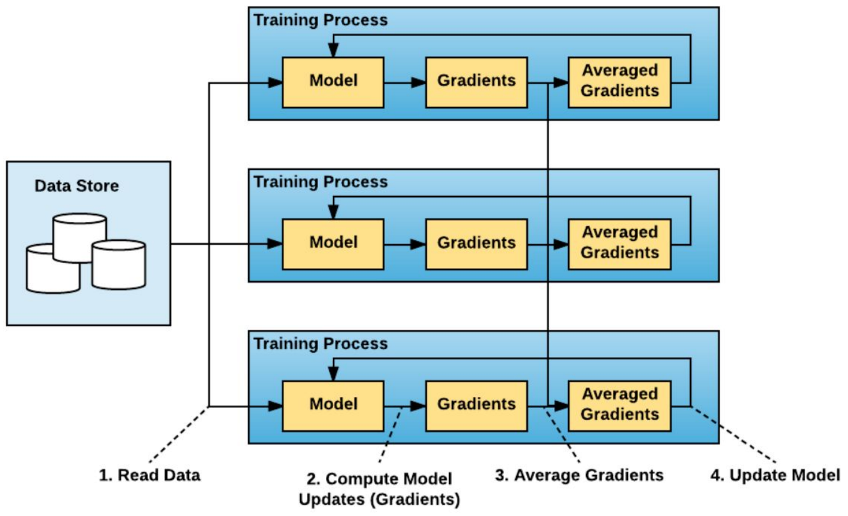


Figure 1: Illustration of basic data parallelism algorithm [2]

As rapid growth of the training dataset and the machine learning model, and the availability of high-performance multi-core GPUs, the performance of primitive versions of TensorFlow cannot satisfy the need of modern business anymore. In the second paper [2], the author introduced a new approach to the distributed deep learning library on TensorFlow, called Horovod. There are several major chances to the old distributed TensorFlow packages. 1) The parameter server approach in the standard distributed TensorFlow packages was replaced by the NCLL's ring-all reduced approach that was originally introduced by Baidu [6]. 2) They converted code into a stand-alone package that is compatible with various releases of TensorFlow. 3) The Horovod library supports the models to run not only on a single GPU, but also on multiple GPUs. 4) The Horovod library support distributed training on various deep learning framework (e.g., PyTorch, MXNet) with minimal modification to the code. As the results shown in Figure 2, the less number of communication between worker and parameter server scaled the training efficiency of original distributed TensorFlow on multi-core GPUs.

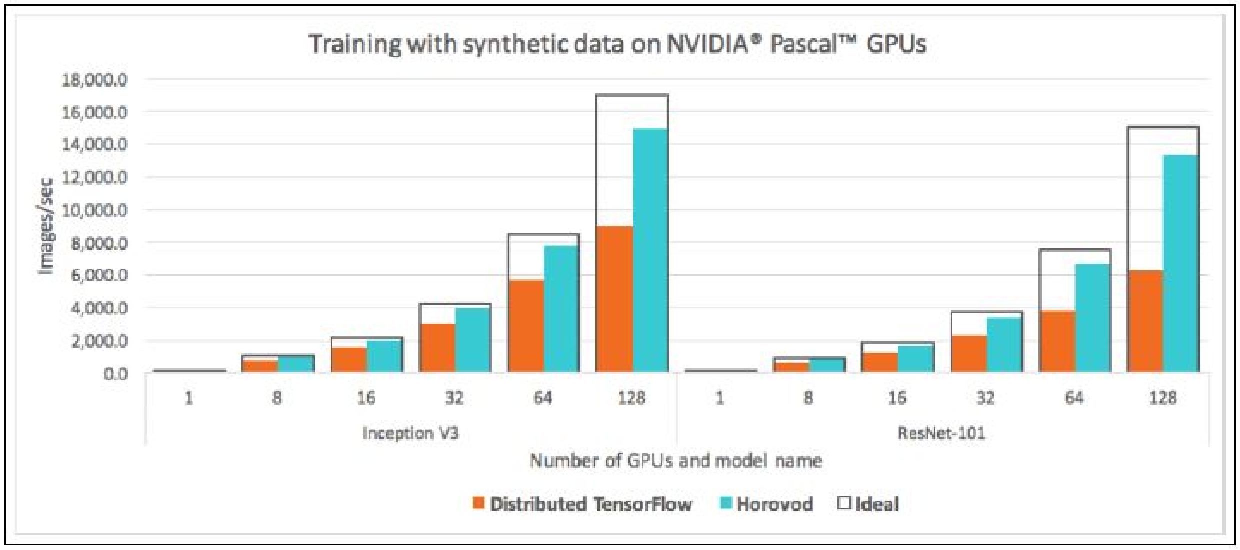


Figure 2: A benchmarks comparison for multi-core GPU scaling performance on TensorFlow [2]

Although approaches like Horovod offer good performance, they must be used for models that reside in the memory of a CPU/GPU. However, there are some larger and deeper models that require more memory than what is available on a single CPU/GPU. This has caused a need for model parallelism which involves distributing a model across multiple GPUs. This means that a layer or multiple layers will be trained on each device. With larger models creating a need for model parallelism approaches, researchers at Google sought to develop an approach for model parallelism to train a large deep learning model. However, one of the major challenges facing them was that traditional model parallelism suffers from under-utilization of GPU resources since at each time only one GPU can perform computations. Figure 3 on the following page illustrates this concept.

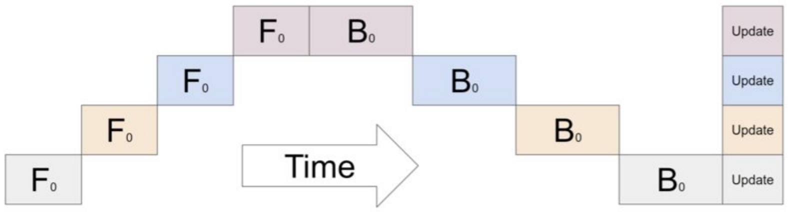
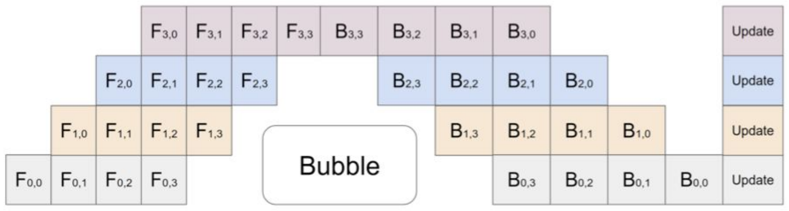


Figure 3: Illustration of the traditional model parallelism approach with both forward ($F_0$) and backward passes ($B_0$) on multiple devices (each device is shown as a row). This shows how the resources are underutilized across devices. [3]

To solve this challenge, the researchers wrote an article [3] where they proposed a new approach called pipeline parallelism. As is shown in Figure 4, for pipeline parallelism, the data that is to be trained on each device is split into multiple batches. This allows for a device to train a small batch and pass to the next device for computation, instead of having to train on all the data before the next device can perform training. The researchers compiled their pipeline parallelism approach into a library called GPipe. This approach exhibited success as GPipe was used to train a model that achieved a top-1 accuracy of 84.4% on ImageNet-2012.



$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$
Bubble
Update

Figure 4: Illustration of the pipeline parallelism approach with both forward ($F_{device,\ batch}$) and backward passes ($B_{device,\ batch}$) on multiple devices (each device is shown as a row). This shows how this approach is able to utilize resources better than traditional model parallelism. [3]

While this approach was successful, it did not exhibit the ability to train state-of-the-art DNNs like ResNet(s) on High-Performance Computing (HPC) systems. In addition, it does not utilize hybrid parallelism which combines data and model parallelism. To solve these challenges, researchers at Ohio State published an article [4] that proposed a hybrid parallelism approach for HPC systems. Their approach called HyPar-Flow utilizes Horovod for data parallelism and implements a pipelining approach for model parallelism. In addition, HyPar-Flow takes advantage of HPC optimizations. HyPar-Flow exhibited success in that it allowed for a hybrid parallelism approach that saw up to a 1.6X speedup over Horovod-based data-parallel training.

However, despite the success of HyPar-Flow, it is unable to operate on multiple GPUs and instead works on multiple CPUs. In addition, the pipeline parallelism approach is limited in performance compared to data parallelism, and the length of the pipeline is limited by the batch size. To mitigate the pipeline parallelism issues and allow for GPU use, researchers at Ohio State published a paper [5] where they propose GEMS (GPU Enabled Memory Aware Model Parallelism System). As was discussed with traditional model parallelism, there is a memory vacuum during forward and backward propagation. Within GEMS, to utilize this memory, two model replicas can be created. The first replica is trained as normal, and the second replica uses the free memory and compute to train in an inverted manner. Following forward and backward passes for both replicas, the parameters of each model are synchronised similar to data parallelism. This entire proposed process is illustrated in Figure 5.

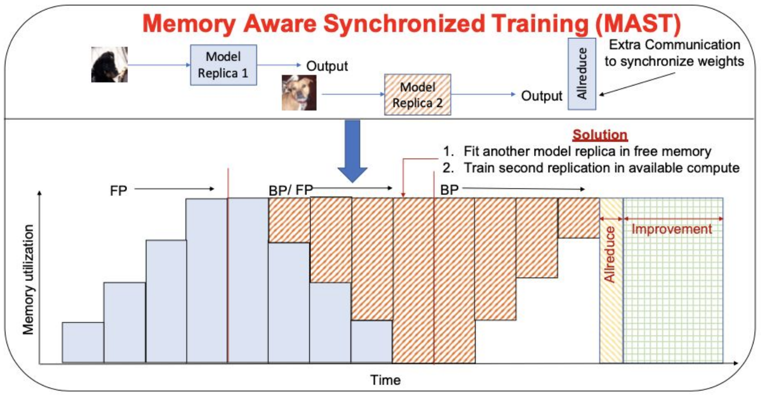

Memory Aware Synchronized Training (MAST)
Model Replica 1
Output
Model Replica 2
Output
Allreduce
Extra Communication to synchronize weights
Solution
1. Fit another model replica in free memory
2. Train second replication in available compute
FP
BP/ FP
BP
Memory utilization
Allreduce
Improvement
Time

Figure 5: Illustration of GEMS parallelism approach [5]

The GEMS approach exhibited success in that it was used to train a 1000-layer ResNet-1k model with 97.32% scaling-efficiency and reduce the training time for ResNet-110-v2 from seven hours to 28 minutes.

- **Research Hypothesis**

There five papers showing different approaches for parallelising the training of deep neural networks set the backdrop for our project. Within this research project, we seek to extend the support for model parallelism to the PyTorch and extend the MASTER approach proposed in [5] to PyTorch. In [5], authors tested their proposed approach on ResNet networks and showed improvement in terms of performance and time to reach the accuracy. In this project, we will explore new architectures like UNET and optimize the MASTER to increase the performance on multi-gpu systems. Unlike ResNet architecture, UNET has large skip connections therefore realizing model parallelism and hybrid parallelism is a challenging task. Using the MASTER approach, we wish to accelerate the training on large image sizes. Images / samples per second metric is a well known metric to estimate the training on large datasets for a given architecture. Therefore, we will test the performance of the proposed approach using Images per second metric and compare it with traditional approaches to show the improvement in training time.

## References:

[1] Vishnu, Abhinav, Charles Siegel, and Jeffrey Daily. "Distributed tensorflow with MPI." arXiv preprint arXiv:1603.02339 (2016).

[2] Sergeev, Alexander, and Mike Del Balso. "Horovod: fast and easy distributed deep learning in TensorFlow." arXiv preprint arXiv:1802.05799 (2018).

[3] Huang, Yanping, et al. “GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism”. arXiv preprint arXiv:1811.06965 (2019).

[4] Awan, Ammar Ahmad, et al. "HyPar-Flow: Exploiting MPI and Keras for Scalable Hybrid-Parallel DNN Training using TensorFlow." arXiv preprint arXiv:1911.05146 (2019).

[5] A. Jain , Ammar Awan , et al. “GEMS: GPU Enabled Memory Aware Model Parallelism System for Distributed DNN Training.” Nov 2020

[6] Andrew Gibiansky. Bringing HPC techniques to deep learning. http://research.baidu. com/bringing-hpc-techniques-deep-learning, 2017. [Online; accessed 6-December-2017].