Investigation Related to Influence of Multi-GPUs on Parallel Algorithms Based on PyTorch Distributed

Luyu Li

Department of Computer Science, Macau University of Science and Technology, Macau, 999078, China
20098535i011004@student.must.edu.mo

Abstract. This paper designs experiments to investigate the impact of different GPU counts on occupancy, accuracy, loss values and time of the PyTorch distributed data parallel and model parallel module in LeNet VGG16 and ResNet models respectively. PyTorch is a powerful deep learning framework but due to the expansion of the database size, it cannot load it all into memory at once. So, it divides large-scale data into small chunks, which are then processed simultaneously on multiple processors or computers, and finally the results are combined. But this leads to a change in the model results. Therefore, it brings out the experiment as follow, n LeNet and VGG16 models, model parallel experiments with different numbers of GPUs were conducted on these both models, and about ResNet model in this paper, performed data parallelism experiments with different number of GPUs. The findings revealed that, in the model parallel experiments, the number of GPUs employed did not have a significant impact on the floating accuracy of the results. However, a slight variation in the accuracy of the output answers was observed among the LeNet models, with an overall average accuracy of 26.2%. In the data parallel experiment, it was observed that the running time increased with an increasing number of GPUs. Specifically, the running time using one GPU was measured at 265.173 seconds, while using five GPUs resulted in a running time of 347.273 seconds. These results suggest that the utilization of multiple GPUs in data parallelism leads to an increase in operating time.

Keywords: Data Parallelism, Model Parallelism, Pytorch, Deep Learning.

1 Introduction

Deep Neural Networks (DNN) are artificial neural networks composed of multi-layer interconnected neurons. DNNs already have their applications in various field, ranging from image classification [1], question and answer systems [2], and cyber-security 3 etc [3]. The pursuit of increased intelligence in many deep neural network (DNN) applications entails the utilization of larger datasets to optimize larger models, with the expectation of advancements in distributed training systems. Among existing solutions, distributed data parallel and model parallel are promising approach to train large DNNs with numerous layers on multiple GPUs, where the DNN is partitioned into multiple stages, each containing a number of layers and running on a GPU.

In recent work about PyTorch distributed, Shen Li introduced the practical experience of PyTorch distributed training [4], including the basic principles of distributed training, the implementation of PyTorch distributed training, and optimization techniques for distributed training. Che-Lun Hung introduces a parallel algorithm based on CUDA for multiple sequence alignment on multiple GPUs [5]. Shinji Sakane presents a parallel computing method based on multiple GPUs for simulating dendrite crystal growth under forced convection conditions [6]. Yakun Huang proposes a new method for accelerating deep neural networks, called Distributed Deep Learning (DDL), which enables data and model parallelization on ubiquitous end devices [7].

About three basic models used in this paper. It can be found that in LeNet model, the average pooling used in this model may lose some important feature information. However, in VGG16 model, the use of max pooling may lose some detail information which may influence the accuracy of results and the loss value during the operation time. The last one is ResNet, which can train very deep neural network models. This model has the drawback of having a high number of model parameters, which leads to a requirement of more computing resources for both training and inference. Additionally, the training speed of this model is slow, and it requires a significant amount of computing resources and a long training time.

To address the limitations mentioned above, this study conducts a series of testing experiments aimed at investigating the efficacy of model parallelism and data parallelism on different neural network models. Specifically, the design, implementation, and evaluation of model parallelism are presented using the LeNet and VGG16 models, while data parallelism is explored using the ResNet model. In LeNet model, the model parallel experiment was implemented to explore how the different numbers of GPUs impact on the GPU footprint of LeNet models and the accuracy of individual results. In VGG16 model, it applies the model parallel experiment using two GPUs to investigate the effect of different number of GPUs on loss values, computing time and accuracy of results. About ResNet model in this paper, the data parallelism experiments with different type and different number of GPUs were performed , in order to figure out the occupancy of each GPU in data parallelism and loss values under the influence of different types of GPUs as well as carrying out data parallel experiments with different numbers of GPUs in the same type of GPU, model loss values and changes in occupancy of each GPU and recording the model running time when using different numbers of GPUs.

For the first experiment the average accuracy of the models reached 26.2%, and there is a slight difference in the accuracy of the answers output by these models. About the occupancy rate of each GPU, the result shows that when it adds a new GPU into the system, the new GPU will take up a lot of the work of the previous GPU, thus making the previous GPU much less occupied. However, this phenomenon is gradually reduced as the number of GPUs increases eventually reaching a relatively balanced value. For the second experiment, 2% increase in output accuracy with two GPUs compared to one GPU. This is not a great improvement. But about computing time, 15s reduction in computing time with two GPUs compared to one GPU. It means that the different number of GPUs did not help more with the reduction in loss values or the increase in accuracy. However, by splitting the model operations between two GPUs, the pressure on a single GPU is reduced, and the two layers increase to a shorter communication time than the original single GPU, resulting in shorter operation times. In the last experiment, it can see that the better the GPU type, the lower the GPU footprint required to run the model. However, the GPU type does not have a significant effect on the loss value of the model. And the situation about occupancy rate of each GPU in this model is just opposite from the first experiment. The running time that using one GPU is 265.173s and using five GPUs are 347.273s. Operating time increased with the quantity of GPUs.

2 Method

2.1 Dataset Description and Preprocessing

In this study, two datasets were employed, namely MINST and ImageNet. The MNIST handwritten digit dataset contains 60,000 grayscale training images and 10,000 testing images, with 10-digit classes (0-9). The image data is converted to floating-point numbers and normalized to between 0 and 1, the labels are converted to one-hot encoding, and the image data is then converted to a 4D tensor in this study.

ImageNet contains over 1 million images with 1000 classes. The image size is 224 $\times$ 224 pixels, and the image channel is RGB. In terms of data augmentation, VGG16 and ResNet use techniques such as random cropping, random scaling, and random horizontal flipping to increase the robustness and generalization ability of the models. The data is split into training and validation sets using the ImageDataGenerator class.

2.2 CNN Model

The first model is the LeNet model, which is based on the Convolutional Neural Network (CNN) theory and is used for research in the field of image recognition. LeNet was proposed by Yann LeCun et al. in 1998 and was the first successful deep learning model applied to handwritten digit recognition [8]. It consists of convolutional layers, pooling layers, and fully connected layers, where the convolutional and pooling layers are used for feature extraction and the fully connected layers are used for classification. LeNet is a milestone in the field of deep learning and laid the foundation for later convolutional neural networks.

The second model is the VGG16 model, which was proposed by Karen Simonyan and Andrew Zisserman in 2014 and is characterized by its deep architecture [9], consisting of 16 convolutional layers which including 13 convolutional layers and 3 fully connected layers, and pooling layers. It uses small 3x3 filters with a stride of 1 and padding of 1.

The third model is the ResNet model, which was proposed by Kaiming He et al. in 2015 and is characterized by its residual blocks [10], which allow for the training of very deep neural networks 10. The difference between the LeNet, VGG16 and ResNet is that ResNet introduced residual blocks into their model. Additionally, it used skip connections to address information loss and gradient vanishing problems in deep neural networks. It can train very deep neural network models.

2.3 Model Parallelism

Model parallelism is a parallel computing method in deep learning that divides a large model into multiple small models and assigns them to different GPUs for processing. Each GPU processes a different part of the model and then passes the results to the next GPU for processing. When all GPUs complete their computations, their results are collected and merged to obtain the final output. Model parallelism is suitable for large models and can reduce the memory pressure on a single GPU. By dividing the model into multiple small models, each GPU only needs to process a portion of the model, reducing the amount of model parameters and gradients that need to be stored on a single GPU. Additionally, model parallelism can increase GPU utilization because while one GPU is computing, other GPUs can process other parts of the model.

2.4 Data Parallelism

Data parallelism is a parallel computing method in deep learning that divides a large dataset into multiple small batches and assigns them to different GPUs for processing. Each GPU uses the same model parameters but processes different data. When all GPUs complete their computations, their results are collected and merged to update the model parameters. Data parallelism is suitable for large datasets and can speed up the training process. By dividing the dataset into multiple small batches, each GPU only needs to process a portion of the data, reducing the memory pressure on a single GPU. Additionally, data parallelism can increase GPU utilization because while one GPU is computing, other GPUs can process other batches of data.

2.5 Implementation Details

In this study, the three models employed possess distinct configurations and training settings. For the LeNet model, the Adam optimizer is utilized along with a cross-entropy loss function. The model's learning rate is set to 0.001, and a batch size of 64 is employed. The evaluation metric employed is accuracy, and the model is trained using an RTX 2080 Ti GPU. Moving on to the VGG16 model, it utilizes the Stochastic Gradient Descent (SGD) optimizer and cross-entropy loss function. The learning rate for VGG16 is set to 0.01, and a batch size of 32 is used. Similar to LeNet, accuracy is employed as the evaluation metric, and the model is trained using an RTX 2080 Ti GPU. As for the ResNet model, it employs the Adam optimizer and the same cross-entropy loss function and learning rate as LeNet. However, ResNet differs in terms of batch size, which is set to 128. Similar to the previous models, accuracy serves as the evaluation metric, and the model is trained using both RTX 3090 and RTX 2080 Ti GPUs. The utilization of multiple GPUs allows for investigating the impact of data parallelism when varying the number of GPUs from 5 to 2, with the GPU occupation rate being compared, and the lower occupancy rate selected for analysis.

3 Experiment Results and Discussion

3.1 Model Parallelism in LeNet

Fig. 1 demonstrates the accuracy and occupancy of each GPU of LeNet model when using 1 to 5 GPUs respectively. Looking at the line part in Fig. 1, it is evident that the accuracy rates fluctuated when using different numbers of GPUs. Notably, the highest accuracy rate of approximately 28% is observed when employing three GPUs. Conversely, a gradual decline in accuracy is observed as the number of GPUs increases from 1 to 2, reaching a nadir of 22% with the use of 2 GPUs. Subsequently, the accuracy levels display an upward trend, ultimately stabilizing at 27% when 4 or 5 GPUs are employed. In terms of the bar chart, it is clear that the occupancy of individual GPUs is highly dependent on the number of GPUs used. The highest occupation of GPU was recorded when using one GPU, at about 9% in one GPU. The average GPU footprint is gradually decreasing across different amounts of GPU usage, reaching the lowest point of 2.4% when using 5 GPUs.

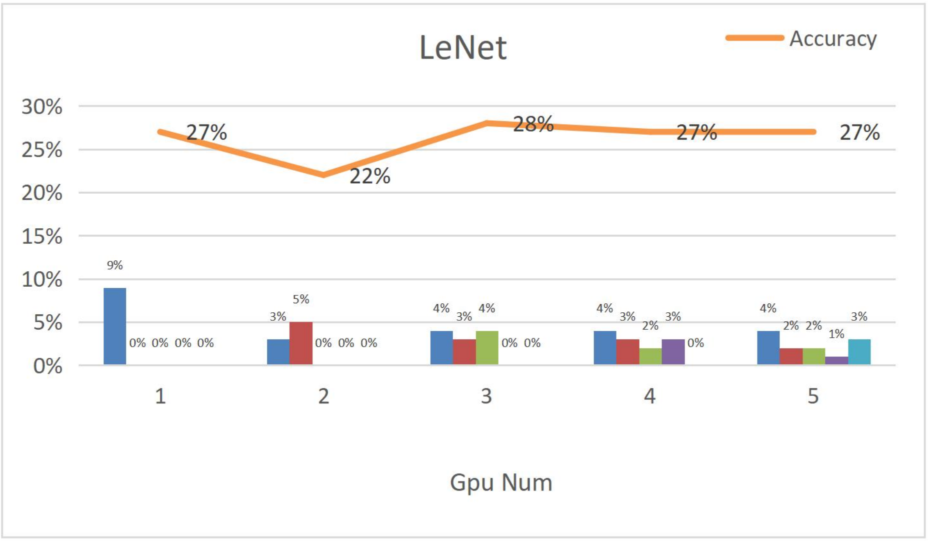

Fig. 1. The accuracy and occupancy of each GPU based on LeNet model (Photo/Picture credit: Original).

From the results in Fig. 1, it can be found that LeNet achieved the lowest accuracy when using two GPUs, while the accuracy is higher when using 1, 3, 4, or 5 GPUs. In this experiment, the accuracy of LeNet is influenced by various factors. When using two GPUs, the accuracy may be lower due to uneven data partitioning, increased communication overhead, and potential asynchronous parameter updates: 1) the data partitioning may not be evenly distributed, resulting in some GPUs having less data while others have more; 2) the frequency of data transfer and synchronization operations increases, leading to increased communication overhead; 3) the synchronization of parameter updates can be more complex, potentially leading to asynchronous updates. However, when using 3, 4, or 5 GPUs, these issues can be mitigated by more balanced data partitioning, reduced communication overhead, and improved parameter synchronization. Additionally, when using 1 GPU, the accuracy may be higher due to simplified data partitioning, reduced communication overhead, and elimination of asynchronous parameter updates. It is important to consider these factors and optimize the number of GPUs to achieve the best accuracy in model parallelism experiments.

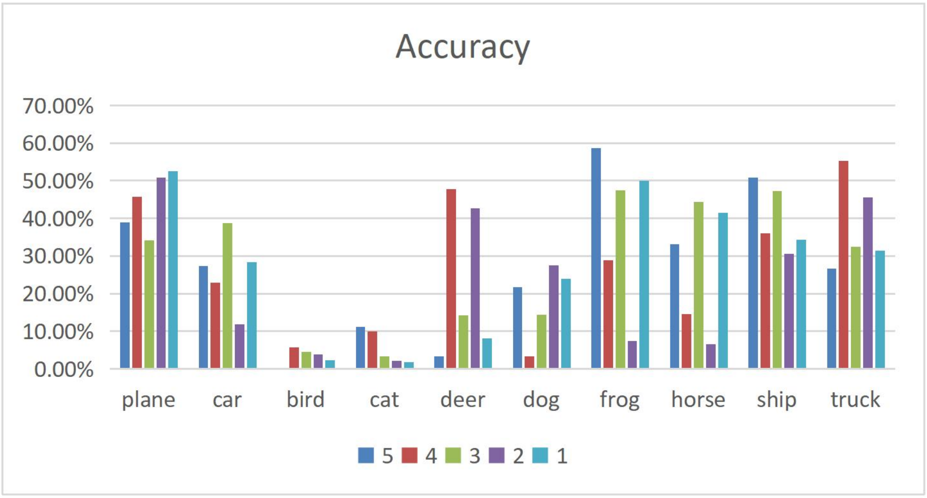

Fig. 2. The accuracy of each output in MNIST (Photo/Picture credit: Original).

The bar chart in Fig. 2. provides information about the accuracy of each output in MNIST trained by the LeNet model. Overall, it is clear that the accuracy of the results of the trained dataset is not strongly dependent on the number of GPUs used.

When training the LeNet model with different numbers of GPUs, the accuracy of individual results can exhibit variability due to several contributing factors. These factors include the disparate handling of data and computation by distinct GPUs, resulting in discrepancies in the computed outcomes. Additionally, the synchronization of updates and the randomness in the training process can also affect the accuracy of individual results. For example, synchronous updates may introduce inconsistencies as some GPUs may need to wait for others to finish computations. Moreover, the inherent randomness in deep learning can lead to different results across different GPUs. Therefore, it is important to be aware that the accuracy of individual results may differ when training the LeNet model with different numbers of GPUs.

3.2 Model Parallelism in VGG16

The variation of the model loss rate during the training process is shown in Fig. 3, it is evident that the loss values in this model experienced fluctuations decline during the training time not only when using one GPU nor using two GPUs.

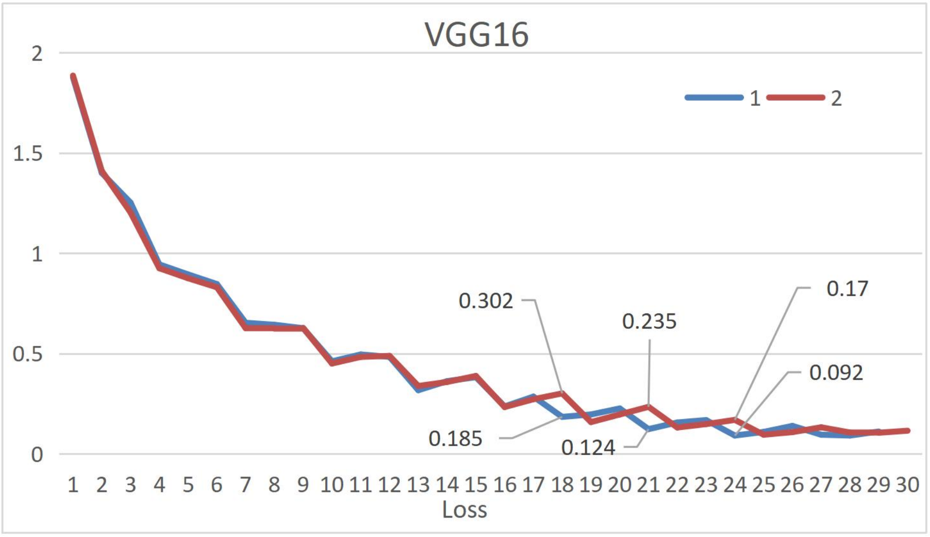

Fig. 3. The loss curve during the training process (Photo/Picture credit: Original).

When training a VGG16 model on ImageNet, the use of one GPU versus two GPUs can result in significant differences in the loss values during the later stages of training. This can be attributed to several factors. Initially, during the early stages of training, the loss values may be similar regardless of the number of GPUs used due to the random initialization of model parameters and the limited learning from the dataset. However, as training progresses, the computational power and data parallelism offered by multiple GPUs can accelerate the training process, leading to faster convergence and potentially lower loss values. Additionally, the synchronization of updates and the inherent randomness in the training process can introduce variations in the loss values between different GPUs.

3.3 Data Parallelism in ResNet

Fig. 4 and Fig. 5 provide information about the different GPU occupations and losses incurred when running the ResNet model with 2080ti and 3090.

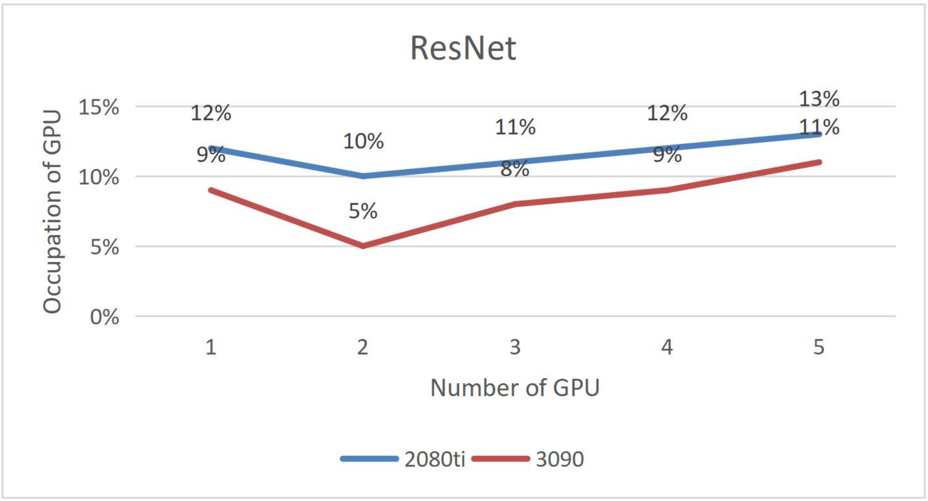

Fig. 4. The occupation of different number of GPU when using 2080ti and 3090 (Photo/Picture credit: Original).

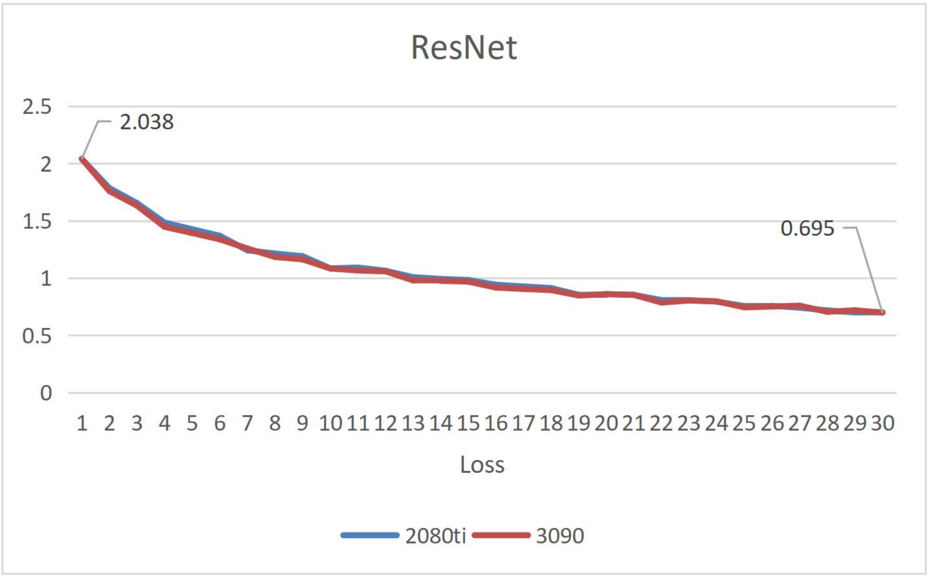

Fig. 5. The loss curve using 2080ti and 3090 during training process (Photo/Picture credit: Original).

In Fig. 4, it is evident that the occupation rates fluctuated when using different numbers of GPUs in different types of GPU. The lowest occupation rate occurred when using two GPUs, with about 10% using 2080ti and 5% using 3090. From using single GPU to two GPUs, there was a gradual decrease in occupation rate, reaching the lowest point when using two GPUs. After that, the accuracy levels started to rise again, eventually reaching a stable 11% using 3090 and 13% using 2080ti with the prerequisite of using five GPUs. The GPU usage with the 3090 is always less than the GPU usage with the 2080ti. Fig. 5 shows the loss values generated when training the ResNet model using 3090 and 2080ti respectively. The loss values for both GPUs steadily decrease throughout the training process, from 2.038 at the beginning to 0.695 at the end, and the loss values for both GPUs remain essentially the same throughout the training process.

It can be observed that the loss values of the model do not depend much on the type of GPU, and as the overall GPU occupancy was lower on all 3090 GPUs than with the 2080ti, so the subsequent experiments were conducted on the 3090 model GPU.

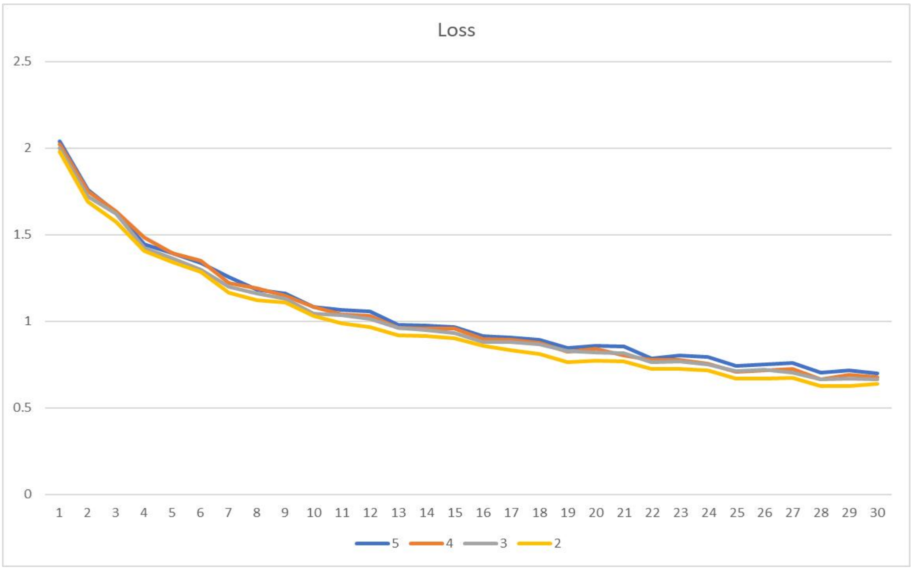

Fig. 6. The loss curve of using different number of GPU during training process (Photo/Picture credit: Original).

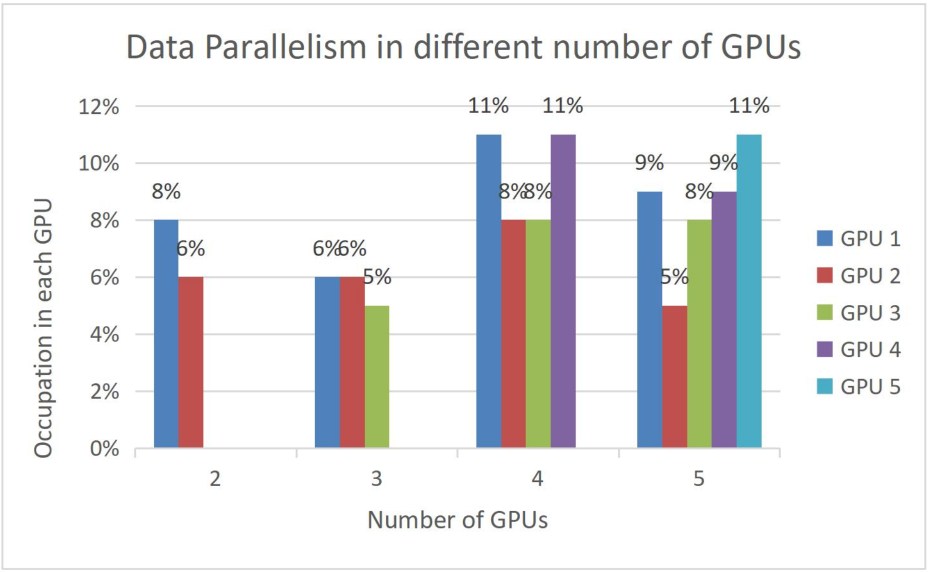

Fig. 7. The occupation in each GPU (Photo/Picture credit: Original).

Fig. 6. consists of 4 lines of different colors, which represent 4 different loss values when using different number of GPUs. As the training process continues, the loss value of the model decreases, in the case of the ResNet model, the experiments with different numbers of GPUs produce essentially the same loss value at the beginning, but there is a slight difference in the loss value as the training progresses, with two GPUs giving the smallest loss value and five GPUs giving the highest loss value compared to other numbers of GPUs.

With the same model of GPU but with different numbers of cards (Fig. 7.), it can be seen that the second GPU still has the lowest occupancy for each GPU, and the overall GPU occupancy is concave, i.e. low in the middle and high at the ends. However, it is worth noting that the GPU occupancy is relatively balanced across the three cards when the data is parallelized across all three cards. It can also be seen that the number of cards does not have a significant effect on the loss value, but as the number of cards decreases it still reduces the loss value by a certain amount.

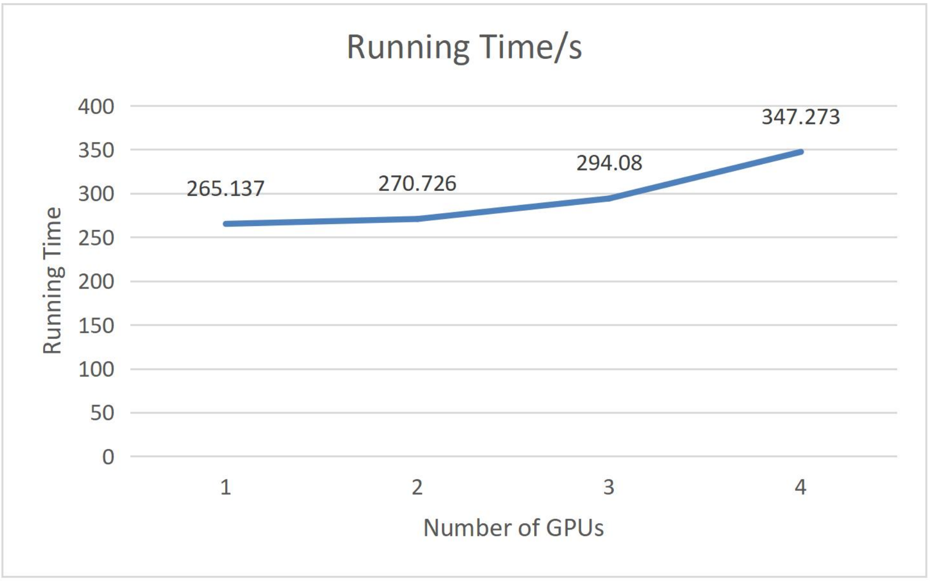

Fig. 8. The running time when using different numbers of GPUs (Photo/Picture credit: Original).

Fig. 8. shows the relationship between GPU and computing time for 2, 3, 4 and 5 cards respectively. It can be seen that as the number of GPUs increases, so does the computing time of the model due to the increase in communication overhead between layers.

4 Conclusion

In this work, it aimed to investigate the efficacy of model parallelism and data parallelism on different neural network models. The methods employed in this study included model parallelism and data parallelism. The findings in this research are that the variation in the accuracy of the model is mainly influenced by the model framework and the variation in the settings of the model parameters. The type and number of GPU's does not have a significant impact on the accuracy of the model, but as the number of GPU's increases the run time of the model will increase. These findings contribute significantly to the impact of the type and number of GPUs used on the accuracy, loss value and running time of distributed algorithms. However, it is important to acknowledge the limitations of this study, such as the model parameters are fixed values and do not include multiple influences in the comparison. Future research should focus on testing the impact of model parameter settings on model accuracy and loss values such as changing the learning rate or batch size to explore their impact.

References

1. Alex, K., et al.: ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6, 84–90 (2017).
2. Young, T., et al.: Recent Trends in Deep Learning Based Natural Language Processing, in IEEE Computational Intelligence Magazine, vol. 13, no. 3, pp. 55-75, (2018).
3. Chen, Z.: Deep Learning for Cybersecurity: A Review. 2020 International Conference on Computing and Data Science (CDS), Stanford, CA, USA, pp. 7-18 (2020).
4. Li, S., et al.: Pytorch distributed: Experiences on accelerating data parallel training. arXiv preprint arXiv:2006.15704 (2020).
5. Hung, C. L., et al.: CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multi-GPUs. Computational biology and chemistry, 58, 62-68 (2015).
6. Sakane, S., Takaki, T., Rojas, R., Ohno, M., Shibuta, Y., Shimokawabe, T., and Aoki, T.: Multi-GPUs parallel computation of dendrite growth in forced convection using the phase-field-lattice Boltzmann model. Journal of Crystal Growth, 474, 154-159 (2017).
7. Huang, Y., Qiao, X., Lai, W., Dustdar, S., Zhang, J.and Li, J.: Enabling DNN Acceleration With Data and Model Parallelization Over Ubiquitous End Devices. IEEE Internet of Things Journal, vol. 9, no. 16, pp. 15053-15065, 15 (2022).
8. Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P.: Gradient-based learning applied to document recognition. in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. (1998).
9. Simonyan, K., and Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).
10. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition pp. 770-778 (2016).