Performance, Energy, and Scalability Analysis and Improvement of Parallel Cancer Deep Learning CANDLE Benchmarks

Xingfu Wu
Argonne National Laboratory
University of Chicago
xingfu.wu@anl.gov

Valerie Taylor
Argonne National Laboratory
University of Chicago
vtaylor@anl.gov

Justin M. Wozniak
Argonne National Laboratory
University of Chicago
woz@anl.gov

Rick Stevens
Argonne National Laboratory
University of Chicago
stevens@anl.gov

Thomas Brettin
Argonne National Laboratory
University of Chicago
brettin@anl.gov

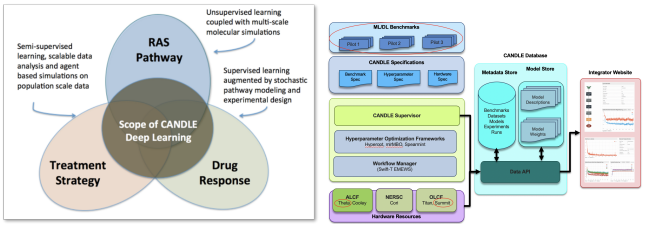
Fangfang Xia
Argonne National Laboratory
University of Chicago
fangfang@anl.gov

ABSTRACT

Training scientific deep learning models requires the significant compute power of high-performance computing systems. In this paper, we analyze the performance characteristics of the benchmarks from the exploratory research project CANDLE (Cancer Distributed Learning Environment) with a focus on the hyperparameters epochs, batch sizes, and learning rates. We discuss the parallel methodology, which use the distributed deep learning framework Horovod to parallelize the CANDLE benchmarks. We then use scaling strategies for both epochs and batch size with linear learning rate scaling to investigate how they impact the execution time and accuracy as well as the power, energy, and scalability of the parallel CANDLE benchmarks under conditions of weak scaling and strong scaling. We use the following two machines for the experimental work: the IBM Power9 heterogeneous system, Summit, at Oak Ridge National Laboratory and the Cray XC40, Theta, at Argonne National Laboratory. This study provides insights into how to set the proper numbers of epochs, batch sizes, and compute resources for these benchmarks to preserve the high accuracy and to reduce the execution time of the benchmarks. We identify the data-loading performance bottleneck and then improve the performance and energy for better scalability. Results with the modified benchmarks on Summit indicate up to 78.25% in performance improvement and up to 78% in energy saving under strong scaling on up to 384 GPUs and up to 79.5% in performance improvement and up to 77.11% in energy saving under weak scaling on up to 3,072 GPUs. On Theta, we achieve up to 45.22% performance improvement and up to 41.78% in energy saving under strong scaling on up to 384 nodes. Moreover, the modification dramatically reduces the broadcast overhead.

1 INTRODUCTION

Training modern deep learning models requires the large amount of computing power provided by high-performance computing (HPC) systems. TensorFlow [2][30] is a widely used open source frameworks for deep learning; it supports a wide variety of deep learning uses, from conducting exploratory research to deploying models in production on cloud servers, mobile apps, and even self-driving vehicles [28]. Horovod [14] [28], developed by Uber, is a distributed training framework for TensorFlow and Keras [16]. In this work, we use Horovod to parallelize Python-based benchmarks [8] from the exploratory research project CANDLE (Cancer Distributed Learning Environment) [5]. We then analyze and improve the Horovod implementation of CANDLE benchmarks in terms of performance, energy, and scalability on the IBM Power9 heterogeneous system Summit with GPUs [29] at Oak Ridge National Laboratory and on the Cray XC40 Theta [11] at Argonne National Laboratory.



(a) Scope **(b) System Overview**

Figure 1: Scope and System Overview of CANDLE [5]

The CANDLE project [5] [33] focuses on building a single scalable deep neural network that can address three cancer challenge problems shown in Figure 1(a): the RAS pathway problem of understanding the molecular basis of key protein interactions in the RAS/RAF pathway presented in 30% of cancers using unsupervised learning; the drug response problem of developing predictive models for drug response to optimize preclinical drug screening and drive precision-medicine-based treatments for cancer patients using supervised learning; and the treatment strategy problem of automating the analysis and extraction of information from millions of cancer patient records to determine optimal cancer treatment strategies using semi-supervised learning. The CANDLE system overview in Figure 1(b) consists of several major components: hardware resources, the CANDLE supervisor and workflow manager, CANDLE benchmarks, database, and integrator website. The CANDLE benchmarks are the main application driver in the CANDLE project.

The CANDLE benchmarks [7] Pilot1, Pilot2, and Pilot3 implement deep learning architectures that are relevant to these three

ACM Conference, 2019,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

cancer problems. The Pilot1 (P1) benchmarks are formed from problems and data at the cellular level. The goal behind the P1 benchmarks is to predict the drug response based on molecular features of tumor cells and drug descriptors. The Pilot2 (P2) benchmarks are formed out of problems and data at the molecular level. The goal behind the P2 benchmarks is molecular dynamic simulations of proteins involved in cancer, specifically the RAS protein. The Pilot3 (P3) benchmarks are formed out of problems and data at the population level. The goal behind the P3 benchmarks is to predict cancer recurrence in patients based on patient-related data.

The CANDLE benchmarks are implemented in Python by using the Keras framework. Each benchmark uses common Python-based CANDLE utilities and implements a common interface used by higher-level Python-based driver systems, such as the CANDLE/Supervisor framework for hyperparameter optimization [33]. These benchmarks, which are intended to run on exascale systems as they emerge, are currently being tested on pre-exascale systems such as Theta and Summit. These pre-exascale systems feature new hardware at ever greater scale, requiring new analysis of performance and power to determine how best to use them. Deep learning is expected to play a greater role in scientific computing on systems such as Summit. Thus, deep learning is critical for studying the performance and power usage of the whole application stack, including the scripting level, numerics, and communication. In this work, we focus on the P1 benchmarks, which include four benchmarks: NT3, P1B1, P1B2, and P1B3.

To accelerate TensorFlow applications by utilizing large-scale supercomputers such as Theta and Summit requires a distributed TensorFlow environment. Currently, TensorFlow has a native method for parallelism across nodes using the gRPC layer in TensorFlow based on sockets [1] [13], but this is difficult to use and optimize [21] [28]. The performance and usability issues with the distributed TensorFlow can be addressed, however, by adopting an MPI communication model. Although TensorFlow has an MPI option, it replaces only point-to-point operations in gRPC with MPI and does not use MPI collective operations. Horovod adapts the MPI communication model by adding an allreduce between the gradient computation and model update, replacing the native optimizer with a new one called the Distributed Optimizer. No modification to TensorFlow itself is required; the Python training scripts are modified instead. The Cray programming environment machine learning plugin (CPE ML Plugin) [21], like Horovod, does not require modification to TensorFlow, but it is designed for Cray systems and is not available to the public on other systems. Therefore, we chose Horovod for this investigation.

In this paper, we have the following contributions.

- We discuss our parallel methodology and use Horovod to parallelize the CANDLE P1 benchmarks (NT3, P1B1, P1B2 and P1B3) on Theta and Summit. This parallelization method can be applied to other CANDLE benchmarks such as the P2 and P3 benchmarks in a similar way.
- We analyze the performance characteristics of the CANDLE benchmarks with the focus on the hyperparameters epochs, batch sizes, and learning rates. We use scaling strategies for both epochs and batch size with linear learning rate scaling to investigate how they impact not only the performance and accuracy but also the power, energy, and scalability under strong scaling on Summit (with GPUs) and Theta (with CPUs).
- We identify the performance bottlenecks and then refine the parallel cancer deep learning benchmarks to improve their performance, energy, and scalability under strong scaling and weak scaling.

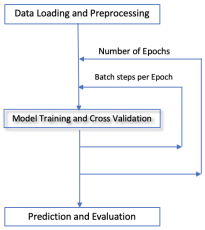


Figure 2: Control Flow of a CANDLE Benchmark

The remainder of this paper is organized as follows. Section 2 briefly describes the CANDLE P1 benchmarks and Horovod and then discusses the parallel methodology and the Horovod implementation. Section 3 depicts the system platforms Summit and Theta. Section 4 analyzes the performance and power characteristics of the Horovod CANDLE benchmarks under strong scaling. Section 5 discusses performance and energy improvement of these benchmarks under strong scaling. Section 6 analyzes the performance and energy improvement of these benchmarks under weak scaling. Section 7 summarizes this work and discusses some future work.

2 CANDLE BENCHMARKS AND THEIR HOROVOD IMPLEMENTATIONS

In this section, we briefly describe the CANDLE benchmarks [7] and the distributed deep learning framework Horovod [28]. We then discuss in detail the methodology for the Horovod implementation of the benchmarks.

2.1 CANDLE Benchmarks

The CANDLE benchmarks are written in Python and Keras, which is a high-level neural network API written in Python and capable of running on top of TensorFlow, CNTK [22], or Theano [32]. Each CANDLE benchmark entails three phases: data loading and preprocessing, basic training and cross-validation, and prediction and evaluation on test data in Figure 2.

Batch steps per epoch is the total number of samples divided by the batch size. Increasing the batch size means decreasing the batch steps per epoch. To achieve high model accuracy, we have to choose the proper (optimal) number of epochs and batch size. If the batch size is too small, the number of batch steps per epoch (the number of iterations) is too large so that it takes a long time to train the model. If the batch size is too large, then few data samples are used for model training, significantly impacting the model accuracy. For a given batch size, one epoch is not enough to achieve high model accuracy. Therefore, the parameters for the model training from

Table 1: Epochs, batch size, data samples, and training and testing file sizes for the P1 benchmarks.

Benchmark	NT3	P1B1	P1B2	P1B3
Training data size	597MB	771MB	162MB	318MB
Testing data size	150MB	258MB	55MB	103MB
Number of epochs	384	384	768	1
Batch size	20	100	60	100
Learning rate	0.001	none	0.001	0.001
Optimizer	sgd	adam	rmsprop	sgd
Total training samples	1,120	2,700	2,700	900,100
Total elements per sample	60,483	60,484	28,204	1,000

this epoch are used for model training in the next epoch, in order to improve the model accuracy.

In this work, we focus on the P1 benchmarks and briefly describe four benchmarks—NT3, P1B1, P1B2 and P1B3—as follows. Table 1 shows the number of epochs, batch size, learning rate, the total number of data samples, and data file sizes for training and testing in each benchmark. Each benchmark uses the importing data function `pandas.read_csv()` [26] to read the data files locally.

2.1.1 NT3 Benchmark. This benchmark [8] is a 1D convolutional network for classifying RNA-seq gene expression profiles into normal or tumor tissue categories. This network follows the classic architecture of convolutional models with multiple 1D convolutional layers interleaved with pooling layers followed by final dense layers. The model is trained on the balanced 700 matched normal-tumor gene expression profile pairs available from the NCI Genomic Data Commons and acts as a quality control check for synthetically generated gene expression profiles. The full dataset of expression features contains 60,483 float columns transformed from RNA-seq FPKM-UQ values [8] that map to a column that contains the integer 0|1. As shown in Table 1, the training data size for this benchmark is 597 MB, and the test data size is 150 MB. The number of epochs is 384. The batch size is 20 (default); and the total training samples are 1,120. Thus, the batch steps per epoch are 56. The optimizer is sgd (stochastic gradient descent).

2.1.2 P1B1 Benchmark. This benchmark [7] is a multilayer perceptron (MLP) network, which is a class of feedforward artificial neural networks, with encoding layers, dropout layers, bottleneck layer, and decoding layers. It has at least three hidden layers: one encoding layer, one bottleneck layer, and one decoding layer. Given a sample of RNA-seq gene expression data, it builds a sparse autoencoder that can compress the expression profile into a low-dimensional vector without much loss of information. In Table 1, the training data size is 771 MB, and the test data size is 258 MB. The number of epochs is 384; the batch size is 100 (default); and the total training samples are 2,700. Thus, the batch steps per epoch are 27. The optimizer is adam (adaptive moment estimation).

2.1.3 P1B2 Benchmark. This benchmark [7] is a multilayer perceptron (MLP) network with regularization and five layers. Given patient somatic SNP data, it builds a deep learning network that can classify the cancer type based on sparse input data and evaluate the information content and predictive value in a molecular assay with auxiliary learning tasks. In Table 1, the training data size is 162 MB, and the test data size is 55 MB. The number of epochs is 768; the batch size is 60 (default); and the total training samples are 2,700. Thus, the batch steps per epoch are 45. The optimizer is rmsprop (root mean square propagation).

2.1.4 P1B3 Benchmark. This benchmark [7] is a multilayer perceptron (MLP) network with convolution-like layers. Given drug screening results on NCI60 cell lines, it builds a deep learning network that can predict the growth percentage of a cell line treated with a new drug from cell line gene expression data, drug concentration, and drug descriptors. This benchmark is a simplified form of the core drug response prediction problem to combine multiple molecular assays and a diverse array of drug feature sets to make a prediction. In Table 1, the training data size is 318 MB, and the test data size is 103 MB. The number of epochs is 1; the batch size is 100 (default); and the total training samples are 900,100. Thus, the batch steps per epoch is 9,001. The optimizer is sgd.

2.2 Horovod

The goal of Horovod [14] [28] is to make distributed deep learning fast and easy to use. The core principles of Horovod are based on MPI concepts such as size, rank, local rank, allreduce, allgather, and broadcast; and it is implemented by using MPI subroutines. A unique feature of Horovod is its ability to interleave communication and computation. Moreover, it is able to batch small allreduce operations by combining all the tensors that are ready to be reduced at a given moment into one reduction operation, an action that results in improved performance. The Horovod source code is based on the Baidu tensorflow-allreduce repository [4]. Horovod provides MPI-based data parallelism for TensorFlow. In its examples [14], it provides the parallelization at the epoch level (keras_mnist.py) and at the batch step level (keras_mnist_advanced.py). Choosing which level depends on the application characteristics.

2.3 Methodology: Using Horovod to Parallelize the CANDLE Benchmarks

In this section, we discuss our methodology using Horovod to parallelize the CANDLE benchmarks on Summit with GPUs. For Theta with CPUs, we simply replace the number of GPUs with the number of nodes for the methodology.

2.3.1 Methodology: Scalable Data Parallelism. Each CANDLE benchmark has two main loops, at the batch step level and at the epoch level. We want to parallelize the two loops to speed up the training process. Figure 3 shows the framework of the data-parallel methodology. Data parallelism is at the epoch level and/or the batch step level. The one model training iteration is inside the two main loops. At the end of the iteration, the allreduce operation is used to average the gradients, and then the averaged gradients are applied to the model update for the next training iteration.

We assume that the number of epoch is E, the batch size is B, and the number of total data samples is S. Thus, the number of batch steps per epoch is S/B, and the total number of iterations is $E \times S/B$, as shown in Figure 3. Decreasing E per GPU or increasing B results in a reduction in the number of iterations, which means fewer allreduce operations and less time. In Figure 4(a), for strong scaling (inverse proportion), we keep the total number of epochs constant, decrease the number of epochs per GPU, and increase the number of GPUs. For weak scaling (direct proportion), we keep the

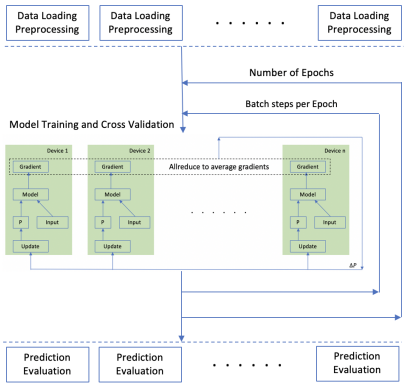
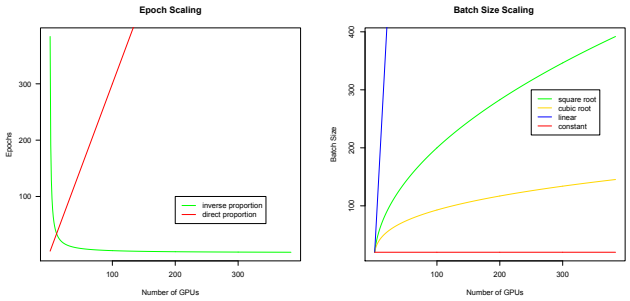


Figure 3: High-Level Control Flow of a Parallel CANDLE Benchmark



(a) Epoch Scaling **(b) Batch Size Scaling**

Figure 4: Scaling Strategies

number of epochs per GPU constant and increase the number of GPUs.

Consider how to scale the batch size to some extent for better performance without reducing the training accuracy (such as linear scaling and square root scaling for batch steps per epoch), as shown in Figure 4(b). If the total number of data samples is large, we increase the batch size based on the number of GPUs as follows: Linear scaling: $batch\_size \times GPUs$; Square root scaling: int $(batch\_size \times (GPUs)^{1/2})$; Cubic root scaling: int $(batch\_size \times (GPUs)^{1/3})$. Which scaling strategy is used depends on the total number of data samples and number of GPUs for training. Overall, choosing the level of data parallelism depends on the application characteristics.

2.3.2 Implementations. As described in [14], to use Horovod, we made the following additions to a benchmark to utilize GPUs:

- For using GPUs on Summit, pin the GPU to be used to the process local rank (one GPU per process). In this case, the first process on the node will be allocated the first GPU, the second process will be allocated the second GPU, and so forth.

```
from keras import backend as K
import tensorflow as tf
import horovod.tensorflow as hvd
hvd.init()
......
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))
```

  For each Summit node with 6 GPUs, the hvd.local_rank() is 0, 1, 2, 3, 4, and 5.
- For using CPUs on Theta, pin the node to be used to the process rank (one process per node). In this case, the first process will be allocated the first node, the second process will be allocated the second node, and so forth. We also set 64 threads per node to accelerate the code.

```
os.environ["KMP_BLOCKTIME"] = "0"
os.environ["KMP_SETTINGS"] = "1"
os.environ["KMP_AFFINITY"]= "granularity=fine,verbose,compact,1,0"
config = tf.ConfigProto(intra_op_parallelism_threads=
int(os.getenv('OMP_NUM_THREADS', 64)),
inter_op_parallelism_threads=1, allow_soft_placement=True)
K.set_session(tf.Session(config=config))
```

- Obtain the size (`hvd.size()`) and rank (`hvd.rank()`), and adjust the number of epochs.
  If the number of epochs n is large, adjust the number of epochs based on the number of GPUs as follows.

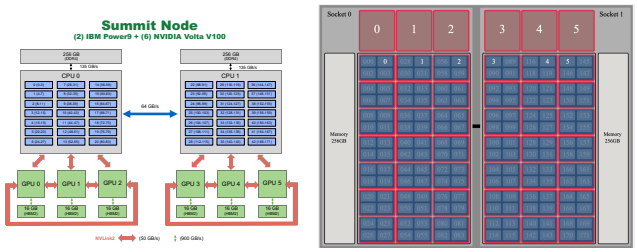
```
nprocs = hvd.size()
myrank = hvd.rank()

def comp_epochs(n, myrank=0, nprocs=1):
        j = int(n // nprocs)
        k = n % nprocs
        if myrank < nprocs-1:
                i = j
        else:
                i = j + k
        return i

epochs = comp_epochs(gParameters['epochs'], myrank, nprocs)
```

  We use `comp_epochs()` to calculate the number of epochs for each GPU. For load balancing, we ensure that the number of epochs is the same for each GPU.
- Scale the learning rate by the number of workers. We scale the learning rate to `learning_rate` × `nprocs`. We keep the batch size constant for the benchmarks NT3, P1B1, and P1B2 because of the small number of samples, and we scale the batch size for the benchmark P1B3 because of the large number of samples in Table 1.
- Wrap the original optimizer in the Horovod distributed optimizer using `hvd.DistributedOptimizer(optimizer)`. The distributed optimizer delegates the gradient computation to the original optimizer, averages gradients using the `Allreduce`, and then applies those averaged gradients.
- Add `hvd.BroadcastGlobalVariablesHook(0)` to the callbacks to broadcast initial variable states from rank 0 to all other processes. This step ensures consistent initialization of all workers when training is started with random weights.

3 SYSTEM PLATFORMS

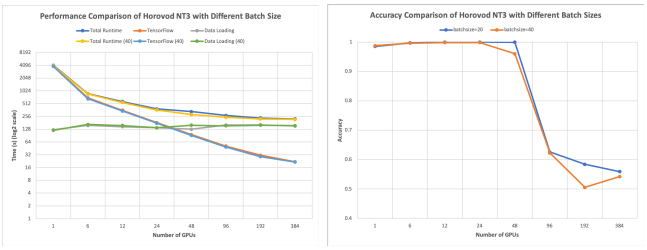
We conduct our experiments on the IBM Power9 heterogeneous system Summit [29] of approximately 200 petaflops at Oak Ridge National Laboratory and the Cray XC40 Theta [11] of approximately 12 petaflops at Argonne National Laboratory. In this section, we briefly describe their specifications.

The basic building block of Summit is the IBM Power System AC922 node. Each of the approximately 4,600 compute nodes on Summit contains two IBM POWER9 processors and six NVIDIA



(a) Node Layout (b) GPUs Partitioning

Figure 5: High-Level Overview of a Summit Node [29]



(a) Performance (b) Accuracy

Figure 6: Horovod NT3 on Summit

Volta V100 accelerators in Figure 5 (a). Each POWER9 processor is connected via dual NVLINK bricks, each capable of a 25 GB/s transfer rate in each direction. Nodes contain 512 GB of DDR4 memory for use by the POWER9 processors and 96 GB of high-bandwidth memory (HBM2) for use by the accelerators. Additionally, each node has 1.6 TB of nonvolatile memory that can be used as a burst buffer. Summit is connected to an IBM Spectrum Scale filesystem providing 250 PB of storage capacity with a peak write speed of 2.5 TB/s. The largest block size of I/O that the Spectrum Scale can issue on Summit is 16 MB [18]. For each Summit node [3], the TDP of each Volta GPU is 300 W, and the TDP of each Power9 is 190 W. The power consumption of each Summit node is 2,200 W. In this work, we use the NVIDIA System Management Interface (nvidia-smi) [24] to measure power consumption for each GPU; however, the power measurement for IBM Power9 is not available to the public. The power sampling rate used is 1 sample per second (default).

Each Cray XC40 node has 64 compute cores (one Intel Phi Knights Landing (KNL) 7230 with the thermal design power (TDP) of 215 W), shared L2 cache of 32 MB (1 MB L2 cache shared by two cores), 16 GB of high-bandwidth in-package memory, 192 GB of DDR4 RAM, and a 128 GB SSD. The Cray XC40 system uses the Cray Aries dragonfly network with user access to a Lustre parallel file system with 10 PB of capacity and 210 GB/s bandwidth. In this work, we simplified the PoLiMEr library [17], which utilizes Cray's CapMC [19] [10] to measure power consumption for the node, CPU, and memory at the node level on Theta. The power sampling rate used is approximately 2 samples per second (default). In a Python code, we import ctypes to export the CDLL for loading the shared PoLiMEr library in order to measure the power. We conduct our experiments with the cache memory mode.

4 PERFORMANCE AND POWER ANALYSIS

In this section, we discuss CPU and GPU partitioning and analyze the performance and power characteristics of the Horovod CANDLE P1 benchmarks with strong scaling on Summit; we find a similar performance trend on Theta. For the constant number of epochs in Table 1, we scale up the number of GPUs to measure the performance and power of the Horovod CANDLE benchmarks. We use Python's cProfile [27] to profile the performance and use nvidia-smi to measure the GPU power consumption.

4.1 Horovod and GPUs Partitioning

Summit supports the NVIDIA Collective Communications Library (NCCL) [23], which implements multi-GPU and multinode collective communication primitives that are performance optimized for NVIDIA GPUs to achieve high bandwidth over PCIe and NVLink high-speed interconnect. Therefore, we configure Horovod using NCCL on Summit. Each Summit node has two Power9 (42 cores) and six Volta GPUs. We use the jsrun visualizer [15] to partition the CPU cores and GPUs in order to achieve better performance. Figure 5(b) shows the GPUs partitioning layout with 6 partitions for our experiments, where each partition consists of one GPU and 7 CPU cores.

4.2 Performance and Power Analysis

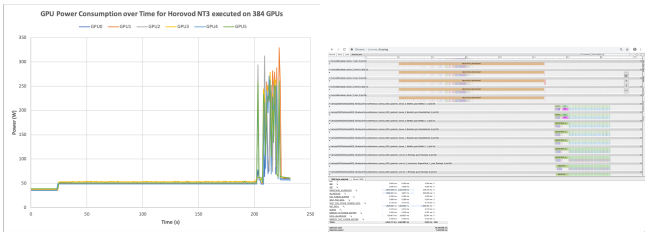
4.2.1 Horovod NT3. We use 384 epochs and the batch sizes of 20 (default) and 40 to conduct the experiments for Horovod NT3 on Summit. This is our strong-scaling study: each GPU executes the number of epochs, which is 384 divided by the number of GPUs. In our previous work [35], we found that the model training and cross-validation phase dominated the time spent in TensorFlow on the Cray XC40 Theta. Therefore, we focus on the time for this phase and data loading, training accuracy, and GPU power on Summit.

Figure 6(a) shows the performance comparison of Horovod NT3 with the batch sizes of 20 (default) and 40 on Summit, where **TensorFlow** stands for the time spent in the model training and cross-validation for a batch size of 20; **Total Runtime (40)** stands for the total runtime for the Horovod NT3 with a batch size of 40; and **Data Loading** stands for the time spent in the data-loading phase using pandas.read_csv() [26] for a batch size of 20. On 384 GPUs (64 nodes), each GPU executes one epoch; and on one GPU, each GPU executes 384 epochs. We observe that with increasing numbers of GPUs, the time in TensorFlow decreases significantly for both cases. Using a larger batch size (40) results in the less runtime because of fewer iterations performed. However, the data-loading time increases slightly for both cases. In particular, on 48 GPUs or more, the data-loading time dominates the total runtime. Therefore, data loading is its performance bottleneck.

Figure 6(b) compares the training accuracy for Horovod NT3 with batch sizes of 20 and 40. With increasing numbers of GPUs, the number of epochs per GPU decreases. The training accuracy reaches 1 when 12, 24, and 48 GPUs are used for the batch size of 20; when 12 and 24 GPUs are used for the batch size of 40, the training accuracy decreases significantly. These results indicate that the

Table 2: Time per epoch (s) and average GPU power (W) for Horovod NT3.

#GPUs	1	6	12	24	48	96	192	384
Time per epoch	10.30	10.93	11.17	11.41	11.91	12.74	15.65	21.82
Power per GPU	224.2	176.8	155.1	130.3	93.88	76.02	64.59	59.14
Time per epoch (40)	10.14	10.45	10.73	11.09	11.50	12.17	14.51	21.75
Power per GPU (40)	228.1	171.2	154	126.7	94.09	77.54	65.69	57.07



(a) Node Power (b) Timeline

Figure 7: Horovod NT3 Using 384 GPUs on Summit

number of epochs for the training and the batch size impact the accuracy. The proper number of epochs per GPU for the training is 8. Using 4 epochs or less per GPU results in a significant decrease in accuracy. Notice that for Horovod NT3, using a batch size of 50 or larger causes running out of memory.

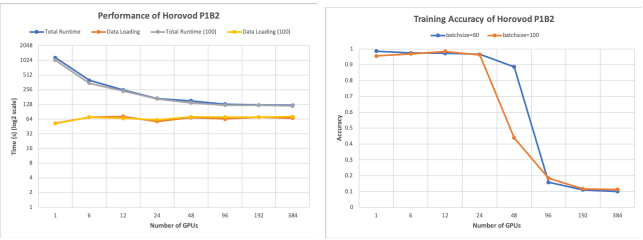
Table 2 compares the time per epoch and average GPU power for the benchmark with batch sizes of 20 and 40. For the given batch size, the time per epoch should be the same. Using a larger batch size (40) results in smaller time per epoch and lower GPU power. With the increased number of GPUs, the time per epoch increases significantly from around 10 s on one GPU to around 22 s on 384 GPUs. This increase is caused by the Horovod allreduce overhead, which is similar to what we found for Horovod NT3 on the Cray XC40 Theta with Intel KNL CPUs [35]. The benchmark is compute-intensive (more than 695 s per epoch) on Theta. On Summit, however, it is not compute-intensive (around 10 s per epoch). The data loading becomes the dominant performance bottleneck for Horovod NT3.

Figure 7(a) shows GPU power per node over time for the Horovod NT3 executed on 384 GPUs (64 nodes). We observe that the benchmark spends most of the time in data loading. For one node of 64 nodes, the power behavior for each GPU is similar. We find that the data loading takes around 153 s. Then what happened? To explain the power behavior in Figure 7(a), we use the Horovod timeline to record the communication activities.

Horovod has the ability to record a timeline of its activity viewed in the Chrome browser through chrome://tracing [9]. Figure 7(b) shows the timeline for the communication of the benchmark on 384 GPUs with allreduce highlighted. This timeline starts the broadcast communication, not the beginning of the benchmark. It consists of two communication types, broadcast (negotiate_broadcast, broadcast, mpi_broadcast) and allreduce (allreduce, NCCL_allreduce, and negotiate_allreduce), where broadcast is implemented based on mpi_broadcast and allreduce is implemented based on the NCCL _allreduce [23].

(a) Performance (b) Loss

Figure 8: Horovod P1B1 on Summit



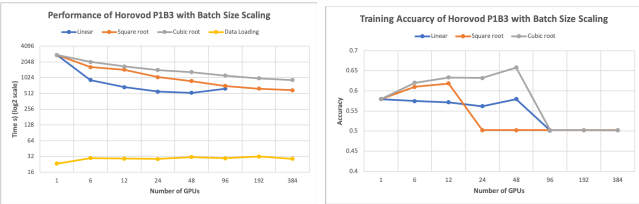
(a) Performance (b) Accuracy

Figure 9: Horovod P1B2 on Summit

Based on the communication activities in Figure 7(b), we can explain the power behavior shown in Figure 7(a). After data loading and preprocessing, the negotiate_broadcast takes place. During the broadcast, the GPU power remains the same; however, it takes around 43 s for the broadcast. Then the gradients are computed so that the GPU power increases. Next, allreduce and NCCL_allreduce are used to average the gradients, and the averaged gradients are applied. After this, the model training batch steps are started. During the training, negotiate_allreduce, allreduce and NCCL_allreduce take place periodically.

4.2.2 Horovod P1B1. As shown in Table 1, the P1B1 has the largest training data size. Its default batch size is 100, so the batch steps per epoch are only 2700/100=27. Figure 8(a) compares the performance of Horovod P1B1 with batch sizes of 100 and 110, where P1B1 requires at least 4 epochs (at most 96 GPUs) for execution. We observe that the data loading dominates the total runtime using 24 GPUs or more. Therefore, data loading is its performance bottleneck, similar to the Horovod NT3. Figure 8(b) compares the training loss of the benchmark with batch sizes of 100 and 110. The loss increases only slightly for both cases.

4.2.3 Horovod P1B2. Figure 9(a) compares the performance of Horovod P1B2 with batch sizes of 60 (default) and 100. Because P1B2 is strong scaling, we observe that the data loading starts to dominate the total runtime with increasing numbers of GPUs. Therefore, data loading is its performance bottleneck, similar to the Horovod NT3. Figure 9(b) compares the training accuracy of the benchmark with the number of GPUs. The accuracy decreases significantly when using 96 GPUs or more. This result indicates that using16 epochs or more per GPU for model training will result in high accuracy.



(a) Performance **(b) Accuracy**

Figure 10: Horovod P1B3 on Summit

4.2.4 Horovod P1B3. In Table 1, P1B3 is different from the other three benchmarks in the number of epochs, total training samples, and total elements per sample. P1B3 has only one epoch as default, and it has 900,100 training samples with only 1,000 elements per sample. Because of the huge number of training samples in P1B3, we focus on how different batch size scaling strategies shown in Figure 4 impact the performance and accuracy of the Horovod P1B3.

Figure 10(a) shows the performance of Horovod P1B3 with three batch size scaling strategies on Summit. The default batch size is 100. For the **linear** scaling, the batch size is $100 \times GPUs$; For the **square root** scaling, the batch size is int $(100 \times (GPUs)^{1/2})$; and for the **cubic root** scaling, the batch size is int $(100 \times (GPUs)^{1/3})$. We observe that the linear scaling scales up the batch size significantly with the number of GPUs. This results in the smallest runtime because of the fewest batch steps per epoch. However, setting the batch size too large (19,200 or 38,400) using 192 or 384 GPUs causes failed execution. The cubic root scaling is the slowest increase in batch size, so that its runtime is the largest. We note that data loading is not a dominant performance factor. Figure 10(b) shows that cubic root scaling results in the best accuracy among these strategies. Using 96 GPUs or larger, however, does not lead to better accuracy. These results indicate that setting the proper batch size can result in high accuracy. For instance, for the given number of GPUs (48), setting the batch size to $int(100 \times (48)^{1/3}) = 363$ leads to the highest accuracy (0.6579).

5 PERFORMANCE AND ENERGY IMPROVEMENT

Based on the performance analysis in the preceding section, performance optimization should focus on the data-loading process in the Horovod CANDLE benchmarks on Summit and Theta. In these benchmarks, pandas.read_csv() is called to read the csv data files. Tables 3 and 4 show the performance (in seconds) for data loading with different file sizes using different methods for the P1 benchmarks on Summit and Theta. We compare data-loading methods, the original one (pandas.read_csv()) and the data loading in chunks with low_memory=False (chunk size is 16 MB). The option low_memory is True in the default case for the original method; it internally processes the file in chunks, resulting in lower memory use while parsing. Setting the option to False improves the data-loading process significantly because of the use of large chunks without being limited to lower memory use. We also tested the Dask DataFrame [12] to see whether it would improve the data-loading performance; the performance is better than the original method but worse than the data loading in chunks with low_memory=False.

Table 3: Performance (in seconds) for data loading with different file sizes using different methods on Summit.

Benchmark	Data Size	pandas.read_csv (original)	Data loading in chunks with low_memory=False
NT3	**Training:** 597MB	81.72	14.30
	Testing: 150MB	22.25	5.25
P1B1	**Training:** 771MB	235.68	30.99
	Testing: 258MB	80.77	14.47
P1B2	**Training:** 162MB	40.98	11.03
	Testing: 55MB	15.95	5.33
P1B3	**Training:** 318MB	5.41	5.34
	Testing: 103MB	3.20	2.52

Table 4: Performance (seconds) for data loading with different file sizes using different methods on Theta

Benchmark	Data Size	pandas.read_csv (original)	Data loading in chunks with low_memory=False
NT3	**Training:** 597MB	52.91	13.84
	Testing: 150MB	13.93	3.62
P1B1	**Training:** 771MB	139.71	27.43
	Testing: 258MB	48.38	11.67
P1B2	**Training:** 162MB	25.07	9.53
	Testing: 55MB	9.56	4.40
P1B3	**Training:** 318MB	4.74	4.53
	Testing: 103MB	2.79	2.49

For instance, the following is the original Python code for data loading with the NT3 training data (nt_train2.csv) of 597 MB.

```
import pandas as pd
df = pd.read_csv('nt_train2.csv',header=None)
```

We replace it with the following the Python code with data loading in chuncks and low_memory=False.

```
csize = 2000000
chunks = []
for chunk in pd.read_csv('nt_train2.csv',header=None,
chunksize=csize,low_memory=False):
    chunks.append(chunk)
df = pd.concat(chunks, axis=0, ignore_index=True)
```

In Tables 3 and 4, for the training data file in NT3 there is approximately five times improvement in the data loading by using the method low_memory=False on Summit and approximately four times improvement on Theta. For the largest training data file (771 MB) in P1B1, there is more than seven times improvement in the data loading on Summit and more than five times improvement on Theta. These are significant improvements. In contrast, for P1B3 there is little data-loading improvement because P1B3 has only 1,000 elements per sample in Table 1. Thus, its csv training file has 1,000 columns per row with a huge number of rows (900,100). This situation is very different from the other three benchmarks with a medium number of rows (1,120 or 2,700) with a large number of columns per row (more than 28,000). It indicates that the types of data samples impact the importing data's I/O performance using pandas.read_csv() significantly. The time spent in the data loading for a single data file on Theta is much less than that on Summit.

In the following sections, we apply the improved data-loading method to the Horovod P1 benchmarks and discuss how it improves the performance and energy.

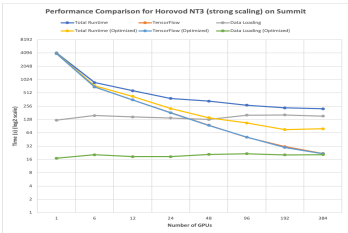


Figure 11: Performance for Horovod NT3 on Summit

Table 5: GPU power (W) and Energy (J) for Horovod NT3

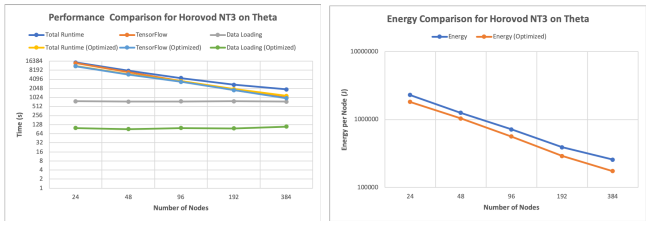
#GPUs	1	6	12	24	48	96	192	384
Power	224.2	176.8	155.1	130.3	93.88	76.02	64.59	59.14
Power (optimized)	228.2	201.7	197.3	188.3	158.4	115.5	97.28	73.84
% Increase	1.82	14.06	27.27	44.51	68.77	51.87	50.61	24.85

(a) Power per GPU

#GPUs	1	6	12	24	48	96	192	384
Energy	919535.66	155572.44	88981.99	50044.33	31270.70	20510.47	15147.06	13227.92
Energy (optimized)	912133.19	150248.40	84508.45	42577.27	22296.14	12377.82	7372.18	5829.51
% Increase	-0.81	-3.42	-5.03	-14.92	-28.70	-39.65	-51.33	-55.93

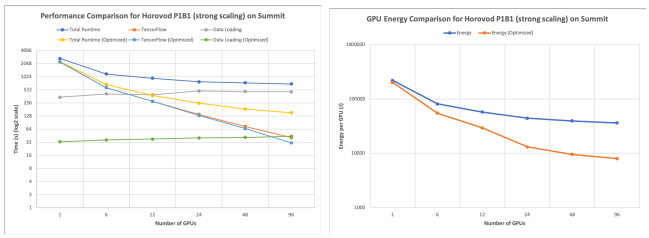
(b) Energy per GPU

Figure 12: Timeline for the Broadcast of the Optimized Horovod NT3 Using 384 GPUs on Summit



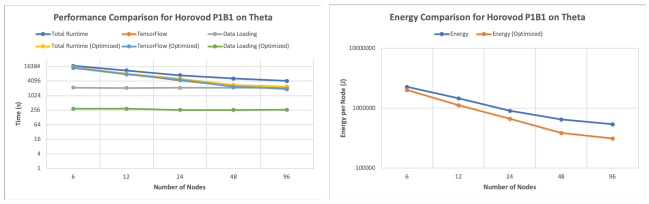
(a) Performance **(b) Energy**

Figure 13: Horovod NT3 on Theta



(a) Performance **(b) Energy**

Figure 14: Horovod P1B1 on Summit



(a) Performance **(b) Energy**

Figure 15: Horovod P1B1 on Theta

5.1 Horovod NT3

We first discuss the performance, power, and energy of the optimized Horovod NT3. We then compare the results with the Horovod NT3 on both Summit and Theta.

Figure 11 compares the performance of the Horovod NT3 with that of the optimized Horovod NT3 on Summit. The optimized method improves the performance of data loading by a factor of at least 5, so that the total time decreases significantly; the performance for TensorFlow is similar. Overall, we achieve up to 67.68% performance improvement on Summit.

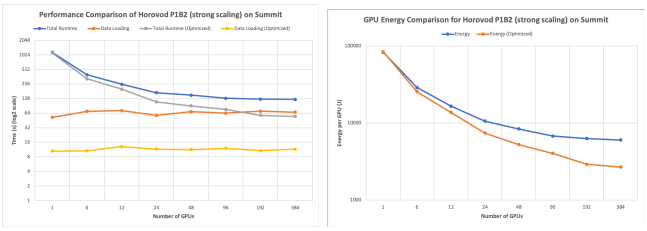
Table 5(a) shows that the average power per GPU for the optimized NT3 increases by up to 68.77% because of the reduced time for low-power data loading. Table 5(b) shows that the energy per GPU for the optimized NT3 decreases by up to 55.93%. Therefore, the performance improvement results in energy saving.

Figure 12 shows the timeline for the broadcast of the optimized Horovod NT3 on 384 GPUs. We find that compared with Figure 7, the optimized method results in a significant decrease in the broadcast overhead, from 43.72 s to 4.65 s, an improvement of 89.36%. This indicates that the slow data-loading delays the data movement. We find a similar improvement in broadcast overhead using other numbers of GPUs.

Figure 13 compares the performance and energy of the Horovod NT3 with those of the optimized Horovod NT3 on Theta. The optimized method improves the performance of data loading by a factor of at least 6, so that the total time decreases significantly; the performance for TensorFlow is similar. Overall, we achieve up to 38.46% performance improvement and up to 32.21% energy saving on Theta. Notice that the time spent in the data loading for Horovod NT3 on Theta is more than four times that on Summit because of the larger I/O contention and smaller I/O bandwidth on Theta. The time per epoch increases from 695 s on 24 nodes to 965 s on 384 nodes on Theta. Compared with Table 2, the time per epoch on Theta is much larger than that on Summit.

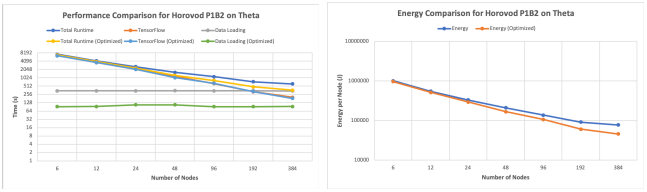
5.2 Horovod P1B1

When we apply the optimized data-loading method to the Horovod P1B1, we observe that the optimized P1B1 results in up to 78.25% performance improvement in Figure 14(a), and up to 78% energy saving in Figure 14(b).



(a) Performance (b) Energy

Figure 16: Horovod P1B2 on Summit



(a) Performance (b) Energy

Figure 17: Horovod P1B2 on Theta

Figure 15 compares the performance and energy of the Horovod P1B1 with those of the optimized Horovod P1B1 on Theta. The optimized method improves the performance of data loading by a factor of at least 7, so that the total time decreases significantly; the performance for TensorFlow is similar. Overall, we achieve up to 45.22% performance improvement and up to 41.78% energy saving on Theta. Notice that the time spent in the data loading for Horovod P1B1 on Theta is more than four times larger than that on Summit.

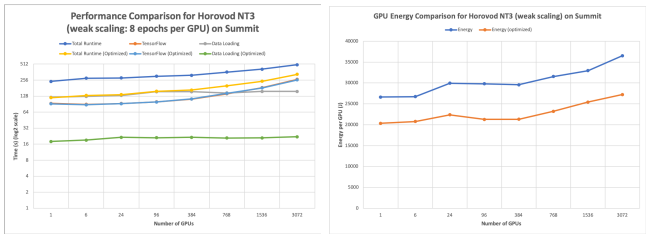
5.3 Horovod P1B2

When we apply the optimized data-loading method to the Horovod P1B2, we observe that the optimized P1B2 results in up to 55.45% performance improvement in Figure 16(a) and up to 55.44% energy saving in Figure 16(b) on Summit.

Figure 17 compares the performance and energy of the Horovod P1B2 with those of the optimized Horovod P1B2 on Theta. The optimized method improves the performance of data loading by a factor of at least 3, so that the total time decreases significantly; the performance for TensorFlow is similar. Overall, we achieve up to 40.72% performance improvement and up to 40.95% energy saving on Theta. Notice that the time spent in the data loading for Horovod P1B2 on Theta is more than five times larger than that on Summit.

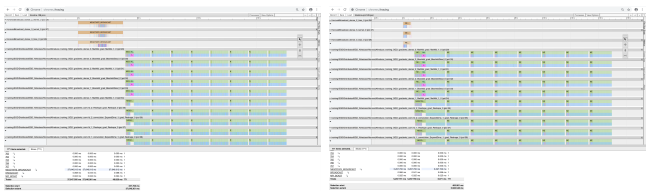
5.4 Horovod P1B3

When we apply the optimized data-loading method to the Horovod P1B3 with cubic root scaling, we observe that the optimized P1B3 results in up to 6.50% performance improvement on Summit. We expect this small performance improvement because of the small data-loading improvement for the data sample type in Table 3. The small performance improvement occurs on Theta as well, and the time spent in the data loading for Horovod P1B3 on Theta is more than five times larger than that on Summit.



(a) Performance (b) Energy

Figure 18: Horovod NT3 (weak scaling) on Summit



(a) Original (b) Optimized

Figure 19: Timeline for the Horovod NT3 with Weak Scaling on 768 GPUs on Summit

6 SCALABILITY ANALYSIS AND IMPROVEMENT

In this section, we analyze the scalability of the Horovod P1 benchmarks and discuss their performance and energy improvements on Summit; the performance improvement on Theta is similar. We use weak scaling (8 epochs per GPU) for these benchmarks to conduct the experiments because the Horovod NT3 with 8 epochs achieves an accuracy of 1 in Figure 6(b). Because the number of epochs is just 1 for P1B3, we focus on the benchmarks NT3, P1B1, and P1B2.

6.1 Horovod NT3

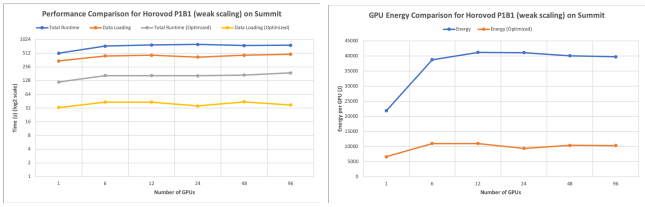
Figure 18(a) shows the performance of the Horovod NT3 with weak scaling on Summit. Our optimized method improves the performance of data loading by a factor of at least 5, so that the total time decreases significantly. Further, we achieve a performance improvement between 34.23% and 52.44% on up to 3,072 GPUs; however, the performance improvement percentage decreases with the number of GPUs because of the large Horovod overhead.

We also find that the optimization method results in a reduction in the broadcast overhead. This is similar to what we found for the strong-scaling case. For instance, Figure 19 shows that the broadcast overhead decreases from 37.65 s to 5.3 s on 768 GPUs (128 nodes), which is an 85.92% improvement. Each GPU performs 8 epochs, so the timeline shows 8 pieces of the communication for 8 epochs. Because the timeline files are very large for 1,536 and 3,072 GPUs, we cannot generate their timeline plots.

In Table 6, the differences in training accuracy and time per epoch for the original and optimized Horovod NT3 are smaller because we focus on improving the data loading performance. The time per epoch for the sequential one is just 10.30 s, but the time per epoch on 3,072 GPUs is more than 3 times larger because of

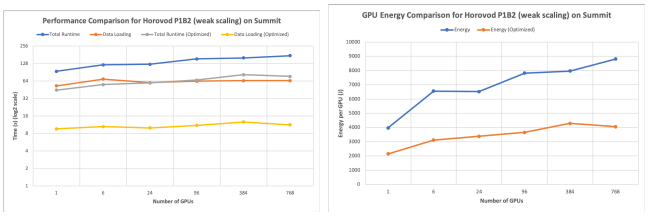
Table 6: Training accuracy, time per epoch (s), and average GPU power (W) for Horovod NT3 on Summit

#GPUs	1	6	24	96	384	768	1536	3072
Training Accuracy	0.83	0.96	0.993	0.64	0.56	0.51	0.52	0.51
Time per epoch	10.30	11.22	11.61	12.47	14.08	17.66	23.10	33.09
Power per GPU	110.1	96.4	106.5	98.75	93.92	87.04	80.31	73.24
Training Accuracy (optimized)	0.81	0.97	0.996	0.61	0.55	0.52	0.52	0.51
Time per epoch (optimized)	10.30	10.99	11.53	12.37	14.19	18.05	22.82	32.37
Power per GPU (optimized)	169.5	157.5	164.5	135.3	127.8	116.5	104.4	83.0



(a) Performance **(b) Energy**

Figure 20: Horovod P1B1 (Weak Scaling) on Summit



(a) Performance **(b) Energy**

Figure 21: Horovod P1B2 (Weak Scaling) on Summit

the large Horovod overhead. For weak scaling, the ideal case is the time per epoch remains constant. This overhead is caused mainly by the allreduce operations using NCCL_Allreduce. The average power per GPU for the optimized Horovod NT3 is larger than for the Horovod NT3 because of the reduced time for low-power data loading. However, because of the large performance improvement, the energy saving is between 22.31% and 28.59% in Figure 18(b).

6.2 Horovod P1B1 and P1B2

For Horovod P1B1 under weak scaling, the optimized method achieves a performance improvement between 75.24% and 79.50% in Figure 20(a), and an energy saving between 69.70% and 77.11% in Figure 20(b).

For Horovod P1B2 under weak scaling, the optimized method achieves a performance improvement between 48.63% and 56.62%, as shown in Figure 21(a), and an energy saving between 45.86% and 53.91% in Figure 21(b).

7 CONCLUSIONS

In this paper, we investigated the performance and power characteristics of the parallelized CANDLE benchmarks with a focus on the hyperparameters epochs, batch sizes, and learning rates under weak scaling and strong scaling on Summit and Theta. We found that the time per epoch on Theta is much larger than that on Summit for these benchmarks. For instance, the NT3 benchmark is compute-intensive (more than 695 s per epoch) on Theta. On Summit, however, it is not compute-intensive (around 10 s per epoch); instead, data loading becomes the dominant performance bottleneck for the CANDLE benchmarks. We identified this bottleneck and improved both the performance and energy for better scalability. We observed that different types of data samples impact the importing data I/O performance using pandas.read_csv() significantly. Overall, the time spent in the data loading for these benchmarks on Theta is more than four times larger than that on Summit because of the larger I/O contention and lower I/O bandwidth on Theta. When we applied the optimized data-loading method to these benchmarks, we achieved up to 78.25% in performance improvement and up to 78% in energy saving under strong scaling on up to 384 GPUs and achieved up to 79.5% in performance improvement and up to 77.11% in energy saving on up to 3,072 GPUs on Summit. We also achieved up to 45.22% performance improvement and up to 41.78% in energy saving under strong scaling on up to 384 nodes on Theta. Moreover, the optimization dramatically reduced the broadcast overhead. We note, however, that the Horovod allreduce overhead for the Horovod NT3 on 3,072 GPUs is almost three times larger than using 6 GPUs on a single node because the time per epoch is very small (only around 10 s).

To further improve the performance of the TensorFlow run, we plan to use NVProf [25] to profile the TensorFlow run and identify the other performance bottlenecks for much larger datasets. We also plan to upgrade NCCL from version 2.3.7 to version 2.4.2 to reduce the communication overhead for the allreduce operations [23]. We will add checkpoint/restart features to the Horovod benchmarks for fault tolerance. Additionally, we plan to use our performance and power modeling work [34] to model and further optimize the CANDLE benchmarks.

REFERENCES

[1] M. Abadi, A. Agarwal, et al., TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, arXiv:1603.04467, 2016.
[2] M. Abadi, P. Barham, et al., TensorFlow: A System for Large-Scale Machine Learning, arXiv:1605.08695, 2016.
[3] S. Atchley, Summit Architecture Overview, Summit Training Workshop, Knoxville, TN, Dec. 3, 2018.
[4] Baidu-allreduce, https://github.com/baidu-research/baidu-allreduce, https://github.com/baidu-research/tensorflow-allreduce.
[5] CANDLE: Cancer Distributed Learning Environment, http://candle.cels.anl.gov.
[6] T. Ben-Nun and T. Hoefler, Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, https://www.arxiv.org/abs/1802.09941, Sep. 2018.
[7] CANDLE Benchmarks: https://github.com/ECP-CANDLE/Benchmarks (Accessed on Dec. 2018).
[8] CANDLE NT3, https://github.com/ECP-CANDLE/Benchmarks/tree/master/Pilot1/NT3.
[9] Chrome trace event profiling tool, https://www.chromium.org/ developers/how-tos/trace-event-profiling-tool.
[10] Cray, Monitoring and Managing Power Consumption on the Cray XC System, Tech Report, S-0043-7204.
[11] Cray XC40 Theta, Argonne National Laboratory, https://www.alcf.anl.gov/theta.
[12] Dask DataFrame, https://docs.dask.org/en/latest/dataframe.html.
[13] Distributed TensorFlow, https://www.tensorflow.org/deploy/distributed, https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/distributed_runtime/README.md.
[14] Horovod: A Distributed Training Framework for TensorFlow, https://github.com/uber/horovod.
[15] jsrun visualizer, https://jsrunvisualizer.olcf.ornl.gov
[16] Keras: The Python Deep Learning Library, https://keras.io/#keras-the-python-deep-learning-library.
[17] I. Marincic, V. Vishwanath, and H. Hoffmann, PoLiMEr: An Energy Monitoring and Power Limiting Interface for HPC Applications, SC2017 Workshop on Energy Efficient Supercomputing, Nov. 13, 2017.

[18] G. S. Markomanolis, Spectrum Scale (GPFS), Summit Training Workshop, Knoxville, TN, Dec. 5, 2018.
[19] S. Martin, D. Rush, et al., Cray XC40 Power Monitoring and Control for Knights Landing. Proceedings of the Cray User Group (CUG), 2016.
[20] S. McCandlish, J. Kaplan, and D. Amodei, An Empirical Model of Large-Batch Training, arXiv:1812.06162v1, 14 Dec 2018.
[21] P. Mendygral, Scaling Deep Learning, ALCF SDL(Simulation, Data and Learning) Workshop, March 2018.
[22] Microsoft Cognitive Toolkit, https://github.com/Microsoft/cntk.
[23] NVIDIA Collective Communications Library (NCCL 2.3.7), https://developer.nvidia.com/nccl, https://devblogs.nvidia.com/scaling-deep-learning-training-nccl/.
[24] NVIDIA System Management Interface (nvidia-smi), https://developer.nvidia.com/nvidia-system-management-interface, 2018.
[25] NVProf, https://docs.nvidia.com/cuda/profiler-users-guide/index.html.
[26] Pandas, https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html.
[27] Python Profilers, https://docs.python.org/2/library/profile.html.
[28] A. Sergeev and M. Del Balso, Horovod: Fast and Easy Distributed Deep Learning in TensorFlow, arXiv:1802.05799v3, Feb. 21, 2018.
[29] Summit, https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
[30] TensorFlow, https://www.tensorflow.org.
[31] TensorFlow Benchmarks, https://www.tensorflow.org/performance/benchmarks.
[32] Theano, https://github.com/Theano/Theano.
[33] J. M. Wozniak, R. Jain, et al., CANDLE/Supervisor: A Workflow Framework for Machine Learning Applied to Cancer Research, SC17 Workshop on Computational Approaches for Cancer, 2017.
[34] X. Wu, V. Taylor, J. Cook, and P. Mucci, Using Performance-Power Modeling to Improve Energy Efficiency of HPC Applications, IEEE Computer, Vol. 49, No. 10, pp. 20-29, Oct. 2016.
[35] X. Wu, V. Taylor, J. M. Wozniak, R. Stevens, T. Brettin and F. Xia, Performance, Power, and Scalability Analysis of the Horovod Implementation of the CANDLE NT3 Benchmark on the Cray XC40 Theta, SC18 Workshop on Python for High-Performance and Scientific Computing, Dallas, USA, Nov. 12, 2018.
[36] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer, ImageNet Training in Minutes, the 47th International Conference on Parallel Processing, August 2018.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).