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

Xingfu Wu, Valerie Taylor  
Argonne National Laboratory  
University of Chicago  
Email: xingfu.wu, vtaylor@anl.gov

Justin M. Wozniak  
Mathematics and Computer Science  
Argonne National Laboratory  
Email: wozniak@mcs.anl.gov

Rick Stevens, Thomas Brettin, Fangfang Xia  
Computing, Environment, and Life Sciences  
Argonne National Laboratory  
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 production 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:

```plaintext
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'
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 num-
  ber of GPUs/CPUs used as follows.

```python
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
  $gParameters[\text{\textquoteleft learning rate\textquoteright}] \times hvd.size()$.
  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
  MPI_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 manage-
ment 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
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 tra-
tional 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 1. Cache mode on Cray XC40 Theta

Figure 2. Flat mode on Cray XC40 Theta
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 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

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Runtime</th>
<th>Node Power</th>
<th>CPU Power</th>
<th>Memory Power</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1567</td>
<td>159.21</td>
<td>105.26</td>
<td>12.08</td>
<td>249,560.42</td>
</tr>
<tr>
<td>50</td>
<td>1497</td>
<td>159.35</td>
<td>99.42</td>
<td>12.12</td>
<td>238,546.95</td>
</tr>
<tr>
<td>100</td>
<td>1476</td>
<td>158.88</td>
<td>101.04</td>
<td>11.60</td>
<td>234,506.88</td>
</tr>
<tr>
<td>150</td>
<td>1469</td>
<td>160.54</td>
<td>100.22</td>
<td>12.12</td>
<td>235,833.26</td>
</tr>
<tr>
<td>200</td>
<td>1462</td>
<td>160.38</td>
<td>101.86</td>
<td>12.14</td>
<td>234,475.56</td>
</tr>
</tbody>
</table>
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 * 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_libsparsers.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. MEMCOPY_IN_FUSION_BUFFER and MEMCOPY_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.

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

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

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

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.

<table>
<thead>
<tr>
<th>#Nodes</th>
<th>400</th>
<th>200</th>
<th>100</th>
<th>50</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (Model.fit)</td>
<td>1.042</td>
<td>1.927</td>
<td>2.640</td>
<td>6.976</td>
<td>10.662</td>
</tr>
<tr>
<td>loss</td>
<td>0.6912</td>
<td>0.6843</td>
<td>0.7164</td>
<td>0.0097</td>
<td>0.0770</td>
</tr>
<tr>
<td>acc</td>
<td>0.5420</td>
<td>0.5804</td>
<td>0.5527</td>
<td>0.9991</td>
<td>0.9821</td>
</tr>
<tr>
<td>val_loss</td>
<td>0.6941</td>
<td>0.7124</td>
<td>0.6973</td>
<td>0.1206</td>
<td>0.1129</td>
</tr>
<tr>
<td>val_acc</td>
<td>0.5143</td>
<td>0.5143</td>
<td>0.4857</td>
<td>0.9786</td>
<td>0.9821</td>
</tr>
</tbody>
</table>

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.
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.

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
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


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).