Efficient Training of Semantic Image Segmentation on Summit using Horovod and MVAPICH2-GDR

Quentin Anthony, Ammar Ahmad Awan, Arpan Jain, Hari Subramoni, and Dhabaleswar K. (DK) Panda
Dept. of Computer Science and Engineering
The Ohio State University
{anthony.301, awan.10, jain.575, subramoni.1, panda.2}@osu.edu

Abstract*—Deep Learning (DL) models for semantic image segmentation are an emerging trend in response to the explosion of multi-class, high resolution image and video data. However, segmentation models are highly compute-intensive, and even the fastest Volta GPUs cannot train them in a reasonable time frame. In our experiments, we observed just 6.7 images/second on a single Volta GPU for training DeepLab-v3+ (DLv3+), a state-of-the-art Encoder-Decoder model for semantic image segmentation. For comparison, a Volta GPU can process 300 images/second for training ResNet-50, a state-of-the-art model for image classification. In this context, we see a clear opportunity to utilize supercomputers to speed up training of segmentation models. However, most published studies on the performance of novel DL models such as DLv3+ require the user to significantly change Horovod, MPI, and the DL model to improve performance. Our work proposes an alternative tuning method that achieves near-linear scaling without significant changes to Horovod, MPI, or the DL model. In this paper, we select DLv3+ as the candidate TensorFlow model and implement Horovod-based distributed training for DLv3+. We observed poor default scaling performance of DLv3+ on the Summit system at Oak Ridge National Laboratory. To address this, we conducted an in-depth performance tuning of various Horovod/MPI knobs to achieve better performance over the default parameters. We present a comprehensive scaling comparison for Horovod with MVAPICH2-GDR up to 132 GPUs on Summit. Our optimization approach achieves near-linear (92%) scaling with MVAPICH2-GDR. We achieved a "mIOU" accuracy of 80.8% for distributed training, which is on par with published accuracy for this model. Further, we demonstrate an improvement in scaling efficiency by 23.9% over default Horovod training, which translates to a $1.3\times$ speedup in training performance.

***Index Terms*—DNN Training, Performance Characterization, MVAPICH2 MPI, TensorFlow, Horovod, Image Segmentation**

I. Introduction and Motivation

Deep Neural Networks (DNNs)[1] have revolutionized many computer vision tasks, and are at the forefront of state-of-the-art image processing problems such as image classification, object detection, and semantic image segmentation. Simply put, a DNN is a directed, weighted graph with non-linear mappings between an input *x* and a learned output *y*. Nodes in the graph are labeled as *neurons*, and each subsequent set of neurons are grouped into *layers*. Input image data is passed through each layer of the DNN sequentially. Training the DNN entails adjusting the weighted connections to produce a function f such that $y = f(x)$. The trained DNN may then be applied to new data to produce predictions. **Training DNN models is compute/communication intensive, and much effort has been extended to train DNNs on HPC systems [1] [2] [3] [4].**

Deep Convolutional Neural Networks (DCNNs) are an extension of traditional DNNs to image-related applications. DCNNs have achieved highly accurate image classification by employing pooling layers, which reduce the input image's dimensions by combining the outputs of several previous neurons into a single neuron. A standard image classification DCNN is ResNet-50. Such pooling layers reduce the size of the network while improving the network's invariance to local image transformations. While invariance is desirable for high-level vision tasks such as image classification, it may be problematic for lower-level tasks such as semantic image segmentation, in which each individual pixel is classified according to its enclosing region.

A better DCNN for such low-level vision tasks should be sensitive to local image details instead of relying upon local pooling layers. Encoder-decoder image segmentation DCNNs achieve this behavior by introducing an encoder-decoder structure that first reduces the input image to its essential features, and then recovers the input image to its original resolution at the output layer. DeepLab-v3+ (DLv3+) is one successful example of the encoder-decoder structure that we evaluate in this paper. **A key difference between an image classification model (e.g. ResNet-50) and an image segmentation model (e.g. DLv3+) is that segmentation models are much more compute-intensive even for the same image size (see Figure 1).** We first compare the training throughput of ResNet-50 on a standard ImageNet image size (224x224) with the larger PASCAL VOC 2012 image size (513x513) typically used to train DLv3+. The ResNet-50 model is still almost 10× faster to train than a DLv3+ model on the same image size. This massive increase in computation indicates that segmentation models like DLv3+ could benefit significantly from distributed training on HPC systems. Further, while semantic image segmentation has become increasingly relevant for applications such as object detection, climate analysis, and autonomous driving, few studies have investigated the feasibility and

[1]We use the terms DNN and model interchangeably in this paper

978-1-7281-7445-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPSW50202.2020.00171

performance of distributed segmentation networks on HPC systems.

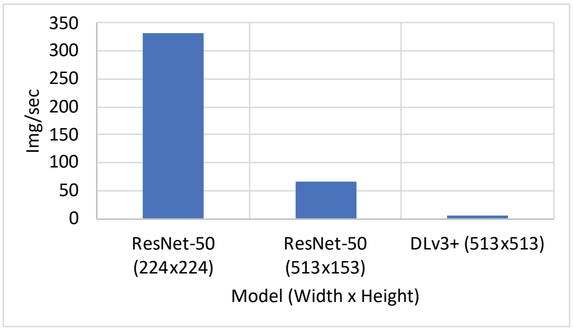

Fig. 1: Single-node performance for classification (ResNet-50) and segmentation (DLv3+) models on a V100 GPU

To the best of our knowledge, only the work of Kurth . [5] has scaled an image segmentation model to an HPC system, albeit with significant changes to Horovod's data-parallel support. We demonstrate that comparable performance is achievable by tuning the knobs of both Horovod and a GPU-Direct MPI communication backend. In this work, we first modify an existing segmentation network (DLv3+) written in Tensorflow to provide multi-node data-parallel support via Horovod, and then we examine the default scaling behavior on MVAPICH2-GDR and NCCL communication backends. Finally, we demonstrate the performance benefits that may be gained through system-level tuning and confirm that segmentation models trained on a distributed system still provide competitive accuracy compared to their counterparts trained on a single node.

A. Challenges

The key challenge addressed in this paper is: *How do we efficiently distribute image segmentation models after taking into account the unique characteristics of both the segmentation model and the HPC system?* We seek a distributed training method that may be easily applied to other novel DNN architectures. To answer this broad question, we solve the following concrete challenges:

- What are the key differences between segmentation and classification networks that affect their training performance at scale?
- How may existing segmentation networks be trained on HPC systems without changes to the model or communication library? (e.g. MPI, NCCL)
- What system-level optimizations may be made to efficiently train segmentation networks on distributed systems, given these key differences?

B. Proposed Approach

To solve these challenges, we propose a two-step process to distribute existing segmentation models. First, introduce data-parallel training support via Horovod to the existing training code. Second, fine-tune Horovod to support the increased model and image sizes required for semantic image segmentation.

C. Contributions

This work provides the user with an efficient approach to scale image segmentation without changes to the model, the distributed training framework (Horovod), or the communication runtime. Further, our proposed training approach is agnostic to the model, DL framework, and system used for DNN training. This is achieved by extensive tuning of various knobs for Horovod and the MPI runtime. To understand the communication characteristics, we rely on a diagnostic tool called *hvprof* [6]. We make the following key contributions in this paper:

- Demonstrate the productivity and performance shortcomings of existing distributed semantic image segmentation training methods.
- Establish the usefulness of Horovod and MPI profiling tools for distributing novel DNNs on GPU clusters, and apply them to demonstrate a 68.1% improvement (Table I) in allreduce
- Propose and evaluate system-level tuning to provide competitive segmentation training performance.
- Ensure that the prediction accuracy of segmentation models is not affected by distributed training
- We demonstrate the superiority of our tuned Horovod/MPI approach and report an improvement in scaling efficiency by 23.9% (Fig. 13), which translates to a $1.3\times$ speedup over the default approach

D. Organization

The rest of the paper is organized as follows. Section 2 provides the necessary background, including details on Horovod, Semantic image segmentation, and DeepLab. Section 3 contains an overview of the proposed optimizations to improve distributed training performance. Section 4 discusses characterization metrics, software libraries, and platforms used in this study. Sections 5 and 6 provide insights into hyperparameter optimization for single-node training and the scaling trend under Horovod and MVAPICH2-GDR's default settings. In Section 7, we show optimized scaling performance using our proposed design, and then we summarize key insights in Section 8. Section 9 contains related work, and we conclude with Section 10.

II. Background

A. DL Frameworks

DL frameworks are a user-transparent interface to define, train, and validate DL models on various CPU and GPU architectures. Such frameworks reduce development effort by providing modular building blocks that may be combined to create novel, reproducible DNN architectures. These frameworks hide complex mathematics from the users, enabling them to define and implement new layers and models tailored according to the end application. DNN training time depends on the DL model, the size of dataset, and the DL framework being used. While DNN training has become increasingly compute-intensive, few DL frameworks provide convenient distributed training features. Even with the limited framework support, however, the choice of communication backend may be complex. Training with TensorFlow, for example, can be distributed across nodes with gRPC, gRPC-X, and MPI/NCCL based solutions [7].

B. Data-Parallelism

Data parallelism is an approach to distributed DNN training that sends a copy of the DNN model to each CPU/GPU, and then training data is partitioned across all the processes. The number of samples sent to each node at each global training step is called the *batch size*. After each training step, the DNN parameters need to be synchronized by averaging the gradients among the processes. This is typically achieved with an MPI_allreduce, which performs an element-wise sum operation and sends the result to every process. The standard data-parallelism approach, synchronous training, requires each device's model copy to send an updated gradient before progressing to the next global training step. While asynchronous training approaches improve the throughput on individual nodes, training convergence becomes complicated to achieve.

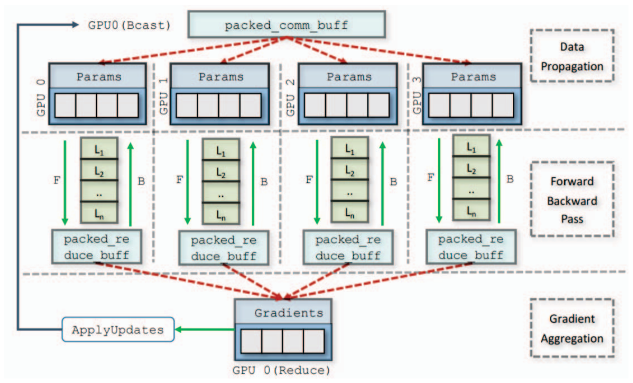

Fig. 2: Example data parallel DNN training on 4 GPUs [8]

C. Horovod

Horovod is a distributed DNN training framework that employs data parallelism to train DNNs [9]. Horovod employs the MPI_Allreduce and MPI_Bcast of a user-provided communication backend to perform data-parallel training. Horovod supports most communication runtimes such as MPI, DDL [10], and NCCL [11]. Horovod initializes a communication engine responsible for synchronization among processes, and employs optimization techniques such as Tensor Fusion to improve distributed training performance. Tensor fusion batches small allreduce operations into a single reduction operation, which improves DNN training performance. Tensor Fusion works as follows:

1) Determine which tensors are ready to be reduced. Select first few tensors that fit in `HOROVOD_FUSION_THRESHOLD` bytes and have the same data type
2) Allocate fusion buffer of size `HOROVOD_FUSION_THRESHOLD` if it was not allocated before. Default fusion buffer size is 64 MB
3) Copy data of selected tensors into the fusion buffer
4) Execute the allreduce operation on the fusion buffer
5) Copy data from the fusion buffer into the output tensors
6) repeat until there are no more update tensors to reduce in current cycle of length HOROVOD_CYCLE_TIME (Default is 3.5 ms)

Tuning these parameters may have significant performance benefits for novel DNN training workloads, and here we exploit this insight for semantic image segmentation. Currently, Horovod supports the TensorFlow, MXNet, PyTorch, and Keras DL frameworks. Horovod acts as a middleware between these frameworks and a communication backend as depicted in Figure 3

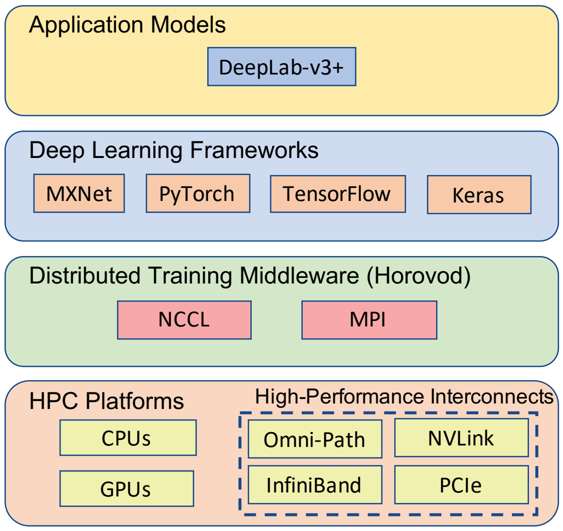

Fig. 3: Overview of a Distributed DNN Training Stack

D. Semantic Image Segmentation and Pascal VOC 2012

In image classification, each individual image of a dataset is assigned a class label. However, in the age of high-resolution multi-class image datasets, DCNNs for image classification do not provide sufficient detail of image contents. Semantic image segmentation, also termed pixel-level classification, produces a mapping from each pixel of an image to a distinct object class. The output of a segmentation model is a set of masks over each pixel cluster, which correspond to a set of distinct object classes. For an example segmentation data sample, see Figure 4. Semantic image segmentation is vital to several emerging applications such as autonomous driving, automatic medical diagnosis from imaging, or climate analysis [5].

Fig. 4: A sample image and ground-truth segmentation mapping from PASCAL VOC 2012. Courtesy: Everingham et al. [12]

In order to provide application-agnostic benchmarks for segmentation models, several freely-available datasets have become available such as Cityscapes, ADE20K, and PASCAL VOC. The PASCAL Visual Object Classes (VOC) dataset consists of 20 foreground object classes and one background class. The training and validation data has 11,530 images containing 27,450 region-of-interest (ROI) annotated objects and 6,929 segmentations. Users may evaluate the accuracy of a segmentation model by calculating the mean intersection-over-union (mIOU), which is defined as the intersection-over-union between the predicted and ground-truth segmentation masks averaged over each class in the image.

E. DeepLab

DeepLab is an extension of the U-Net segmentation architecture [13] based on Fully Convolutional Networks (FCNs) [14], [15]. DeepLab [16] is one of the most popular image segmentation models, and introduced three major improvements over past DCNNs for semantic image segmentation: upsampled filters or 'atrous convolution,' atrous spatial pyramid pooling (ASPP), and improved localized object boundaries. The latest iteration of DeepLab, DeepLab-v3+ (DLv3+), introduced an encoder-decoder structure that compresses image features while maintaining fine-grained details. A depiction of the encoder-decoder architecture of DLv3+ is contained in Figure 5. DLv3+ obtains state-of-the-art mIOU (89.0%) on the PASCAL VOC 2012 image segmentation dataset [17] without any post-processing. We chose DeepLab-v3+ as our evaluation model due to its popularity, performance, and open-source implementation.

III. Proposed Optimization Approach

We now describe our proposed approach to improve the performance of Deeplab distributed training. Broadly, we follow a three-phase optimization approach:

1) As DeepLab's default implementation is only for a single CPU/GPU, we must first realize a distributed version of DLv3+. Based on our findings in [7], we choose Horovod to implement distributed DLv3+. *Design details are discussed further in Section III-A.*

2) With a basic version of DLv3+ in hand, we identify performance bottlenecks using an in-house Horovod profiler called *hvprof* [6]. The major insight is that the default implementation using Horovod is not efficient at scale (See Figure 7). *More details are described in Section III-B.*

3) Implement a simple grid-search tuning strategy for Horovod as well as for the MPI runtime. Based on the profiling insights, we found the best parameters for Horovod and MPI. *Exact parameters used in this study are provided in Section III-C.*

A. Extending DeepLab to Support Horovod

Given the simplicity and formulaic structure of DNN training using a DL framework, Horovod support can be added in a general and model-agnostic manner. DNN training with a DL framework broadly follows the following guidelines:

1) Setup training data and apply preprocessing, if necessary.
2) Define model structure.
3) Declare optimizer and training hyperparameters.
4) Create a training loop or computational graph to carry out each training iteration.

With this DNN training pipeline in mind, we added Horovod support to DLv3+ by following the practice described by Horovod developers. The guidelines are as follows.

1) Map the processes to the GPUs on each node (typically one GPU per process).
2) Add a Horovod broadcast operation to set up the initial model parameters at each device.
3) Wrap the training optimizer in Horovod's distributed optimizer.
4) Scale the learning rate of the optimizer by the number of devices (Optional, but good practice to counteract the effective increased batch size).
5) Add logging at each training step to monitor training.

Following this approach, we have added Horovod distributed training to an implementation of DLv3+ in Tensorflow.

B. Profiling and Improving Performance of Communication

Our goal is to improve performance in a generalizable and simple manner. While creating a specialized Horovod extension as in [5] should provide nearly ideal performance, we seek to achieve comparable performance without changes to Horovod, MPI, and the model itself. MVAPICH2-GDR [18], a GPU-Direct RDMA (GDR) MPI implementation, coupled with effective Horovod tuning, can provide competitive performance without sacrificing user productivity or explicit changes to the application code. Applying this method, we collected the default distributed DNN training performance data discussed in Section VI.

We ran a DLv3+ training job for 100 steps with *hvprof* [6], which provides the user with a detailed profile of Horovod's

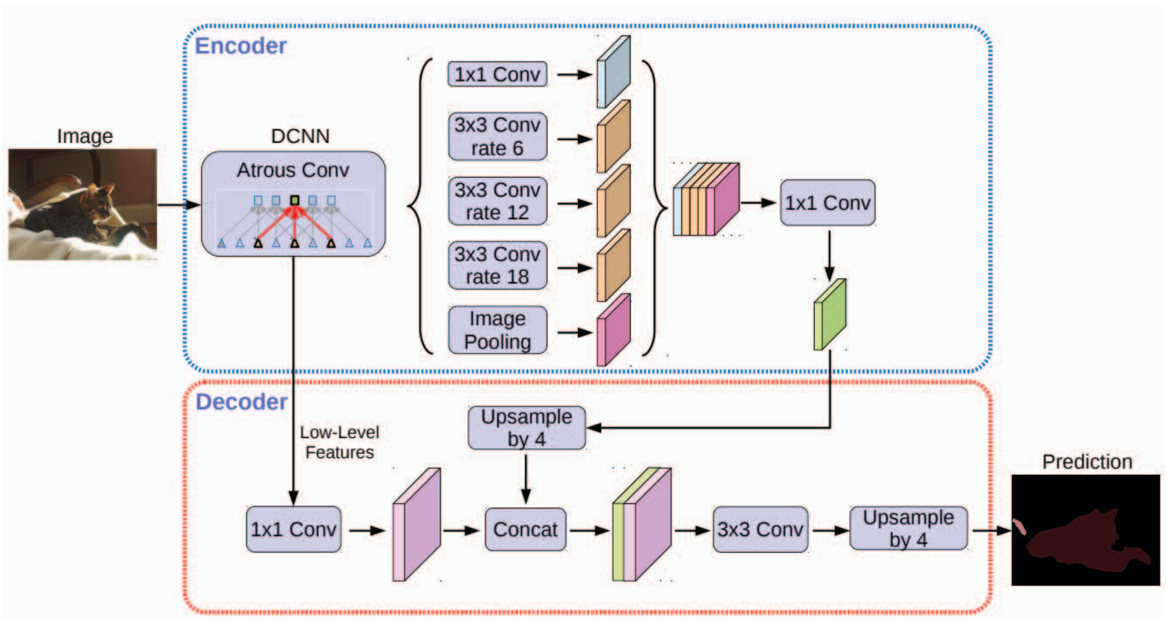

Fig. 5: The DeepLab-v3+ architecture, containing an encoder module that compresses image features via atrous convolution, and a decoder that recovers segmentation details at object boundaries. Courtesy: Chen et al. [17]

communication backend performance (MPI or NCCL) organized by message size and collective used. The *hvprof* output is depicted in Figure 11 and a speedup achievable for allreduce is in Section VII. After applying *hvprof* to a training run, we discovered the following insights into the communication bottlenecks present in the basic version of distributed semantic image segmentation models.

- The high-resolution dataset significantly increased the average message size for `MPI_Broadcast` operations.
- Large tensors were being delayed by a short default `HOROVOD_CYCLE_TIME` and a small default `HOROVOD_FUSION_THRESHOLD`.

C. Tuning Strategies to Improve Performance

With these insights in hand, we applied the following communication optimizations to provide efficient, reproducible, and scalable DLv3+ training without requiring any code changes to the model or to Horovod.

- We develop a grid-search tuning framework to find the best possible combination of `HOROVOD_FUSION_THRESHOLD` and `HOROVOD_CYCLE_TIME` to provide the best performance at every scale.
- We exploit MVAPICH2-GDR's tuning infrastructure to optimize the library for large message sizes being used by collectives like MPI_Broadcast and MPI_Allreduce.

The results of these tuning operations are discussed in detail in SectionVII. The performance benefits of tuning are directly compared with default performance in Section VI.

```
/* Create Horovod Parameters grid   */
Horovod_cycle_array     ← {};
Horovod_threshold_array ← {};
/* Create grid to store throughput  */
Result_array ← {{ }};
/* Search Grid                      */
foreach cycle_time in Horovod_cycle_array do
    foreach threshold in Horovod_threshold_array
     do
        train_deeplab(cycle_time, threshold);
        record(cycle_time, threshold, Result_array)
    end
end
```

Algorithm 1: Horovod Tuning with Grid Search

IV. Characterization Metrics and Platforms

We discuss different software libraries, evaluation platforms, and experiments needed to fully characterize DLv3+ training performance.

A. Evaluation Platforms

We performed all experiments on the Summit supercomputer at Oak Ridge National Laboratory (ORNL). It is the #1-ranked machine in the TOP500 as of November 2019 [19], and is composed of 4,608 nodes each with two IBM POWER 9 CPUs each connected to 3 NVIDIA Volta GPUS (V100) via NVIDIA NVLink. Each V100 GPU has 16 GB HBM2 memory.

B. Software Libraries

We use Tensorflow v1.14.0 compiled with CUDA 10.1.168 and CUDNN 7.6.1 on Horovod 0.18.1. Horovod was built against the MVAPICH-GDR 2.3.3 GPU-direct MPI library [20]. Evaluations with NCCL used NCCL 2.5.7-1 from GitHub[2]. The DeepLab-v3+ model was pulled and modified from the publicly-available implementation[3]

C. DeepLab Training

We used an *xception_65* backbone, with atrous rates of [6, 12, 18] and an output stride of 16. After performing the single-node evaluation depicted in Figure 6, we chose a training batch size of 4 images. Distributed training took approximately 300,000 training steps to converge for accuracy measurements.

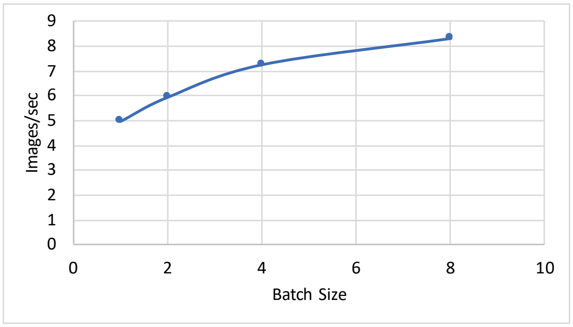

Fig. 6: Single-Node Batch Size Evaluation

V. Single-Node - Hyperparameter Optimization

Before distributing DLv3+, we performed an exhaustive single-node training evaluation to find the best hyperparameters. In particular, we sought the best batch size performance and hyperparameters to achieve a highly accurate mIOU on PASCAL VOC. For the single-node evaluation, we chose a batch-size of 4 and hyperparameters listed in Section IV-C.

VI. Default Scaling - Shortcomings and Insights

Once default Horovod support was added to DLv3+ as detailed in Section III-A, the scaling behaviour was taken up to 22 Summit nodes (132 V100 GPUs) by adding benchmarking support ($images/second$) to the DeepLab model. We compare results between NCCL and MVAPICH2-GDR to ensure that poor scaling results are not due to issues with any particular communication backend.

From Figures 7 and 8, we can see that, while performance is acceptable for a small number of nodes, the $images/sec$ quickly degrades at scale. This is due to the segmentation model's communication requirements detailed in Section III-B. Further, scaling efficiency drops to 60% for large node counts. The shortcomings of default scaling performance led us to perform an investigation into the system workload that segmentation models require. We performed an analysis with our Horovod/MPI profiler *hvprof* as detailed in Section III-B.

[2]https://github.com/NVIDIA/nccl
[3]https://github.com/tensorflow/models/tree/master/research/deeplab

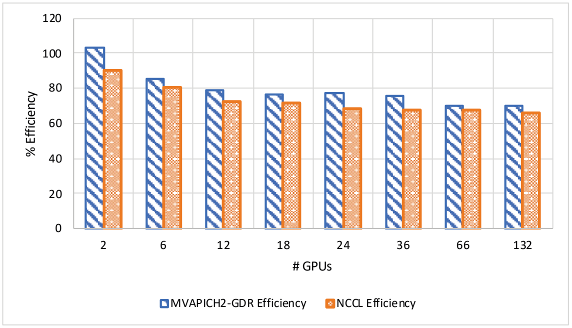

Fig. 7: Default DLv3+ Scaling Efficiency for Horovod built against MVAPICH2-GDR and NCCL

We note that we compare our performance to NCCL to identify the root cause of scaling inefficiencies. Clearly, the problem exists for both NCCL as well as MPI, which highlights that the high-level model characteristics and low-level communication runtime's knobs are not well-tuned to support each other. We seek to achieve a scaling efficiency above 90% by tuning both Horovod and the underlying MPI distribution, while leaving the basic structure of MPI/Horovod and the model unchanged.

VII. Optimized Scaling - Performance and Insights

After applying the system-level optimizations presented in Section III, we re-ran our training experiments and took scaling data up to 22 Summit nodes (132 GPUs). The optimized results are depicted below in Figures 9 and 10.

Further, we applied hvprof to demonstrate the benefits of optimized horovod by profiling 100 training steps of DLv3+ under default and optimal horovod parameters. The improvement for allreduce is depicted in Figure 11 and Table I. **We demonstrate a 68.1% improvement in allreduce over the default**

Message Size (Bytes)	Time (ms)		Percentage Improvement
	Default	Optimized	
0-10^5	1866.0	1635.1	12.3
$10^5 - 10^6$	1292.4	342.4	73.5
$10^6 - 10^7$	3503.4	724.5	79.3
$10^7 - 10^8$	21911.8	3234.0	85.2
$>10^8$	-	3176.8	
Total Time	28573.8	9112.9	68.1

TABLE I: Allreduce time performance improvement

With the tuned system, we achieve both near-linear scaling up to 132 GPUs and above 90% scaling efficiency. These results demonstrate the feasibility and benefits of our proposed approach. **In short, distributed training of novel DNN architectures may be significantly improved given an in-depth understanding of the characteristics of DNN**

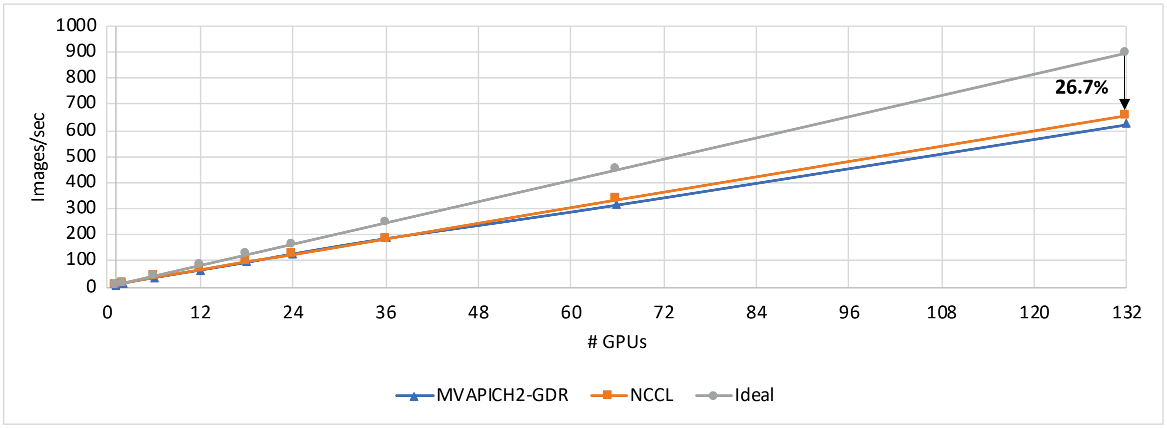

Fig. 8: Distributed DLv3+ Training Performance for Horovod built against MVAPICH2-GDR and NCCL

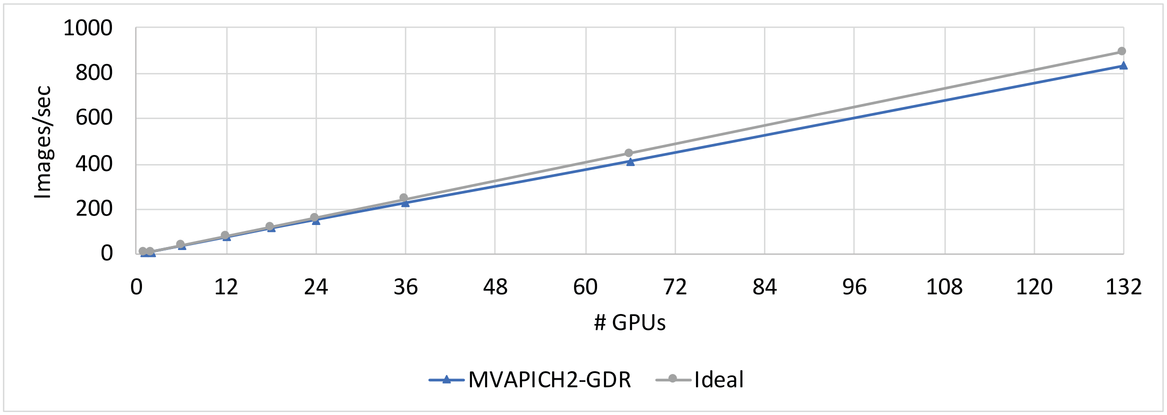

Fig. 9: Optimized Distributed DLv3+ Training Performance for Horovod built against MVAPICH2-GDR

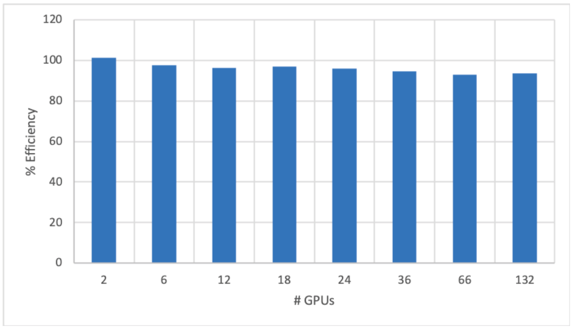

Fig. 10: Optimized DLv3+ Scaling Efficiency for Horovod built against MVAPICH2-GDR

workloads. This performance benefit doesn't necessarily require significant changes to the model or Horovod/MPI. With these results in hand, we find that a general and simple development pipeline to provide distributed training support to a novel DNN is as follows:

- Add Horovod support to an existing single-node training script via the steps in Section III-A.

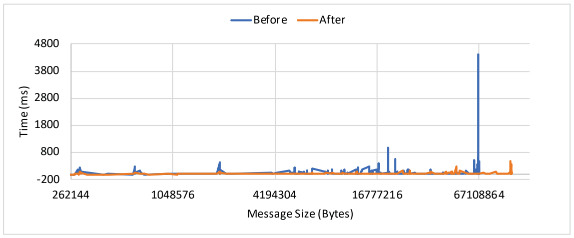

Fig. 11: Hvprof allreduce training profile for 100 training steps of DLv3+ on 12 GPUs

- Apply Horovod diagnostic tools to generate a profile and locate bottlenecks
- Given the insights gained from profiling, tune the relevant Horovod, MPI, and model parameters to provide efficient scaling

Finally, we seek to ensure that the model's segmentation accuracy is not significantly affected by distributed training. Using the hyperparameters listed in Section IV, we trained the model on 22 nodes (132 GPUs) on the PASCAL VOC 2012 dataset. We were able to achieve a mIOU of **80.84%**, which is comparable with the official published DLv3+ results.

VIII. Key Insights

We present our key insights as follows:

- Horovod support for existing models is feasible, and an abundance of documentation makes this step approachable, even if the model is a novel DNN architecture (e.g. DeepLab-v3+).
- While default Horovod support provides acceptable scaling efficiency for a small number of nodes, efficiency quickly drops off at larger scales.
- Without changing the underlying model/Horovod/MPI implementations, it is possible to tune the system scaling to provide competitive training performance.

We believe that this work provides a general training optimization pipeline for other compute and communication-intensive DNNs that could benefit from distributed training on HPC systems. Figures 12 and 13 illustrate these insights for distributed training with novel DNN architectures (in particular semantic image segmentation).

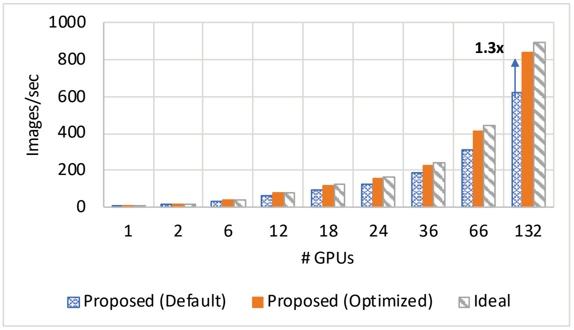

Fig. 12: Performance Improvement for DLv3+ Training after Tuning/Optimization

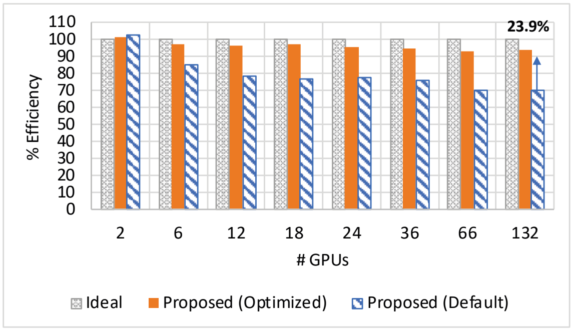

Fig. 13: Scaling Efficiency Improvement for DLv3+ Training after Tuning/Optimization

IX. Related Work

There are several studies available in the literature which use segmentation DNNs for climate analytics such as extreme weather prediction, [21]–[23], tropical cyclone detection [24], etc. DeepLab [16] is a widely used segmentation model based on CNNs, which is used in various applications like object detection [25], style transfer [26], etc.

Kurth et al. [5] extended the DeepLabv3+ and Tiramisu segmentation models to perform climate analytics. TensorFlow was coupled with Horovod to distribute DNN training using Data Parallelism and implemented a Hierarchical allreduce to scale the training to 27,360 GPUs. However, in this paper we used the MVAPICH2-GDR communication runtime and Horovod/MPI tuning to achieve competitive allreduce performance *without changing the code for Horovod/MPI*. Jacobs et al. [27] scaled Deep Generative models on scientific datasets using LBANN, a deep learning framework for distributed training. Data parallelism is coupled with model parallelism to scale a DNN to 1,024 GPUs with 64 trainers and achieved 109% parallel efficiency. CosmoFlow [28] uses DNN to determine the physical model which can be used to describe the universe. CosmoFlow uses TensorFlow and CPE ML plugin to implement distributed training for 3D convolution operation on 8192 KNL nodes and achieved 77% parallel efficiency. Cycle-Consistent Adversarial Networks (CycleGANs) [29] can be used to visualize the effect of climate change. Before and after images of locations, which have experienced floods, forest fires, and other natural disasters are used to train the network so that it can predict the outcome of natural disasters for other locations.

X. Conclusion

The computational workloads for Deep Learning are rapidly increasing for emerging image processing applications such as autonomous driving, automatic medical image diagnosis, and climate analysis [5]. The extreme computation and communication requirements of these applications provide an excellent opportunity for distributed DNN training. We demonstrate that scaling image segmentation analysis models to HPC systems is both feasible and approachable for existing single-node DNN implementations. Near-linear scaling may be achieved without significantly changing the structure of the model, Horovod, or MPI communication backend. Given an understanding of the communication requirements of the novel DCNN, we have achieved strong distributed training with above 90% scaling efficiency while maintaining a mIOU accuracy of 80.8% that is competitive with state-of-the-art single-node results. Further, our tuning method achieves an improvement in scaling efficiency by 23.9%, which translates to a 1.3× speedup in training performance. We believe that these results pave the way for efficiently training semantic image segmentation models and other novel DNNs that require long training times.

Acknowledgement

This research is supported in part by NSF grants #CNS-1513120, #ACI-1450440, #CCF-1565414, #ACI-1664137, #OAC-1818253, and #ACI-1931537.

REFERENCES

[1] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters," in *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. ACM, 2017, pp. 193–205.

[2] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, "Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes," *CoRR*, vol. abs/1807.11205, 2018. [Online]. Available: http://arxiv.org/abs/1807.11205

[3] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman, "Mesh-tensorflow: Deep learning for supercomputers," *CoRR*, vol. abs/1811.02084, 2018. [Online]. Available: http://arxiv.org/abs/1811.02084

[4] M. Yamazaki, A. Kasagi, A. Tabuchi, T. Honda, M. Miwa, N. Fukumoto, T. Tabaru, A. Ike, and K. Nakashima, "Yet another accelerated SGD: resnet-50 training on imagenet in 74.7 seconds," *CoRR*, vol. abs/1903.12650, 2019. [Online]. Available: http://arxiv.org/abs/1903.12650

[5] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, Prabhat, and M. Houston, "Exascale deep learning for climate analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 51:1–51:12. [Online]. Available: https://doi.org/10.1109/SC.2018.00054

[6] A. A. Awan, A. Jain, C.-H. Chu, H. Subramoni, and D. Panda, "Communication Profiling and Characterization of Deep Learning Workloads on Clusters with High-Performance Interconnects," in *Hot Interconnects 26 (HotI '19)*, August 2019.

[7] A. A. Awan, J. Bedorf, C.-H. Chu, H. Subramoni, and D. Panda, "Scalable Distributed DNN Training using TensorFlow and CUDA-Aware MPI: Characterization, Designs, and Performance Evaluation," in *The 19th Annual IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGRID 2019)*, May 2019.

[8] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. New York, NY, USA: ACM, 2017, pp. 193–205. [Online]. Available: http://doi.acm.org/10.1145/3018743.3018769

[9] A. Sergeev and M. Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018. [Online]. Available: http://arxiv.org/abs/1802.05799

[10] M. Cho, U. Finkler, S. Kumar, D. S. Kung, V. Saxena, and D. Sreedhar, "Powerai DDL," *CoRR*, vol. abs/1708.02188, 2017. [Online]. Available: http://arxiv.org/abs/1708.02188

[11] NVIDIA, "NVIDIA Collective Communication Library (NCCL)," https://docs.nvidia.com/deeplearning/sdk/nccl-developer-guide/docs/index.html, 2016, Accessed: March 16, 2020.

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[13] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351. Springer, 2015, pp. 234–241, (available on arXiv:1505.04597 [cs.CV]). [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a

[14] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[15] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[16] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[17] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[18] "NVIDIA GPUDirect RDMA," Accessed: March 16, 2020. [Online]. Available: http://docs.nvidia.com/cuda/gpudirect-rdma/

[19] H. Meur, E. Strohmaier, J. Dongarra, and H. Simon, "TOP 500 Supercomputer Sites," http://www.top500.org, 1993, [Online; accessed March 16, 2020].

[20] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE, https://mvapich.cse.ohio-state.edu/, 2001, [Online; accessed March 16, 2020].

[21] Y. Liu, E. Racah, Prabhat, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. F. Wehner, and W. D. Collins, "Application of deep convolutional neural networks for detecting extreme weather in climate datasets," *CoRR*, vol. abs/1605.01156, 2016. [Online]. Available: http://arxiv.org/abs/1605.01156

[22] G. Iglesias, D. C. Kale, and Y. Liu, "An examination of deep learning for extreme climate pattern analysis," in *The 5th International Workshop on Climate Informatics*, 2015.

[23] S. Kim, H. Kim, J. Lee, S. Yoon, S. E. Kahou, K. Kashinath, and M. Prabhat, "Deep-hurricane-tracker: Tracking and forecasting extreme climate events," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2019, pp. 1761–1769.

[24] D. Matsuoka, M. Nakano, D. Sugiyama, and S. Uchida, "Detecting precursors of tropical cyclone using deep neural networks," in *The 7th International Workshop on Climate Informatics, CI*, 2017.

[25] J. Han, D. Zhang, G. Cheng, N. Liu, and D. Xu, "Advanced deep-learning techniques for salient and category-specific object detection: a survey," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 84–100, 2018.

[26] F. Luan, S. Paris, E. Shechtman, and K. Bala, "Deep photo style transfer," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[27] S. A. Jacobs, B. V. Essen, D. Hysom, J.-S. Yeom, T. Moon, R. Anirudh, J. J. Thiagaranjan, S. Liu, P.-T. Bremer, J. Gaffney, T. Benson, P. Robinson, L. Peterson, and B. Spears, "Parallelizing training of deep generative models on massive scientific datasets," 2019.

[28] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook, K. Maschhoff, J. Sewall, N. Kumar, S. Ho, M. F. Ringenburg, P. Prabhat, and V. Lee, "Cosmoflow: Using deep learning to learn the universe at scale," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2018, pp. 819–829.

[29] V. Schmidt, A. Luccioni, S. K. Mukkavilli, N. Balasooriya, K. Sankaran, J. Chayes, and Y. Bengio, "Visualizing the consequences of climate change using cycle-consistent adversarial networks," *CoRR*, vol. abs/1905.03709, 2019. [Online]. Available: http://arxiv.org/abs/1905.03709