Comparative Analysis of Mirrored Strategy and Horovod for Data Parallelism in Semantic Segmentation Model

Myllena de Almeida Prado
Department of Computer Science
Texas State University
San Marcos, TX 78666, USA
xeh13@txstate.edu

Tanzina Akter Tani
Department of Computer Science
Texas State University
San Marcos, TX 78666, USA
dtu14@txstate.edu

***Abstract*—This report explores distributed deep learning training using data parallelism approaches, with a focus on the use of the mirrored strategy and Horovod framework. We tackle the problem of effectively synchronizing data between several processing units. We illustrate the scalability and speed improvements of our method through experiments. Furthermore, this study presents a comparison analysis between the mirrored method and the Horovod framework and provides insights into maximizing training using data parallelism.**

I. Introduction

Deep Neural Networks (DNNs) are instrumental in advancing the field of artificial intelligence, providing the computational architecture required for sophisticated tasks such as natural language processing, image recognition, autonomous navigation. These sophisticated models, characterized by their deep layers and intricate connections, demand substantial computational power, particularly for training on extensive datasets.

TensorFlow, an open-source platform developed by Google, is specifically designed for creating and training such DNNs. Its flexibility and extensive feature set make it a preferred choice for both research and production in numerous AI domains. TensorFlow supports both CPU and GPU computing, allowing for versatile deployment scenarios from mobile devices to distributed computing environments on cloud platforms [1] [2].

One of the advanced applications of DNNs in TensorFlow is in the field of semantic image segmentation, where DeepLabV3+ with Transformer models represent a significant advancement. These models combine the strengths of convolutional neural networks (CNNs) with transformers, an architecture that excels in capturing global dependencies within data. DeepLabV3+ efficiently handles the localization of objects at a pixel level, while transformers process contextual relationships over entire images, making this combination highly effective for detailed and accurate segmentation tasks [3]. However, the complexity and size of DeepLabV3+ with transformers demand an extraordinary number of computational resources, particularly for training over large datasets.

To overcome the inherent challenges in training large-scale models, such as extensive training times and memory limitations on single GPU systems, the field has turned towards High-Performance Computing (HPC) solutions. These include advanced GPU setups and, more critically, distributed and parallel training strategies. This shift is crucial as the size of the models and the datasets often exceed the capabilities of individual computing units, necessitating a distributed approach to DL training [4]. In the realm of distributed training, data parallelism is a core technique used to enhance the performance of model training. This method involves partitioning the input data among multiple processing units, each running a replica of the model. The local gradients from each unit are aggregated to update the global model weights, ensuring consistent learning across processors.

Horovod, a distributed training framework developed by Uber, significantly improves the scalability of Deep Neural Networks (DNNs) across multiple GPUs and nodes. It expands these capabilities to multi-node environments, enabling efficient and scalable training across numerous GPUs and multiple servers. Compatible with TensorFlow, Keras, PyTorch, and Apache MXNet, Horovod facilitates the effective distribution of workloads and synchronization of model parameters across all nodes. This efficiency not only reduces the bandwidth demands during training but also cuts down on the time needed for synchronization, allowing for the training of large models on extensive datasets in less time [5].

Furthermore, TensorFlow's Mirrored Strategy plays a pivotal role in this context on single machines equipped with multiple GPUs. This strategy ensures that each GPU handles a segment of the data while maintaining synchronous updates across the models, thereby optimizing resource utilization and reducing training times [6].

Data parallelism, through tools like Horovod and TensorFlow's strategies, addresses the substantial computational needs of training state-of-the-art models like DeepLabV3+ with transformers. These technologies ensure that developers can leverage the full spectrum of available computational resources, thus facilitating more efficient and faster training

processes for complex machine learning models in real-world applications.

II. RELATED WORK & BACKGROUND

A. Deep Learning Framework

Deep Learning (DL) has been a transformative force within the Artificial Intelligence (AI) landscape, impacting fields such as image recognition, natural language processing, and robotics. The renewed enthusiasm for DL is largely due to the advent of expansive datasets like ImageNet and advancements in powerful computational technologies, including GPUs and sophisticated multi-core CPUs. These improvements have enabled the training of intricate models on an unprecedented scale, ushering in a new era of AI capabilities. To facilitate this development, several DL frameworks like TensorFlow, PyTorch, and Caffe have been introduced [7]. These platforms streamline the process of model development by providing intuitive interfaces and abstracting complex mathematical details, allowing for the creation of custom DNN architectures suited to varied applications [8].

B. Semantic Segmentation Model

In the enhanced version of the DeepLabV3+ semantic segmentation model, a Swin Transformer has been integrated into the encoder to evaluate its effectiveness on a dataset specifically designed for a 256x256 image dimension. This model retains the core features of the original DeepLabV3+, including atrous convolution to manage context at multiple scales and an enhanced Atrous Spatial Pyramid Pooling (ASPP) module with image-level features for robust multi-scale segmentation [9]. The encoder incorporates the Swin Transformer, while the decoder utilizes the DCNN model Xception, enhancing segmentation accuracy, particularly around object edges. The architecture integrating the Swin Transformer with the DeepLabV3+ system is depicted in Fig.1.

However, swin Transformers, which process shifted windows of attention in hierarchical structures, require more computation and memory compared to traditional DCNNs, especially as image sizes increase. To manage these demands, data parallelism techniques are implemented to optimize training speed and resource utilization.

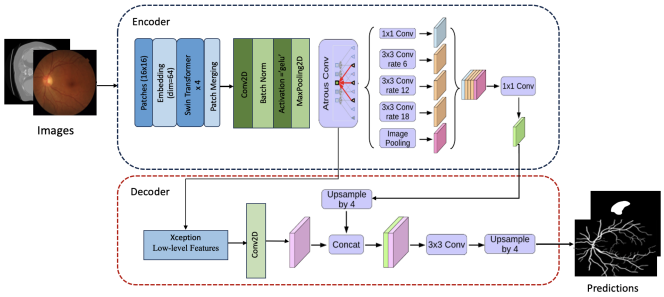


Fig. 1: Proposed DeepLabV3+ Architecture

C. Data Parallelism

Data parallelism is an essential technique in deep learning that enhances the training of complex neural networks by distributing workloads across multiple processing units, such as GPUs or nodes in a cluster. This method significantly reduces computation times by allowing tasks to be executed simultaneously rather than sequentially.

A key benefit of data parallelism is its ability to scale. It divides large datasets into smaller batches that are then processed in parallel across different hardware, speeding up the training process while bypassing the memory limitations of single machines by leveraging the combined memory and processing power of multiple units. Various libraries support this approach, including TensorFlow, PyTorch, Horovod [10], Apache MXNet [11], Microsoft's DeepSpeed [12], NVIDIA's NCCL , Intel oneAPI Deep Neural Network Library (oneDNN), and H2O. These tools provide robust frameworks that enable concurrent data processing across various GPUs or nodes, significantly enhancing the speed and efficiency of training deep learning models.

GPU-accelerated computing: In 2011, Nvidia introduced the first GPGPU, the Femi GPU, with 01 Teraflop performance and 175 GBytes/second bandwidth. Since then, Nvidia has released improved models like Tesla T4, P100, V100, and A100, optimized for AI and Deep Learning. The A100, based on Volta architecture with Tensor Core tech, delivers a 900% boost over P100 and 700% over V100 in AI inference. With TF32 precision, it achieves 20X higher performance without code changes, and an extra 2X with automatic mixed precision and FP16 [4].

D. Horovod

Horovod is an open-source framework that scales deep learning models across multiple GPUs or machines. By using the ring-Allreduce algorithm to facilitate fast inter-GPU communication, it minimizes the normal overheads associated with large-scale training and addresses the scaling issues associated with moving from one to numerous GPUs. Horovod has demonstrated its ability to cut training times in a variety of applications, including semantic segmentation and image classification. Rakshith et al. [13] used Horovod for distributed training on standard datasets like CIFAR-10 and Cats vs. Dogs. They highlight the Horovod's abilities to enhance scalability and reduce training times while maintaining or improving accuracy, setting a performance benchmark for these datasets. Quentin Anthony et al. [8] demonstrated the potential of Horovod combined with MVAPICH2-GDR on the Summit supercomputer for training semantic segmentation models, such as DeepLab-v3+. Their results show that Horovod can be optimized for near-linear scaling on HPC systems, significantly cutting down the training time for complex tasks.

E. Mirrored strategy

Ponnuswami et al. [14] evaluated TensorFlow's Mirrored Strategy (MS), Multi-Worker Mirrored Strategy (MWMS),

and Parameter Server Strategy (PSS) across various models and datasets. It highlighted MS as suitable for smaller datasets, MWMS for larger synchronized environments, and noted PSS's potential improvements with enhanced network infrastructure. Another paper evaluates distributed and parallel training strategies for deep learning, comparing TensorFlow and PyTorch APIs across tasks like prediction, detection, and classification using datasets such as MNIST, COCO-2017, and flowers. It highlights the effectiveness of strategies like Mirrored and TPU Strategy in TensorFlow and Distributed Data Parallel Training in PyTorch in reducing training times and enhancing performance [6]. Nguyen et al. (2020) [4] assess the efficacy of TensorFlow 2.2's MirroredStrategy in distributed training, demonstrating significant reductions in training times across various datasets and GPU configurations. This study highlights the practical advantages of distributed training in optimizing deep learning processes within high-performance computing settings. The paper [15] introduces an innovative approach to driver attention assessment, combining deep learning and eye-tracking technology. It extensively explores parallelization techniques, showcasing the effectiveness of both the Horovod framework for scaling CNNs and the mirrored strategy in TensorFlow for multi-threaded processing. Horovod demonstrates superior computational efficiency compared to the mirrored strategy in TensorFlow for multi-threaded processing.

III. EXPERIMENT & EVALUATION

A. Data Description

The study leverages two pivotal datasets to advance the field of medical image segmentation.

- **Retinal vessel segmentation dataset:**
  FIVES is a Fundus Image Dataset specifically designed for AI-based Vessel Segmentation, introduced in 2022. It stands as the largest vessel segmentation dataset to date, encompassing 800 RGB fundus images, each meticulously annotated at the pixel level. These images boast a high resolution of 2048x2048 pixels and are divided into a 75:25 train-test split ratio. The dataset features four distinct categories: healthy, diabetic retinopathy (DR), age-related macular degeneration (AMD), and glaucoma.
- **Spleen dataset from MSD:**
  The spleen dataset extracted from the Medical Segmentation Decathlon (MSD 09) comprises 61 portal-venous phase CT scans, meticulously segmented into 41 training volumes and 20 test volumes. The training set contains a total of 3,653 slice images, each accompanied by corresponding segmentation masks. However, the test volumes are provided without masks. In this report, the spleen dataset is referenced as MSD, indicating its origin from the Medical Segmentation Decathlon.

B. Data Pre-Processing

In this project, images and corresponding masks were captured at a resolution of 256x256 pixels. The training dataset was split into an 80:20 ratio for training and validation sets, respectively. Augmentation techniques were applied, including random horizontal and vertical flips, as well as a random rotation range of -30 to 30 degrees, enhancing the diversity of the dataset.

Hyper-parameter settings: Batch Size– 8/16, Optimizer– Adam, Learning rate– 0.01, Loss function- Dice loss with Binary cross-entropy, Epochs- 100.

C. System specifications

The parallel distribution ran in the Lonestar6 server. The execution configuration varied from 1 GPU to 6 GPUs and used 1 and 2 nodes. The specifications of server is described in Table I. The GPU model present in Lonestar6 is NVIDIA A100 PCIe with 40GB of memory.

TABLE I: System specifications

Specification	Value
GPU	NVIDIA A100 PCIe 40GB
GPU Memory	40 GB HBM2
Total cores per node	128 cores on two sockets (64 cores/socket)
Cache	• 32KB L1 data cache per core • 512KB L2 per core • 32 MB L3 per core complex

D. Horovod

The first method for data parallelism utilizes Horovod [10], which employs the "ring-allreduce" algorithm to synchronize data such as averaging gradients and updating model parameters across all participating computing units, as illustrated in Figure 2. This efficient data aggregation and distribution process is managed through the Message Passing Interface (MPI), optimizing bandwidth usage and reducing communication overhead. Horovod's approach significantly enhances the scalability and speed of training deep learning models across multiple GPUs or servers.

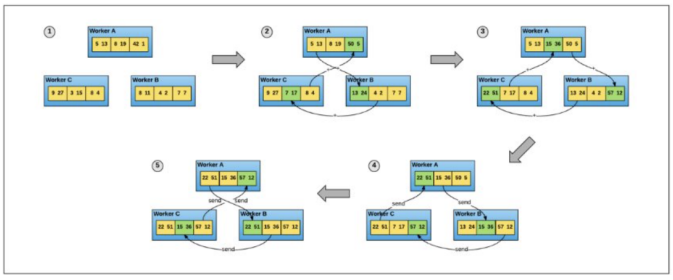


Fig. 2: Ring-allreduce process of Horovod [10].

1) Tasks: The tasks for this method were divided into three main types:

- **Data Loading**
  - This task is independent across the different processes. Each rank loads a subset of the dataset.

- **Distributed Model Training**
  - The distributed train behaves similarly to a workpool model. The workers produce the gradients and consume the synchronized parameters. The synchronization process is done by all-reduce operation in a circular process, as illustrated in Figure 2.
  - This task has independence between processes, thus it may introduce bottlenecks.
- **Metrics Calculation**
  - The last task is the final metric calculation. This task is done by process 0. Therefore, rank 0 needs to wait for the completion of model training across all ranks.

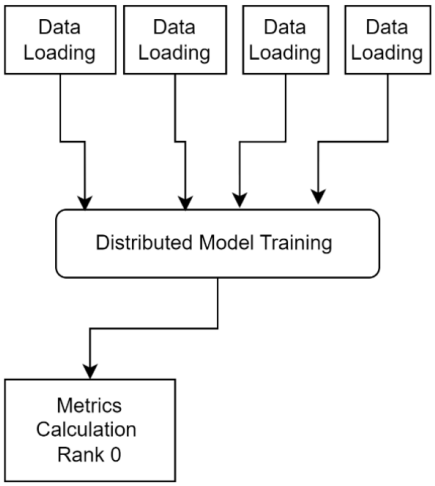


Fig. 3: Task dependency graph of Horovod approach.

2) Implementation: The setup of Horovod consists of setting the GPU for multiple processes, then setting the gradient to be processed with ring-allreduce method, lastly, it is necessary to set the broadcasting hook from rank 0, which synchronizes initial parameters across all nodes, ensuring uniform training conditions and maximizing computational efficiency. Figure 4 illustrates the basic setup steps.

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]
```

Fig. 4: Horovod setup implementation [10].

E. Mirrored strategy

The Mirrored Strategy in TensorFlow, termed 'tf.distribute.MirroredStrategy', utilizes synchronous data parallelism, ensuring each training step is synchronized across all devices. It's specifically engineered to distribute the training workload across multiple GPUs within a single machine.

1) Defining Tasks:: The key tasks involved in TensorFlow's MirroredStrategy are:

- Data Distribution: Distribute batches of data evenly across all available GPUs.
- Local Computation: Each GPU performs forward and backward passes independently using its batch of data.
- Gradient Calculation: Each GPU calculates gradients based on its data.
- Gradient Aggregation: All GPUs synchronize by aggregating gradients to ensure consistent updates across all models.
- Parameter Update: Update the model parameters on each GPU based on the aggregated gradients.

2) Execution model:: The execution model for the MirroredStrategy can be best described as a workpool model as each GPU acts independently, pulling its own "work" (data batches) and performing computations. The coordination point is the gradient aggregation, which synchronizes the state across all workers (GPUs) to maintain consistency.

3) Task Dependency graph:: The Fig.4 illustrates a distributed deep learning training cycle using a Mirrored Strategy with two GPUs. Each GPU independently computes gradients and loss from its data subset, using identical copies of the model. The computed gradients are sent to the CPU, where they are aggregated to synthesize the learnings from both GPUs. The CPU then updates the model's parameters based on these combined gradients and redistributes the updated parameters back to each GPU. This ensures that both GPUs operate with the same model state in subsequent iterations, maintaining consistency and synchronicity across devices, which is crucial for the effective convergence and accuracy of the model

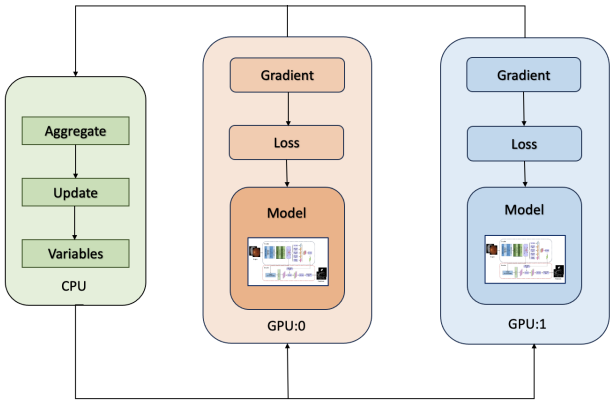


Fig. 5: Mirrored Strategy Task Dependency Graph

4) Implementation:: An example of how to implement mirror-strategy with a simple DL model is shown in Fig.6.

```
import tensorflow as tf

# Initialize MirroredStrategy to use all available GPUs.
strategy = tf.distribute.MirroredStrategy()

# Initialize MirroredStrategy to use specific GPUs
strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])

#Print the number of devices being used by the strategy.
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))

def build_and_compile_cnn_model():
    model = tf.keras.models.Sequential([
      tf.keras.layers.Dense(512, activation='relu', input_shape=(784,)),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(10)
    ])
    model.compile(
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
        metrics=['accuracy'])

    return model

# Define the strategy scope for model building and compilation.
with strategy.scope():
    multi_model = build_and_compile_cnn_model()

# Train the model using a distributed strategy.
multi_model.fit(dataset, epochs=3, steps_per_epoch=70)
```

Fig. 6: Mirrored strategy implementation

F. Limitation:

The Lonestar6 server has 6 gpus, with one node having 3 gpus.However, the Mirrored_strategy only work for single device with multple gpus. That means it only worked for one node with 3 gpus. The multi-worker mirrored strategy could access multi nodes, however, Lonestar6 can not access the port to connect which means it does not work for multi worker with tensorflow mulit-worker mirrored strategy library. The code is included in the github about the issue. Because of this, we can not compare Horovod with Mirrored_strategy for different nodes with 6 gpus.

G. Evaluation Metrics

The methods evaluation used three metrics: speed-up, efficiency loss, and image throughput. The speed-up (Equation 1) and image throughput (Equation 3) help us understand the scalability of the model. Meanwhile, the efficiency loss (Equation 2) indicates how effectively the computational resources are utilized.

- Speed up:

$$\text{Speedup}_p = \frac{T_1}{T_c} \tag{1}$$

where
T_1 = Execution time using 1 process,
T_c = Execution time using p processors

- Efficiency:

$$\text{Efficiency} = \frac{\text{Speedup}_p}{p} \tag{2}$$

where p = processors

- Image throughput:

$$\text{Throughput (img/sec)} = \frac{\text{Image}}{T_c} \tag{3}$$

where
Image = Number of Images,
T_c = Execution time using p processors

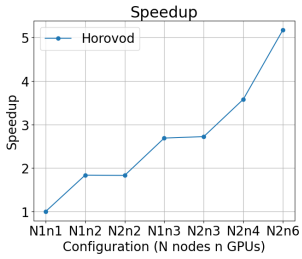
IV. Result

A. Horovod Result

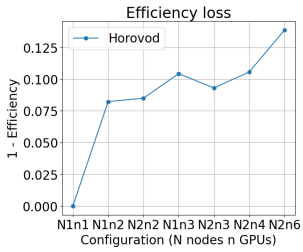
Data parallelism for Horovod was implemented across various configurations, with the number of nodes varying between 1 and 2. The number of GPUs ranged from 1 to 6, with each node accommodating a maximum of 3 GPUs. These configurations were used for the Retinal Vessel Segmentation dataset and for the combined dataset of Retinal Vessel Segmentation and Spleen from the Medical Segmentation Decathlon (MSD).

1) Retinal vessel segmentation dataset: The data parallelism using the Horovod approach demonstrated strong performance. As illustrated in Figure 8a and Figure 8b, increasing the number of processes improves speed-up and efficiency. However, while the speed-up remained consistent across different numbers of nodes with the same number of GPUs, efficiency decreased in the configuration N2n3. This reduction is attributed to the increased communication time between processes when using two nodes instead of just one. Similarly, throughput followed the trends observed in speed-up, further highlighting the model's robust scalability.

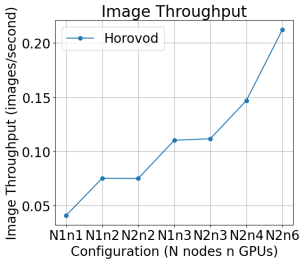
According to Figure 10d, the model's accuracy varied, with the last configuration showing a drop in value while the other configurations maintained a consistent average.



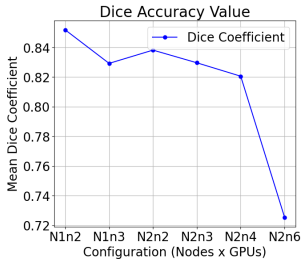
(a) Speed up for Fives dataset.



(b) Efficiency for Fives dataset.



(c) Throughput (img/sec)for Fives dataset.

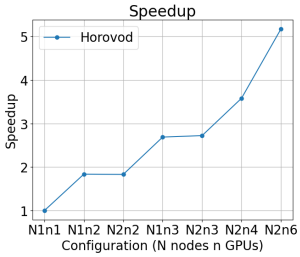


(d) Dice accuracy for Fives dataset.

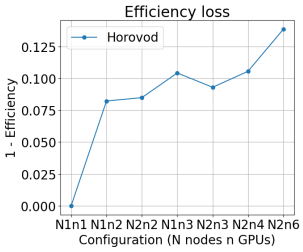
Fig. 7: Performance metrics and Dice accuracy for the Fives dataset with Horovod across different configurations.

2) Retinal vessel segmentation dataset plus Spleen dataset from MSD: The second experiment with Horovod involved training the model using both datasets combined. This integration led to an increase in data volume, training time, and complexity of data parallelism. However, the metrics for speed up and image throughput exhibited similar trends to those observed with the previous dataset, as illustrated in Figure 8a and Figure 8c. Additionally, there was a decrease in efficiency

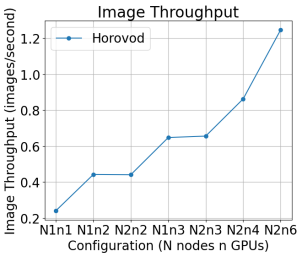
for configurations with two nodes and three GPUs, attributable to the increased communication time between nodes. Finally, while the accuracy remained satisfactory for the first dataset, it exhibited significant variability and poor performance for the MSD dataset, as shown in Figure 8d. These performance results demonstrate the challenges of scaling the model while maintaining consistent performance. A viable solution for this issue is to perform further fine-tuning of the model for the configuration with bad results.



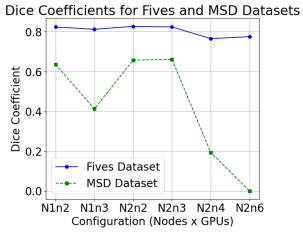
(a) Speed up for both datasets.



(b) Efficiency for both datasets.



(c) Throughput (img/sec) for both datasets.



(d) Dice accuracy for both datasets.

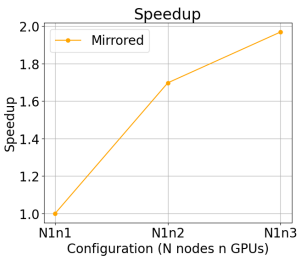
Fig. 8: Performance metrics and Dice accuracy of Horovod across different configurations.

B. Mirrored Strategy Result

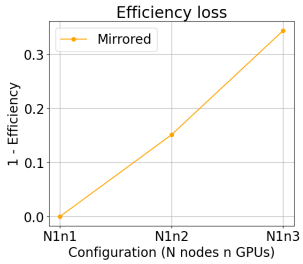
Data parallelism for Mirrored Strategy was implemented across various configurations, with the number of node1 and GPUs ranged from 1 to 3. These configurations were used for the Retinal Vessel Segmentation dataset and for the combined dataset of Retinal Vessel Segmentation.

1) Retinal vessel segmentation dataset:

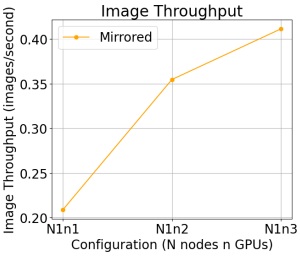
- Speedup:This plot in Fig.9 (a) shows the speedup gained by using mirrored configurations in nodes with different GPU counts. As the number of GPUs/nodes increases, the speedup also increases, suggesting improved parallel processing efficiency.
- Efficiency: The plot in Fig.9 (b) shows the efficiency loss as more nodes or GPUs are added. As the system scales from Nn1 to Nn3, efficiency loss increases, suggesting diminishing returns due to overhead costs such as communication and synchronization, which worsen as more units are added. This plot highlights the scalability limits and the challenge of maintaining efficient resource utilization in larger configurations.
- Throughout: The plot in Fig.9 (c) shows the image throughput (images processed per second) across different configurations. The plot indicates that throughput increases with more GPUs/nodes, demonstrating enhanced processing capability with more resources.
- Dice Accuracy: The plot in Fig.9 (d) shows the dice score of the model result on test dataset across node1-2 with 3 GPUS. As the number of GPUs increases, the dice scored showed some improvement making the range from 0.82 to 0.855.



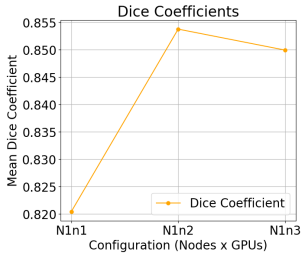
(a) Speed up



(b) Efficiency



(c) Throughput



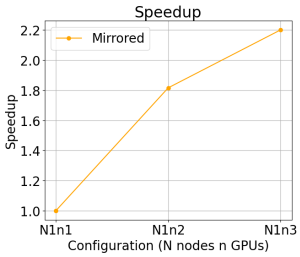
(d) Dice accuracy for fives datasets.

Fig. 9: Mirrored result evaluation for FIVES dataset

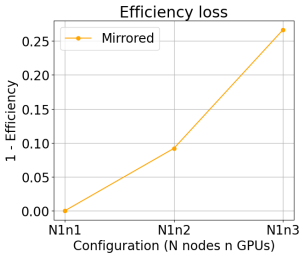
2) Retinal vessel segmentation dataset plus Spleen dataset from MSD:

- Speedup:This plot in Fig.10 (a) shows the speedup gained by using mirrored configurations in nodes with different GPU counts. As the configurations scale up, the speedup increases significantly, suggesting that the application or task benefits from additional parallelism.
- Efficiency: The plot in Fig.10 (b) shows the efficiency loss as more GPUs are added. As the system scales from N1n1 to N1n3, efficiency loss increases, suggesting diminishing returns due to overhead costs such as communication and synchronization, which worsen as more units are added. This plot highlights the scalability limits and the challenge of maintaining efficient resource utilization in larger configurations.
- Throughout: The plot in Fig.10 (c) shows the image throughput (images processed per second) across different GPU configurations. The throughput increases with each configuration upgrade, reflecting improved capability to process more data as more computational resources are added.
- Dice Accuracy: The plot in Fig.10 (d) shows the dice score of the model result on test dataset across node 1 with 3 GPUS. For five dataset, as the number of GPU

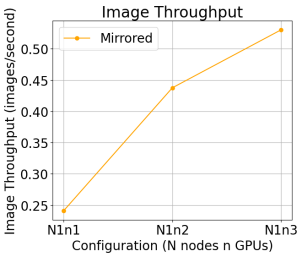
increases, the dice score stayed almost stable in between 0.85 to 0.76. However, For two dataset, as the number of GPU becomes 3, the dice score decreased drastically from 0.89 to 0.3. The accuracy differences happen because of how well the GPUs communicate and sync up. Factors like network speed and workload distribution affect how effectively they work together, leading to poor performance.It can be improved if the model training configuration could fine tuned with different batch size, epoch or optimization function.



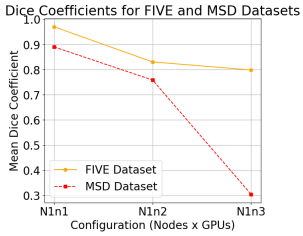
(a) Speed up



(b) Efficiency



(c) Throughput



(d) Dice accuracy for both datasets.

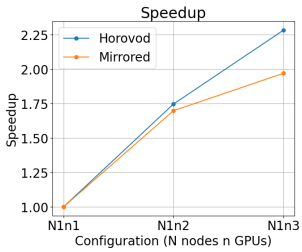
Fig. 10: Mirrored result evaluation for FIVES+MSD dataset

C. Comparative Analysis

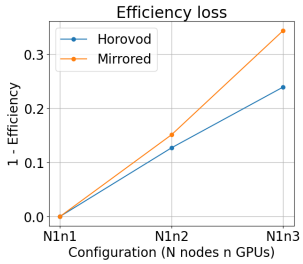
Lastly, we compared the performance of both approaches. The comparison focused on configurations with one node and the number of GPUs in the range from 1 to 3.

1) Retinal vessel segmentation dataset: The small dataset demonstrated good performance for both methods. However, as illustrated in Figures 11a and 11c, the Horovod approach achieved superior speed and throughput, indicating enhanced scalability. However, the Horovod's efficiency was lower than the mirrored indicating that the computational resources were better used in mirrored.

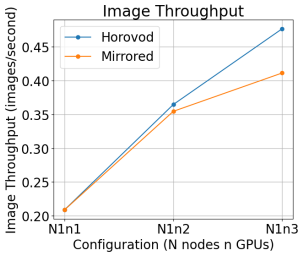
2) Retinal vessel segmentation dataset plus Spleen dataset from MSD: The comparison of both approaches as data volumes increase shows a behavior consistent with that observed in smaller datasets. However, from the efficiency loss graph presented in Figure 12b, it is evident that the Mirrored Strategy outperforms Horovod in terms of resource optimization. The graphs for Speedup (Figure 12a) and Throughput (Figure 12c) confirm previous findings that Horovod offers superior scalability compared to the Mirrored Strategy.



(a) Comparison of speed up between Horovod and Mirrored for Fives dataset.

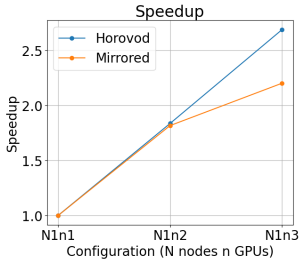


(b) Comparison of efficiency between Horovod and Mirrored for Fives dataset.

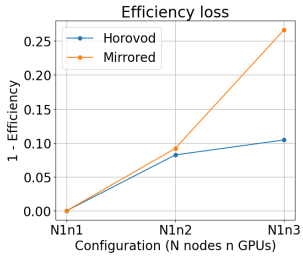


(c) Comparison of throughput between Horovod and Mirrored for Fives dataset.

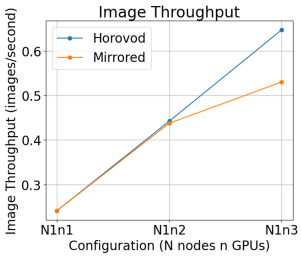
Fig. 11: Comparison analysis for the FIVES dataset



(a) Comparison of speed up between Horovod and Mirrored for both datasets.



(b) Comparison of efficiency between Horovod and Mirrored for both datasets.



(c) Comparison of throughput between Horovod and Mirrored for both datasets.

Fig. 12: Comparison analysis for FIVES plus MSD dataset

V. Conclusion

In this project, we have worked with two data parallelism techniques, Horovod and Mirrored Strategy, for a TensorFlow Deep Learning semantic segmentation model. Based on the experimental dataset, Horovod emerges as significantly

more effective than the Mirrored strategy when handling distributed computing tasks across various GPU configurations. Horovod demonstrates superior scalability, efficiency, and throughput, positioning it as the preferred option for large-scale and intricate machine learning operations within high-performance computing environments. Future research can delve into investigating the impact of different batch sizes on performance within Horovod and Mirrored strategies. Additionally, conducting comparisons between Horovod and Mirrored alongside other data parallelism libraries can further elucidate the optimal frameworks for scaling purposes.

REFERENCES

[1] Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020.

[2] Developers TensorFlow. Tensorflow. *Site oficial*, 2018.

[3] Bhakti Baheti, Shubham Innani, Suhas Gajre, and Sanjay Talbar. Semantic scene segmentation in unstructured environment with modified deeplabv3+. *Pattern Recognition Letters*, 138:223–229, 2020.

[4] Nguyen Quang-Hung, Hieu Doan, and Nam Thoai. Performance evaluation of distributed training in tensorflow 2. In *2020 International Conference on Advanced Computing and Applications (ACOMP)*, pages 155–159. IEEE, 2020.

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

[6] Hassam Tahir and Eun-Sung Jung. Performance analysis and comparison of distributed training strategies for deep learning. 01 2022.

[7] Ammar Ahmad Awan, Jereon Bédorf, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K Panda. Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 498–507. IEEE, 2019.

[8] Quentin Anthony, Ammar Ahmad Awan, Arpan Jain, Hari Subramoni, and Dhabaleswar K DK Panda. Efficient training of semantic image segmentation on summit using horovod and mvapich2-gdr. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1015–1023. IEEE, 2020.

[9] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[10] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.

[11] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[12] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[13] RM Rakshith, Vineet Lokur, Prateek Hongal, Vivek Janamatti, and Satyadhyan Chickerur. Performance analysis of distributed deep learning using horovod for image classification. In *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 1393–1398. IEEE, 2022.

[14] Ganesan Ponnuswami, Sriram Kailasam, and Dileep Aroor Dinesh. Evaluating data-parallel distributed training strategies. In *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 759–763. IEEE, 2022.

[15] Laura Rocchia, Cesare Sassolia, Roberto Da Viàb, and Eric Pascolob. Shape project nier: Deep learning for video and time series analysis.