Performance Characterization of DNN Training using TensorFlow and PyTorch on Modern Clusters

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

***Abstract*—The recent surge of Deep Learning (DL) models and applications can be attributed to the rise in computational resources, availability of large-scale datasets, and accessible DL frameworks such as TensorFlow and PyTorch. Because these frameworks have been heavily optimized for NVIDIA GPUs, several performance characterization studies exist for GPU-based Deep Neural Network (DNN) training. However, there exist very few research studies that focus on CPU-based DNN training. In this paper, we provide an in-depth performance characterization of state-of-the-art DNNs such as ResNet(s) and Inception-v3/v4 on multiple CPU architectures including Intel Xeon Broadwell, three variants of the Intel Xeon Skylake, AMD EPYC, and NVIDIA GPUs like K80, P100, and V100. We provide three key insights: 1) Multi-process (MP) training should be used even for a single-node, because the single-process (SP) approach cannot fully exploit all the cores, 2) Performance of both SP and MP depend on various features such as the number of cores, the processes per node (ppn), and DNN architecture, and 3) There is a non-linear and complex relationship between CPU/system characteristics (core-count, ppn, hyper-threading, etc) and DNN specifications such as inherent parallelism between layers. We further provide a comparative analysis for CPU and GPU-based training and profiling analysis for Horovod. The fastest Skylake we had access to is up to 2.35× better than a K80 GPU but up to 3.32× slower than a V100 GPU. For ResNet-152 training, we observed that MP is up to 1.47× faster than SP and achieves 125× speedup on 128 Skylake nodes.**

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

I. Introduction

Deep Learning (DL) has become the primary approach to solve a multitude of established as well as emerging artificial intelligence (AI) challenges. Novel Deep Neural Network (DNN) architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) were proposed in the late 80s and early 90s, but their usage was limited due to the *overfitting* problem [1] and *limited computational* resources. Overfitting and the availability of small datasets led to the advancement in the unsupervised training of Machine Learning (ML) models such as Restricted Boltzmann Machines (RBMs) [2] and Deep Belief Networks (DBNs) [3]. The AlexNet [4] model proved the effectiveness of the DNN-based approach and reignited interest in DL. Broadly, the rise of DL can be attributed to the availability of large scale datasets, abundant computing resources, an ever-growing collection of DL models, and the development of accessible DL frameworks such as TensorFlow [5] and PyTorch [6].

With the availability of large datasets, models are becoming more complex. This in turn has led to a significant increase in the memory and computational resources required to train these models. DNN variants exhibit a certain degree of inherent parallelism; therefore, they can be distributed across multiple computing nodes [7], [8]. Today, NVIDIA Graphics Processing Units (GPUs) are widely used to train complex DNNs effectively by applying optimized cuDNN primitives and the CUDA programming model. There are several performance studies that offer insights into DNN training performance on GPUs [9]–[11]. Like all other DL frameworks, TensorFlow and PyTorch are highly optimized for GPUs and use cuDNN and CUDA kernels to provide excellent performance. However, little exists in the literature that deals with CPU-based DNN training, especially using Horovod with TensorFlow and PyTorch. Existing performance studies for CPU-based DNN training are largely limited to blog posts and best practice documents [12], [13]. To address this, we investigate the following broad challenges:

- What is the impact of the number of cores of a CPU on the performance of DNN Training?
- Does the batch size (BS) used to train a DL model impact the training performance on different CPU architectures?
- Do single process (SP) per node and multiple processes (MP) per node configurations exhibit different performance behavior? Which better exploits all the available CPU cores?
- What is the interplay between a CPU's clock speed and core count with a DNN's architecture?
- How can CPU and GPU performance be compared for various models and frameworks?
- What are the performance trends for TensorFlow and PyTorch?

A. Contributions

To the best of our knowledge, this is the first study that offers a systematic performance characterization of 1) CPU-based DNN Training using TensorFlow and PyTorch for

978-1-7281-4734-5/19/$31.00 ©2019 IEEE

computation and Horovod for communication across compute elements, and 2) describes how to achieve the best possible performance for a given HPC platform. In this paper, we evaluate the DNN training performance for different CPU architectures including the latest Intel Xeon Skylake and AMD EPYC processors. We utilize different CPU architectures and distributed training approaches to improve the training performance. Furthermore, we discuss how efficient multi-node training on CPUs can be realized by selecting the best process per node (ppn) and batch size configurations. In addition to CPU-based training, we also discuss some GPU-based results and offer a comparative analysis. We make the following key contributions:

- Provide performance characterization for ResNet(s) and Inception-based models on five different HPC platforms in single-process (SP), multi-process (MP), and multi-node (MN) configurations.
- Offer improvement strategies like using a multi-process (MP) approach over single-process (SP) (up to 1.47$\times$ better) for single-node training.
- Evaluate large-scale (up to 128 Skylake nodes) training experiments using ResNet (50, 101, and 152) as well as Inception v3 and v4.
- Report up to 125$\times$ speedup on 128 nodes for ResNet-152 training using TensorFlow and the MVAPICH2 MPI library.
- Summarize key insights (cf. Section IX) gained from our systematic evaluation and performance characterization of different DNNs on various HPC platforms.

II. Background

A. Deep Learning Frameworks

DL frameworks provide an interface to define, train, and validate DL models on various CPU and GPU architectures. They offer building blocks which can be used to design different types of DNNs and provide Automatic Differentiation methods to implement the back-propagation algorithm (Section II-B) transparent to the user for the designed DL model. Most DL frameworks such as TensorFlow, PyTorch, MXNet, and CNTK are optimized for GPUs. However, Intel has also optimized TensorFlow and PyTorch for Intel CPUs by providing and integrating the Intel MKL library.

B. Distributed DNN Training

DNNs are variants of neural networks that have more than 2 hidden layers. There are three types of layers in the DNN: 1) Input layer, 2) Hidden layers (compute intensive and map the input to the output), and 3) Output layer. DNN training is a compute-intensive task that consists of two phases: 1) Forward pass and 2) Backward pass. In the forward pass, the output is calculated using the feed-forward method, while in the backward pass the back-propagation algorithm is used to calculate the gradients that update the weights of the DNN. DNN training performance depends on several hyperparameters like learning rate and BS.

Broadly, there are two approaches to distribute DNN training: 1) Data parallelism and 2) Model parallelism. In data parallelism, the DL model is replicated across all the processes and data is distributed among these processes. All the processes perform forward and backward pass simultaneously and then communicate the gradient among themselves to update the weights of the model in a synchronous manner. In Data parallelism, the effective BS increases but the BS per worker remains constant. In model parallelism, the model is split across all the processes. Send and Recv communication operations are used to implement distributed forward and backward pass.

C. Horovod

Horovod is a popular distributed DNN training framework which uses data parallelism approach to train DNNs [14]. Horovod uses MPI_Allreduce and MPI_Bcast to implement distributed training and provides a high-level Python interface to the users. Currently, Horovod supports TensorFlow, MXNet, PyTorch, and Keras. Horovod can utilize communication runtimes like MPI, DDL [15], and NCCL [16]. Horovod initializes a communication engine which is responsible for the communication among different processes and uses optimization techniques like Tensor Fusion to improve the performance on multiple nodes. Tensor Fusion uses runtime parameters such as HOROVOD_FUSION_THRESHOLD and HOROVOD_CYCLE_TIME to tune communication performance based on the availability (readiness) of tensors that need to be exchanged.

III. Challenges in Characterizing Performance of CPUs for Deep Learning Workloads

There is a need to evaluate the DNN training process on CPUs in a holistic manner. This is because CPUs have evolved in terms of core counts, clock speed, and other characteristics like hyper-threading. In this section, we highlight some of the key challenges in characterizing the performance of CPU-based DNN training for both single-node as well as multi-node configurations.

A. What is the impact of the number of cores of a CPU on the performance of DNN Training?

As most of the current generation CPUs are multi-core, it is important to understand the performance all the way from a single CPU core to the maximum number of cores available. The previous generation Xeon (Skylake) CPUs have up to 48 cores, but it is not clear if all 48 cores can be fully exploited by frameworks like TensorFlow and PyTorch.

B. How can the BS of a DL model impact the training performance on different CPU architectures?

Most modern ML/DL frameworks use mini-batches of training data to perform DNN training. However, the BS is an important hyperparameter that requires adjustment in tandem with other hyperparameters like learning rate and the number of layers in a DL model. This is clearly a major challenge as most systems (CPUs and GPUs) offer better performance for larger BS because I/O (for reading training data from storage) and communication (for multi-process training) can be overlapped with computation. BS effectively increases the time of computation on GPUs before the next phase on I/O and communication is performed. However, the impact of BS on performance, purely from a throughput (images processed per second) standpoint, needs to be better understood for CPUs.

C. Will a single process per node or a multiple processes per node configuration better exploit the available cores?

As multi-core CPUs are the focus, it is not clear if multiple cores can be better utilized by using multiple threads (OpenMP/MKL-DNN) or by using multiple processes (MPI/ Horovod). This is true even for a single node without any inter-node (over the network) communication as well. Furthermore, it is important to investigate the effect of CPU mapping of the MPI processes for DNN training as well, which is handled differently by different MPI libraries.

D. What is the interplay between a CPU's clock speed and core count with a DNN's architecture?

CPU clock speed and core counts can impact the DNN training throughput but it is challenging to define a clear trend or a linear equation for these two characteristics. This is further exacerbated by the fact that DNN architectures vary widely in terms of inherent parallelism in the model itself. For example, ResNet(s) are very linear and there are few inter-op parallelism opportunities. However, Inception-based models are different and offer more potential to exploit inter-op parallelism. It is thus non-trivial to develop a relationship for these CPU characteristics and properties of various DNNs.

IV. Characterization Metrics and Platforms

We discuss different software libraries, evaluation platforms, and types of experiments needed to fully characterize performance across various hardware architectures and DL frameworks.

A. Evaluation Platforms

We are using four different HPC clusters for our experiments. Table I summarizes the CPUs on different HPC platforms being used for our experiments.

Architecture	Cluster	Speed (GHz)	Cores	Threads Per Core	label
Skylake	RI2	2.6	28	1	**Skylake-1**
Skylake	Pitzer	2.4	40	1	**Skylake-2**
Skylake	Stampede2	2.1	48	2	**Skylake-3**
Broadwell	RI2	2.4	28	1	**Broadwell**
EPYC	AMD-Cluster	2.0	32	4	**EPYC**

TABLE I
Evaluation Platforms

RI2 is an in-house cluster at The Ohio State University. There are 32 nodes on the RI2 cluster that are connected using Mellanox SB7790 and SB7800 InfiniBand switches. There are 20 nodes with two 14-core Intel (**Broadwell**) Xeon E5-2680 v4 2.4 GHz processors and 128 GB memory. The rest of the 12 nodes are equipped with two 14-core Intel (**SkyLake-1**) Xeon Gold 6132 2.6 GHz processors and 192 GB Memory. Skylake nodes are equipped with 2 NVIDIA K80 GPUs with 12 GB memory. Both Skylake and Broadwell nodes are equipped with a single-port InfiniBand EDR HCA.

Pitzer is the latest cluster at the Ohio Supercomputer Center (OSC) [17]. It is powered by Dell PowerEdge C6420 servers equipped with a dual-socket Intel Xeon 6148 CPU (**Skylake-2**) with 20 cores per socket clocked at 2.4 GHz and 192 GB main memory. The nodes are equipped with Mellanox InfiniBand EDR HCAs. There are 32 GPU nodes equipped with two Nvidia Volta V100 GPUs with 16 GB memory.

Stampede2 [18] is an NSF funded cluster at Texas Advanced Computing Center (TACC). There are two types of computing nodes: 1) KNL nodes and 2) Skylake nodes. We have conducted our experiments on the Skylake nodes, which are equipped with dual-socket Intel Xeon (**Skylake-3**) CPUs with 24 cores per socket clocked at 2.1 GHz. The nodes have 192 GB memory and are equipped with Intel Omni-Path Network adapter.

AMD-Cluster: We use a small 8-node cluster with the latest AMD processors. The nodes are connected to each other using the InfiniBand network. Each node has one dual-socket AMD **EPYC** 7551 CPU with 32 cores per socket. The nodes have 256 GB of main memory and have one Mellanox IB EDR HCA.

B. Software Libraries

We have conducted our experiments using the Intel optimized TensorFlow (v1.12), TensorFlow v1.12 (for GPUs and AMD processors), and PyTorch (v1.1) for DNN training. To facilitate distributed training using TensorFlow and PyTorch, we have used Horovod [14] built against the MVAPICH2 2.3 MPI library [19]. Training experiments use the *tf_cnn_benchmarks* [20] program for TensorFlow and Horovod's *pytorch_synthetic_benchmark* for PyTorch. In our experiments, **intra-op** refers to *–num_intra_threads* and **inter-op** refers to *–num_inter_threads* flag in the tf_cnn_benchmarks program. We ran each experiment three times and averaged the result to mitigate any jitters in the result.

C. Type of Experiments

Broadly, we perform five different types of experiments:

1) Single Node Single Process (SP) Experiments
2) Single Node Multi-Process (MP) Experiments
3) Multi-Node Multi-Process (MN) Experiments
4) GPU-CPU Comparisons
5) Horovod Profiling Experiment

Single Node Single Process (SP): The performance of a single process on a single node is evaluated, and the insights gained from these experiments are used to find an appropriate number of processes and BS for the multi-process experiments.

Single Node Multi-Process (MP): Through these experiments, our motivation is to investigate if the performance of a single node can be increased by running distributed training (via Horovod) using multiple MPI processes. Multi-process performance can be improved by increasing the BS. However, we focus on increasing the performance of DNN training while keeping the BS minimal on a single node. TensorFlow (tf_cnn_benchmarks program) allows inter-operator (computing independent operations simultaneously) and intra-operator parallelism (parallelism within one operation). For MP, we have to select an appropriate number of intra-op and inter-op threads based on the number of cores per process. In addition, we also evaluate the effect of hyper-threading on the number of intra-op and inter-op threads.

Multi-Node Multi-Process (MN): The goal of SP and MP experiments is to optimize the performance of DNN training on a single node and come up with a strong sequential baseline. After that, DNN training performance is evaluated on multiple nodes by utilizing the best configuration discovered by SP and MP experiments. As there are different types of DL models in the literature, we evaluate a broad variety of models so that the scalability trends can be better understood. We evaluate state-of-the-art models like: 1) ResNet-50, 2) ResNet-101, 3) ResNet-152, 4) Inception-v3, and 5) Inception-v4. These models cover a broad spectrum both in terms of model size (number of parameters and layers) as well as complexity (inception modules and residual blocks).

GPU-CPU Comparison Experiments: The main focus of this study is characterization of CPU-based training. However, we want to provide a high-level comparison of CPUs and GPUs as well. In these experiments, we evaluate different GPU architectures: 1) Kepler K80, 2) Pascal P100, and 3) Volta V100. We provide both single-node and multi-node results for TensorFlow as well as PyTorch. We also compare the best CPU-based training results to these GPU-based results to produce a comprehensive performance comparison.

Horovod Profiling: To gain better insights at the communication level for distributed DNN training, we have implemented profiling variables for the Horovod library. The motivation for these profiling variables is to differentiate the number of communication operations into two categories: 1) Allreduce operation called by DL framework, and 2) Allreduce operation called by Horovod Engine. This type of profiling cannot be achieved by mpiP [21] and other similar profiling tools as they intercept MPI operations and cannot differentiate between different MPI_Allreduce calls effectively. Horovod Engine is basically a communication thread that runs in the background. Through profiling, we correlate end-to-end performance numbers to the impact of low-level runtime parameters like HOROVOD_CYCLE_TIME, which influences the number of Allreduce operations called by Horovod Engine.

V. Performance Characterization: Single Node

We now provide analysis of different experiments we performed (see Section IV for the details of software libraries, hardware platforms, and experimental setup).

A. Single Node Single Process (SP) Experiments

1) Skylake-1: We train ResNet-50 on a single node using a single process with multiple threads. As Skylake-1 has 28 cores, we first study the performance impact for number of threads (1–28) as shown in Figure 1(a). We observed a good scaling trend up to 14 threads per process depending on the BS. However, the performance improvement for 28 threads is the least. To understand the impact of BS better, we show the same results but in a different configuration, i.e., performance for different BS (16–1024) in Figure 1(b). We observed that the performance improvement when using eight threads with increasing BS is not significant. When using 28 threads, however, performance improvements can be observed up to BS 512. Although increasing the BS is beneficial, we can clearly see that the advantage of increasing the BS diminishes beyond a certain point (e.g. 256 for 28 threads). The reason for diminishing returns are twofold: 1) The TensorFlow runtime seems to be under-optimized with respect to BS increase and 2) There is not enough increase in parallelizable work as we increase the BS that could keep all the cores busy. In ideal cases, the BS increase should scale performance in a near-linear fashion. However, this is not the case even with Intel-optimized TensorFlow.

An orthogonal but important problem with BS is its impact on convergence of the learning process or accuracy of the trained model [22]. As a rule of thumb, smaller BS are preferred by DL scientists as they offer better convergence. To address this, we choose a BS of 128 as the benefits of BS increase start to diminish past that point. We will use this BS for multi-node scaling presented in Section V-B.

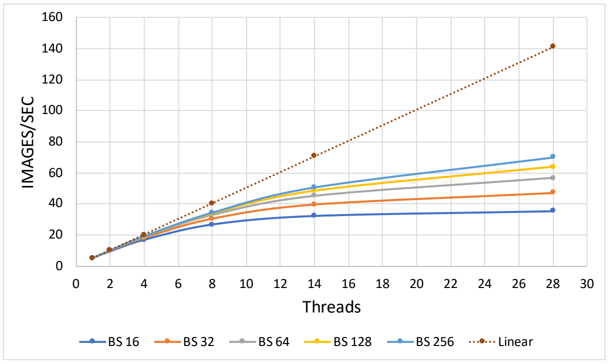
2) Broadwell: Similar experiments were conducted on the Broadwell processor available on the same RI2 cluster. Figure 2 shows the performance of ResNet-50 model for a single process. Just like Skylake-1, we also observed considerable scaling up to 14 threads for the Broadwell processor. Performance improvement after 14 threads is minimal.

3) Skylake-2: Skylake on Pitzer has a lower clock speed compared to Skylake-1 (2.4 GHz vs 2.6 GHz) but more number of cores compared to Skylake-1 (40 vs 28). Figure 3 shows the performance of ResNet-50 training with respect to the number of threads. We observed that the single thread performance of Skylake-2 (Pitzer) is higher than Skylake-1 (RI2).

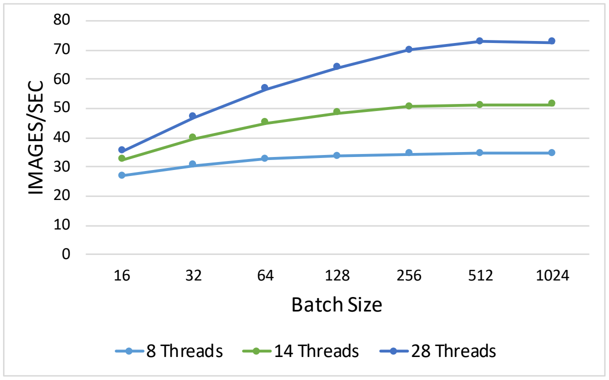
4) Skylake-3: Skylake-3 (Stampede2) has a higher number of cores but lower clock speed compared to Skylake-1 and Skylake-2. There are two hardware threads per core (hyper-threading) so we can run up to 96 threads on Skylake-3. Figure 4 shows the performance of ResNet-50 for varying number of threads. We found that running a single process with 96 threads does not improve the performance. Instead, we observed worse performance compared to the 48 threads case.

B. Single Node Multi-Process (MP) Experiments

Figure 5 shows the impact of different processes per node (ppn) configurations and varying BS for ResNet-152 model. We observed that *ppn* and *BS* have a non-linear relationship. For BS=64, 4ppn seems to offer the best performance whereas



(a) Varying Number of Threads



(b) BS Variation

Fig. 1. ResNet-50: Training Performance on Skylake-1 (TensorFlow)

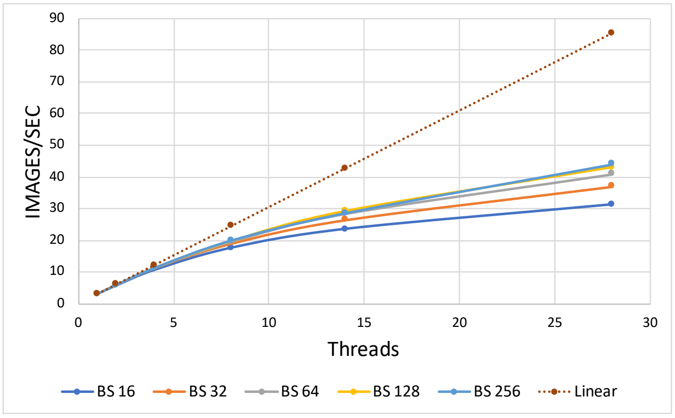


Fig. 2. ResNet-50: Training Performance on Intel Broadwell Processors (TensorFlow)

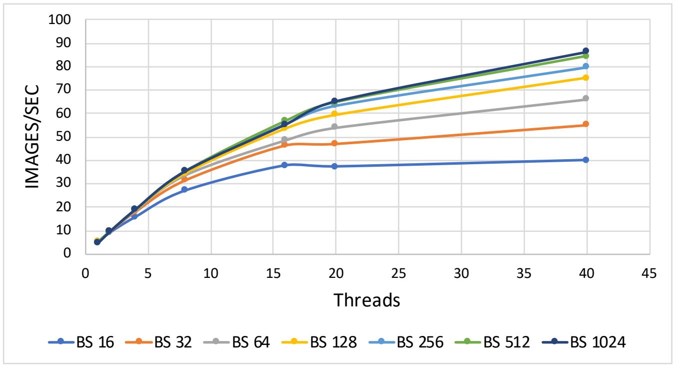


Fig. 3. TensorFlow: Impact of threads on ResNet-50 performance (Skylake-2)

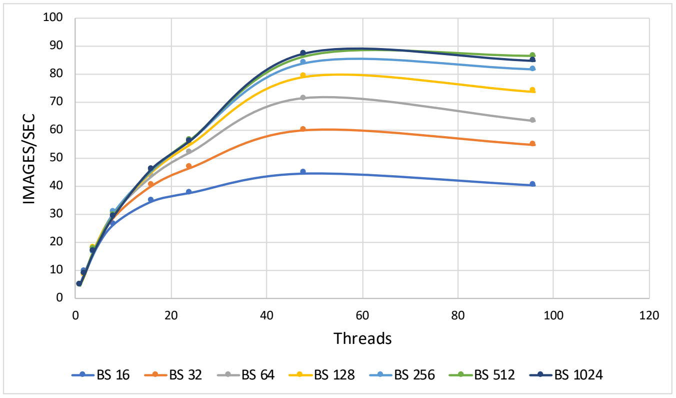


Fig. 4. TensorFlow: Impact of threads on ResNet-50 performance (Skylake-3)

8ppn is better for BS=32. These trends vary between models as well.

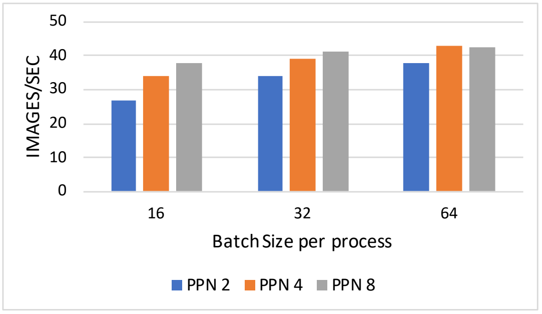
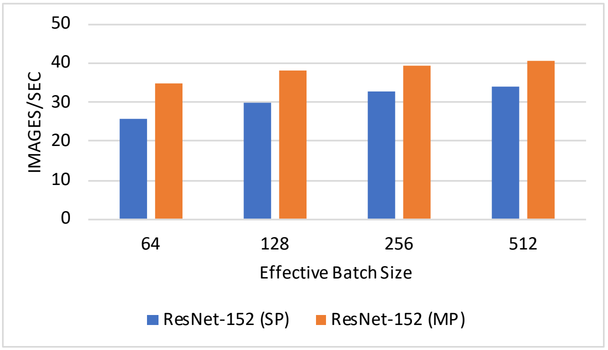
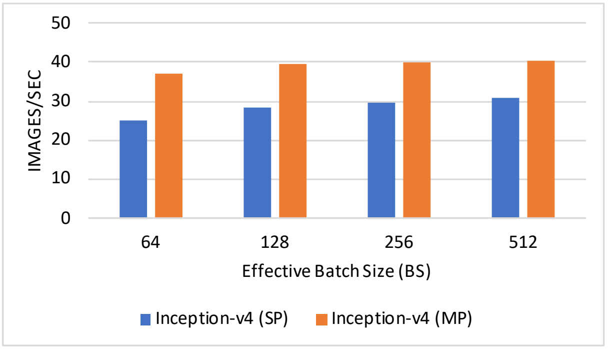


Fig. 5. TensorFlow: BS and Multiple PPN Configurations for ResNet-152 (Skylake-3)

To deal with bad scaling behavior for the single process case, we propose that a multi-process (MP) approach should be used even on a single node. Figures 6(a) and 6(b) show performance improvement for multi-process (MP) over single-process (SP) for ResNet-152 and Inception-v4. We found that 4ppn and 11 *intra-op* threads provide the best performance. Unlike Skylake-1 (RI2), which has one hardware thread per core, Skylake-3 (Stampede2) has two hardware threads per core (hyper-threading enabled). Thus, we ran experiments for both values (1 and 2) for *inter-op* threads. *inter-op=2* offered better performance on this hyper-threading enabled system.



(a) ResNet-152 performance comparison



(b) Inception-v4 performance comparison

Fig. 6. TensorFlow: Performance comparison of Single Process (SP) and Multi-Process (MP) on Skylake-3 (Stampede2)

VI. Performance Characterization: Multi-Node

Based on single node results, we conducted experiments with various combinations of process count and number of intra-op threads per process. In this section, we provide performance evaluation of ResNet-50, ResNet-101, ResNet-152, Inception-v3, and Inception-v4 models across multiple nodes. For every model, we conducted the experiments described in Section V-A to find out the best number of inter-op threads, intra-op threads, and batch size.

A. Skylake-1

We found that for the 28-core Skylake-1, 2ppn yields better performance compared to 1ppn, but slightly worse performance compared to 4ppn. However, the difference between 2ppn and 4ppn is minimal. Therefore, doubling the batch size by using 4ppn makes little sense for this slight improvement. This is to make sure that multi-node scaling can be performed without going to very large batch sizes that hurt accuracy [22]. Figure 7 shows multi-node performance for Skylake-1 for all five models.

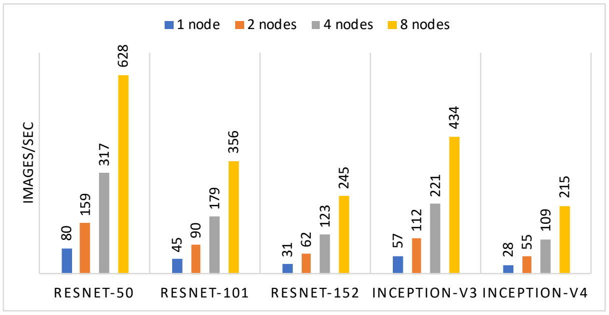


Fig. 7. TensorFlow: Multi-Node performance of ResNet and Inception models (Skylake-1)

B. Broadwell

The best performance for Broadwell processors is achieved when two processes are used with 13 intra-op threads and one inter-op thread. Figure 8 shows the multi-node performance on Broadwell processors for ResNet and Inception models. We found that a BS of 64 gave similar performance to a BS of 128 for the ResNet and Inceptions models except ResNet-50. We present the performance of ResNet-50 with a BS of 128 and other models with a BS of 64.

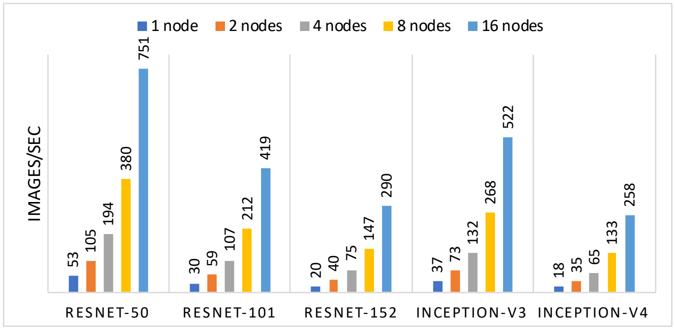


Fig. 8. TensorFlow: Multi-Node performance of ResNet and Inception models (Broadwell)

C. Skylake-2

Figure 9 shows the multi-node performance of ResNet and Inception models on Skylake-2 (Pitzer). We observed that an average speedup of 15.6× for 16 nodes can be achieved using the Horovod design for data-parallel training using TensorFlow and MPI.

D. Skylake-3

Skylake-3 has hyper-threading enabled so we need to tune intra-op and inter-op threads as well. Figure 10 shows the performance benefit for MP on a single node when appropriate intra-op and inter-op threads are used on Skylake-3. MP-Tuned corresponds to tuned intra-op and inter-op threads, MP-Default corresponds to the default settings, and SP corresponds to Single Process. MP-Tuned offers up to 1.5× better performance than SP and 1.1× better performance than MP-Default for Inception-v4 model on 32 nodes.

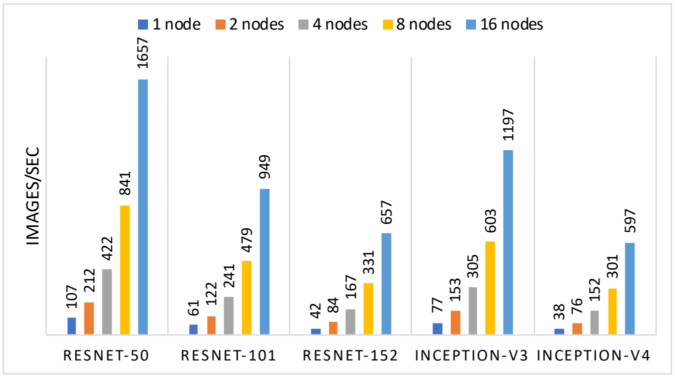


Fig. 9. TensorFlow: Multi-Node performance of ResNet and Inception models (Skylake-2)

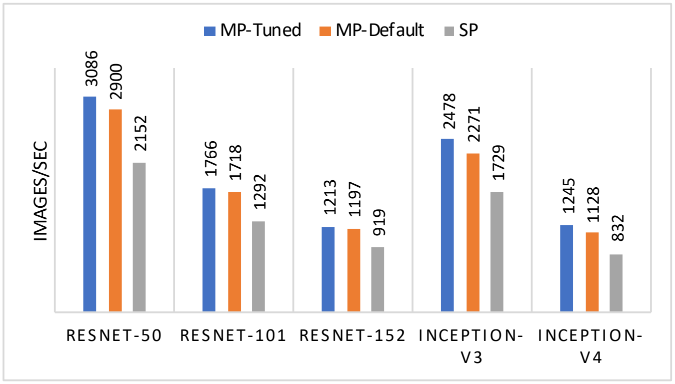


Fig. 10. TensorFlow: Performance comparison: MP-Tuned, MP-Default, and SP on 32 nodes (Skylake-3)

The largest scale we have tested is 128 nodes on Skylake-3 (Stampede2). Figure 17[1] shows multi-node performance (up to 128) for all the five models: 1) ResNet-50, 2) ResNet-101, 3) ResNet-152, 4) Inception-v3, and 5) Inception-v4. Using Horovod and MVAPICH2 2.3, we were able to achieve *125*× speedup for ResNet-152 using 128 nodes.

Figure 11 shows the performance of ResNet(s) and Inception (v3/v4) models for different BS on 128 nodes. It is apparent that a smaller BS offers slower performance compared to a larger BS. The performance impact of BS also depends on the size and type of the model. E.g. ResNet-50 definitely offers an advantage with large BS as it is a relatively smaller model.

We now present some early experience results for the PyTorch framework on Skylake-3. We observed very slow performance (2.1 images/sec for ResNet-50 model) with PyTorch for single process training. We also observed that it can be significantly improved using multiple processes on single node (the proposed MP approach). We found that 48 ppn yields the best performance for all DNN models. We have used BS 16

[1]Please note the out-of-order number for this figure. This was done to bring other figures closer to the text.

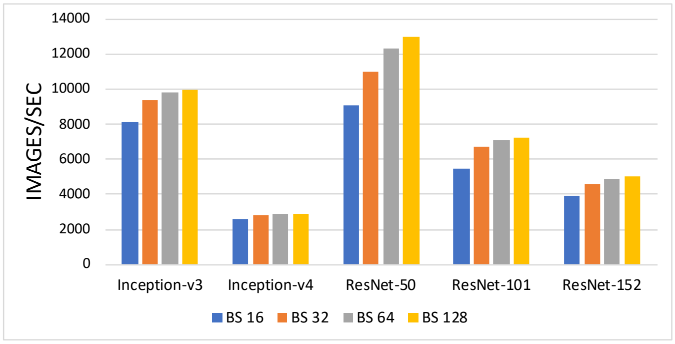


Fig. 11. TensorFlow: Performance of Various Models on 128 Nodes (Skylake-3) with different BS

for ResNet-50 and ResNet101 and BS 8 for ResNet-152 and Inception-v3. Figure 12 shows the multi-node performance of ResNet and Inception models on Skylake-3.

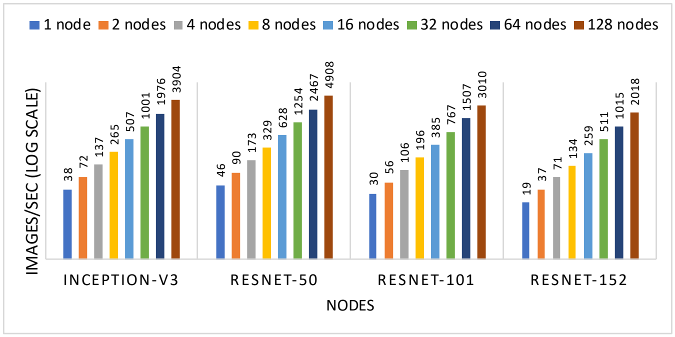


Fig. 12. PyTorch: Multi-Node performance of ResNet and Inception models (Skylake-3)

E. EPYC

We conducted extensive experiments to find appropriate values for ppn, intra-op, and inter-op threads to get good performance for ResNet and Inception models on the AMD system as well. Figure 13 shows multi-node (up to 8 nodes) performance for ResNet and Inception models for 16 ppn, 5 intra-op threads, and 2 inter-op threads. We get 7.8× speedup on 8 nodes for ResNet-152. Unfortunately, Intel optimizations in TensorFlow only benfit Intel systems. On the AMD platform, we observed no performance improvement for Intel-optimized TensorFlow. As a result, the raw performance on Intel systems (Skylake-3) is 4.5× better than AMD EPYC.

In addition to TensorFlow, we evaluated PyTorch on the AMD EPYC system as well. Best numbers were obtained by setting both *ppn* and BS to 32. Figure 14 shows multi-node (up to 8 nodes) performance for ResNet-50, ResNet-101, ResNet-152, and Inception-v3 models. We get 7.98× speedup on 8 nodes for ResNet-50 model. Like Intel-optimized TensorFlow, we did not observe any benefit of Intel-optimized PyTorch on the AMD system. Thus, Skylake-3 offers 1.5×

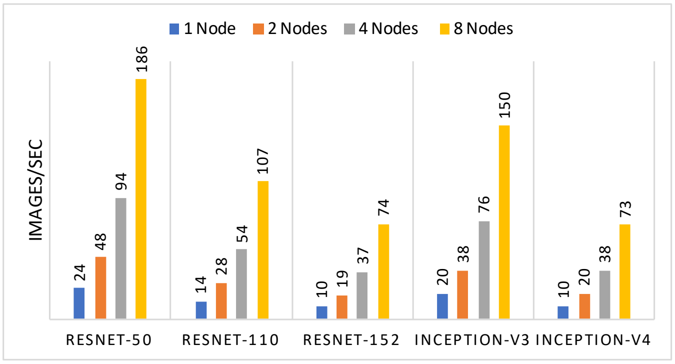


Fig. 13. TensorFlow: Multi-Node performance of ResNet and Inception models (AMD EPYC)

better performance for the ResNet-101 model compared to EPYC. On a side note, PyTorch is 1.2× faster than TensorFlow on 8 EPYC nodes for ResNet-152.

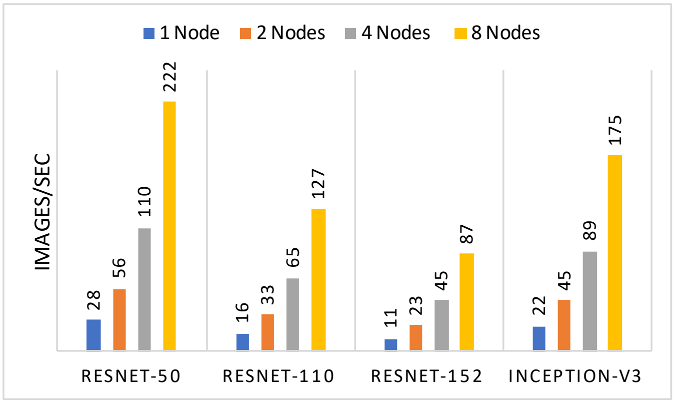


Fig. 14. PyTorch: Multi-Node performance of ResNet and Inception models (AMD EPYC)

VII. GPU-CPU Comparison

In this section, we present the comparison of distributed DNN training across single and multiple nodes on different GPU architectures for ResNet and Inception models. Figure 15 shows the performance comparison of TensorFlow for K80, P100, V100, and Skylake-3 architecture for the best BS. The Skylake-3 architecture is up to 2.35× faster than K80 GPUs for the inception-v4 model whereas V100 is up to 3.32× faster than Skylake-3 for ResNet-101.

Figure 16 presents the comparison of PyTorch and TensorFlow on different GPU architectures for ResNet and Inception-v3. #-TF corresponds to the # number of GPUs using TensorFlow, and #-PT corresponds to the # number of GPUs using PyTorch. In our evaluation, PyTorch always gave better performance compared to TensorFlow, and it is 1.12× faster on 4 GPUs for ResNet-152.

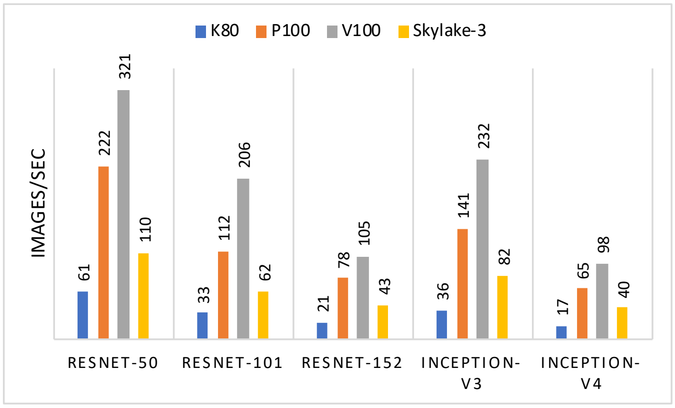


Fig. 15. TensorFlow: Performance Comparison of K80, P100, V100, and Skylake-3 for ResNet and Inception models

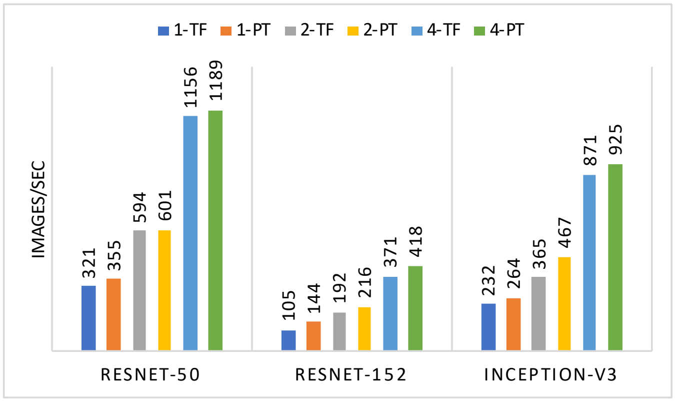


Fig. 16. Performance comparison of PyTorch and TensorFlow DL framework up to 4 nodes for ResNet and Inception models

VIII. Horovod Profiling

1) TensorFlow: In this section, we present a relation between the end-to-end performance numbers and the number of Allreduce operations called by the Horovod Engine for ResNet models. Figure 18 shows the performance and number of Allreduce operations called by Horovod Engine over 40 iterations of DNN training for ResNet models on Skylake-3. The default value of HOROVOD_CYCLE_TIME is 3.5 milliseconds. In Figure 18, ResNet-50 corresponds to the end-to-end performance while HE ResNet-50 corresponds to the number of allreduce operations called by Horovod Engine. The same nomenclature is used for ResNet-101 and ResNet-152. We have observed a small gain (1.04×) for ResNet-101 when HOROVOD_CYCLE_TIME is 90 milliseconds compared to default settings. In the case of TensorFlow, HOROVOD_CYCLE_TIME tuning does not yield significant improvement over default settings.

2) PyTorch: Figure 19 shows the relationship among End-to-end performance, HOROVOD_CYCLE_TIME, and Allreduce operations called by the Horovod Engine for PyTorch. We have observed up to 1.25× better performance to the

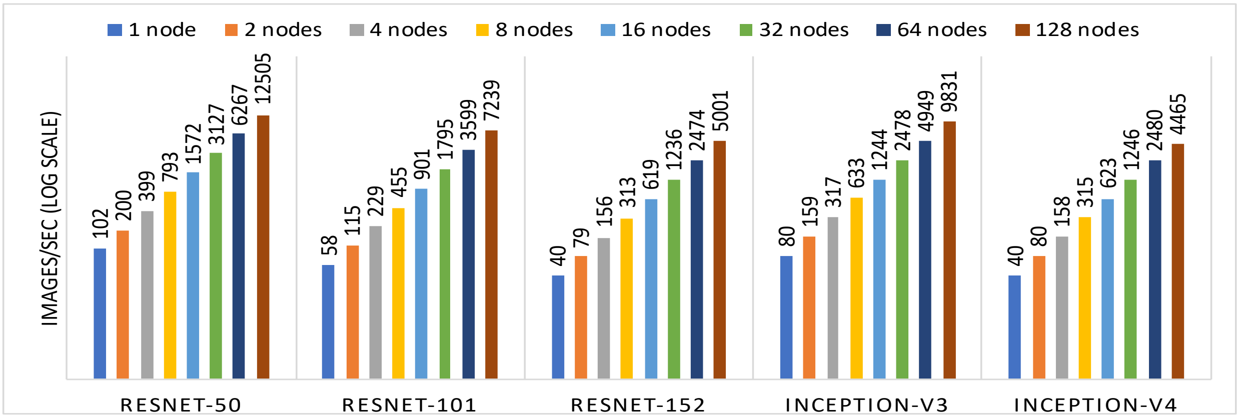


Fig. 17. TensorFlow: Multi-Node performance of ResNet and Inception models (Skylake-3)

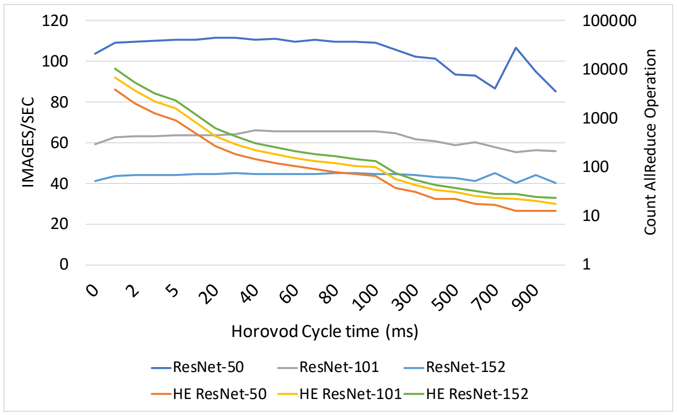


Fig. 18. TensorFlow: Relationships among End-to-end performance, HOROVOD_CYCLE_TIME, and Allreduce operations called by Horovod Engine

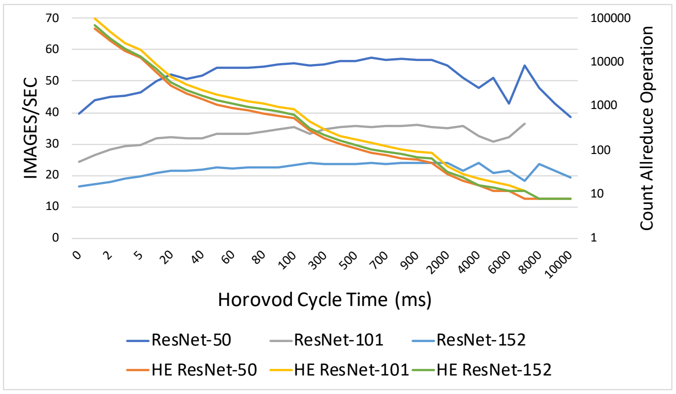


Fig. 19. PyTorch: Relationships among end-to-end performance, HOROVOD_CYCLE_TIME, and Allreduce operations called by Horovod Engine

Default HOROVOD_CYCLE_TIME for ResNet-50, while decreasing number of Allreduce operation used by Horovod 199 times when HOROVOD_CYCLE_TIME is 600 milliseconds. Based on Figure 19, we can infer that PyTorch requires HOROVOD_CYCLE_TIME tuning to achieve better performance unlike TensorFlow, which did not yield significant improvements over default setting.

IX. Key Insights

- We propose that single node training should be run with a multi-process (MP) approach as it performs better than single-process (SP) despite the MKL-DNN and multi-threading optimizations in the Intel-optimized TensorFlow build.
- The best process per node (ppn) configuration for the SP approach depends on the number of cores available on a CPU. Our TensorFlow experiments show that 2, 4, and 4 ppn offer the best performance for Intel processors with 28, 40, and 48 cores, respectively. For the AMD EPYC processors, the best ppn configuration is 32 for TensorFlow.
- The best ppn configuration for PyTorch is very different from TensorFlow, it should be equal to the number of cores in the processor.
- The number of intra-op threads should be one less than the number of cores available per process when using the Horovod library for distributed training on Intel architectures. This is to allow Horovod to use a dedicated thread for progressing gradient-readiness related communication efficiently.
- The proposed MP approach, when compared to SP on a single CPU, provides up to 1.35× and 1.47× better performance for ResNet-152 and Inception-v4, respectively, for TensorFlow.
- TensorFlow gives better performance on CPUs compared to PyTorch. On GPUs, PyTorch gives better performance, but TensorFlow scales well up to 4 nodes. Skylake gives up to 2.35× better performance compared to K80s, but V100 is up to 3.32× faster than Skylake.
- For PyTorch on Skylake, HOROVOD_CYCLE_TIME tuning provides up to 1.25× better performance for ResNet-50.

X. Related Work

There are several works which have characterized the performance of DNN training on NVIDIA GPUs [9]–[11]. Shi et al. [23] have benchmarked several DL frameworks like Caffe, Torch, CNTK, and TensorFlow for multiple CPU and GPU architectures. The main focus of the study was to evaluate different DL frameworks and characterize their performance. However, we focus on the in-depth performance characterization of real TensorFlow models including ResNet(s) and Inception-based models on newer CPU architectures like Intel Xeon Skylake. Intel Xeon Phi for DNN training has been evaluated in [24], but the study was focused on a single node and the Intel Xeon Phi co-processor, which is no longer in production. There is a best practice guideline published by Intel in [12] that focuses on multi-node training with Intel-optimized TensorFlow. Authors of [25] have also provided a large-scale training experiment for Intel CPUs. The study, however, is done with the Caffe framework, and the intent is to train models with large batch sizes on many nodes.

XI. Conclusion

DL applications are rapidly emerging and models are becoming increasingly complex and computationally intensive. Distributed training is an excellent solution for training large DNNs in a reasonable time frame. However, distributed training has been mostly focused on GPU-based systems. Little exists that shows how CPUs can be effectively utilized for DNN training. In this paper, we have evaluated several DNN architectures using TensorFlow and PyTorch on Intel Xeon Skylake, Intel Xeon Broadwell, and AMD EPYC processors at scale. Based on in-depth performance characterization, we offer unique guidelines regarding processes per node (SP vs. MP approach), multi-threading, and batch size (BS) configuration for various DNNs and CPU architectures. We show that the proposed multi-process (MP) approach can provide up to 1.47$\times$ better performance compared to the single-process (SP) approach. Further, we were able to achieve up to 125$\times$ speedup on 128 nodes for ResNet-152 training (up to 5,001 images/second) using TensorFlow with Horovod and the MVAPICH2 MPI library. We also compared the best CPU configuration to NVIDIA K80, P100, and V100 GPUs. The characterization and key insights presented in this paper can guide application programmers and DL scientists to effectively exploit CPU systems by tuning process count, ppn, and BS configurations for different DNNs.

Acknowledgement

We would like to thank Vardaan Gangal for assisting us in conducting GPU-based experiments. This research is supported in part by NSF grants #CNS-1513120, #ACI-1450440, #CCF-1565414, and #ACI-1664137.

References

[1] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in neural information processing systems*, 2001, pp. 402–408.

[2] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 791–798.

[3] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch," 2017.

[7] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 171–180.

[8] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[9] A. A. Awan, J. Bedorf, C.-H. Chu, H. Subramoni, and D. K. Panda, "Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation," *arXiv preprint arXiv:1810.11112*, 2018.

[10] A. A. Awan, H. Subramoni, and D. K. Panda, "An in-depth performance characterization of cpu-and gpu-based dnn training on modern architectures," in *Proceedings of the Machine Learning on HPC Environments*. ACM, 2017, p. 8.

[11] P. Sun, W. Feng, R. Han, S. Yan, and Y. Wen, "Optimizing network performance for distributed dnn training on gpu clusters: Imagenet/alexnet training in 1.5 minutes," *arXiv preprint arXiv:1902.06855*, 2019.

[12] A. Bhandare, D. Karkada, K. Datta, A. Kurpad, V. Sripathi, S. Choi, and V. Saletore, "Best practices for scaling deep learning training and inference with tensorflow on intel xeon processor-based hpc infrastructures," Connectivity Group & AI Products Group, Data Center Group Customer Solutions Technical Enabling, Intel Corporation, Tech. Rep., January 2019.

[13] A. Viebke and S. Pllana, "The potential of the intel (r) xeon phi for supervised deep learning," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 2015, pp. 758–765.

[14] 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

[15] 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

[16] NVIDIA, "NVIDIA Collective Communication Library (NCCL)," https://docs.nvidia.com/deeplearning/sdk/nccl-developer-guide/docs/index.html, 2016, Accessed: August 28, 2019.

[17] O. S. Center, "Ohio supercomputer center," 1987. [Online]. Available: http://osc.edu/ark:/19495/f5s1ph73

[18] D. Stanzione, B. Barth, N. Gaffney, K. Gaither, C. Hempel, T. Minyard, S. Mehringer, E. Wernert, H. Tufo, D. Panda, and P. Teller, "Stampede 2: The Evolution of an XSEDE Supercomputer," in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, ser. PEARC17. New York, NY, USA: ACM, 2017, pp. 15:1–15:8. [Online]. Available: http://doi.acm.org/10.1145/3093338.3093385

[19] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE, https://mvapich.cse.ohio-state.edu/, 2001, [Online; accessed August 28, 2019].

[20] "tf_cnn_benchmarks: High performance benchmarks," 2019, [Online; accessed August 28, 2019]. [Online]. Available: https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks

[21] J. S. Vetter and M. O. McCracken, "Statistical scalability analysis of communication operations in distributed applications," in *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, ser. PPoPP '01. New York,

NY, USA: ACM, 2001, pp. 123–132. [Online]. Available: http://doi.acm.org/10.1145/379539.379590

[22] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *CoRR*, vol. abs/1706.02677, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[23] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, Nov 2016, pp. 99–104.

[24] A. Viebke and S. Pllana, "The potential of the intel (r) xeon phi for supervised deep learning," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, Aug 2015, pp. 758–765.

[25] V. Codreanu, D. Podareanu, and V. Saletore, "Large minibatch training on supercomputers with improved accuracy and reduced time to train," in *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*, Nov 2018, pp. 67–76.