Accelerating DNN Training on two GPUs using Data and Model Parallelism

[1]Chavala Mutyala Rao, [2]Kattula Shyamala

[1]Research Scholar, [2]Professor,
Computer Science and Engineering,
Osmania University, Hyderabad, India

***Abstract*— Deep Neural Networks (DNNs) have gained a lot interest in recent years as it has demonstrated success for many classification and regression tasks, including image recognition language translation and speech recognition. Training a DNN model is compute-intensive and often takes days to weeks, due to the large amount of training data and the ever-increasing model size. As such, multi-GPU platforms are widely adopted to speed up DNN training through parallel execution. In multi-GPU platforms, data parallelism is a commonly used parallelization approach. When data parallelism is applied, each GPU holds a complete copy of DNN model and processes a subset of the training data. Data parallelism suffers from excessive inter-GPU communication over-heads, since weights updated at each individual GPU need to be synchronized. Communication overhead increases as the model size increases, severely hindering thescalability of data parallelism.**

Another parallelization approach is model parallelism, which partitions a DNN model among GPUs. Each GPU is responsible for the weight updates of the assigned model layers. Therefore, the amount of data communicated among GPUs is significantly less than data parallelism. Furthermore, model parallelism enables training a large model exceeding the size constraint limited by the accelerator memory capacity. However, for a naive model parallelism implementation, only one GPU is active at a time, even with pipelining we could increase GPU utilization.

In this research article, we discussed two distributed deep learning techniques to train DNN models which have massive model parameters shows superior performance advantage over data parallelism on a 2-GPU platform. However, model parallelism shows unstable learning and leads to worse model accuracy than data parallelism due to the weight staleness problem. Overall communication cost in model parallelism is better than data parallelism approach.

***Index Terms*— Parallel Computing, Distributed Deep Learning, Deep Neural Networks, Data Parallelism, Model Parallelism.**

I. 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. This approach can significantly reduce inter-GPU communication cost compared to data parallelism [1].

Deep Neural Network (DNN) models have continuously been growing in size in orderto improve the accuracy and quality of the models. Moreover, for training of large DNNmodels, the use of heterogeneous GPUs is inevitable due to the short release cycle of new GPU architectures. In this research work, we investigate how to enable training oflarge DNN models on a multiple GPU's called a virtual worker, processes mini batchesin a pipelined manner, and multiple such virtual workers employ data parallelism for higher performance. [2].

Deep Neural Networks have been popularly used to solve various problems such as image classification, speech recognition, topic modeling, and text processing. The sizeof 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 featuresof data. The size of input data and batches used for training has also increased to achieve higher accuracy and throughput [3].

Training of a DNN model is processed by a forward pass followed by a backward passfor each mini-batch, which is a subset of training samples, in a popularly used stochastic gradient descent (SGD) method. For each mini-batch, the weight updates, i.e., gradients,are computed to update weights (or parameters) w of the model. In the basic form of MP, k GPUs, individually, act as one virtual worker to process a mini-batch as follows:

For each mini-batch, execution of the forward pass starts from GPU_1 up to GPU_k. When each GPU_i, where 1<= i< k, completes the forward pass of the assigned partition, it sends the computed activations of only the last layer in its partition to GPU_{i+1}.Once GPU_k finishes the forward pass of its partition, the backward pass of the mini- batch is executed from GPU_k down to GPU_1. When each GPU_i, where 1 < I, 0 <= k, finishes the backward pass, it sends the computed local gradients of only the first layerin its assigned partition to GPU_i. This basic form of MP results in low GPU utilizationas only one GPU is actively executing either the forward or backward pass.

Research article organized organized in this way, section II we discuss an overview of DNN training, section III demonstrates the relatedworks of data and model parallel approaches. In section IV data parallel DNN training which includes steps and algorithmic flow of data parallel DNN training with results and summary are demonstrated. Section V explains model parallel DNN training with steps and algorithmic flow of model parallel DNN training with results and summary. Finally we discuss summary of conclusions and future work.

II. Related Work

Multiple studies have been conducted investigating ways to improve the speedup of compute intensive applications. There are several moving parts contributing to the abysmal performance of parallel computing implementations. This section discusses several research articles related to solve compute intensive problems like parallelizing convolution neural networks to improve performance and speedup. This section begins with a discussion of literature survey on data and model parallelism. The discussion progresses, with more emphasis on highly computationally intensive applications in computer vision like distributed training of DNNs on GPU platforms in the context of image classification applications which deals with large datasets. Works related to MNIST dataset is of most interest, also NORB and CIFAR datasets were also considered. Additionally, work done in distributed deep learning is briefly addressed [4].

A. Data Parallel DNN Training

Training DNNs on large datasets is an extremely compute intensive application and takes an enormous execution time. Most related studies regarding DNN training paral lelization are based on data and model parallelism. Data parallelism is the most commonly used parallelization in training DNNs. Proposed 1-bit SGD, which quantizes thegradients to one bit per value to lower the communication cost and proposes to send gradients only when it is larger than a threshold to reduce communications. Design another gradient quantization approach to do compression, and their proposed scheme allows users to trade off accuracy and performance [5]. Speed up data parallelism on convolution neural networks by using low bit-width weights, activations, and gradients. Propose Gradient Dropping to prevent frequent gra- dient transfers. Implement Gradient dropping, and develop many techniques to increase the robustness. Improve the performance of parameter server by leveraging distributed shared memory to reduce the networking time. Apply MPI parallelism to replace the gradient aggregation process at the parameter server, in order to provide better scalability [6]. Propose to give up slow workers at run-time, solving the straggler problem without affecting algorithmic correctness. Novel variant of SGD called asynchronous data parallelism, which skips the synchronization between workers [7]. Many deep learning frameworks such as Tensor-flow, PyTorch and Caffe provide high-level APIs for applying data parallel training. It is well known that the huge com- munication overhead of data parallelism limits its scalability. Some prior studies focuson reducing the communication overheads for data parallelism. Another issue, straggler problem, further downgrades the performance of data parallelism. Due to unstable networking, some workers may encounter heavy traffic and consume much time on weight synchronization, making other workers stall. This straggler problem reduces GPU utilization [8].

B. Model Parallel DNN Training

Model parallelism is another appealing approach to parallelize DNN training on fully-connected layers of AlexNet design a two-stage pipeline for training recurrent neural network. Use reinforcement learning to come up with a device placement policy for model parallelism. The above methods cannot parallelize some portions of a model if dependencies exist, i.e. inter-layer parallelization [9].

Gpipe updates the weights synchronously and reduce the GPU idle time by dividing the mini-batch into multiple micro-batches and pipelining the micro-batches divided from the same mini-batch. Some recent works consider not only data parallelism and model parallelism when finding parallelization strategy [10].

Partition the tensor along multiple dimensions and search for the best parallelizationstrategy. Chandra et al. have a good, somewhat formal, discussion of the process of identifying and removing these dependencies. The mechanisms for dealing with these data dependencies often require some restructuring of code. For example, it may be necessary to split a loop that computes multiple quantities [11].

Multiple GPUs to train CNNs on the ImageNet dataset using both data and model parallelism, i.e., either the input space is divided into mini-batches where each GPU train its own batch (data parallelism) or the GPUs train one sample together (model parallelism). There is no direct comparison with the training time on CPU; however, using 4 GPUs (Nvidia Titan) and model and data parallelism, the network was trained for 4.8 days [12]. Propose PipeDream as the first work using pipeline in model parallelism to train DNN models, enabling inter-layer parallelization. Propose algorithms to break back- ward locking for model parallelism so that the backward pass can be performed in parallel.

C. Experimental Setup

We have used two NVIDIA RTX 3090 GPUs with 1049 CUDA Cores, 1.70GH BoostClock, 24GB Memory Size with GDDR6X Memory Type. Dataset is CIFAR-10, DNN Model is ResNet-50 ,Pytorch Framework, Visual Studio Code Editor and Horward, DDP packages were also used.

III. Data Parallel DNN Training

Data parallelism is a popular approach to train Deep Neural Networks (DNNs) when thesize of the training data is too large to fit into the memory of a single machine. In dataparallelism, the training data is divided into multiple parts, and each part is assigned to a separate computing node. The nodes train the model independently and exchange their updates with each other to keep the model synchronized.

A. Algorithmic flow of Data Parallel DNN Training

Algorithm 1: Data Parallel DNN Training

```
Initialize the Horovod Environment
Define batch size and the number of epochs
Load CIFAR10 dataset
Define ResNet50 model
Define the loss function and optimizer
Wrap the optimizer with Horovod Distributed Optimizer
Move model to GPU
Training loop
        Move inputs and labels to GPUForward pass
        Backward pass
        Update inter-GPU communication time and data transferredUpdate weights
        Compute batch metrics
Compute epoch metrics
Print Results
```

B. Results and Discussions

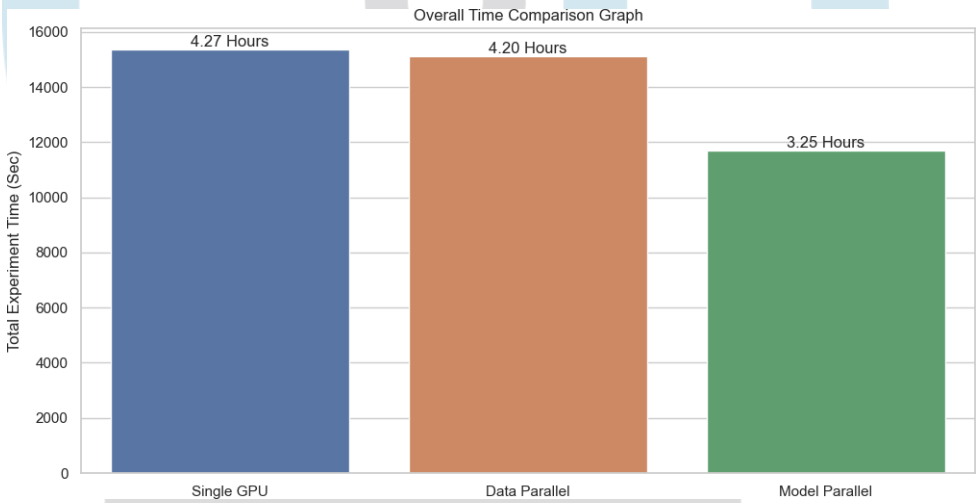

Fig 1. Overall Time Comparison of DNN training on Single GPU, Data Parallel and Model Parallel

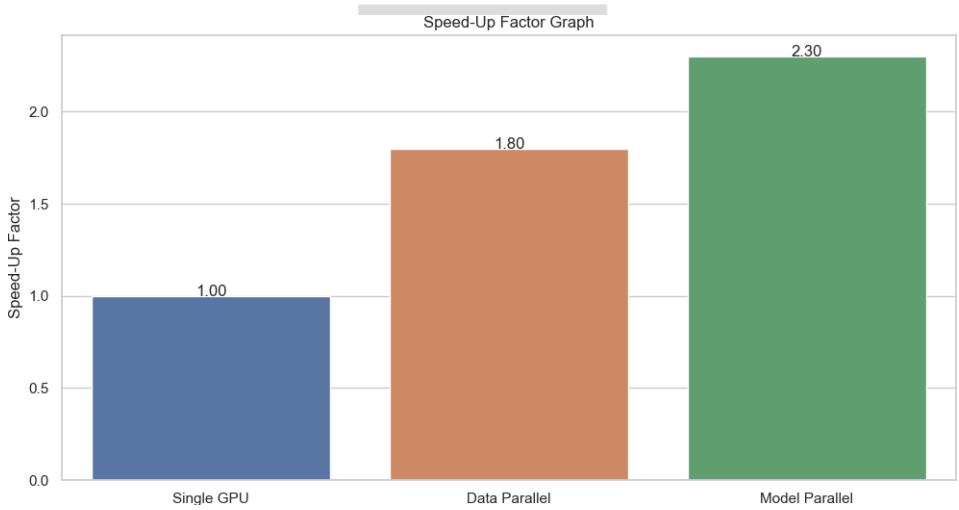

Fig 2. Speedup Factor DNN training on single GPU, data parallel and model parallel

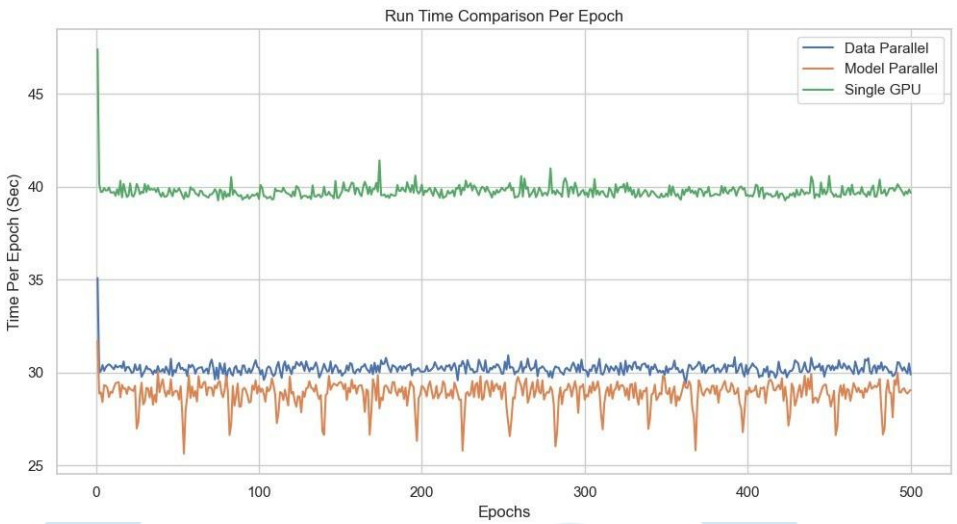

Fig 3. Runtime Comparison in secs for single GPU, data parallel and model parallel

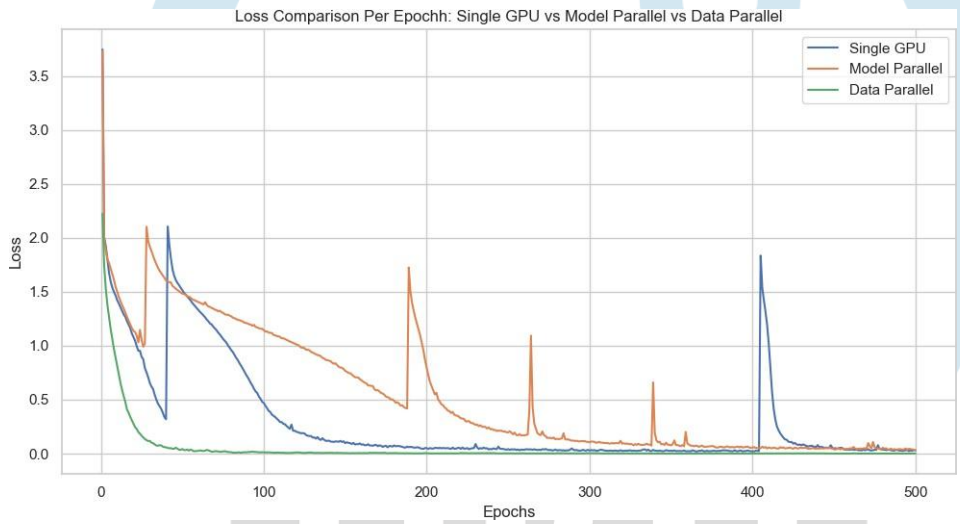

Fig 4. Overall Loss per epoch on single GPU, data parallel andmodel parallel approaches

Data parallelism is implemented and over all time comparison of DNN training on single GPU, Data Parallel and Model Parallel is compared and it is shown that data parallelapproach performs better than other approaches. Meanwhile speedup factor for DNN training on single GPU, data parallel and model parallel is also observed and it is notedthat data parallel speedup factor is far better than other approaches. Runtime Comparison of Single GPU, Data Parallel and Model Parallel is observed in secs in graph and data parallel approach takes less time when compared with other approaches. Finally overall loss per epoch is observed while performing DNN training.

IV. Model Parallel DNN Training

Model parallelism is a technique used to train large deep neural networks (DNNs) thatdo not fit in the memory of a single device. In this approach, the model is partitioned into multiple segments, and each segment is trained on a separate device, such as a CPU or GPU. The output of one segment is passed as input to the next segment until the entire model is trained.

A. Algorithmic flow of Model Parallel DNN Training

1: Define the ResNet50 model from torchvision.models
2: Define the SplitModel with the first half of the ResNet50 model on GPU 1 and thesecond 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
6: Training loop
    Define the number of epochs
    Move inputs and labels to GPU 1
    Zero the gradients

Forward pass
Compute the loss
Backward pass and update the weights
7: Compute the number of correct predictions in the batch
8: Compute the train loss
9: Update inter-GPU communication time and data transferred
10: Compute the average train loss
11: Compute the average train accuracy
12: Update history

B. Results and Discussions

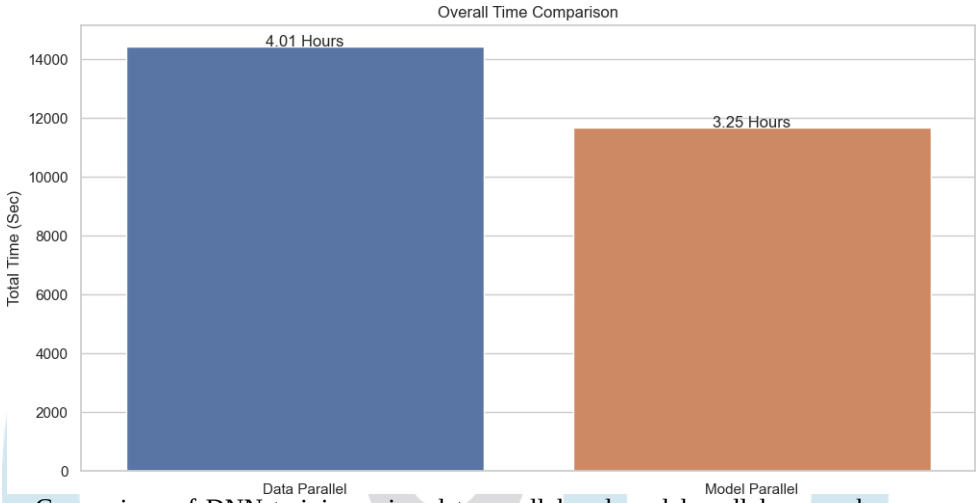

Fig 5. Overall Time Comparison of DNN training using data parallel and modelparallel approaches

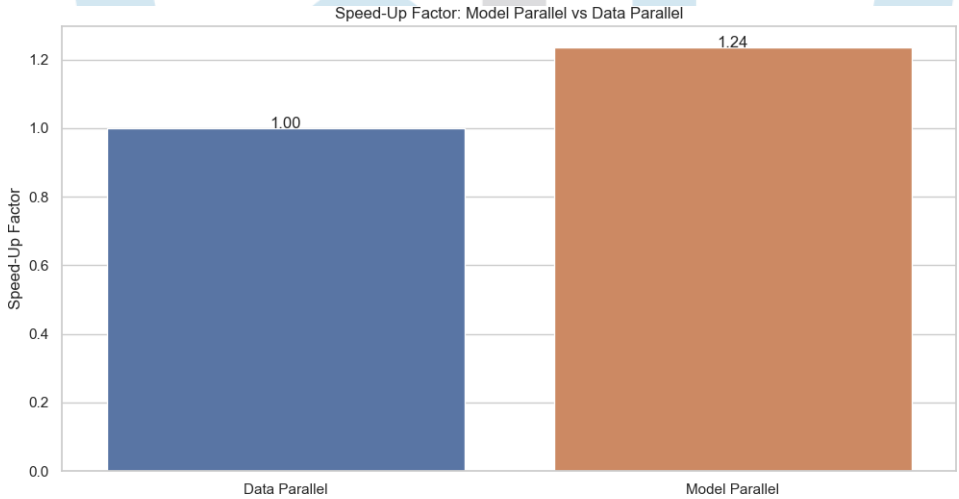

Fig 6. Speedup Factor of DNN training using data parallel and model parallel approaches

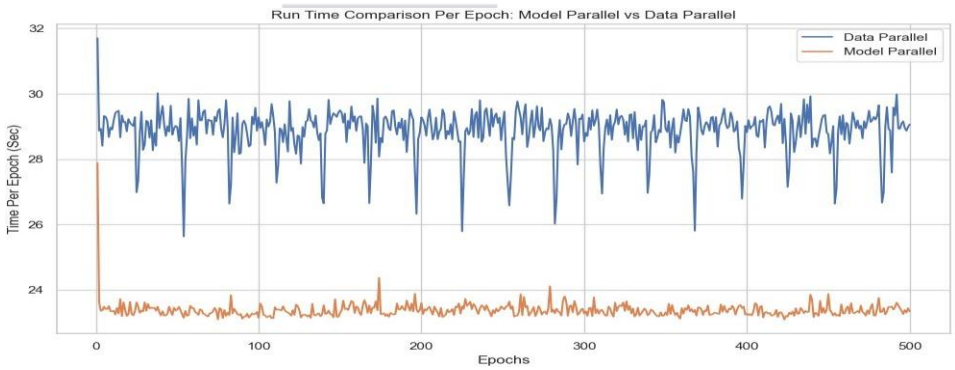

Fig 7. Runtime comparison in secs of DNN training using data parallel and modelparallel approaches

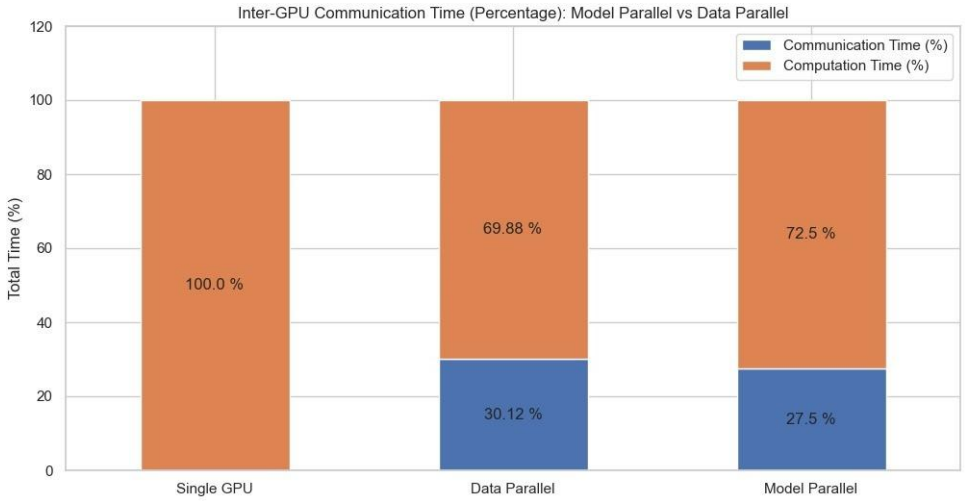

Fig 8. Inter-GPU Communication Time Comparison in Percentages

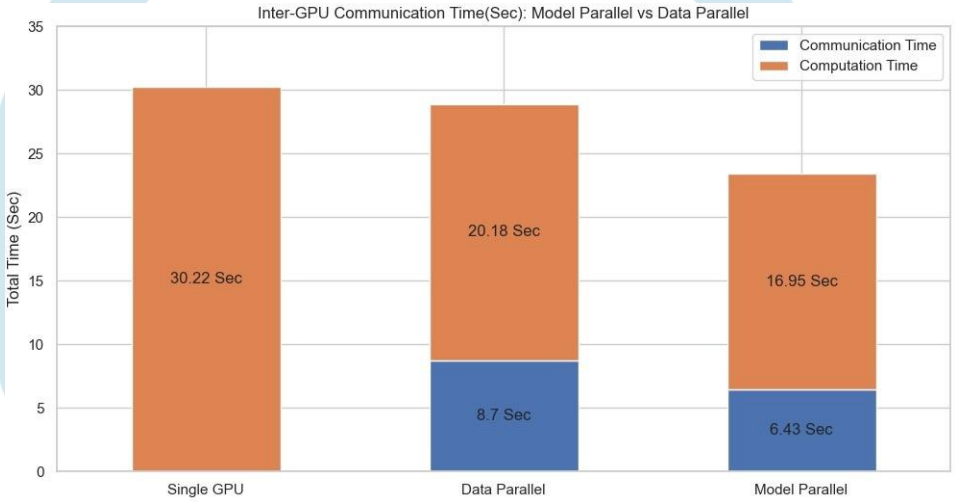

Fig 9. Inter-GPU Communication Time Comparison in Seconds

In Model Parallel DNN training overall time comparison of data parallel and model parallel approaches were compared and speedup factor of DNN training using data parallel and model parallel approaches is also observed and runtime comparison in secs of DNN training using data parallel and model parallel approaches were also showed. From the results it is observed that as the DNN model and data sets are smaller, data parallel approach performs better than model parallel approach. Model parallelism is an effective technique for training large deep neural networks and is commonly used in research and production environments to reduce training time and improve performance.

V. Conclusion and Future Work

With the growing size of the network and larger data-set, ability to extract more complicated representations are becoming challenging at the cost of insanely high computation time. Some DNN models may take several weeks to train on a single core CPU. Also the size of the deep neural network may not fit in a single machine. This made us to explore the possible ways to parallelize deep neural network training.DNN models may be trained using distributed deep learning techniques like Data Parallelism, Model Parallelism. In Data parallelism, inter-GPU communications are more andthe amount of data transferred among GPU's to synchronize the gradients are also more because all the participating GPU's have to communicate gradients and synchronize gradients before updating the model parameters. If we train large DNN model using Model Parallelism obviously amount of data transferred among GPU's per mini batch or epoch may be reduced as the layers of DNN model are splitted across GPU's. During the process of training a large DNN model using model parallelism techniques, the model accuracies may be dropped because of staled weights used during forward and backward phases of DNN training. In future work we would like to implement tensor parallel approach which splitsmodel across activations and model parameters.

REFERENCES

[1] W. Xu, Y. Zhang, and X. Tang, "Parallelizing dnn training on gpus: Challenges and opportunities," in *ompanion 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," arXiv 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] 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 Distributed Computing, vol. 137, pp. 65–76, 2020.

[5] D. Cires¸an, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," Neural networks, vol. 32, pp. 333–338, 2012.

[6] M. Kane, J. W. Emerson, and S. Weston, "Scalable strategies for computing with massive data," Journal of Statistical Software, vol. 55, pp. 1–19, 2013.

[7] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," Neural networks, vol. 64, pp. 39–48, 2015.

[8] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Sci- ence and Technology Congress (DASC/PiCom/DataCom/CyberSciTech). IEEE, 2018, pp. 949–957.

[9] A. Viebke, S. Memeti, S. Pllana, and A. Abraham, "Chaos: a parallelization scheme for training convolutional neural networks on intel xeon phi," The Journal of Supercomputing, vol. 75, pp. 197–227, 2019.

[10] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, G. E. Dahl, G. Saon, H. Soltau, 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.

[11] 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, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III 20. Springer, 2010, pp. 82–91.

[12] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Sci- ence and Technology Congress (DASC/PiCom/DataCom/CyberSciTech). IEEE, 2018, pp. 949–957.

[13] S. G. Akl, The design and analysis of parallel algorithms. Prentice-Hall, Inc., 1989.

[14] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Sci- ence and Technology Congress (DASC/PiCom/DataCom/CyberSciTech). IEEE,2018, pp. 949–957.

[15] B. Chandrashekhar, H. Sanjay, and H. Lakshmi, "Prediction model for scheduling an irregular graph algorithms on cpu–gpu hybrid cluster framework," in 2020 In- ternational Conference on Inventive Computation Technologies (ICICT). IEEE,2020, pp. 584–589.

[16] J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier, "High-performance computing: clusters, constellations, mpps, and future directions," Computing in Science & Engineering, vol. 7, no. 2, pp. 51–59, 2005.

[17] K. Sajay and S. S. Babu, "A study of cloud computing environments for high performance applications," in 2016 International Conference on Data Mining and Advanced Computing (SAPIENCE). IEEE, 2016, pp. 353–359.

[18] T. Sterling, M. Brodowicz, and M. Anderson, High performance computing: modern systems and practices. Morgan Kaufmann, 2017.

[19] M. AbdelBaky, M. Parashar, H. Kim, K. E. Jordan, V. Sachdeva, J. Sexton, 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.

[20] 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.

[21] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in 2016 ieee international conference on acoustics, speech and signal processing (icassp). IEEE, 2016, pp. 5880–5884.