Hybrid Model Parallel and All-Reduce on Distributed GPU Clusters

Gyeongchan Yun*
UNIST
rugyoon@unist.ac.kr

Seungmin Lee*
UNIST
tmdalslsm@unist.ac.kr

Changmin Yi*
UNIST
ulistar93@unist.ac.kr

ABSTRACT

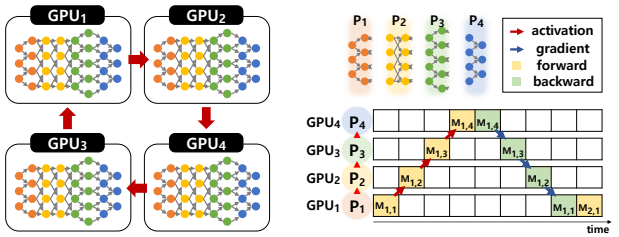
As the size and complexity of Deep Neural Network(DNN) models have been increasing, a variety of parallel techniques such as data parallelism, model parallelism, and hybrid parallelism have been proposed. In this study, we focus on the communication behavior of the above parallelism. We make the observation that inter-node communication for parameter synchronization drastically degrades the training performance in data parallel training.

We propose a hybrid parallel strategy that integrates model and data parallel training by grouping multiple GPUs as a worker for model parallel training and adopts allreduce to synchronize parameters with other workers. This strategy mediates inter-node communication overhead of both data and model parallel training. We evaluate our system on CNN models for image classification compared to Horovod which is state-of-the-art communication schemes. The experimental results show that our system is effective to apply to distributed training with high inter-node communication.

1 INTRODUCTION

Recently, the size and complexity of Deep Neural Network(DNN) models have continuously been increasing in order to improve the accuracy and quality of models. Therefore, it is necessary to train DNN on multi GPUs to address a problem that requires a long time as training needs intensive computations and large datasets. Thus, there have been a variety of parallel techniques to accelerate training on multiple GPUs: *Data parallelism* has a replica of the entire model and divides dataset across each GPU [2, 15], *model parallelism* holds a partition of the model among multiple GPUs [5, 13, 14], *hybrid parallelism* combines these two techniques [9, 10, 13]. Furthermore, due to the low GPU utilization of model parallelism, techniques to apply pipelining has recently been proposed [8, 19]. In this study, at first, we analyze the communication overhead of parallel training in distributed GPU clusters. We propose a system that integrates model and data parallel training to minimize inter-node communication overhead.

There are two widely-used communication architectures in data parallel training - Parameter server and Allreduce. A parameter server [15] was initially proposed for machine learning, for example, sparse logistic regression, it also has been commonly used in distributed DL training. However, this centralized architecture has a communication bottleneck with the high communication cost on the central nodes. Also, DL models with dense parameters are not suitable for the parameter server [12]. Accordingly, a decentralized scheme called Allreduce [1] is adopted to distributed DL systems. This method takes a peer-to-peer system so that reduce the network bottleneck with decreasing the amount of communication. Since Allreduce shows better performance compared to the parameter server [16, 25], the Allreduce architecture has been adopted recently in distributed DNN training.



(a) Data Parallel (b) Model Parallel

Figure 1: Parallelism on multiple GPUs

The existing frameworks support two communication modes in data parallel training: (1) synchronous communication with parameter server or Allreduce - Bulk Synchronous Parallel (BSP) [26] (2) asynchronous communication with parameter server - ASynchronous Parallel (ASP) [24, 26]. Also, there is a synchronization strategy called Stale Synchronous Parallel (SSP) [7] which compromises the middle side of synchronous and asynchronous training. Some systems [11, 15] exploit SSP where the staleness of model parameter between the fastest worker and the slowest one cannot exceed more than a predefined staleness bound. In this study, we only focus on synchronous data parallel training. We adopt Allreduce to communication scheme for data parallel training which is state-of-the-art communication for CNN models. We can observe that the performance of data parallel training significantly suffers from inter-node communication in Figure 2.

In this study, we divide the model on multiple GPUs into different nodes and group these as a worker. The worker performs Allreduce to synchronize parameters with other workers. In our system, Allreduce is done through the intra-communication and only a small amount of data is transferred through inter-communication by the model partitioner.

As the performance and use of distributed DL systems has been gradually increased through various parallel and synchronization strategies, GPU clusters are also growing in size. In line with this trend, using a heterogeneous GPU cluster is inevitable due to the short release cycle of new GPU architectures and expensive cost [11]. However, most prior distributed deep learning system only considers homogeneous GPU cluster that each GPU has similar computation and network capacity [4, 27, 28]. Therefore, we build a model partitioning algorithm that can reflect both homogeneous and heterogeneous GPU clusters.

*Authors contributed equally to this research.

With the above configuration, what kind of data will use the interconnection or intra-connection is defined. Gradients are only communicated through intra-connection. Updating parameters could be done fast using high bandwidth. For inter-connection,a small amount of data such as intermediate results transferred because the model is divided across nodes. It has relatively low bandwidth and usually causes communication overhead, but it is efficiently utilized in this case.

Our system starts with profiling the cluster configuration such as computation capability and memory size of GPUs, network bandwidth by doing virtual iteration. Then, resource allocator makes a worker group composed of some GPUs based on profiling data. Finally, the model partitioner divides the model based on cluster configuration and model information.

In our experiments, we use CNN models for image classification with ImageNet [6] because distributed Tensorflow CNN benchmarks [26] support Allreduce including horovod and provide throughput (images/second) and accuracy.

Our contributions are as follows:

- We propose hybrid strategy merging data parallel and model parallel training which mediates communication between intra-node and inter-node connections.
- We can efficiently utilize network bandwidth in topology-aware manner.
- We achieve higher training performance than data parallel training with state-of-the-art communication schemes, Allreduce, and the possibility of training a large model through model parallel training.

2 BACKGROUND

2.1 Distributed Model Parallel Training

As the size and complexity of DNN have been growing rapidly and the memory size of GPU has the limitation, some giant DNN model can not be loaded to single GPU for training. To facilitate the training of the large DNN model, model parallel training is to divide the model and assign partition of the model to multiple GPUs. The main problem of model parallel training is the low utilization of GPUs due to forward and backward dependency and small size of the computation load of fragmented models assigned to each GPU shown in Figure 1 (b). From the perspective of communication overhead in model parallel training, there is no parameter synchronization between multiple GPUs, but activation outputs and gradients transfer between operators.

2.2 Distributed Data Parallel Training

As the size of the neural network and datasets has been increasing, training a model on multiple GPUs becomes common. As shown in Figure 1 (a), data parallel training learn the same replicated model with multiple GPU workers and mainly use two architectures to share trained parameters between workers - Parameter Server and Allreduce.

Parameter server is a centralized architecture consisting of the server group and several worker group. The server group maintains global shared parameters and communicates with worker group to update or broadcast the shared parameters. However, this centralized architecture has a communication bottleneck with the high communication cost on the central nodes. Also, DL models with dense parameters are not suitable for the parameter server [12]

All reduce is the method presented in [1]. It has recently received attention because of its good performance by utilizing network efficiently. Allreduce consists of peer-to-peer communication and each worker delivers a portion of the gradients to nearby workers. By doing so, Allreduce reduces the amount of data that needs to be transferred and mitigates the centralized network bottlenecks [25]. Nevertheless, Allreduce still suffers from communication overhead when synchronizing gradients, because Allreduce communication is only possible when all workers complete their application simultaneously [19].

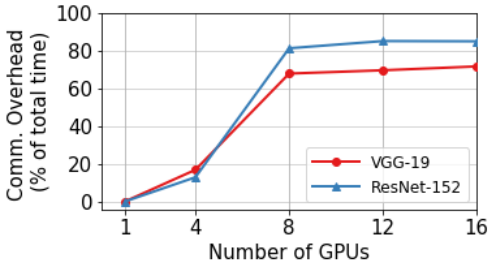


Figure 2: Communication overhead of data parallel training using multi-GPU server instances with 4 GPUs.

3 MOTIVATION

3.1 Intra- and Inter-Communication Overhead

Data parallel training should perform parameter synchronization among multiple workers for each step. Figure 4 (a) shows Allreduce communication behavior of Horovod using 16 GPUs. In multi-GPU server clusters, the workers on the same node come up with intra-node communication and do inter-node communication with the worker on the other node. Figure 2 shows the fraction of time spent by communication stalls in data parallel training on GPU clusters which consists of 4 GPUs in each node. Data parallel training with 4 GPUs only takes place intra-node communication. However, as shown in Figure 2, the communication overhead drastically increases at data parallel training using 8 GPUs where it also happens inter-node communication. We can observe that the performance of data parallel training significantly suffers from inter-node communication. Furthermore, it is inavoidable as the scale of data parallel training increases.

Model parallel training doesn't need to perform parameter synchronization, but consider data transfer between operators assigned to multi-GPU. Highly depending on the number and placement of partitioned layers of model for model parallel training, inter-node communication has increased numerous times, making it inefficient. However, model partitioning for optimal model parallel training is not trivial.

3.2 Effect of Batch Size in Model Training

It is known that training performance increases up to some point with larger batch size [3]. In data parallel training on especially

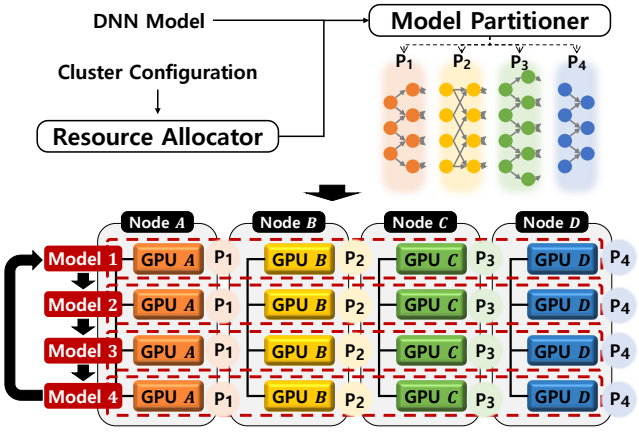


Figure 3: System architecture

heterogeneous GPU, the DNN model with high batch size can not be loaded to the GPU which has a small size of memory. In our measurement, the batch size of 32 in VGG-19 and 16 in ResNet-152 respectively is the maximum size to fit into GeForce RTX 2060 [20] which has the memory size of 6GB. However, in model parallel training, we have advantage in GPU memory because one DNN model is divided into multiple GPUs. It enables training with larger batch size than usual data parallel training and finally leads to training performance improvement.

4 SYSTEM DESIGN

In this section, we give a design of the system that we propose. As we mention in Section 3, both model and data parallel training especially suffer from inter-node communication as the number of using GPUs increases in distributed GPU clusters. We propose our system that integrates model and data parallel training by grouping multiple GPUs as a worker for model parallel training, denoted by *MP worker*. Also, each MP worker adopts Allreduce to synchronize parameters with other MP workers.

4.1 Resource Awareness

As mentioned previously, which data is transmitted by inter-node communication is very critical to model training performance. Therefore, resource allocator distinguishes what kind of network interfaces exist in the cluster configuration and composes the worker group named *MP worker* to minimize inter-node communication overhead. Also, for intra-node communication, resource allocator considers GPU order in a MP worker to share the weight of the model partition only through the intra-node communication.

4.2 Model Partitioning

How to divide a model for model parallel training is difficult and is another subject of research in itself [5, 14]. In this paper, our model partitioner finds the model partition to minimize 1 iteration time composed of estimated computation time and calculated communication time. The total computation time is represented as sum of the computation time of each partition P in the given GPU device D. The computation time for each partition is estimated from the profiled data. The communication time is calculated by the activation output size of last layer in each partition with theoretical network bandwidth. It is multiplied by 2 because the training is bi-directional - forward and backward pass. The equation for the objective function is as follows:

$$Objective = \sum_{P\in\bar{P},D\in\bar{D}} \left(\sum_{l\in P} Comp(l,D)\right) + 2 * \sum_{P\in\bar{P}} \frac{OutputSize(l_{max(x|l_x\in P)})}{NetworkBW}$$

$$when\ \bar{P} = \{P_1, P_2, \ldots\}, \bar{D} = \{D_1, D_2, \ldots\}$$

Through minimizing the objective function, the model partitioner finds an appropriate partition for the MP worker given by the resource allocator. For example, in figure 7, the model partitioner devides the model after the layer which has small activation output size such as `mpool` layer.

	Architecture	CUDA Core	Memory Size(GB)
TITAN RTX	Turing	4608	24
TITAN V	Volta	5120	12
GeForce RTX 2060	Turing	1920	6
Quadro P4000	Pascal	1792	8

Table 1: Heterogeneous GPUs

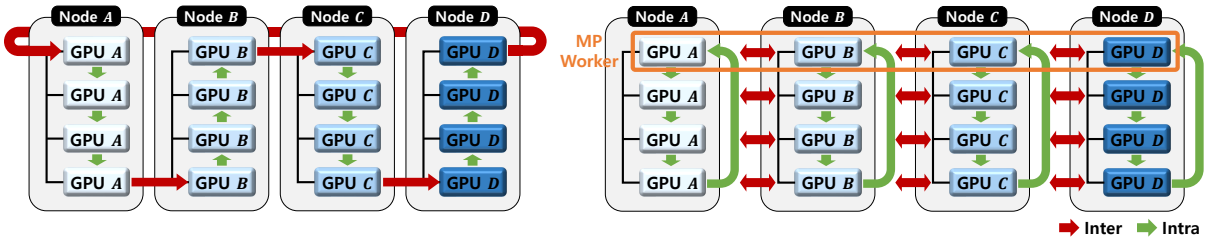
4.3 MP worker with Allreduce

As shown in Figure 3, several MP workers in the distributed GPU clusters update their parameters with their forward and backward pass respectively, and then perform parameter synchronization with other MP workers through Allreduce communication. Figure 4 (b) shows the overall communication behavior in these MP + Allreduce. When running forward and backward pass within the MP worker, bi-directed inter-node communication occurs. The parameter synchronization with Allreduce is performed through intra-node communication because the partition of model parameters in the multi-MP worker is in the same node. The main challenge in this architecture is to reduce the amount of bi-directed inter-node communication. This system executes hybrid parallel strategy that reduces inter-node communication between DP and MP in situations where inter-node communication between DP and MP is unavoidable as described in Section 3.1.

5 IMPLEMENTATION

Model Partitioner We used CPLEX, the MIP(mixed integer programming) problem optimizer, to find a proper partition. Also, `tf.device` provided by Tensorflow is used for placing the computation graph to specific device on the particular servers. Model partitioner leverages `tf.device` to assign the partition of layers of DNN model graph to the device group in MP worker.

MP worker We implemented MP worker by modifying Tensorflow CNN benchmarks. The method `create_MP_worker()` groups multiple GPUs to each MP worker with the number of MP worker provided by the resource allocator. Then, the number of worker and worker configuration in TF CNN benchmarks is changed to treat the MP worker. In figure 3, TF CNN benchmarks creates 4 DNN model computation graphs and performs Allreduce with 4 workers.



(a) Horovod Allreduce

(b) MP + Allreduce

Figure 4: Communication Behavior of Horovod and MP + Allreduce

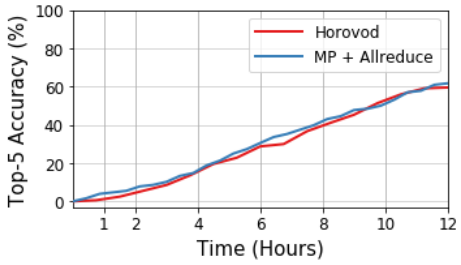


Figure 5: Top-5 accuracy vs time for VGG-19 using 16 GPUs.

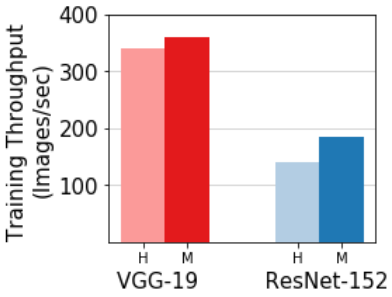


Figure 6: Training throughput on VGG-19 and ResNet-152.

6 EVALUATION

6.1 Experiment Setup

GPU cluster configuration In heterogeneous GPU cluster, we use four nodes with two Intel Xeon Octa-core E5-2620 v4 processors (2.10 GHz) connected via Infiniband (56 Gbps). Each node has 64 GB memory and 4 GPUs. Each node is configured with a different type of GPUs: TITAN RTX [22], TITAN V [23], GeForce RTX 2060 [20], and Quadro P4000 [21]. The detailed hardware spec of each GPU is explained in Table 1. Thus, the total number of GPUs in our cluster is 16. Each GPU is equipped with PCIe-3×16 (15.75 GB/s). Ubuntu 16.04 LTS with Linux kernel version 4.4 is used.

Frameworks We select Tensorflow v1.12 and CNN benchmarks provided by Tensorflow [26]. As baselines, we adopt data parallel training with Horovod [25] on Tensorflow with OpenMPI v4.0.0 for Allreduce.

Models and Datasets We evaluate two CNN models for image classification on ImageNet-1K [6] dataset: VGG-19 and ResNet-152. VGG and ResNet is relatively large models among benchmarks.

Training Methodolgy We use the maximum batch size that fits in all GPUs in our clusters. In data parallel training, the batch size of VGG-19 is 32 and ResNet-152 is 16 respectively. In MP + Allreduce, we use batch size of 128 for both VGG-19 and ResNet-152. Other hyper-parameters including optimizer, learning rate, precision are used the default value in TF CNN benchmarks and are not different in our system and baseline.

6.2 Comparision to Data Parallel Training

Performance We evaluate training performance by throughput (the number of processed images per second). We run 100 iterations after warm-up phases and average all throughput measured at each iteration. Figure 6 shows training throughput of VGG-19 and ResNet-152 on our system and Horovod respectively. In Figure 6, "H" denotes Horovod, "M" denotes MP + Allreduce, that is our system. The performance of our system of VGG-19 and ResNet-152 improves by 6 % and 31% respectively compared to that of Horovod. In figure 2, we can observe that inter-node communication overhead of ResNet-152 is higher than that of VGG-19 in data parallel training. Based on our results, our system is efficient in the case of high inter-node communication overhead in data parallel training. This result also supports that inter-node communication overhead of MP worker in our system is less than that of Horovod.

Model Convergence We evaluate convergence by top-5 validation accuracy versus time for VGG-19. We train Horovod and MP + Allreduce for 12 hours and evaluate top-5 accuracy. We don't modify parameter synchronization algorithm, thus training performance is proportional to convergence rate. Because of time limit and low improvement of training performance on VGG-19, the difference of convergence rate is not noticeable in Figure 5.

7 RELATED WORK

7.1 Communication schemes for Distributed Training on Heterogeneous cluster

There have been studies about decentralized communication topology on heterogeneous cluster. D-PSGD [17] assumes that a network is formed as a connected graph. Every node has its local parameter of the model. In each clock, all node average its parameter only with its neighbors. It still has idle time caused by waiting for the slowest node finishes its job. AD-PSGD [18] is an another work that shows zero idle time. It randomly selects one neighbor and

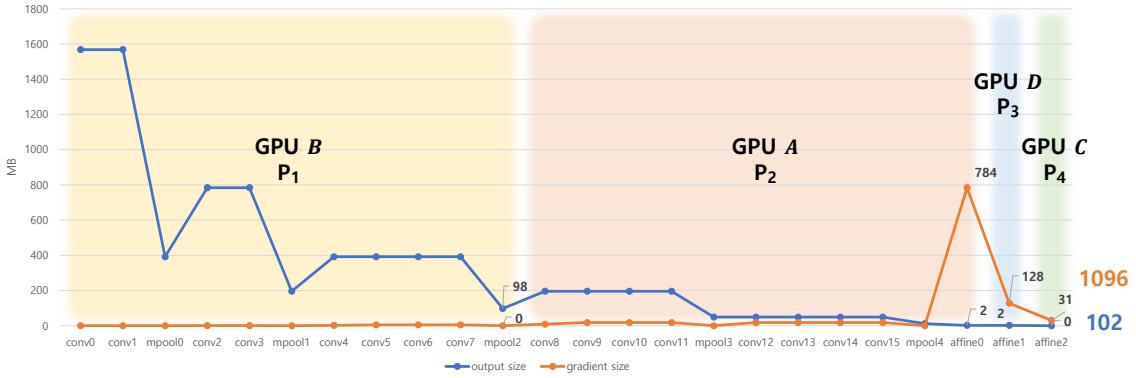


Figure 7: Partition result of VGG19 (128)

averages parameters. This averaging step is done in asynchronous manner on each node. However, it only considers heterogeneous network topology, not a computational resources such as GPUs.

7.2 Hybrid Parallelism

Recently, research on hybrid parallelism has been developing, where it combines model and data parallel strategy for distributed training to improve sub optimal parallel strategy. OptCNN [9] proposed layer-wise parallelism for distributed CNN model training. They allow each layer in DNN to use an individual parallel method by solving a graph search problem. However, they didn't show the convergence results to support their argument not losing accuracy compared to conventional methods. There is another hybrid parallelism that introduces SOAP (Sample-Operator-Attribute-Parameter) search space of parallel strategies for DNNs [10]. Still, it does not consider heterogeneous configuration and execution time of application must be predictable.

7.3 Pipelined Model Parallel Training

Because of sequential dependencies between layers in DNN, model parallel training usually shows low GPU utilization. Also, sending and receiving parameters between nodes through interconnection is too expensive. PipeDream [19] suggests hybrid parallelism that adopts pipelining methods to alleviate these problems. It reduces the overhead by overlapping communication and computation. Gpipe [8] suggests another pipelining model parallelism for optimizing memory efficiency. It supports large DNN model training by dividing mini-batch into micro-batches and re-computing activation output for backward pass, but suffers from poor performance.

8 CONCLUSION

Training large and complex DNN models needs to use distributed GPU clusters. With multiple GPUs and nodes on cluster, inter-node communication for synchronizing parameters is inavoidable in data parallel training. To handle this problem, we analyze intra- and inter- node communication behavior of Horovod Allreduce according to cluster topology and tried to minimize inter-node communications.

The system we designed is basically hybrid parallel scheme. We groups GPUs in different nodes into MP workers and they perform Allreduce to synchronize parameters with other workers. In our system, allreduce is done through the intra-communication and only small amount of data is transferred through inter-communication by model partitioner. Therefore, it is possible to train with larger batch size or large model which can not fit into single GPU by adopting model parallel training. Even though we do not perform pipelining to reduce the idle time of model parallel training, it achieves higher training performance on CNN models for image classification. Based on our experimental results, our system is efficient to mitigate high inter-node communication overhead in data parallel training.

REFERENCES

[1] Mike Barnett, Lance Shuler, Robert van De Geijn, Satya Gupta, David G Payne, and Jerrell Watts. Interprocessor collective communication library (intercom). In *Proceedings of IEEE Scalable High Performance Computing Conference*, 1994.

[2] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of COMPSTAT*. 2010.

[3] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

[4] Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2016.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Ng Andrew. Large Scale Distributed Deep Networks. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2012.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[7] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2013.

[8] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

[9] Zhihao Jia, Sina Lin, Charles R Qi, and Alex Aiken. Exploring hidden dimensions in parallelizing convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[10] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *In Proceedings of the 2nd SysML Conference(SysML)*, 2018.

[11] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2017.

[12] Soojeong Kim, Gyeong-In Yu, Hojin Park, Sungwoo Cho, Eunji Jeong, Hyeonmin Ha, Sanha Lee, Joo Seong Jeong, and Byung-Gon Chun. Parallax: Sparsity-aware data parallel training of deep neural networks. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2019.

[13] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[14] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. On Model Parallelization and Scheduling Strategies for Distributed Machine Learning. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2014.

[15] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[16] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. Pipe-SGD: A Decentralized Pipelined SGD Framework for Distributed Deep Net Training. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2018.

[17] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017.

[18] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017.

[19] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[20] NVIDIA. GeForce RTX 2060. https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2060/.

[21] NVIDIA. Quadro P4000. https://www.nvidia.com/en-us/design-visualization/quadro-desktop-gpus/.

[22] NVIDIA. TITAN RTX. https://www.nvidia.com/en-us/titan/titan-rtx/.

[23] NVIDIA. TITAN V. https://www.nvidia.com/en-us/titan/titan-v/.

[24] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2011.

[25] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[26] TensorFlow. TensorFlow benchmarks. https://github.com/tensorflow/benchmarks/.

[27] Minjie Wang, Chien-chin Huang, and Jinyang Li. Supporting Very Large Models using Automatic Dataflow Graph Partitioning. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2019.

[28] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2017.