Novel Weight Prediction Technique to Mitigate Staleness and Consistency Issue for weight updates in DNN Training

[1]Chavala Mutyala Rao, [2]Kattula Shyamala
[1]Research Scholar, [2]Professor,
Computer Science and Engineering,
Osmania University, Hyderabad, India
[1]cmutyalarao@gmail.com, [2]prkshyamala@gmail.com

Abstract

DNN training is compute-intensive wherein model is partitioned across GPUs because of large amount of training data and model size. GPU is responsible for the weight updates of assigned model layers and one only GPU is active at a time in parallelism implementation to enable parallel execution. In pipelined parallelism multiple mini-batches are injected into the model concurrently and introduce staleness and consistency issue for weight updates. Since multiple mini-batches are simultaneously processed, later mini-batch could start thetraining process before its prior mini-batch updates weights. Consistency issue alleviated by ensuring the forward and backward passes of a mini-batch in the sameGPU as the same weights through storing multiple versions of weights. However, weight staleness problem still unsolved. In this research article pipelined parallelism addresses unstable learning which leads to worse model accuracy due to weight staleness problem. Proposed mechanism, predicts the future weights in early pipeline stages, so that a mini-batch can adopt the predicted future weights rather than the stale weights to perform its computation. The design is based on the observation that smoothed gradients used in momentum-based optimizers reflect the trend of weight updates and can be leveraged for accurate weight prediction.

Keywords: *Parallel Computing, Distributed Deep Learning, Deep Neural Networks, Data Parallelism, Model Parallelism, Pipeline Model Parallelism, Weight Staleness.*

1. Introduction

The training process of Deep Neural Network (DNN) is compute-intensive, often taking days to weeks to train a DNN model. Therefore, parallel execution of DNN training on GPUs is a widely adopted approach to speed up the process nowadays. Data parallelism is currently the most commonly used parallelization method. Nonetheless, data parallelism suffers from excessive inter- GPU communication overhead due to frequent weight synchronization among GPUs. Another approach is pipelined model parallelism, which partitions a DNN model among GPUs, and processes multiple mini- batches concurrently [1].

1.1 Motivation

Deep Neural Networks have been popularly used to solve various problems such as image classification, speech recognition, topic modeling, and text processing. The size of DNN models (i.e., the number of parameters) have continuously been increasing in order to improve the accuracy and quality of models and to deal with complex features of data. The sizes of input data and batches used for training have also increased to achieve higher accuracy and throughput [3].

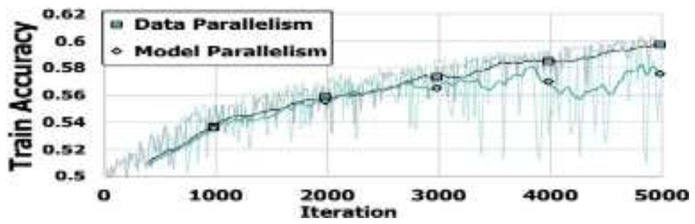


Figure 1.1: The accuracy curve while training a DNN model

1.2 Research Gaps

Training of a DNN model is processed by a forward pass followed by a backward pass for each minibatch, which is a subset of training samples, in a popularly used stochastic gradient descent (SGD) method. For each minibatch, the weight updates, i.e., gradients,are computed to update weights (or parameters) w of the model.

Data parallelism (DP) utilizes multiple workers to speed up training of a DNN model. It divides the training dataset into subsets and assigns each worker a different subset. Each worker has a replica of the DNN model and processes each minibatchin the subset, thereby computing the weight updates.

Model parallelism (MP) is typically exploited for large DNN models that are too large to be loaded into memory of a single GPU. In particular, a DNN model composedof multiple layers is divided into k partitions and each partition is assigned to a differentGPU. Each GPU executes both the forward and backward passes for the layers of the assigned partition. Note that it is important to execute the forward and backward passesof a partition on the same GPU as the activation result computed for the minibatch during the forward pass needs to be kept in the GPU memory until the backward pass of the same minibatch for efficient convergence. Otherwise, considerable extra overheadwill incur for managing the activation through either recomputation or memory man- agement. Several works have been published on pipeline model parallel DNN training. Someof the most notable ones include:

"Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism" presents Megatron-LM, a system for training large language models using pipelined model parallelism. The authors show that Megatron-LM can efficiently scaleto models with billions of parameters [24].

"PipeDream: Generalized Pipeline Parallelism for DNN Training" introduces a framework for pipelined model parallelism that can be applied to a wide range of DNNarchitectures. The authors demonstrate the effectiveness of PipeDream on several pop-ular models, including ResNet-50 and Inception-v4 [25].

"Scalable Training of Deep Learning Machines by Incremental Model Parallelism" proposes an incremental model parallelism approach for pipelined DNN training, whichallows for efficient scaling to very large models. The authors demonstrate the effective- ness of their approach on ResNet-101, achieving state-of-the-art performance on the ImageNet dataset [26].

"GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism" in- troduces GPipe, a framework for pipelined model parallelism that uses micro-batching to reduce memory requirements. The authors demonstrate the effectiveness of GPipe on several large-scale models, including the Transformer and ResNet-50 [27].

Overall, these works demonstrate the effectiveness of pipelined model parallelism for accelerating the training of deep neural networks on large-scale clusters [27].

1.2 Research Contributions

Pipelined DNN training is bi-directional that is, a mini-batch flows through the pipeline (forward pass) and then traverse back for weight updates (backward pass). It issues a forward task and a backward task in a round-robin manner as demonstrated in Figure. 1.2 Once a task is completed, it asynchronously executes the next one to maximize the utilization. The output data will be sent to the next GPU via a background thread to hide the latency. Staleness is a critical issue that should be resolved in pipelined model parallelism. The boxes with index I denotes the time spent on processing i-th mini-batch. Cyan and yellow boxes indicate forward and backward tasks respectively. A round trip of processing a mini-batch is presented by the arrows used an example, as shown in figure to explain the staleness issue in pipelining. In this example, Wt represents the version of weights at time unit t.

In the staleness-free single GPU implementation of training, both the forward and backward passes of a mini-batch are performed based on the latest weights. At time t, the forward and backward passes of a mini-batch both perform computation based on the latest weights Wt, which was generated by the backward pass of the previous mini-batch at time t_1. For example, in Figure. 1.2 (a), the 3-rd mini-batch produces W4, and the processing of the 4-th mini-batch at the 4-th time unit is based on W4, Figure. 1.2 (b), the 4-th mini-batch adopts various versions of weights, ranging from W4 to W6, during its round trip. From the 4-th mini-batch's perspective, W4 and W5 are stale and the only staleness-free version of weights is W6, as W6 is derived after the 3-rd mini-batch updates the weights. Such a weight update is called the weight staleness issue and it leads to unstable and inferior convergence. The round trip of the 5-th mini-batch and its adopted weight versions are illustrated.

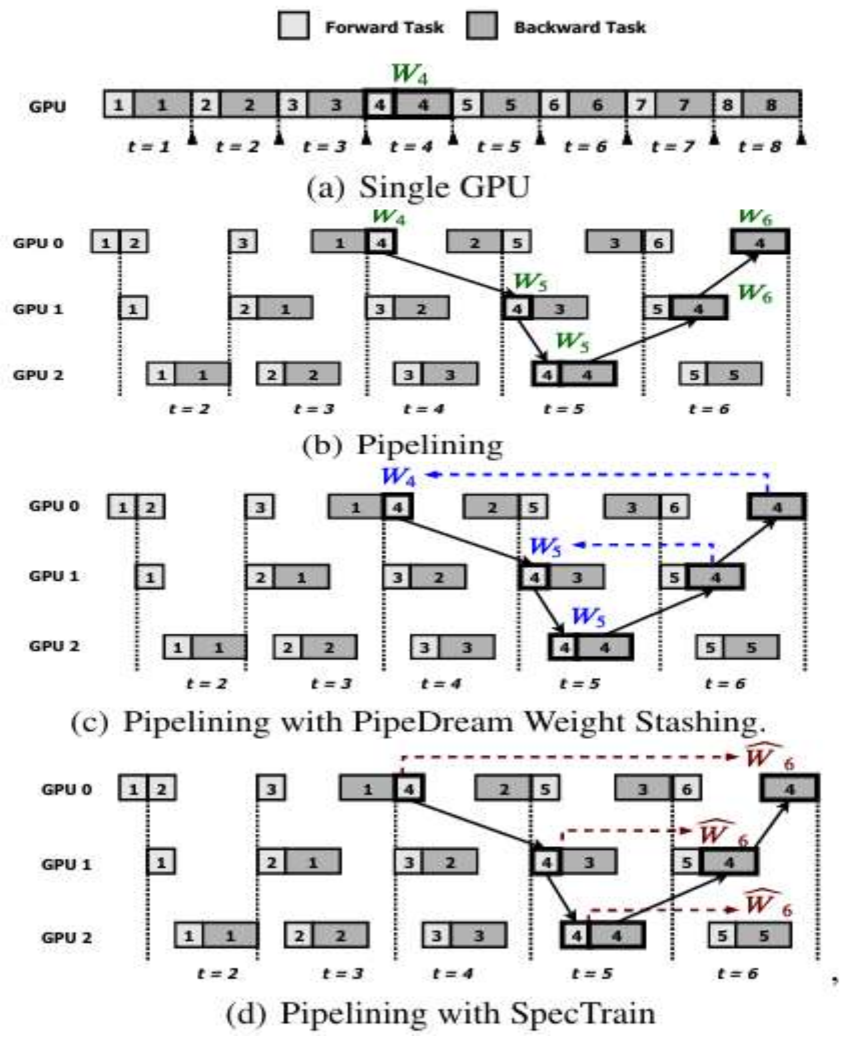


Figure 1.2: Timeline of Staleness in Pipeline Model Parallelism

1.4 Organization of the research article

In Section 2 we discuss related works of distributed deep learning and DNN training. Section 3 explains existing approach pipeline model parallel DNN Training with steps and algorithmic flow of pipeline model parallel training with results and summary. Section 4 demonstrates the proposed technique a novel weight prediction technique using NAG for model parallel DNN training. At end summary of conclusions and future directions were discussed.

2. Related Work

2.1 Distributed Deep Learning

There are different approaches to distributed deep learning, such as data parallelismand model parallelism. In data parallelism, each device works on a different subset of the data, and the model is replicated on each device. In model parallelism, the model is partitioned across different devices, and each device is responsible for computing a portion of the model's output. Distributed deep learning has been used to train large neural networks in fields such as computer vision, natural language processing, and speech recognition. It allows for faster training times and enables the use of larger datasets and more complex models. However, it also requires a significant amount of computational resources and expertise in distributed computing [4].

2.2 DNN Training

DNN models are typically trained using Stochastic Gradient Descent (SGD) and training data are randomly sampled into mini-batches. Mini-batches are fed into the model to traverse the model in two phases: forward and backward passes. The forward pass generates predictions and calculates the loss between the prediction and the ground truth. Then the backward pass back propagates errors to obtain gradients to up-date model weights. A mini-batch passing though the forward and backward phase is referred to as iteration and an epoch is defined as performing the forward-backward pass through the entire training dataset. The training process iterates multiple epochs until the model converges [5].

2.3 Pipeline Model Parallel DNN Training

Pipeline model parallelism is a popular technique for accelerating the training of deep neural networks (DNNs) on large-scale clusters with multiple GPUs. In this technique, the DNN is partitioned into multiple segments, and each segment is assigned to a different GPU for computation. The GPUs work in a pipelined manner, passing data fromone segment to another as the training progresses.

2.4 Algorithmic flow of Pipelined Parallel DNN Training

```
1. Define class PipelinedParallel
2. Initialize Resnet50 Model by defining split size
3. Do forward phase
        Mention splits Navigate to next split
4. for all splits
        prev split runs on cuda-1next split runs on cuda-0
5. append all splits
6. Setup DNN Model
7. Capture pipeline parallel run times
8. Record metrics values
9. Plot graphs with metrics
```

2.5 Results and Discussions

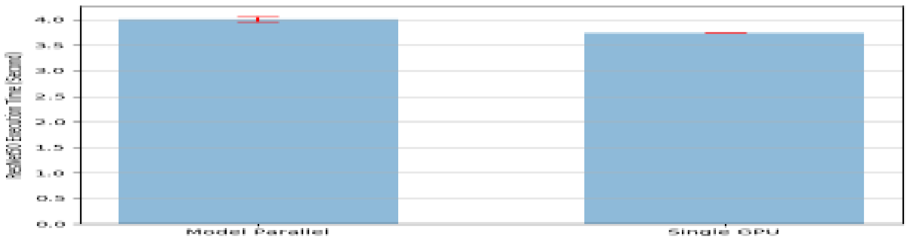


Figure 2.1: DNN training using Model Parallel and Single GPU

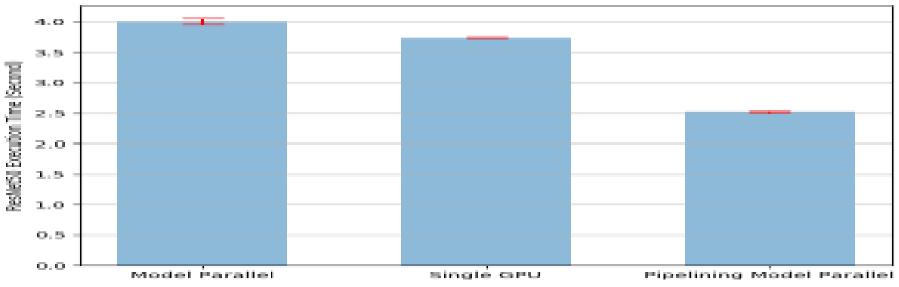


Figure 2.2: Model, Single GPU and Pipleline Parallelism

DNN training using Model Parallel and Single GPU approaches were compared it is observed that model parallel takes more time compared to single GPU approach. After applying pipeline parallelism technique it is observed from the results that pipeline parallelism takes less time when compared to model parallel and single GPU approach. In Pipeline Split Size for DNN training is noted that if the spilt size is increasing the DNN model takes more time for training. Pipeline model parallelism is a powerful technique for training large deep neural networks, and is increasingly being used to increase model accuracies while training large DNN models. [28].

3. Proposed Framework - Novel Weight Prediction Technique (NWPT)

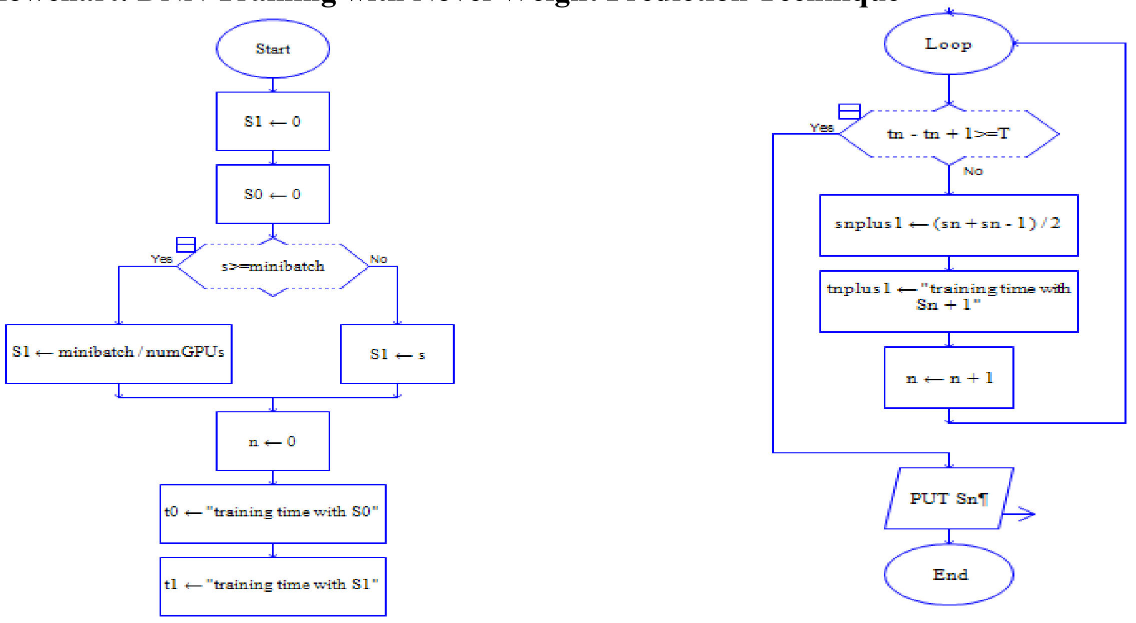
Novel Weight Prediction Technique is a variation of the standard gradient descent algorithm commonly used for optimizing deep learning models. It is a gradient-based optimization algorithm that combines the advantages of two techniques:momentum and gradient descent. Momentum is a technique that helps accelerate the convergence of the gradient de-scent algorithm by adding a fraction of the previous update to the current update. The idea behind momentum is to prevent oscillations and improve the convergence of the optimization process.

NWPT is similar to momentum in that it also adds a fraction of the previous update to the current update. However, NWPT differs from momentum in that it takes into account the current gradient and the previous gradient when computing the update. This helps to avoid overshooting the optimal solution and improves the stability of the algorithm. In other words, MWPT uses a look-ahead approach to compute the gradient, by evaluating the gradient at a future point in the parameter space that is estimated based on the current gradient and the previous update. This helps the algorithm to take into account the direction of the momentum and adjust the update accordingly, leading to faster convergence and better generalization.

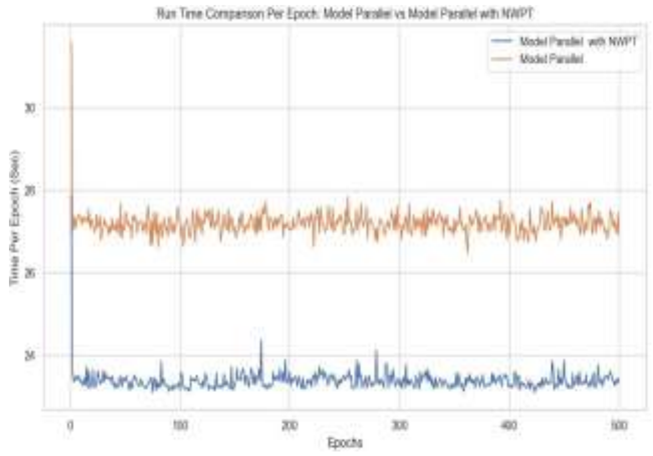
3.1 Algorithmic flow of Model Parallel with NWPT

1. Define the ResNet50 model from torchvision.models
2. Define the SplitModel with the first half of the ResNet50 model on GPU 1 and
   the second half on GPU 0
3. Define the training and validation datasets and dataloaders
4. Define the loss function and optimizer
5. Initialize history for training metrics and validation metrics
6. Define the number of epochs
7. Define predict weights with param, lookahead grad with lr and momentum
8. Training loop
   Move inputs& labels to GPU 1 in Forward pass
   Compute the loss in Backward pass
9. Update the weights with NAG weight prediction
10. Compute the number of correct predictions in the batch
11. Compute the train loss
12. Compute the average train loss
13. Compute the average train accuracy
14. Update history
15. Validation loop
16. Upate history
17. Print results

3.2 Flowchart: DNN Training with Novel Weight Prediction Technique


Start
S1 ← 0
S0 ← 0
s>=minibatch
Yes
No
S1 ← minibatch / numGPUs
S1 ← s
n ← 0
t0 ← "training time with S0"
t1 ← "training time with S1"
Loop
tn - tn + 1>=T
Yes
No
snplus1 ← (sn + sn - 1) / 2
tnplus1 ← "training time with Sn + 1"
n ← n + 1
PUT Sn¶
End

3.2 Results and Discussions



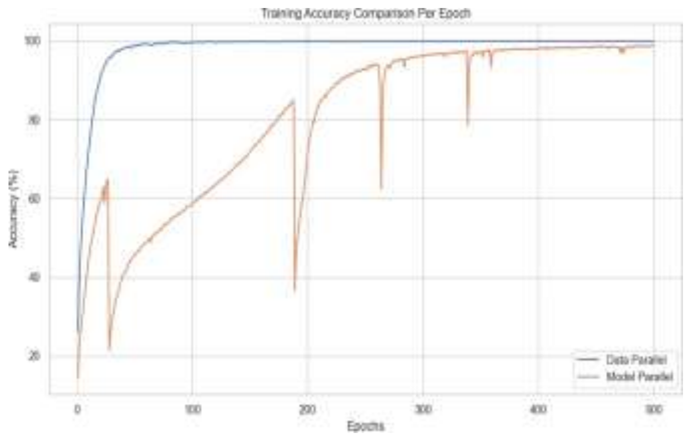


Figure 3.1: Overall Time model parallel with NWPT Figure 3.2: Accuracy Comparison

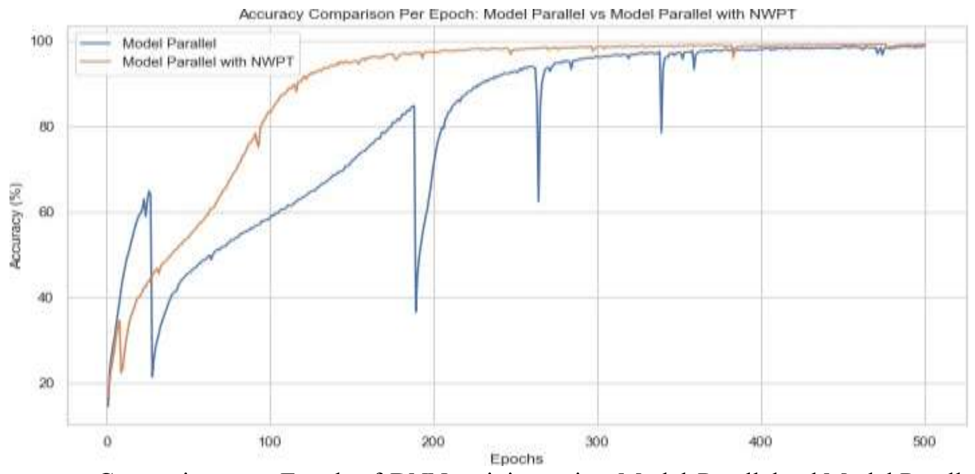


Figure 3.3: Accuracy Comparison per Epoch of DNN training using Model Paralleland Model Parallel with NWPT

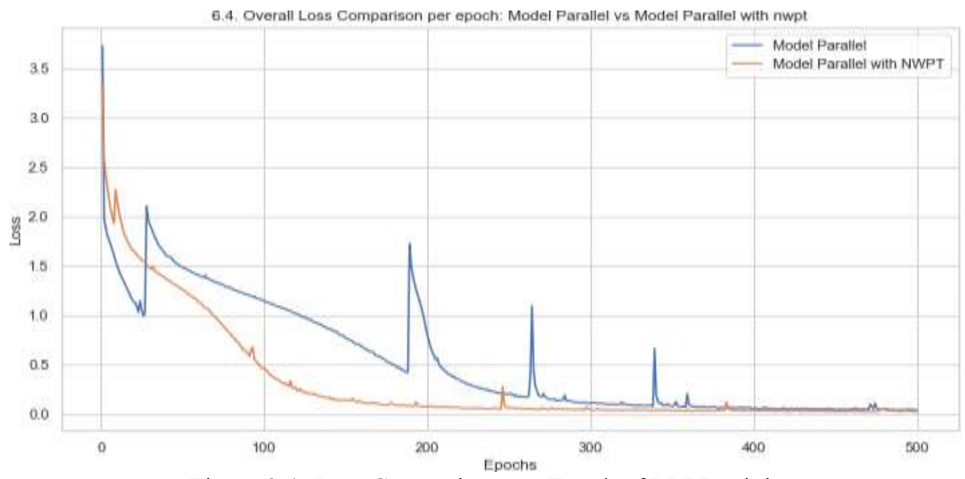


Figure 3.4: Loss Comparison per Epoch of DNN training

In summary, Nesterov Accelerated Gradient Descent is a popular optimization algorithm for deep learning models which helps to improve the convergence and stability ofthe optimization process. In model parallel with NWPT the overall time comparison of model parallel and MP with NWPT approaches were compared and the runtime comparison of DNN training were also showed. From the results it is observed that, DNNtraining with model parallel with NWPT performs better than model parallel approach.

4. Conclusion and Future Work

During the process of training a large DNN model using model parallelism techniques accuracies may be dropped because of staled weights used during forward and backward phases of DNN training. Hence we are proposing a novel weight prediction technique using Nesterov Accelerated Gradient Descent approach to increase the model training accuracies by making the good weight predictions and reduce the differences between staled and predicted weights. In the proposed work we have implemented pipeline parallel approach using a novel weight prediction technique for model weight updates which split layers across multiple GPUs to reduce DNN model training times to reach our modeling goals. In future work we would like to implement tensor parallel approach which splits model across activations and model parameters.

Conflict of Interest:

The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest or non financial interest in the subject matter or materials discussed in this manuscript.

REFERENCES

1. W. Xu, Y. Zhang, and X. Tang, “Parallelizing dnn training on gpus: Challenges and opportunities,” in *Companion Proceedings of the Web Conference 2021*, 2021, pp. 174–178.
2. C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, “Efficient and robust parallel dnn training through model parallelism on multi-gpu platform,” *preprint arXiv:1809.02839*, 2018.
3. J. H. Park, G. Yun, C. M. Yi, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.r. Choi, “Hetpipe: Enabling large dnn training on (whimpy) heterogeneous gpu clusters through integration of pipelined model parallelism and data parallelism,”in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Con-ference*, 2020, pp. 307–321.
4. Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” *arXiv preprintr:1712.01887*, 2017.
5. D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, “Pipedream: Generalized pipeline parallelism for dnn training,” in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.
6. K. S. Chahal, M. S. Grover, K. Dey, and R. R. Shah, “A hitchhiker’s guide on distributed training of deep neural networks,” *Journal of Parallel and DistributedComputing*, vol. 137, pp. 65–76, 2020.
7. S. Lee, D. Jha, A. Agrawal, A. Choudhary, and W.-K. Liao, “Parallel deep convolutional neural

network training by exploiting the overlapping of computation and communication," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 2017, pp. 183–192.
8. R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9,pp. 611–629, 2018.
9. T. Paine, H. Jin, J. Yang, Z. Lin, and T. Huang, "Gpu asynchronous stochas- tic gradient descent to speed up neural network training," *arXiv preprint :1312.6186*, 2013.
10. D. Cires¸an, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neuralnetwork for traffic sign classification," *Neural networks*, vol. 32, pp. 333–338, 2012.
11. M. Kane, J. W. Emerson, and S. Weston, "Scalable strategies for computing withmassive data," *Journal of Statistical Software*, vol. 55, pp. 1–19, 2013.
12. T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and
13. B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural networks*, vol. 64, pp. 39–48, 2015.
14. T. N. Sainath, B. Kingsbury, A.-r. Mohamed, G. E. Dahl, G. Saon, H. Soltau,
15. T. Beran, A. Y. Aravkin, and B. Ramabhadran, "Improvements to deep convolu-tional neural networks for lvcsr," in *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013, pp. 315–320.
16. D. Scherer, H. Schulz, and S. Behnke, "Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors," in *Artificial Neural Networks–ICANN 2010: 20th International Conference,* Springer, 2010, pp. 82–91.
17. S. G. Akl, *The design and analysis of parallel algorithms*. Prentice-Hall, Inc., 1989.
18. S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in *2018 IEEE* , pp. 949–957.
19. B. Chandrashekhar, H. Sanjay, and H. Lakshmi, "Prediction model for scheduling an irregular graph algorithms on cpu–gpu hybrid cluster framework," in *2020 International Conference on Inventive Computation Technologies (ICICT)*. IEEE,2020, pp. 584–589.
20. J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier, "High-performance computing: clusters, constellations, mpps, and future directions," *Computing in Sci- ence & Engineering*, vol. 7, no. 2, pp. 51–59, 2005.
21. K. Sajay and S. S. Babu, "A study of cloud computing environments for high performance applications," in *2016 International Conference on Data Mining andAdvanced Computing (SAPIENCE)*. IEEE, 2016, pp. 353–359.
22. T. Sterling, M. Brodowicz, and M. Anderson, *High performance computing: mod- ern systems and practices*. Morgan Kaufmann, 2017.
23. M. AbdelBaky, M. Parashar, H. Kim, K. E. Jordan, V. Sachdeva, J. Sexton,
24. H. Jamjoom, Z.-Y. Shae, G. Pencheva, R. Tavakoli *et al.*, "Enabling high- performance computing as a service," *Computer*, vol. 45, no. 10, pp. 72–80, 2012.
25. M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.
26. D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline paral-lelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.
27. K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-updatefiltering," in *2016 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE, 2016, pp. 5880–5884.