

# Performance, Power, and Scalability Analysis of the Horovod Implementation of the CANDLE NT3 Benchmark on the Cray XC40 Theta

Xingfu Wu, Valerie Taylor      Justin M. Wozniak      Rick Stevens, Thomas Brettin, Fangfang Xia  
Argonne National Laboratory      Mathematics and Computer Science      Computing, Environment, and Life Sciences  
University of Chicago      Argonne National Laboratory      Argonne National Laboratory  
Email: xingfu.wu, vtaylor@anl.gov      Email: wozniak@mcs.anl.gov      Email: {stevens, brettin, fangfang}@anl.gov

**Abstract**—Training scientific deep learning models requires the large amount of computing power provided by parallel and distributed systems with CPUs and/or GPUs. In this paper, we apply the distributed deep learning framework Horovod to a Python benchmark from the exploratory research project CANDLE (Cancer Distributed Learning Environment) to conduct performance, power, and scalability analysis of its Horovod implementation. We use Horovod to parallelize the benchmark; and we analyze its scalability, performance, and power characteristics with different batch sizes and learning rates under two memory modes: cache and flat on the DOE pre-exascale production system Cray XC40 Theta at Argonne National Laboratory. Our experimental results indicate that power profiling for node, CPU and memory is very useful to show how the Horovod NT3 benchmark exactly behaves on the underlying system. The communication timeline of this benchmark allows for an interpretation of its power behavior. This benchmark under the cache mode results in smaller runtime and lower power consumptions for node and CPU. Increasing the batch size leads to a runtime decrease and slightly impacts the power. Increasing the learning rate results in a decrease in runtime and node power and an increase in accuracy. Several issues in the Horovod NT3 benchmark are discussed for further work.

## 1. Introduction

Today, training modern deep learning models requires the large amount of computing power provided by parallel and distributed systems with CPUs and/or GPUs. TensorFlow is one of the most widely used open source frameworks for deep learning, and it supports a wide variety of deep learning use cases, from conducting exploratory research to deploying models in production on cloud servers, mobile apps, and even self-driving vehicles [19]. Horovod [19] [11], developed by Uber, is a distributed training framework for TensorFlow [21] and Keras [13]. In this work, we use a benchmark, NT3 [6], from the exploratory research project CANDLE (Cancer Distributed Learning Environment) [4] to conduct performance, power, and scalability analysis of its Horovod implementation on the DOE pre-exascale pro-

duction system Cray XC40 Theta [9] at Argonne National Laboratory.

The CANDLE project [4] [12] focuses on building a single scalable deep neural network code that can address three cancer challenge problems: the RAS pathway problem: understanding the molecular basis of key protein interactions in the RAS/RAF pathway presented in 30% of cancers; the drug response problem: developing predictive models for drug response to optimize preclinical drug screening and drive precision-medicine-based treatments for cancer patients; and the treatment strategy problem: automating the analysis and extraction of information from millions of cancer patient records to determine optimal cancer treatment strategies. CANDLE benchmark codes [5] implement deep learning architectures that are relevant to the three cancer problems. The NT3 benchmark [6] is one of the Pilot1 (P1) benchmarks [5] that are formed out of problems and data at the cellular level. The goal behind these P1 benchmarks is to predict the drug response based on molecular features of tumor cells and drug descriptors.

This research is an exemplar of the utility of Python for high-performance computing (HPC) problems. The NT3 benchmark, like other CANDLE benchmarks, is implemented in Python by using the Keras framework. Python allows for the rapid development of the application. It also enables code reuse across the CANDLE benchmarks, since each benchmark uses common Python-based CANDLE utilities, and each benchmark implements a common interface used by higher-level Python-based driver systems, such as the CANDLE/Supervisor framework for hyperparameter optimization [12]. These benchmarks, which are intended to be run on exascale systems as they emerge, are currently being tested on pre-exascale systems such as Theta. Such 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 [20]. Thus, it is critical to study the performance and power usage of the whole application stack, including the scripting level, numerics, and communication.

To speed TensorFlow applications by utilizing large-scale supercomputers such as Theta requires a distributed TensorFlow environment. Currently, TensorFlow has a na-

tive method for parallelism across nodes using the gRPC layer in TensorFlow based on sockets [10]; but this is difficult to use and optimize [19] [16]. The performance and usability issues with distribute TensorFlow can be addressed, however, by adopting an MPI communication model. Although TensorFlow has an MPI option, it only replaces 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 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) [16], like Horovod, does not require modification to TensorFlow, but it is designed for Cray systems, and is not available to the public. Therefore, we choose Horovod in this work.

In this paper, we use Horovod to parallelize the Python NT3 benchmark; and we analyze its scalability, performance, and power characteristics with different batch sizes and learning rates under two memory modes, cache and flat, on the DOE pre-exascale production system Cray XC40 Theta. Our experimental results indicate that power profiling for node, CPU and memory is very useful to show how the Horovod NT3 benchmark exactly behaves on the underlying system. The communication timeline of this benchmark allows for an interpretation of its power behavior. This benchmark under the cache mode results in smaller runtime and lower power consumptions for node and CPU. Increasing the batch size leads to a runtime decrease and slightly impacts the power. Increasing the learning rate results in a decrease in runtime and node power and an increase in accuracy. Several issues in the Horovod NT3 benchmark are discussed for further work.

The remainder of this paper is organized as follows. Section 2 describes the CANDLE NT3 benchmark and Horovod in brief, then discusses the Horovod implementation. Section 3 depicts the system platform Cray XC40 Theta. Section 4 analyzes the scalability of the Horovod implementation of the NT3 benchmark with increasing numbers of nodes. Section 5 uses the experimental results to analyze performance and power characteristics of the NT3 benchmarks. Section 6 summarizes this work and discusses future work.

## 2. CANDLE NT3 Benchmark and Its Horovod Implementation

In this section, we describe the CANDLE NT3 benchmark and Horovod briefly, then discuss the Horovod implementation of the benchmark in detail.

### 2.1. CANDLE NT3 Benchmark

The CANDLE NT3 benchmark [5] [6] is written in Python and Keras, which is a high-level neural networks API written in Python and capable of running on top of

TensorFlow, CNTK [17], or Theano [22]. The NT3 benchmark entails four phases: data loading, preprocessing, basic training and cross-validation, and prediction and evaluation on test data. The benchmark uses the following main global parameters:

```
data_url = 'ftp://ftp.mcs.anl.gov/pub/candle/public/
benchmarks/Pilot1/normal-tumor/'
train_data = 'nt_train2.csv'
test_data = 'nt_test2.csv'
model_name = 'nt3'
conv=[128, 20, 1, 128, 10, 1]
dense=[200,20]
activation='relu'
out_act='softmax'
loss='categorical_crossentropy'
optimizer='sgd'
metrics='accuracy'
epochs=400
batch_size=20
learning_rate=0.001
drop=0.1
classes=2
pool=[1, 10]
save='.'
```

Where the size of the training data file `nt_train2.csv` is 597 MB and the size of the test data file `nt_test2.csv` is 149 MB. The benchmark uses ftp to access the data files; the optimizer is SGD (Stochastic gradient descent); the number of epochs is 400; the batch size is 20; and the learning rate is 0.001.

### 2.2. Horovod

Horovod [19] [11] is a distributed training framework for TensorFlow and Keras and is a stand-alone Python package developed by Uber. The goal of Horovod 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 attempting to combine 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 was based on the Baidu tensorflow-allreduce repository [3].

### 2.3. Using Horovod to Parallelize the NT3 Benchmark

Horovod core principles are based on the MPI concepts of size, rank, local rank, allreduce, allgather, and broadcast. As described in [11], to use Horovod, we make the following additions to the NT3 benchmark to utilize GPUs/CPUs:

- Add `import horovod.tensorflow as hvd` to import the Horovod package.
- Add `hvd.init()` to initialize Horovod.
- For the GPU node only, pin the GPU to be used to 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.

- Obtain the size (`hvd.size()`) and rank (`hvd.rank()`), and adjust the number of epochs based on the number of GPUs/CPUs used 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 the above number of epochs for each GPU/CPU. For the load balancing, the user should ensure that the number of epochs is the same for each CPU/GPU.

- Scale the learning rate by the number of workers. We scale the learning rate to `gParameters['learning_rate'] * hvd.size()`. We keep the batch size the same for our tests.
- Wrap the original optimizer in the Horovod distributed optimizer such as `optimizer = hvd.DistributedOptimizer(optimizer)`. The distributed optimizer delegates gradient computation to the original optimizer, averages gradients using `MPL_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 ensures consistent initialization of all workers when training is started with random weights.

After these steps, we have the Horovod version of the NT3 benchmark.

### 3. System Platform: Cray XC40 Theta

We conduct our experiments on the Cray XC40 Theta [9], which is a pre-exascale production system at Argonne National Laboratory. 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. Cray XC40 [15] [8] provides power management to operate more efficiently by monitoring, profiling, and limiting power usage. The aim is to increase system stability by reducing heat dissipation, reduce system cooling requirements, and reduce utility costs by minimizing power



Figure 1. Cache mode on Cray XC40 Theta

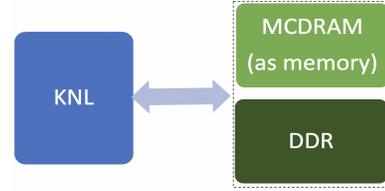


Figure 2. Flat mode on Cray XC40 Theta

usage when rates are the highest and calculating the actual power cost for individual users and/or jobs. In this work, we use a simplified PoLiMER library [14], which utilizes Cray’s CapMC [15] to measure power consumption for the node, CPU, and memory at the node level on Theta. The power sampling rate used is approximately two 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 for the code.

Each XC40 node has one Intel KNL, which brings in new memory technology, an on-package memory called Multi-Channel DRAM (MCDRAM) in addition to the traditional DDR4. MCDRAM has a high-bandwidth (4 times more than DDR4 RAM) and low-capacity (16 GB) memory. MCDRAM can be configured as a shared L3 cache (cache mode) shown in Figure 1 or as a distinct NUMA node memory (flat mode) shown in Figure 2, or somewhere in between. With the different memory modes by which the system can be booted, understanding the best mode for an application becomes challenging for the user.

## 4. Scalability Analysis

In this section, we investigate the performance and power characteristics of the original NT3 benchmark under different memory modes, then analyze the scalability of the Horovod NT3 benchmark for our weak scaling study.

### 4.1. Original NT3 Benchmark under Different Memory Modes

The number of epochs is 400 for the NT3 benchmark. For simplicity, in this section we use just one epoch to conduct the experiments under a learning rate 0.001 and a batch size of 20.

Figure 3 shows power over time on Theta, with cache mode on one node and epochs = 1. The runtime is 1567s, and the average power is 159.21W for node, 105.26W for CPU, and 12.08W for memory. We can see that the NT3 takes around 800s to do the data loading and preprocessing because of the ftp remote data access.

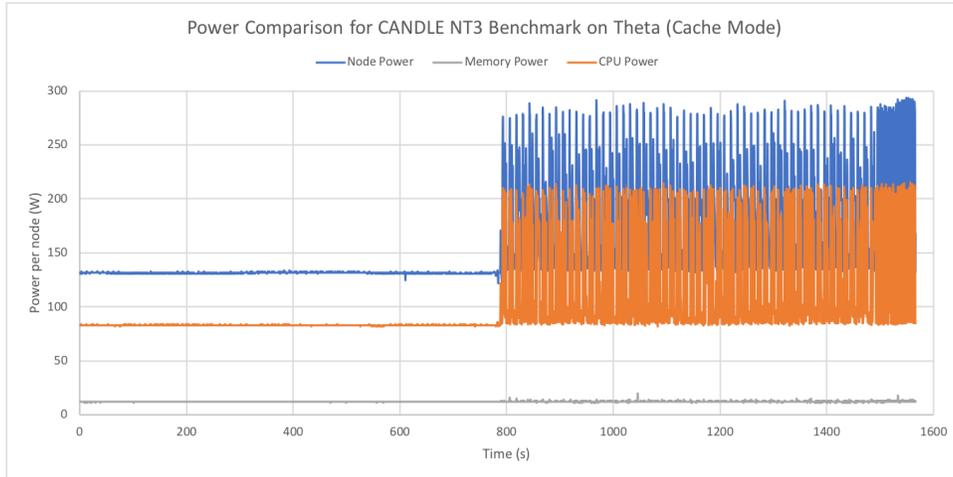


Figure 3. Power over time on Theta (with cache mode on one node and epochs = 1)

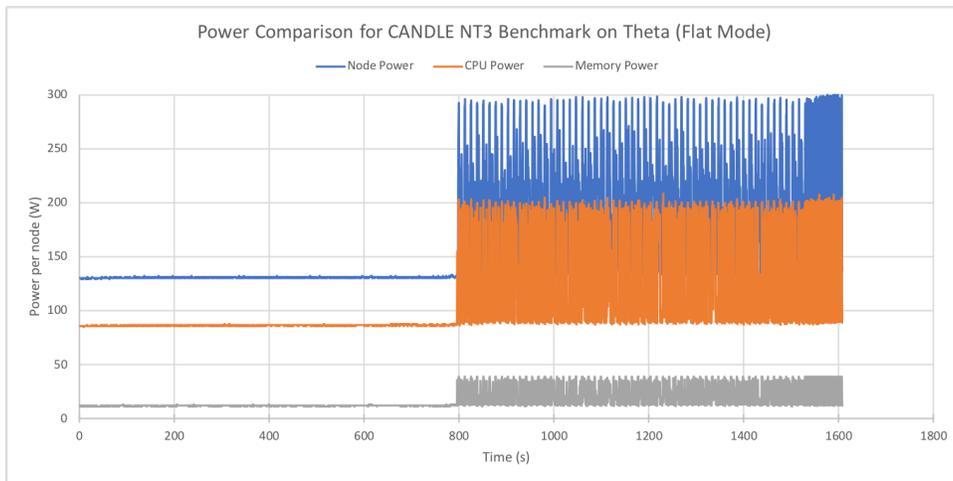


Figure 4. Power over time on Theta (with flat mode on one node and epochs = 1)

Figure 4 shows a power comparison on Theta (with flat mode on one node and epochs = 1). The runtime is 1608s, and the average power is 164.37W for node, 110.37W for CPU, and 17.0W for memory. Comparing the cache mode with the flat mode, we can see that the NT3 benchmark under the cache mode results in better performance and lower power consumptions in node and CPU. The memory power consumption when using the cache mode is lower than that when using the flat mode because the MCDRAM configured as the L3 cache is enough to hold both data files in the cache.

In the experiments, we use a batch size of 20 as default. If we change the batch size, how does this change affect the performance and power of the NT3 benchmark? Table 1 shows runtime, power, and energy of the NT3 benchmark with different numbers of batch sizes under the cache mode. Increasing the batch size results in a decrease in runtime and impacts the power slightly. The benchmark with a batch size of 200 achieves the lowest energy. Because of training on

TABLE 1. RUNTIME (S), POWER (W), AND ENERGY (J) OF THE NT3 BENCHMARK WITH DIFFERENT BATCH SIZES UNDER CACHE MODE.

Batch Size	Runtime	Node Power	CPU Power	Memory Power	Energy
20	1567	159.21	105.26	12.08	249,560.42
50	1497	159.35	99.42	12.12	238,546.95
100	1476	158.88	101.04	11.60	234,506.88
150	1469	160.54	100.22	12.12	235,833.26
200	1462	160.38	101.86	12.14	234,475.56

1,120 samples and validating on 280 samples in the NT3 benchmark, it fails if the batch size is 300 or larger. Thus, the batch size can be adjusted properly only to some extent.

## 4.2. Horovod NT3 Benchmark

In this section, we still use one epoch and a batch size of 20 to conduct the experiments under the cache mode on

Theta. However, we scale up the number of nodes with one epoch per node for our weak scaling study. We measure the performance of the Horovod version of the NT3 and use Python’s cProfile [18] to profile the performance and analyze its scalability on Theta.

Table 2 shows the time for different parts of the NT3 benchmark with increasing learning rate ( $0.001 * \text{hvd.size}()$ ). Where "TensorFlow" is the time for the method `_pywrap_tensorflow_internal.TF_Run`; "Method read" is the time for method read of `pandas._libs.parsers.TextReader` objects; "Keras callback" is the time for the Keras callbacks.py; "Horovod callbacks" is the time for Horovod callbacks.py; "Distributed Optimizer" is the time for the Horovod Distributed Optimizer; "Broadcast" is the time for Horovod broadcast itself; "Allreduce" is the time for Horovod allreduce itself; and "Model.fit" is the time spent in the model training and validation.

The Horovod distributed optimizer has small overhead, around 1.4s even with increasing numbers of nodes, because this optimizer delegates gradient computation to the original optimizer, averages gradients using allreduce, and then applies those averaged gradients. Horovod callbacks introduce some overhead, but it is relatively small. It is interesting to see that the Horovod broadcast overhead is around 0.22s and the Horovod allreduce overhead is around 0.24s. The dominated parts are data loading (Method read) and TensorFlow. Model.fit is the main part of TensorFlow because the NT3 benchmark uses deep learning neural networks to train the model, and the model.fit takes most of the execution time. Because the Python cProfile only provides the time for TensorFlow as a whole, we have to add the inline timing for the function model.fit to measure its runtime.

We further test the communication overhead introduced by Horovod with the same learning rate shown in Table 3 on Theta. Overall, the total runtime results are slightly larger than that shown in Table 2. The Horovod overheads for the callbacks, optimizer, broadcast and allreduce are very similar. Compared with the total runtime, the Horovod overhead is relatively small. Notice that, from Tables 2 and 3, the times for TensorFlow and model.fit increase with the numbers of nodes. It means that Horovod causes some overhead within TensorFlow, but this overhead is not reflected from the Horovod functions provided by the cProfile. In the following section, we will explain this overhead using the Horovod timeline in detail.

## 5. Performance and Power Analysis of the Horovod NT3 Benchmark

In this section, we use the problem size from the original NT3 benchmark for our strong scaling study, where we fix the number of epochs at 400, the learning rate at 0.001, and the batch size at 20. We conduct our experiments with different numbers of nodes under different memory modes to analyze the performance and power behavior of the Horovod NT3 benchmark. We focus on the following metrics: the time spent in the model.fit, loss, and accuracy.

### 5.1. Cache Mode

Table 4 shows the performance for the original dataset with epochs = 400 and the increased learning rate on Theta. "loss" stands for training loss; "acc" stands for training accuracy; "val\_loss" stands for validation (testing) loss; and "val\_acc" stands for validation (testing) accuracy. Each node executes the number of epochs, which is 400 divided by the number of nodes. On 400 nodes, each node executes one epoch, and the total runtime is 1,042s. On 100 nodes, each node executes 4 epochs, and it takes 2,640s. On 25 nodes, each node executes 16 epochs, and it takes 10,662s.

Figure 5 shows power over time for the Horovod NT3 benchmark on 400 nodes. We found that loading the clib for the power measurement using ctypes takes around 11s based on the profile data from the Python cProfile. Therefore, the data loading takes around 781s. Compared with Figure 3, what happened after the data loading phase? To explain the power behavior in Figure 5, we use the Horovod timeline feature [11] to record the communication activities.

Horovod can record the timeline of its activity viewed in the Chrome browser through `chrome://tracing` [7]. Figure 6 shows the timeline for the communication of the benchmark on 400 nodes with the highlight of allreduce from Figure 5. This timeline starts the broadcast communication, not the beginning of the benchmark. It consists of six communication types: `negotiate_broadcast`, `broadcast`, `mpi_broadcast`, `allreduce`, `mpi_allreduce`, and `negotiate_allreduce`, where `broadcast` is implemented based on `mpi_broadcast`; `allreduce` is based on the baidu ring-allreduce algorithm [3] and `mpi_allreduce`. `MEMCPY_IN_FUSION_BUFFER` and `MEMCPY_OUT_FUSION_BUFFER` are to copy data into and out of the fusion buffer. There are two major phases for each tensor broadcast/reduction in the Horovod NT3 benchmark:

- Negotiation phase (`negotiate_broadcast`, `negotiate_allreduce`) when all workers send a signal to rank 0 that they are ready to broadcast/reduce the given tensor. Each worker is represented by a tick under the `negotiate_broadcast/negotiate_allreduce` bar. Immediately after negotiation, rank 0 sends a signal to the other workers to start broadcasting/reducing the tensor.
- Processing phase, when the operation actually happens. These communications shown in Figure 6 indicate the time taken to do the actual operation on the CPU and highlight that the operation was performed by using pure MPI such as `mpi_broadcast` and `mpi_allreduce`.

Based on the communication activities in Figure 6, we can explain the power behavior in Figure 5. After data loading and preprocessing, the `negotiate_broadcast` takes place. During the broadcast, the node power and CPU power decrease because of the dynamic power management on Cray XC40. Then the gradients are computed, so the node and CPU power increase. Both `allreduce` and `mpi_allreduce` are used to average the gradients, and the averaged gradients

TABLE 2. PERFORMANCE (SECONDS) WITH THE INCREASED LEARNING RATE (WEAK SCALING) ON THETA.

#Nodes	2	4	8	16	32	64	128	256	512
<b>Total Runtime</b>	1561	1592	1604	1621	1683	1735	1769	1823	1916
<b>TensorFlow</b>	773	803	816	841	900	954	969	1020	1085
<b>Method read</b>	737	740	739	731	734	730	743	741	748
<b>Keras callbacks</b>	5.67	13.89	16.01	27.85	27.49	28.13	20.08	30.64	25.67
<b>Horovod callbacks</b>	5.66	13.88	16.00	27.84	27.48	28.13	20.08	30.62	25.60
<b>Distributed Optimizer</b>	1.40	1.42	1.47	1.39	1.37	1.47	1.44	1.36	1.43
<b>Broadcast</b>	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22
<b>Allreduce</b>	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.25
<b>Model.fit</b>	735	756	767	780	840	893	916	976	1028

TABLE 3. PERFORMANCE (SECONDS) WITH THE SAME LEARNING RATE (WEAK SCALING) ON THETA.

#Nodes	2	4	8	16	32	64	128	256	512
<b>Total Runtime</b>	1560	1600	1610	1619	1691	1732	1785	1839	1922
<b>TensorFlow</b>	774	813	817	837	905	947	986	1041	1082
<b>Method read</b>	735	734	743	733	735	734	744	735	737
<b>Keras callbacks</b>	10.27	19.92	15.17	24.01	22.49	26.17	24.92	34.85	30.14
<b>Horovod callbacks</b>	10.26	19.91	15.16	24.01	22.47	26.16	24.91	34.83	30.08
<b>Distributed Optimizer</b>	1.37	1.36	1.47	1.37	1.41	1.39	1.38	1.44	1.38
<b>Broadcast</b>	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22
<b>Allreduce</b>	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
<b>Model.fit</b>	731	761	769	780	850	888	929	974	1020

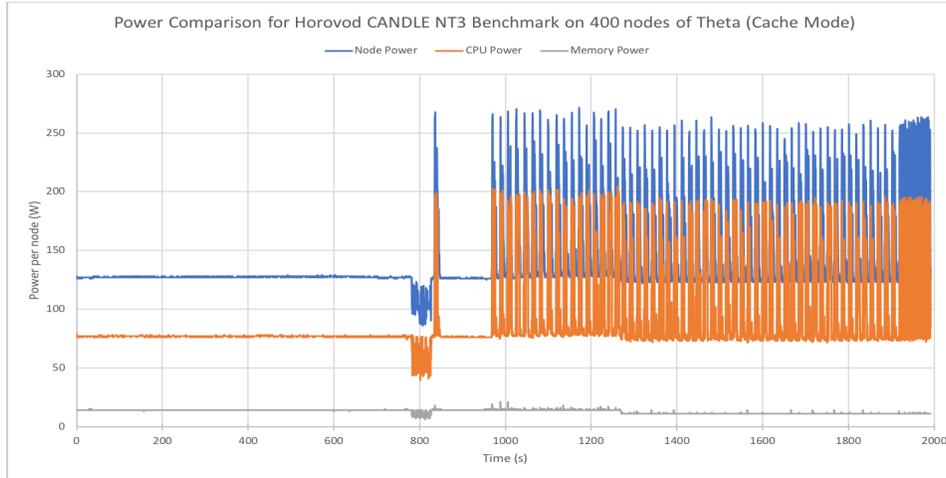


Figure 5. Power over time of the Horovod NT3 with the increased learning rate on 400 nodes.

TABLE 4. TIME (SECONDS), LOSS AND ACCURACY FOR THE ORIGINAL DATASET WITH THE INCREASED LEARNING RATE (STRONG SCALING) ON THETA.

#Nodes	400	200	100	50	25
<b>Time (Model.fit)</b>	1,042	1,927	2,640	6,976	10,662
<b>loss</b>	0.6912	0.6843	0.7164	0.0097	0.0770
<b>acc</b>	0.5420	0.5804	0.5527	0.9991	0.9821
<b>val_loss</b>	0.6941	0.7124	0.6973	0.1206	0.1129
<b>val_acc</b>	0.5143	0.5143	0.4857	0.9786	0.9821

are applied. This process takes around 131s as the first allreduce and mpi\_allreduce shown in Figure 6. The process takes place between 800s and 1000s shown in Figure 5. The model training then is started. During the training, negotiate\_allreduce, allreduce, and mpi\_allreduce take place between two consecutive training steps periodically, as shown in Figure 6. Compared with Figure 3, this Horovod overhead enlarges the gaps between two consecutive training steps. This explains the overhead caused by Horovod within the TensorFlow run.

Table 5 shows the performance for the original dataset with epochs = 400 with the same learning rate on Theta.

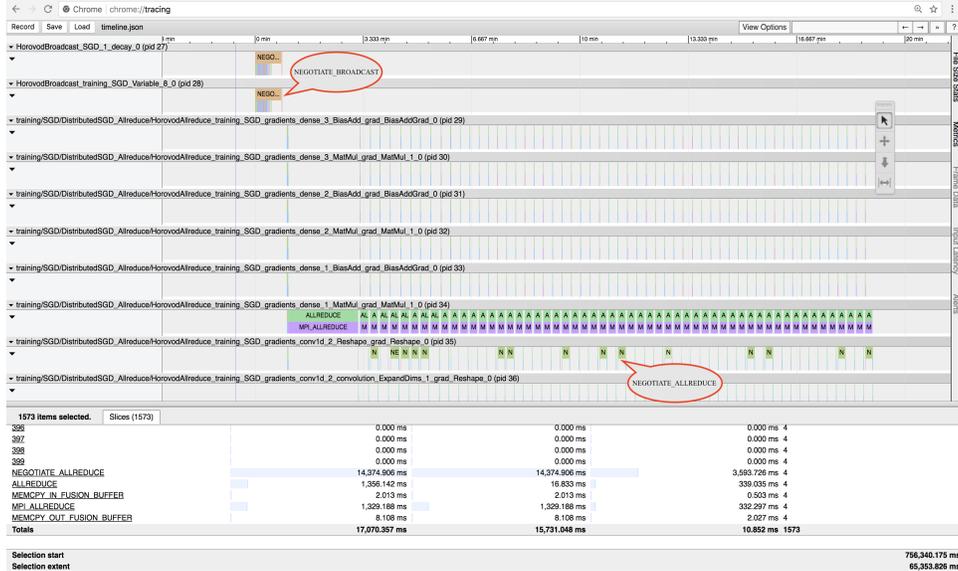


Figure 6. Timeline for the communication of the Horovod NT3 on 400 nodes (cache mode).

TABLE 5. TIME (SECONDS), LOSS AND ACCURACY FOR THE ORIGINAL DATASET WITH THE SAME LEARNING RATE (STRONG SCALING) ON THETA.

#Nodes	400	200	100	50	25
Time (Model.fit)	1,053	2,229	3,709	7,015	13,308
loss	0.6874	0.6806	0.6345	0.5948	0.3335
acc	0.6750	0.8054	0.8491	0.8696	0.9304
val_loss	0.6815	0.6789	0.6324	0.5960	0.3276
val acc	0.6964	0.8036	0.8571	0.8786	0.9536

Compared with Table 4, we observe that the execution time increases slightly. For 100 nodes or more, the accuracy using the same learning rate is much higher than that using the increased learning rate. We note, however, that for 50 and 25 nodes, the accuracy using the same learning rate is lower than that using the increased learning rate.

## 5.2. Flat Mode

Table 6 shows the performance for the original dataset with epochs = 400 with the increased learning rate using the flat mode on Theta. Compared with Table 4 using the cache mode, the NT3 benchmark benefits more from using the cache mode than the flat mode.

Figure 7 shows power behavior similar to that shown in Figure 6. Compared with Figure 4, the Horovod overhead enlarges the gaps between two consecutive training steps because of the allreduce operations as discussed in the previous section. For the benchmark, using the cache mode results in smaller runtime and lower power consumptions for node and CPU.

TABLE 6. TIME (SECONDS), LOSS AND ACCURACY FOR THE ORIGINAL DATASET WITH THE INCREASED LEARNING RATE (STRONG SCALING) ON THETA.

#Nodes	400	200	100	50	25
Time (Model.fit)	1,188	2,089	4,062	7,860	>3hours
loss	0.7071	0.6783	0.6755	0.0111	
acc	0.5464	0.5991	0.5920	0.9991	
val_loss	0.6910	0.6841	0.6718	0.1241	
val acc	0.6714	0.8821	0.8393	0.9714	

## 6. Conclusions

In this paper, we used the Python NT3 benchmark from the exploratory research project CANDLE to conduct performance, power, and scalability analysis of its Horovod implementation with weak scaling and strong scaling cases on Cray XC40 Theta. We used Horovod to parallelize the NT3 benchmark, and we analyzed its scalability, performance, and power characteristics with different batch sizes and learning rates under two memory modes: cache and flat on Theta. Our experimental results indicate that power profiling for node, CPU and memory was very useful to show that the Horovod NT3 benchmark exactly behaved on the underlying system. The communication timeline of this benchmark allowed for an interpretation of its power behavior. This benchmark under the cache mode resulted in smaller runtime and lower power consumptions for node and CPU. Increasing the batch size led to a runtime decrease and slightly impacted the power. Increasing the learning rate resulted in a decrease in runtime and node power and an increase in accuracy.

We plan to address several issues raised by our use of the NT3 benchmark. (1) The data-loading time becomes the

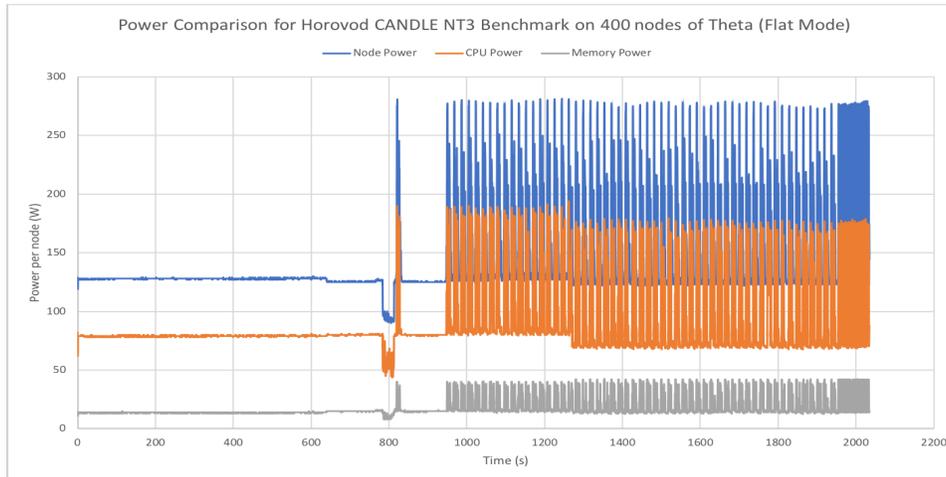


Figure 7. Power over time of the Horovod NT3 with the increased learning rate on 400 nodes.

bottleneck after speeding the model-training process. We have to consider how to speed the input dataset reading operations. (2) The performance profiling using the Python cProfile includes only the total time for the whole TensorFlow run, as shown in Tables 2 and 3. It does not give any detail about how TensorFlow behaves. To further improve the performance of the TensorFlow run, we may need a fine-grained performance profiling tool to profile the TensorFlow run. (3) We developed the Horovod version of the NT3 benchmark to support both CPUs and GPUs. We plan to test the benchmark on some heterogeneous systems with CPUs and GPUs, such as Summit [20]. (4) We plan to add checkpoint/restart features to the Horovod benchmark for fault tolerance purposes. (5) We plan to use our performance and power modeling work [23] to model and optimize the CANDLE benchmarks because the Python codes, like other scripting languages, do not have compiler optimization support, and they often depend on the library, resource, and environment settings for better performance. We can utilize our previous work to identify the better resource and environment settings for performance improvement.

## Acknowledgments

This work was supported in part by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under contract DE-AC02-06CH11357, and in part by NSF grants CCF-1801856. We acknowledge the Argonne Leadership Computing Facility for use of the Cray XC40 Theta under the DOE INCITE project PEACES and ALCF project EE-ECP.

## 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] Baidu-allreduce, <https://github.com/baidu-research/baidu-allreduce>, <https://github.com/baidu-research/tensorflow-allreduce>.  
 [4] CANDLE: Cancer Distributed Learning Environment, <http://candle.cels.anl.gov>.  
 [5] CANDLE Benchmarks: <https://github.com/ECP-CANDLE/Benchmarks>.  
 [6] CANDLE NT3 Benchmark, <https://github.com/ECP-CANDLE/Benchmarks/blob/frameworks/Pilot1/NT3>.  
 [7] Chrome trace event profiling tool, <https://www.chromium.org/developers/how-tos/trace-event-profiling-tool>.  
 [8] Cray, Monitoring and Managing Power Consumption on the Cray XC System, Tech Report, S-0043-7204.  
 [9] Cray XC40 Theta, Argonne National Laboratory, <https://www.alcf.anl.gov/theta>.  
 [10] Distributed TensorFlow, <https://www.tensorflow.org/deploy/distributed>, [https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/distributed\\_runtime/README.md](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/distributed_runtime/README.md).  
 [11] Horovod: A Distributed Training Framework for TensorFlow, <https://github.com/uber/horovod>.  
 [12] J. M. Wozniak, R. Jain, P. Balaprakash, J. Ozik, N. Collier, J. Bauer, F. Xia, T. Brettin, R. Stevens, J. Mohd-Yusof, C. G. Cardona, B. Van Essen, and M. Baughman, CANDLE/Supervisor: A Workflow Framework for Machine Learning Applied to Cancer Research. SC17 Workshop on Computational Approaches for Cancer Workshop, November 2017.  
 [13] Keras: The Python Deep Learning Library, <https://keras.io/#keras-the-python-deep-learning-library>.  
 [14] 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.  
 [15] S. Martin, D. Rush, M. Kappel, M. Sandstedt, and J. Williams. 2016. Cray XC40 Power Monitoring and Control for Knights Landing. Proceedings of the Cray User Group (CUG), 2016.  
 [16] P. Mendygral, Scaling Deep Learning, ALCF SDL(Simulation, Data and Learning) Workshop, March 2018.  
 [17] Microsoft Cognitive Toolkit, <https://github.com/Microsoft/cntk>.  
 [18] Python Profilers, <https://docs.python.org/2/library/profile.html>.  
 [19] A. Sergeev and M. Del Balso, Horovod: Fast and Easy Distributed Deep Learning in TensorFlow, arXiv:1802.05799v3, Feb. 21, 2018.  
 [20] Summit, <https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/>

- [21] TensorFlow, <https://www.tensorflow.org>.
- [22] Theano, <https://github.com/Theano/Theano>.
- [23] 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.

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>).