Data Parallel Deep Learning

Huihuo Zheng, Kaushik Velusamy
Argonne Leadership Computing Facility
August 12, 2022

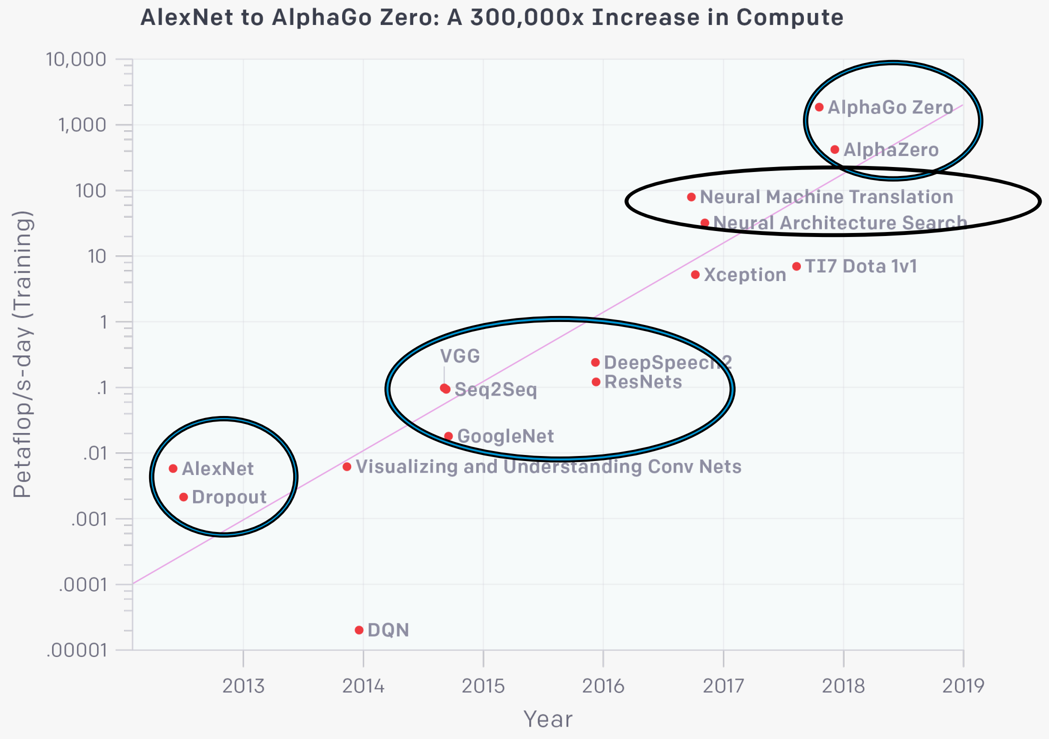
huihuo.zheng@anl.gov,
kaushik.v@anl.gov

Outline

- Why do we need for distributed / parallel deep learning on HPC
- Distribution schemes: model parallelism vs data parallelism
- Steps to change your serial code to data parallel code
- Challenges and tips on data parallel training
- I/O and data management
- Science use cases
- Hands on exercises

Need for distributed (parallel) training on HPC

"Since 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.5 month doubling time (by comparison, Moore's Law had an 18 month doubling period)." https://openai.com/blog/ai-and-compute/



Eras:

- Before 2012 …
- 2012 – 2014: single to couple GPUs
- 2014 – 2016: 10 – 100 GPUs
- 2016 – 2017: large batch size training, architecture search, special hardware (etc, TPU)

Finishing a 90-epoch ImageNet-1k training with ResNet-50 on a NVIDIA M40 GPU takes 14 days. (10^{18} SP Flops) $\longrightarrow$ ~1s on OLCF Summit (~200 petaFlops) if it "scales ideally"

GPT-3

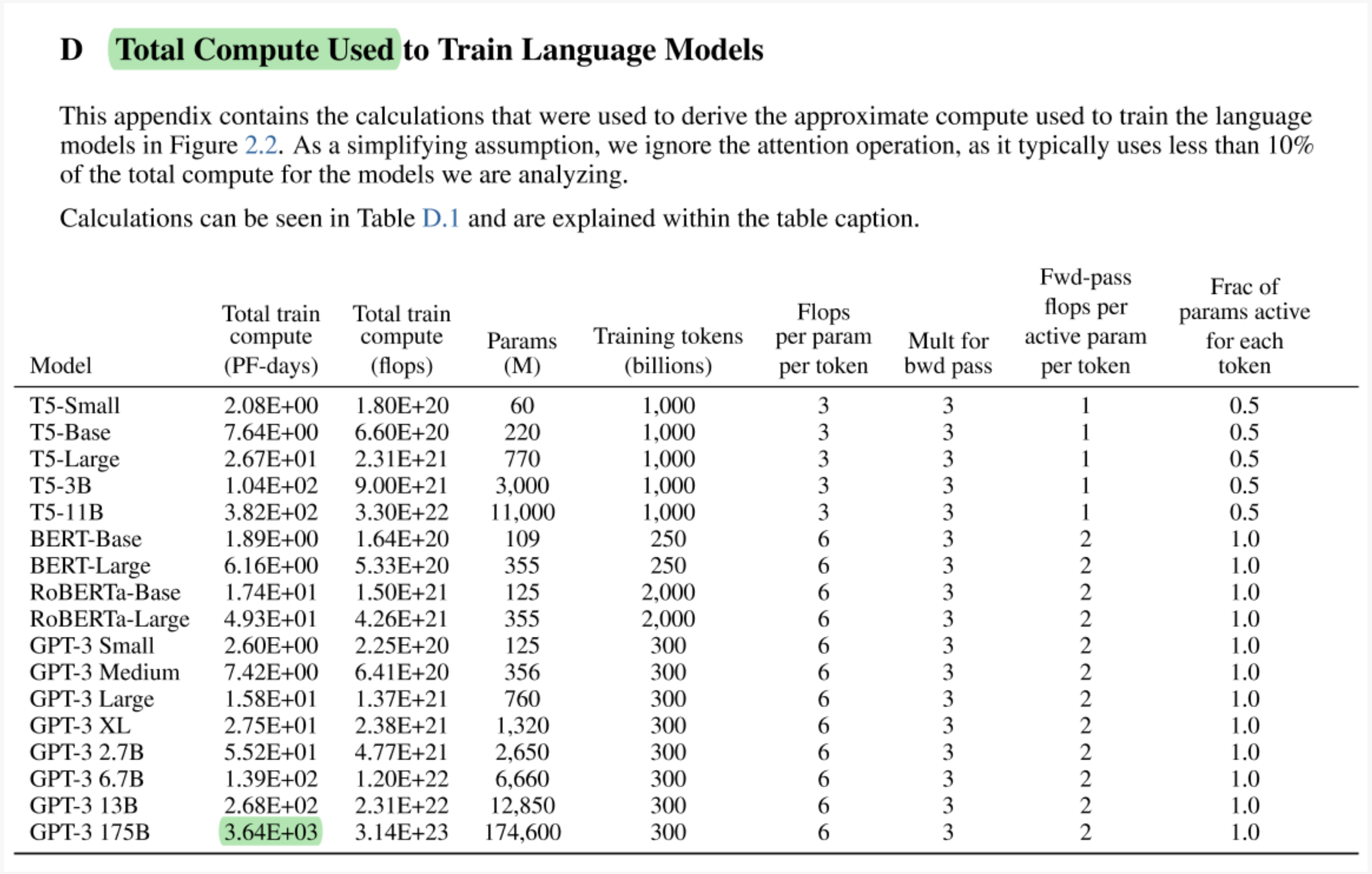
Training time for GPT 3 = 3640 Days
= 9.97 Years

It would take 355 years to train GPT-3 on a single NVIDIA Tesla V100 GPU.

OpenAI launched GPT-3 in May/2020.

Using 1,024x A100 GPUs, researchers calculated that OpenAI could have trained GPT-3 in as little as 34 days.

Estimated that it cost around $5M in compute time to train GPT-3.



D Total Compute Used to Train Language Models

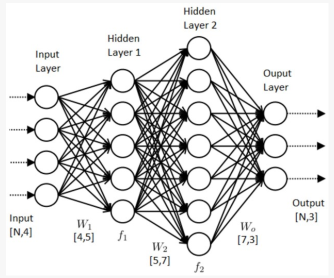
This appendix contains the calculations that were used to derive the approximate compute used to train the language models in Figure 2.2. As a simplifying assumption, we ignore the attention operation, as it typically uses less than 10% of the total compute for the models we are analyzing.

Calculations can be seen in Table D.1 and are explained within the table caption.

Model	Total train compute (PF-days)	Total train compute (flops)	Params (M)	Training tokens (billions)	Flops per param per token	Mult for bwd pass	Fwd-pass flops per active param per token	Frac of params active for each token
T5-Small	2.08E+00	1.80E+20	60	1,000	3	3	1	0.5
T5-Base	7.64E+00	6.60E+20	220	1,000	3	3	1	0.5
T5-Large	2.67E+01	2.31E+21	770	1,000	3	3	1	0.5
T5-3B	1.04E+02	9.00E+21	3,000	1,000	3	3	1	0.5
T5-11B	3.82E+02	3.30E+22	11,000	1,000	3	3	1	0.5
BERT-Base	1.89E+00	1.64E+20	109	250	6	3	2	1.0
BERT-Large	6.16E+00	5.33E+20	355	250	6	3	2	1.0
RoBERTa-Base	1.74E+01	1.50E+21	125	2,000	6	3	2	1.0
RoBERTa-Large	4.93E+01	4.26E+21	355	2,000	6	3	2	1.0
GPT-3 Small	2.60E+00	2.25E+20	125	300	6	3	2	1.0
GPT-3 Medium	7.42E+00	6.41E+20	356	300	6	3	2	1.0
GPT-3 Large	1.58E+01	1.37E+21	760	300	6	3	2	1.0
GPT-3 XL	2.75E+01	2.38E+21	1,320	300	6	3	2	1.0
GPT-3 2.7B	5.52E+01	4.77E+21	2,650	300	6	3	2	1.0
GPT-3 6.7B	1.39E+02	1.20E+22	6,660	300	6	3	2	1.0
GPT-3 13B	2.68E+02	2.31E+22	12,850	300	6	3	2	1.0
GPT-3 175B	3.64E+03	3.14E+23	174,600	300	6	3	2	1.0

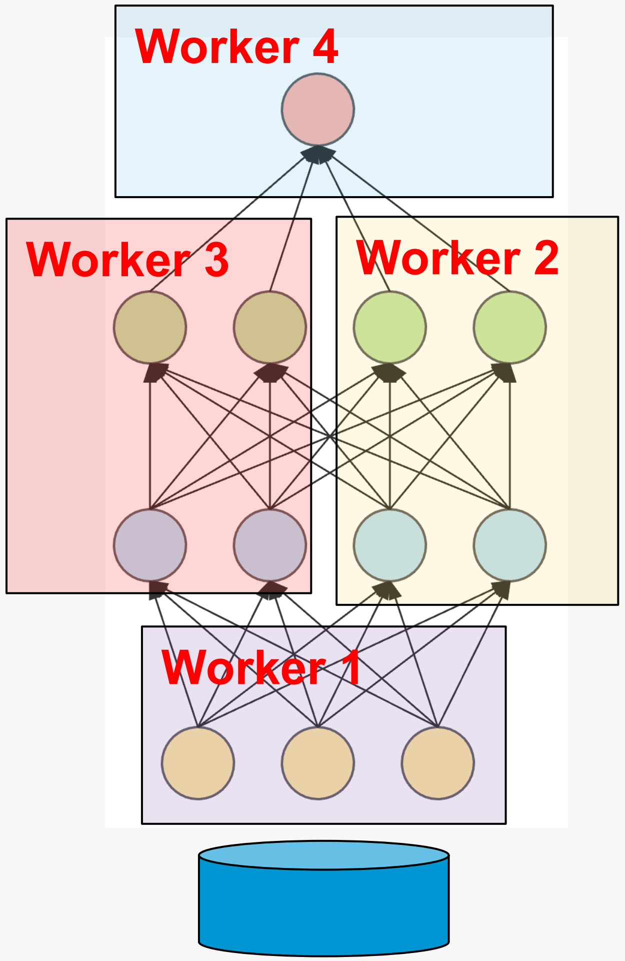
Tom B. Brown et al, “Language Models are Few-Shot Learners”, 2020.

Need for distributed (parallel) training on HPC

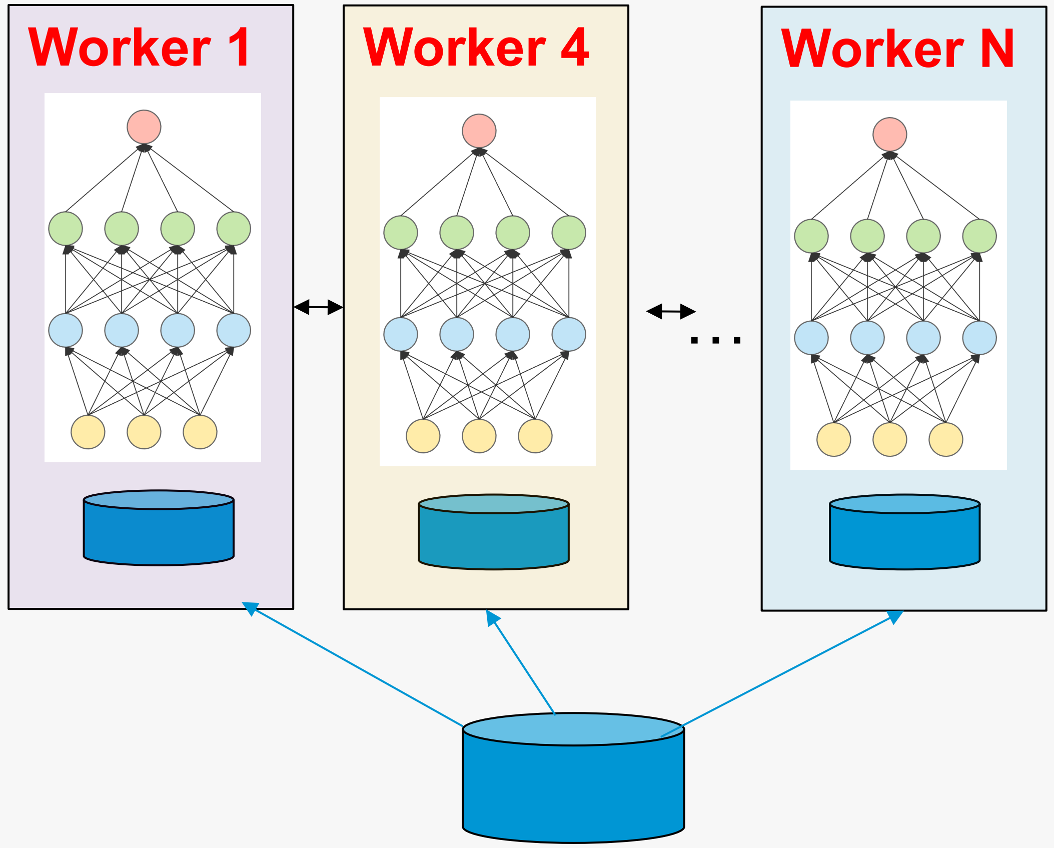


- Increase of model complexity leads to dramatic increase of computation.
- Increase of the amount of dataset makes sequentially scanning the whole dataset increasingly impossible.
- The increase in computational power has been mostly coming (and will continue to come) from parallel computing.
- Coupling of deep learning to traditional HPC simulations might require distributed inference.

Parallelization schemes for distributed learning



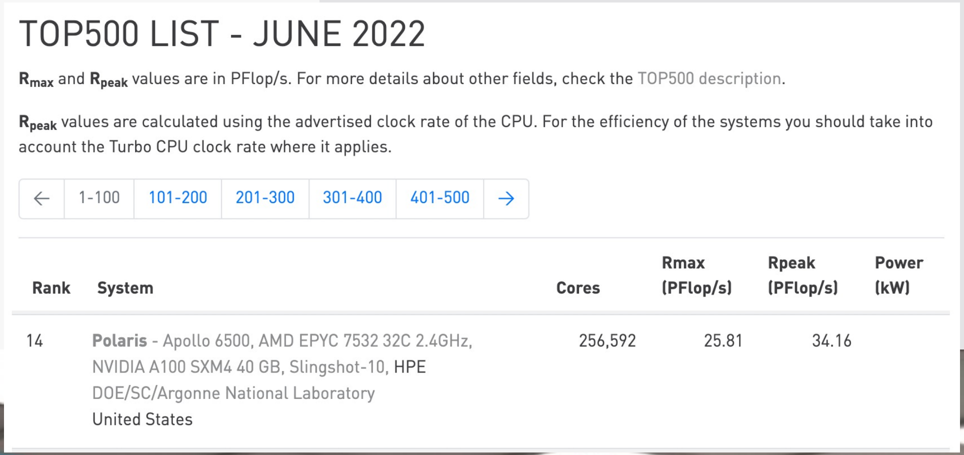
Model parallelism



Data parallelism

Polaris

# of River Compute racks	40
# of Apollo Gen10+ Chassis	280
# of Nodes	560
# of AMD EPYC 7543P CPUs	560
# of NVIDIA A100 GPUs	2240
Total GPU HBM2 Memory	87.5TB
Total CPU DDR4 Memory	280 TB
Total NVMe SSD Capacity	1.75 PB
Interconnect	HPE Slingshot
# of Cassini NICs	1120
# of Rosetta Switches	80
Total Injection BW (w/ Cassini)	28 TB/s
Total GPU DP Tensor Core Flops	44 PF
Total Power	1.8 MW


TOP500 LIST - JUNE 2022

R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← 1-100 101-200 201-300 301-400 401-500 →

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
14	**Polaris** - Apollo 6500, AMD EPYC 7532 32C 2.4GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/Argonne National Laboratory United States	256,592	25.81	34.16	

Argonne NATIONAL LABORATORY

7 Steps to Horovod

How to change a serial code into a data parallel code:

1. Initialize Horovod
2. Pin GPU to each process
3. Checking pointing / printing training time on rank 0
4. Scale the learning rate
5. Set distributed optimizer / gradient tape
6. Broadcast the model & optimizer parameters from rank 0 to other ranks
7. Adjusting dataset loading: number of steps (or batches) per epoch, dataset sharding, etc.

https://eng.uber.com/horovod/

7 Steps to Horovod

Step 1. Initialize Horovod

```
import horovod.tensorflow.keras as hvd
hvd.init()
```

Step 2. Pin GPU to each process

```
# Pin GPU to the rank - we set one GPU per process
tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU’)
```

Step 3. Checkpointing on root rank

```
# It is important to let only one process to do the checkpointing I/O.
if hvd.rank() == 0:
  callbacks.append(tf.keras.callbacks.ModelCheckpoint('./checkpoints-km{epoch}.h5’))
if (hvd.rank()==0):
  print("Hvd Procs %d Total time: %s second" %(hvd.size(),t1-t0))
```

Gradient Descent

Minimizing the loss:

$$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w).$$

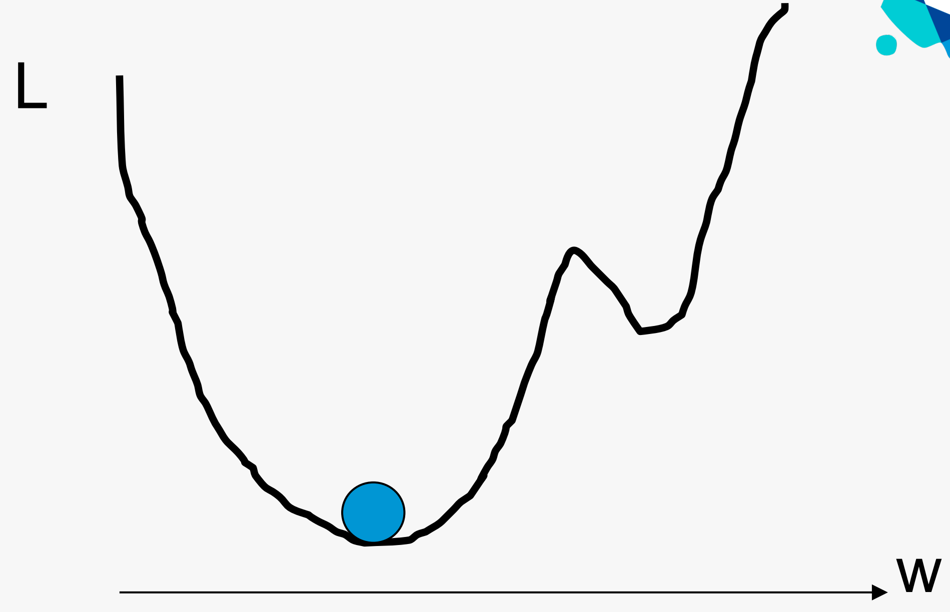
Dataset Weight

Stochastic Gradient Descent (SGD) update

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Minibatch

Model is updated at each step.

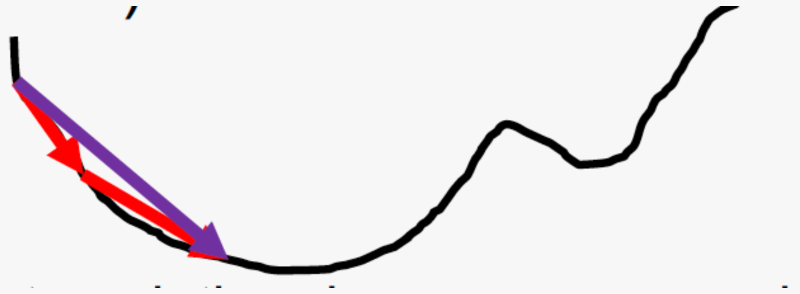


- One minibatch is divided into many sub minibatches and each is feed into one of the workers

7 Steps to Horovod

Step 4: Scale the learning rate with number of workers

If we keep the local batch size on each rank the same, the global batch size increases by n times The learning rate should increase proportionally



When the minibatch size is multiplied by k, multiply the learning rate by k.

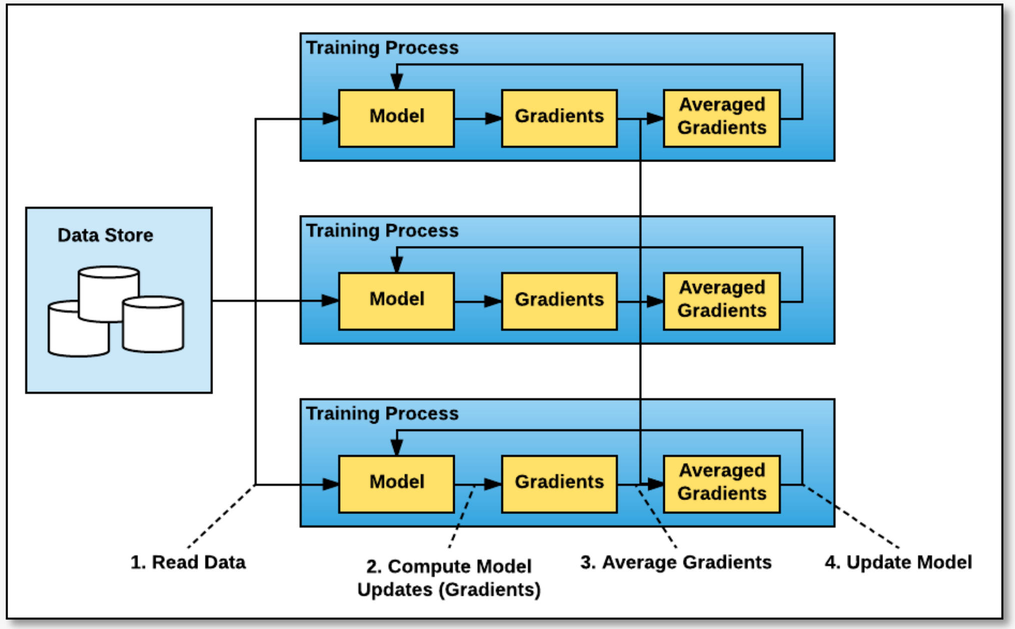
```
opt = tf.optimizers.Adam(lr * hvd.size())
```

Step 5 : Wrap tf.optimizer with Horovod DistributedOptimizer

```
opt = hvd.DistributedOptimizer(opt)
```

Gradients are aggregated over all the workers through MPI_Allreduce

Gradients are averaged at each step (not each epoch)



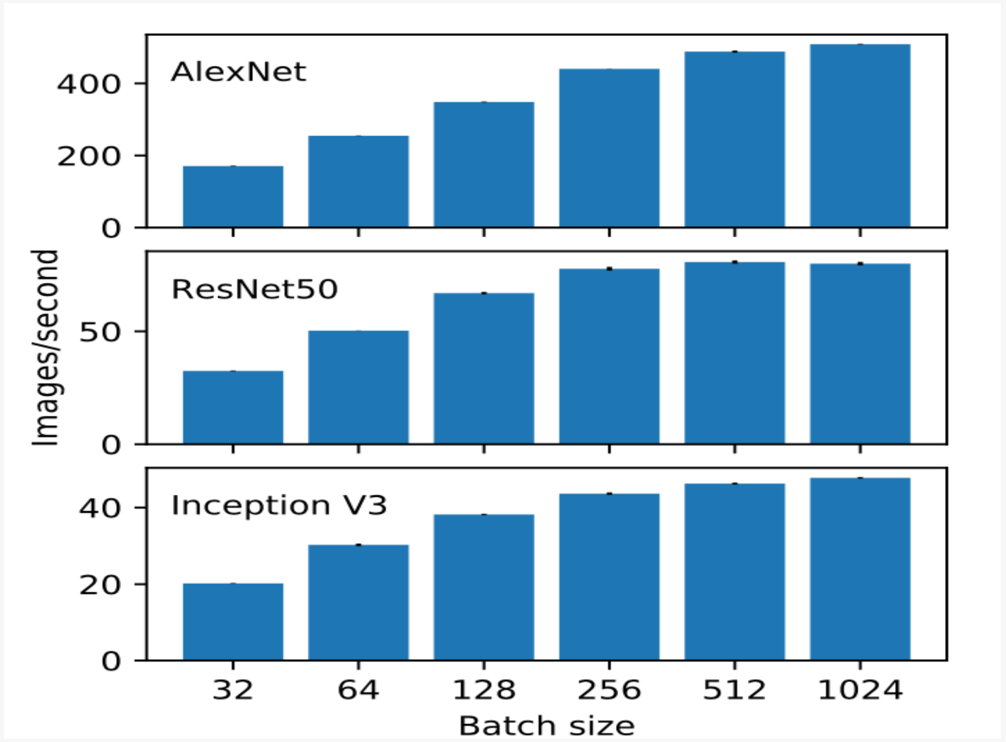
Large minibatch training

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

$\uparrow$

Minibatch

- Option 1. Keeping the same global minibatch size with each worker processing B/N batch (strong scaling)
- Option 2. Increasing the global minibatch size by N times, so that each worker processes batches of size B (week scaling)



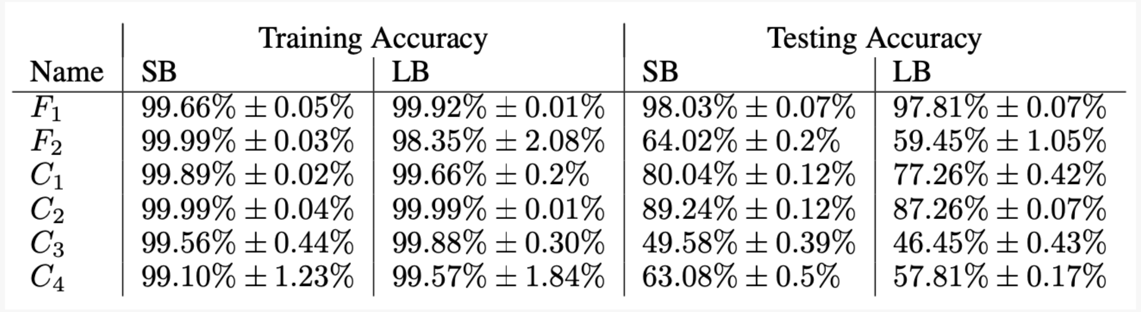
Per node throughput of different local batch size

H. Zheng, https://www.alcf.anl.gov/files/Zheng_SDL_ML_Frameworks_1.pdf

1. Decrease of local batch size reduces the per node throughput;
2. Increase of global minibatch size reduces the number of updates on each epoch (n=X/B); thus it increases the compute/communication ratio

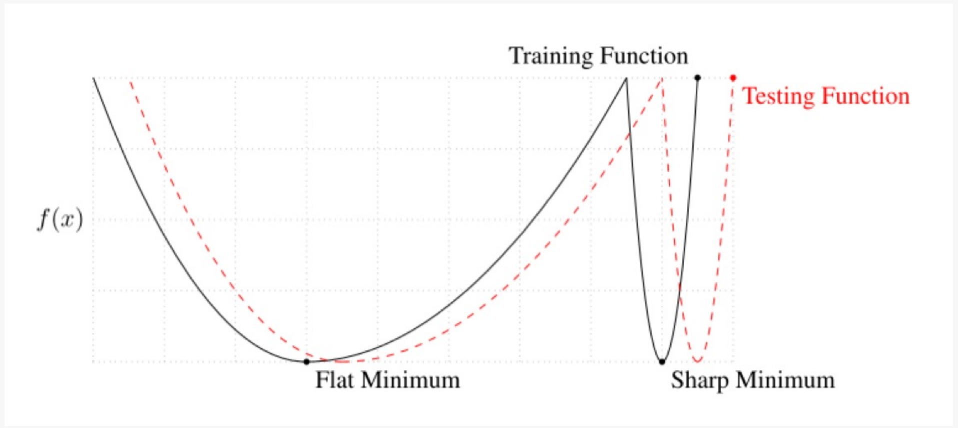
Challenges with large batch training

- Convergence issue: at the initial stages of training, the model is far away from optimal solution $\nabla l(x, \omega_{t+j}) \sim \nabla l(x, \omega_t)$ breaks down. Training is not stable with large learning rate in the beginning;
- Generalization gap: large batch size training tends to be trapped at local minimum with lower testing accuracy (generalize worse).



	Training Accuracy		Testing Accuracy	
Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks



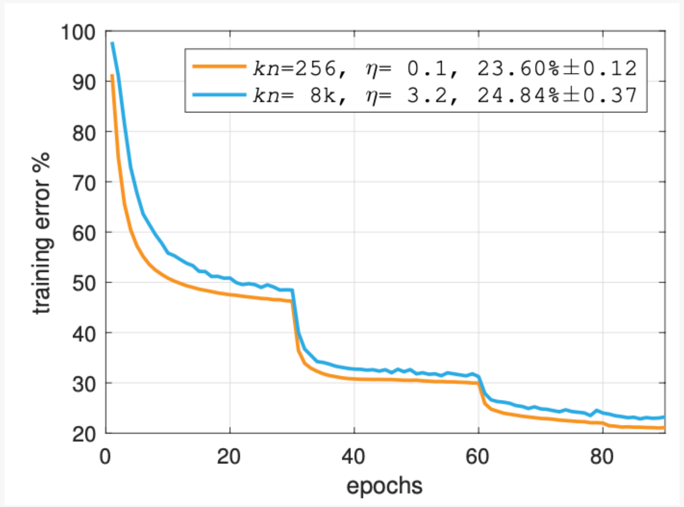
"... large-batch ... converge to sharp minimizers of the training function ... In contrast, small-batch methods converge to flat minimizers"

Keskar et al, arXiv:1609.04836

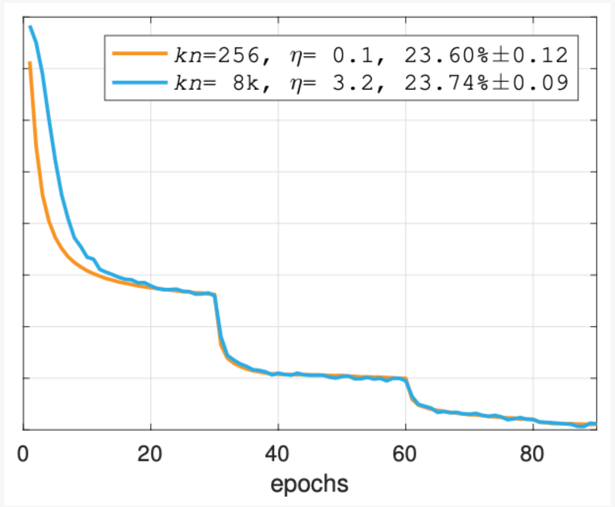
Challenges with large batch training

Solutions: using warm up steps

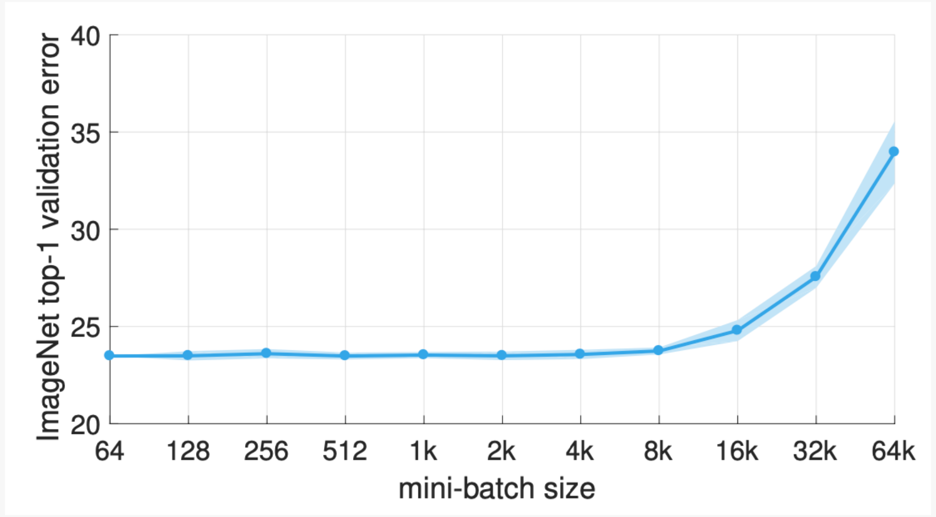
- Using a smaller learning rate at the initial stage of training (couple epochs), and gradually increase to $\hat{\eta} = N\eta$
- Using linear scaling of learning rate ($\hat{\eta} = N\eta$)



No warm up



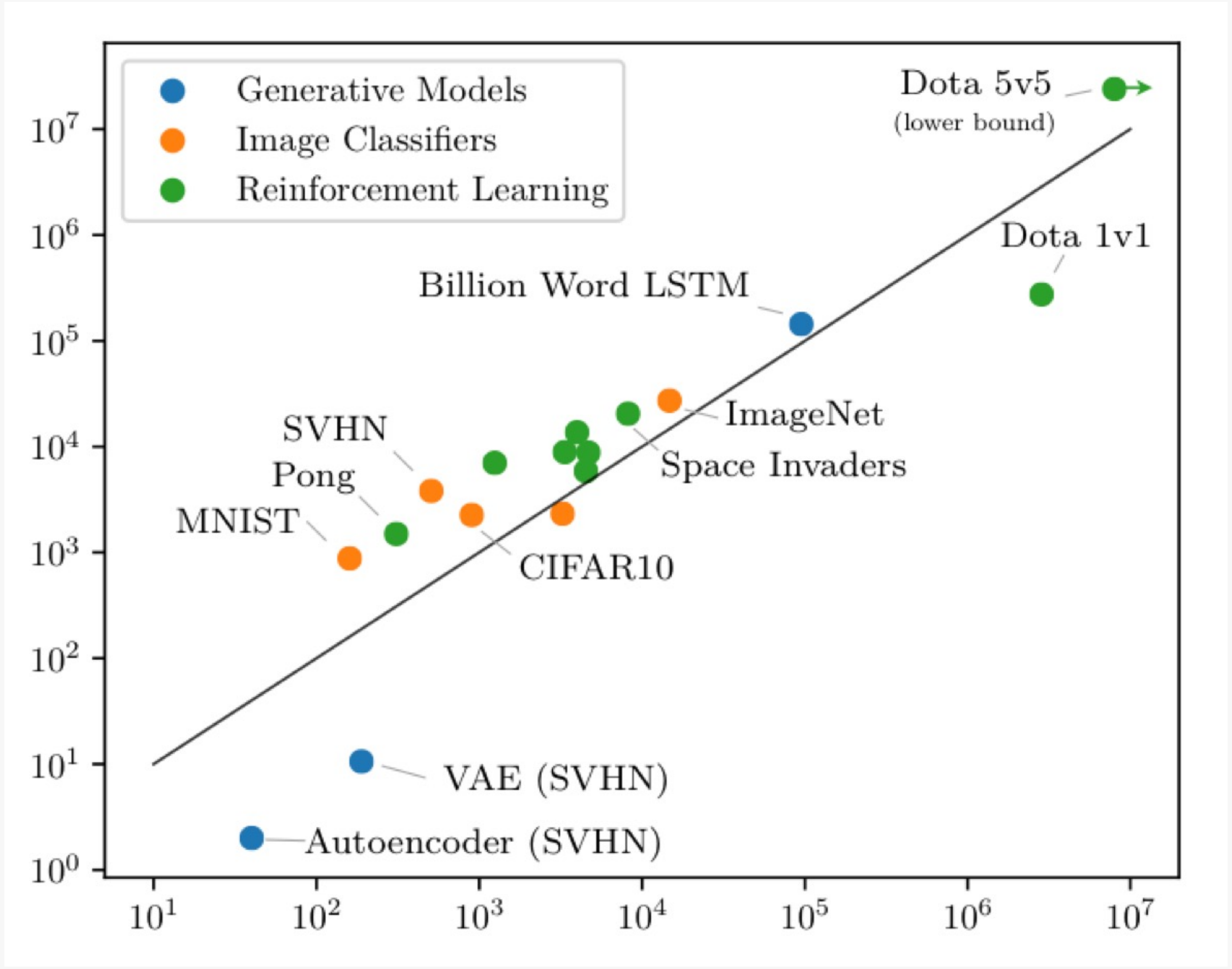
Gradual warm up



This scheme works up to 8k batch size

P. Goyal et al,arXiv: 1706.02677

Challenges with large batch training



Predicted critical **maximum batch size** beyond which the model does not perform well.

S. McCandlish, J. Kaplan, D. Amodei, arXiv:1812.06162

Maximum batch size place limit to data parallel:

N_workers < Maximum batch size

7 Steps to Horovod

6. Broadcast the model & optimizer parameters to other rank

```
callbacks = [  # broad cast
               hvd.callbacks.BroadcastGlobalVariablesCallback(0),
               # Average metric at the end of every epoch
               hvd.callbacks.MetricAverageCallback(),
               # Warmup
               hvd.callbacks.LearningRateWarmupCallback(warmup_epochs=3,initial_lr=_lr),
            ]
```

7. Adjusting dataset loading: number of steps (or batches) per epoch, dataset sharding, etc.

```
       steps_per_epoch=60000/hvd.size()/batch_size
```

Tensorflow with Horovod

```
import tensorflow as tf
import horovod.tensorflow as hvd
layers = tf.contrib.layers
learn = tf.contrib.learn
def main():
   # Horovod: initialize Horovod.
   hvd.init()
   # Download and load MNIST dataset.
   mnist = learn.datasets.mnist.read_data_sets('MNIST-data-%d' % hvd.rank())
   # Horovod: adjust learning rate based on number of GPUs.
   opt = tf.train.RMSPropOptimizer(0.001 * hvd.size())
   # Horovod: add Horovod Distributed Optimizer
   opt = hvd.DistributedOptimizer(opt)
   hooks = [
        hvd.BroadcastGlobalVariablesHook(0),
        tf.train.StopAtStepHook(last_step=20000 // hvd.size()),
        tf.train.LoggingTensorHook(tensors={'step': global_step, 'loss': loss},
                                   every_n_iter=10),
    ]
   checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None
    with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                           hooks=hooks,
                                           config=config) as mon_sess
```

More examples can be found in https://github.com/uber/horovod/blob/master/examples/

PyTorch with Horovod

```python
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                               transform=transforms.Compose([
                                   transforms.ToTensor(),
                                   transforms.Normalize((0.1307,), (0.3081,))
                               ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

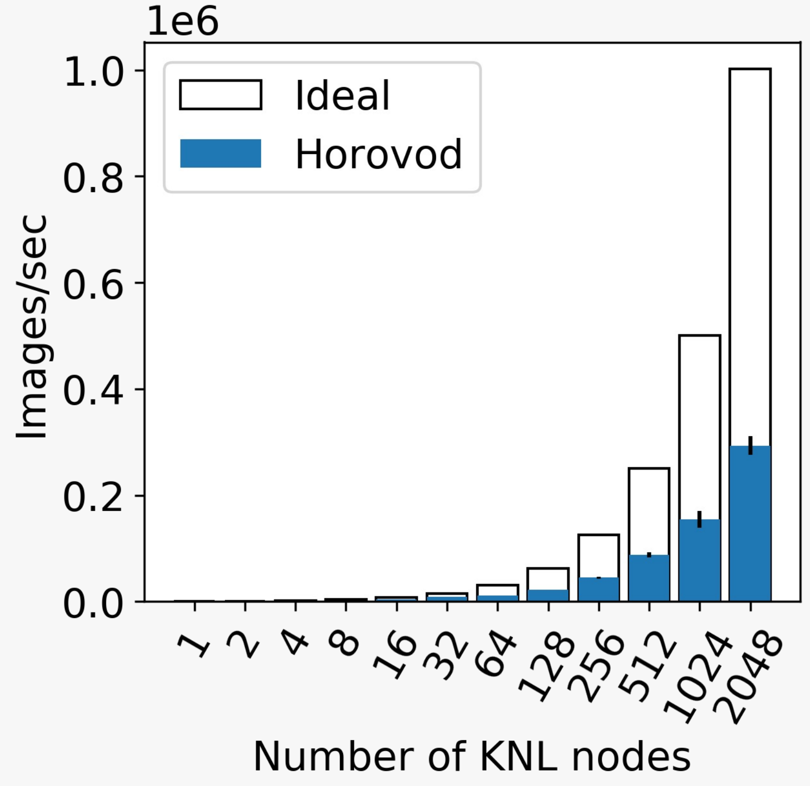
More examples can be found in https://github.com/uber/horovod/blob/master/examples/

Keras with Horovod

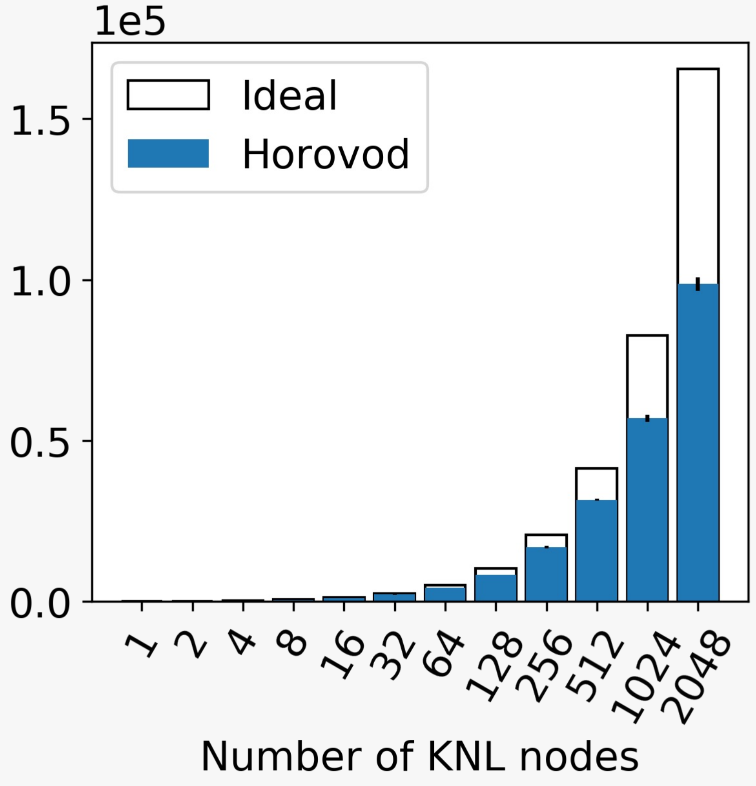
```
import keras
import tensorflow as tf
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])
callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]
# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))
model.fit(x_train, y_train, batch_size=batch_size,
          callbacks=callbacks,
          epochs=epochs,
          verbose=1, validation_data=(x_test, y_test))
```

More examples can be found in https://github.com/uber/horovod/blob/master/examples/

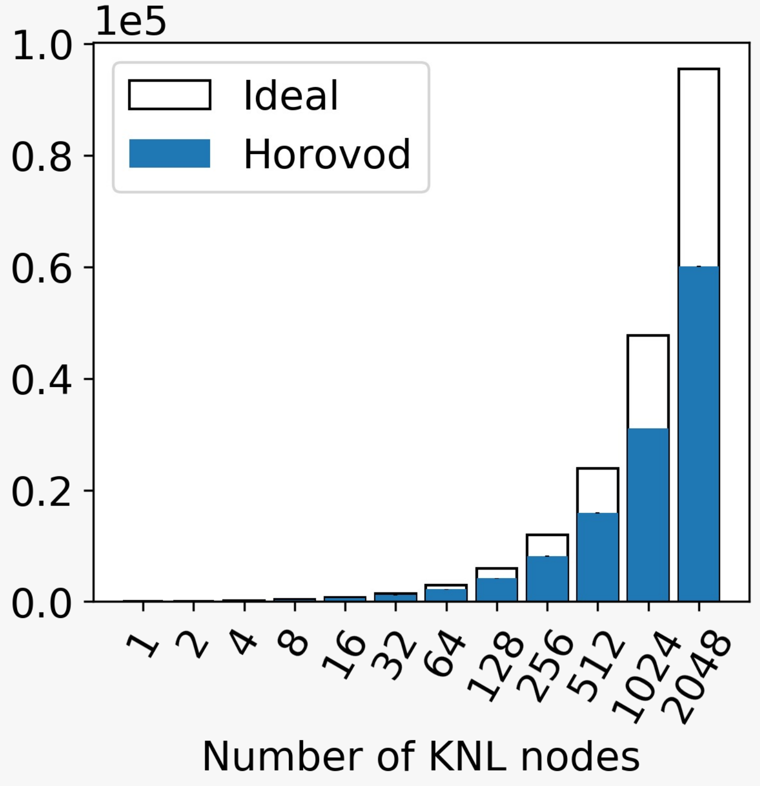
Scaling TensorFlow using Horovod on Theta @ ALCF (Intel Knights Landing): batch size = 512



AlexNet

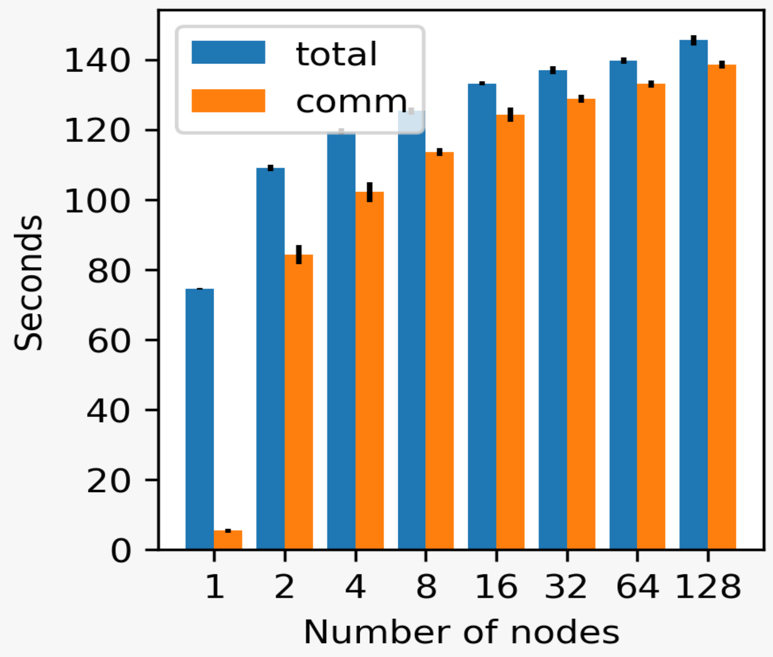


ResNet-50

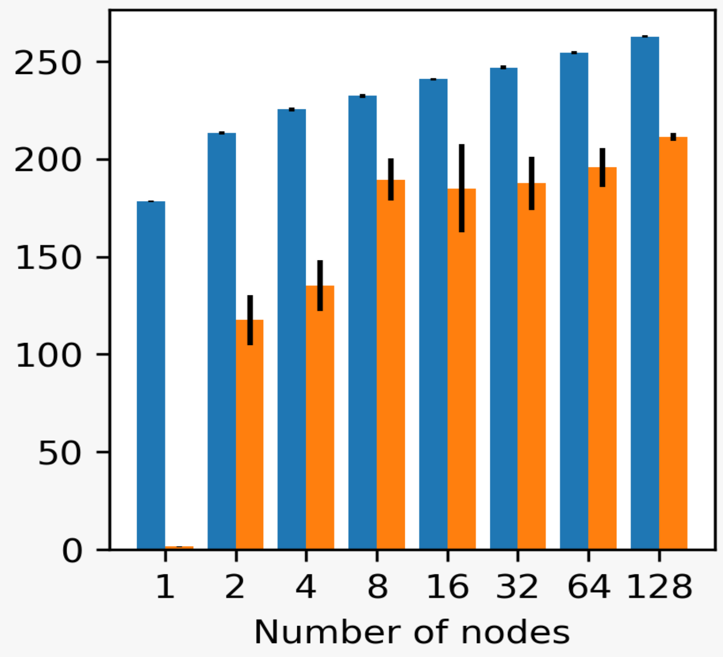


Inception V3

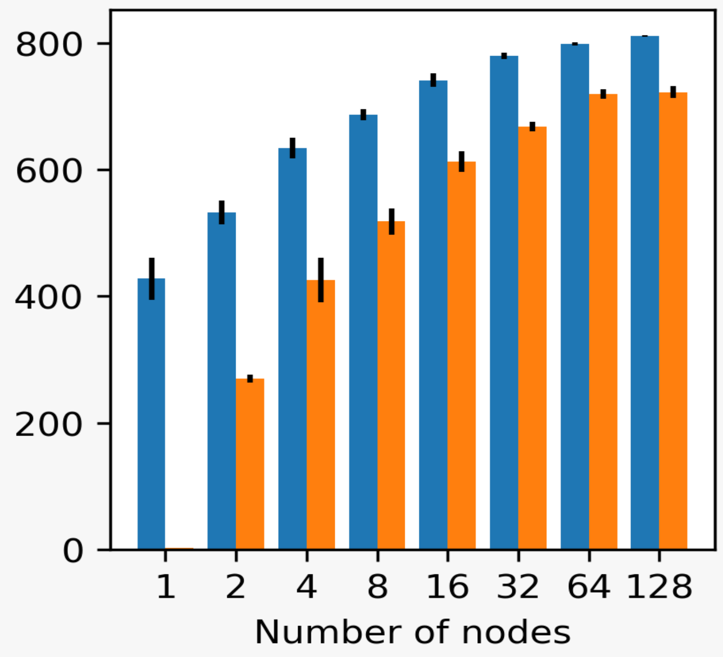
Overlap of communication and compute in Horovod



AlexNet
(batch size = 512,
50 steps)



ResNet-50
(batch size =64,
50 steps)

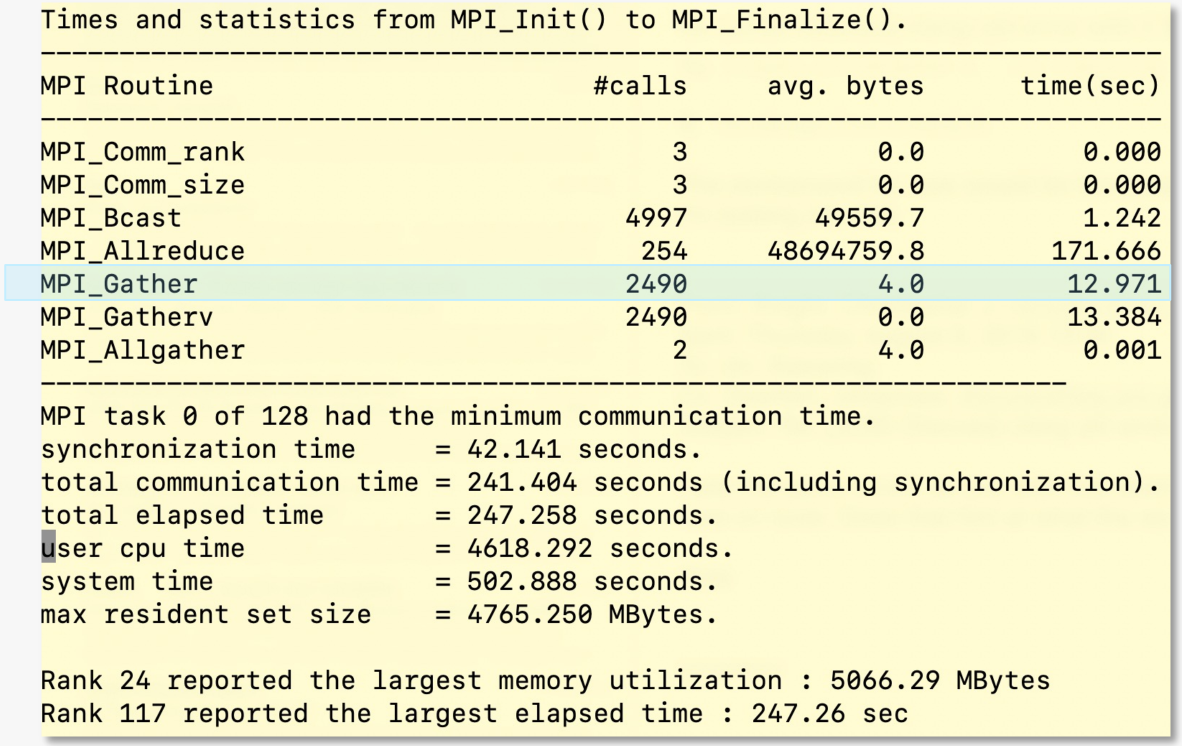


Inception V3
(batch size =128,
50 steps)

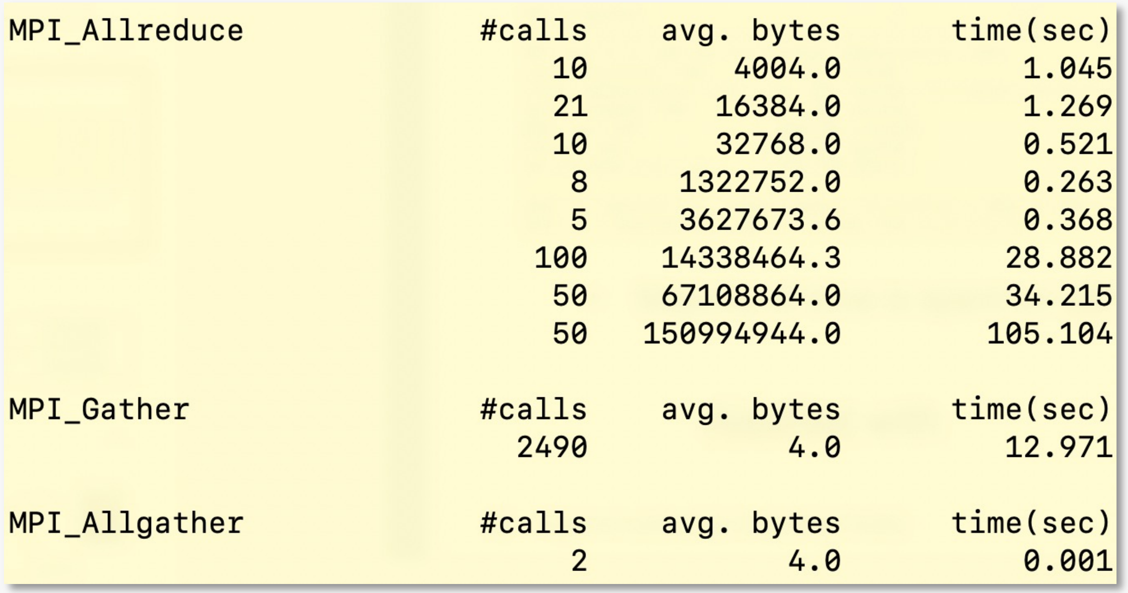
Increase of total time is smaller than the increase of the communication time, which indicates large overlap between compute and communication.

MPI flat profile for Horovod

(AlexNet, batch size=512, 128 KNL nodes)



```
Times and statistics from MPI_Init() to MPI_Finalize().
-----------------------------------------------------------------------
MPI Routine                        #calls     avg. bytes      time(sec)
-----------------------------------------------------------------------
MPI_Comm_rank                           3            0.0          0.000
MPI_Comm_size                           3            0.0          0.000
MPI_Bcast                            4997        49559.7          1.242
MPI_Allreduce                         254     48694759.8        171.666
MPI_Gather                           2490            4.0         12.971
MPI_Gatherv                          2490            0.0         13.384
MPI_Allgather                           2            4.0          0.001
-----------------------------------------------------------------
MPI task 0 of 128 had the minimum communication time.
synchronization time     = 42.141 seconds.
total communication time = 241.404 seconds (including synchronization).
total elapsed time       = 247.258 seconds.
user cpu time            = 4618.292 seconds.
system time              = 502.888 seconds.
max resident set size    = 4765.250 MBytes.

Rank 24 reported the largest memory utilization : 5066.29 MBytes
Rank 117 reported the largest elapsed time : 247.26 sec
```



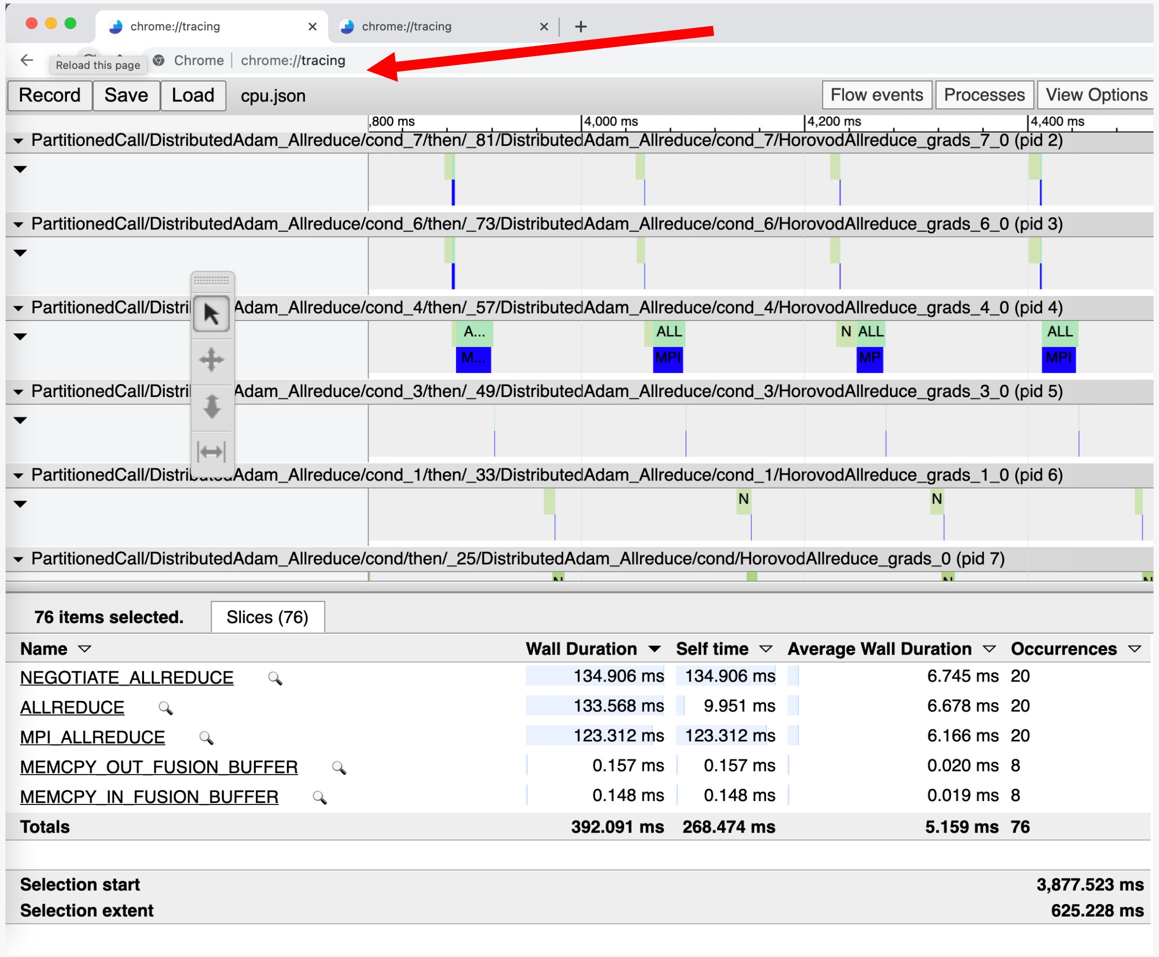
```
MPI_Allreduce                #calls     avg. bytes      time(sec)
                                 10         4004.0          1.045
                                 21        16384.0          1.269
                                 10        32768.0          0.521
                                  8      1322752.0          0.263
                                  5      3627673.6          0.368
                                100     14338464.3         28.882
                                 50     67108864.0         34.215
                                 50    150994944.0        105.104

MPI_Gather                   #calls     avg. bytes      time(sec)
                               2490            4.0         12.971

MPI_Allgather                #calls     avg. bytes      time(sec)
                                  2            4.0          0.001
```

- Majority of time is spent on MPI_Allreduce with message size ranging from KB-GB
- There is load imbalance (synchronization time)

```
LD_PRELOAD=/soft/perftools/hpctw/lib/libmpitrace.so mpirun -np 8 python 03_keras_cnn_concise_hvd.py --epochs 10
```

Horovod Timeline

```
HOROVOD_TIMELINE=gpu.json mpirun -np 8 python 03_keras_cnn_concise_hvd.py
HOROVOD_TIMELINE=cpu.json mpirun -np 8 python 03_keras_cnn_concise_hvd.py --device cpu
```

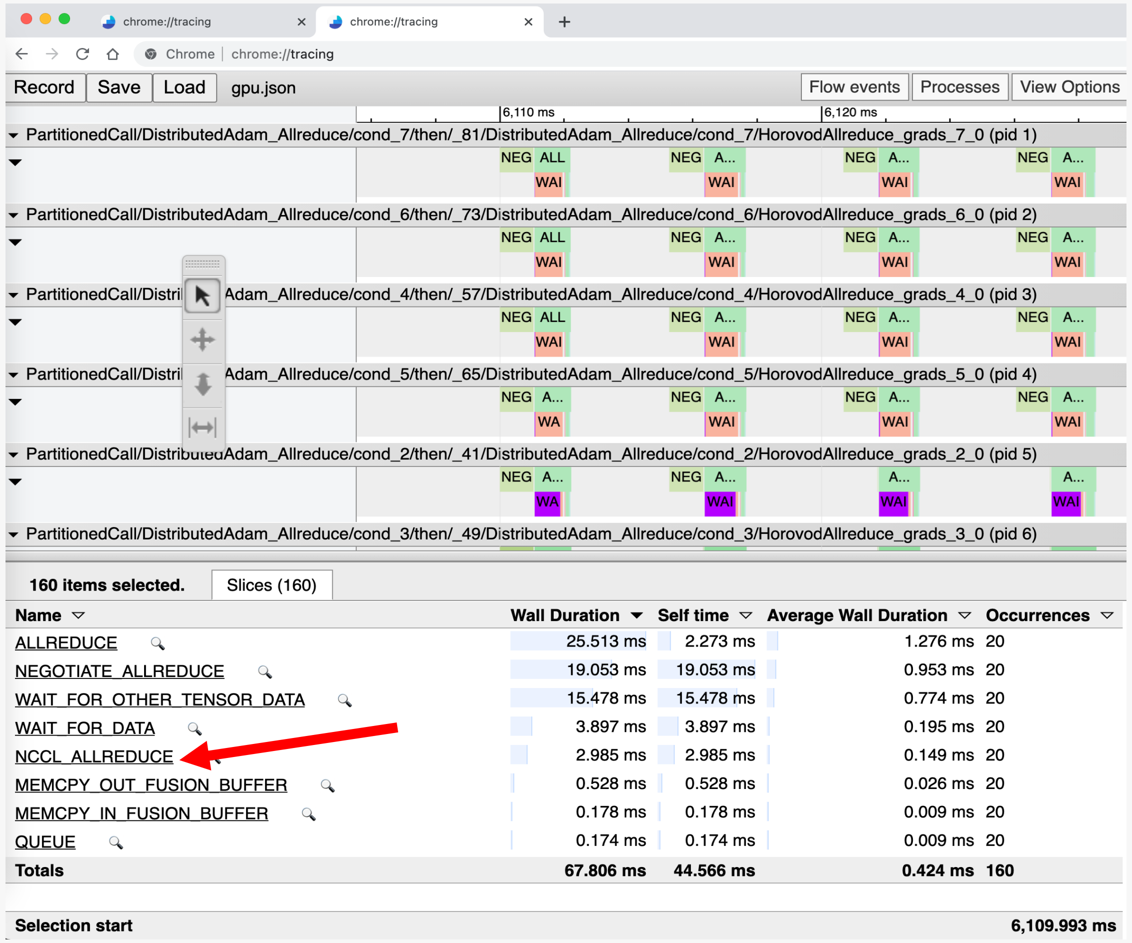


chrome://tracing
Reload this page
Chrome | chrome://tracing
Record
Save
Load
cpu.json
Flow events
Processes
View Options
3,800 ms
4,000 ms
4,200 ms
4,400 ms
PartitionedCall/DistributedAdam_Allreduce/cond_7/then/_81/DistributedAdam_Allreduce/cond_7/HorovodAllreduce_grads_7_0 (pid 2)
PartitionedCall/DistributedAdam_Allreduce/cond_6/then/_73/DistributedAdam_Allreduce/cond_6/HorovodAllreduce_grads_6_0 (pid 3)
PartitionedCall/Distri Adam_Allreduce/cond_4/then/_57/DistributedAdam_Allreduce/cond_4/HorovodAllreduce_grads_4_0 (pid 4)
A...
ALL
N ALL
ALL
M...
MPI
MP
MPI
PartitionedCall/Distri Adam_Allreduce/cond_3/then/_49/DistributedAdam_Allreduce/cond_3/HorovodAllreduce_grads_3_0 (pid 5)
PartitionedCall/Distri Adam_Allreduce/cond_1/then/_33/DistributedAdam_Allreduce/cond_1/HorovodAllreduce_grads_1_0 (pid 6)
N
N
PartitionedCall/DistributedAdam_Allreduce/cond/then/_25/DistributedAdam_Allreduce/cond/HorovodAllreduce_grads_0 (pid 7)

76 items selected. Slices (76)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
NEGOTIATE_ALLREDUCE	134.906 ms	134.906 ms	6.745 ms	20
ALLREDUCE	133.568 ms	9.951 ms	6.678 ms	20
MPI_ALLREDUCE	123.312 ms	123.312 ms	6.166 ms	20
MEMCPY_OUT_FUSION_BUFFER	0.157 ms	0.157 ms	0.020 ms	8
MEMCPY_IN_FUSION_BUFFER	0.148 ms	0.148 ms	0.019 ms	8
Totals	392.091 ms	268.474 ms	5.159 ms	76

Selection start	3,877.523 ms
Selection extent	625.228 ms



chrome://tracing
Chrome | chrome://tracing
Record
Save
Load
gpu.json
Flow events
Processes
View Options
6,110 ms
6,120 ms
PartitionedCall/DistributedAdam_Allreduce/cond_7/then/_81/DistributedAdam_Allreduce/cond_7/HorovodAllreduce_grads_7_0 (pid 1)
PartitionedCall/DistributedAdam_Allreduce/cond_6/then/_73/DistributedAdam_Allreduce/cond_6/HorovodAllreduce_grads_6_0 (pid 2)
PartitionedCall/Distri Adam_Allreduce/cond_4/then/_57/DistributedAdam_Allreduce/cond_4/HorovodAllreduce_grads_4_0 (pid 3)
PartitionedCall/Distri Adam_Allreduce/cond_5/then/_65/DistributedAdam_Allreduce/cond_5/HorovodAllreduce_grads_5_0 (pid 4)
PartitionedCall/DistributedAdam_Allreduce/cond_2/then/_41/DistributedAdam_Allreduce/cond_2/HorovodAllreduce_grads_2_0 (pid 5)
PartitionedCall/DistributedAdam_Allreduce/cond_3/then/_49/DistributedAdam_Allreduce/cond_3/HorovodAllreduce_grads_3_0 (pid 6)
NEG
ALL
A...
WAI
WA

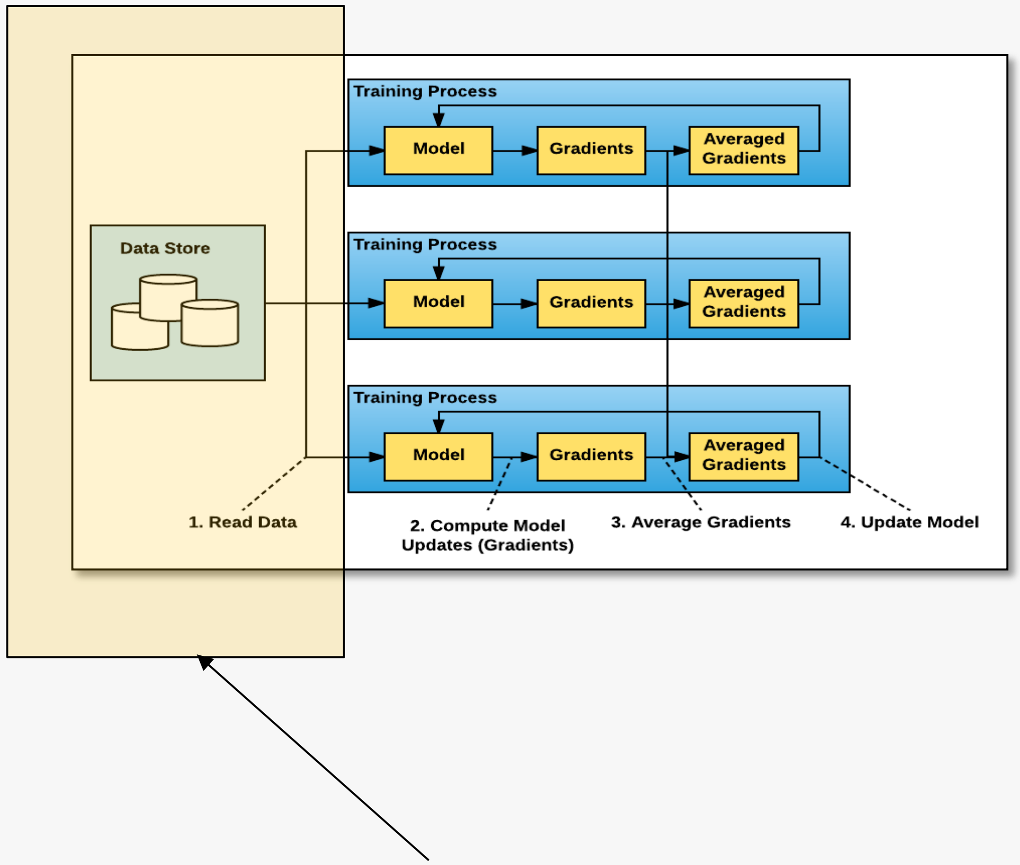
160 items selected. Slices (160)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
ALLREDUCE	25.513 ms	2.273 ms	1.276 ms	20
NEGOTIATE_ALLREDUCE	19.053 ms	19.053 ms	0.953 ms	20
WAIT_FOR_OTHER_TENSOR_DATA	15.478 ms	15.478 ms	0.774 ms	20
WAIT_FOR_DATA	3.897 ms	3.897 ms	0.195 ms	20
NCCL_ALLREDUCE	2.985 ms	2.985 ms	0.149 ms	20
MEMCPY_OUT_FUSION_BUFFER	0.528 ms	0.528 ms	0.026 ms	20
MEMCPY_IN_FUSION_BUFFER	0.178 ms	0.178 ms	0.009 ms	20
QUEUE	0.174 ms	0.174 ms	0.009 ms	20
Totals	67.806 ms	44.566 ms	0.424 ms	160

Selection start	6,109.993 ms

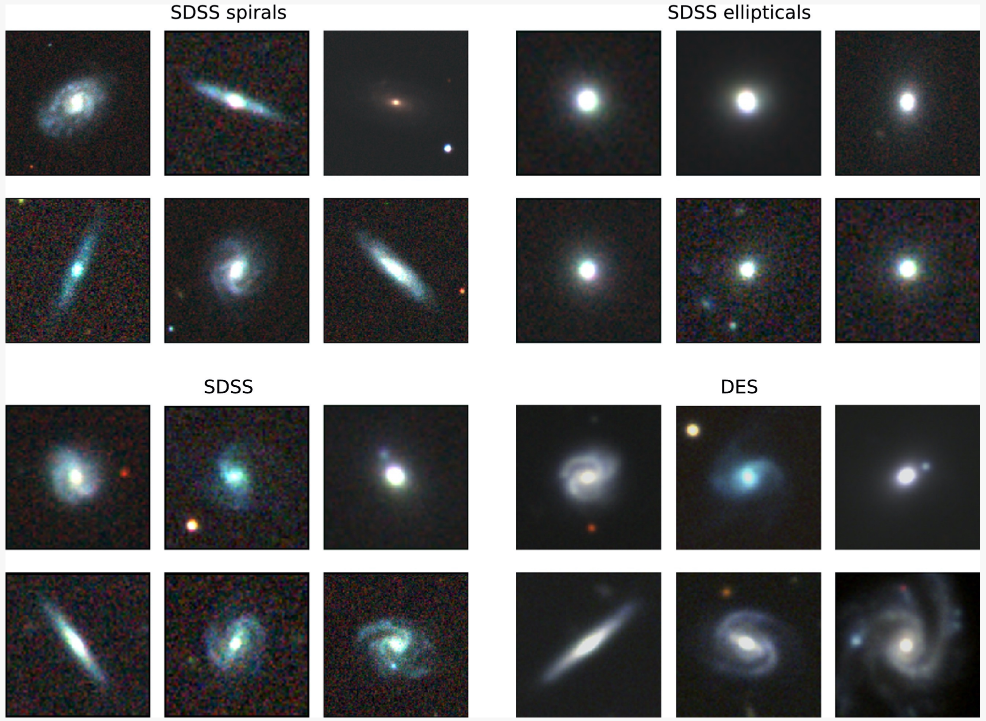
I/O and data management

- Parallel IO is needed: each worker only reads part of the dataset they needed(using MPIIO / parallel HDF5)
- Preprocess the raw data (resize, interpolation, etc) into binary format before the training. *Shuffling in the memory instead of in I/O*
- Store the dataset in a reasonable way (avoiding file per sample)
- Prefetch the data (from disk; from host to device) *Streaming I/O provided by frameworks*

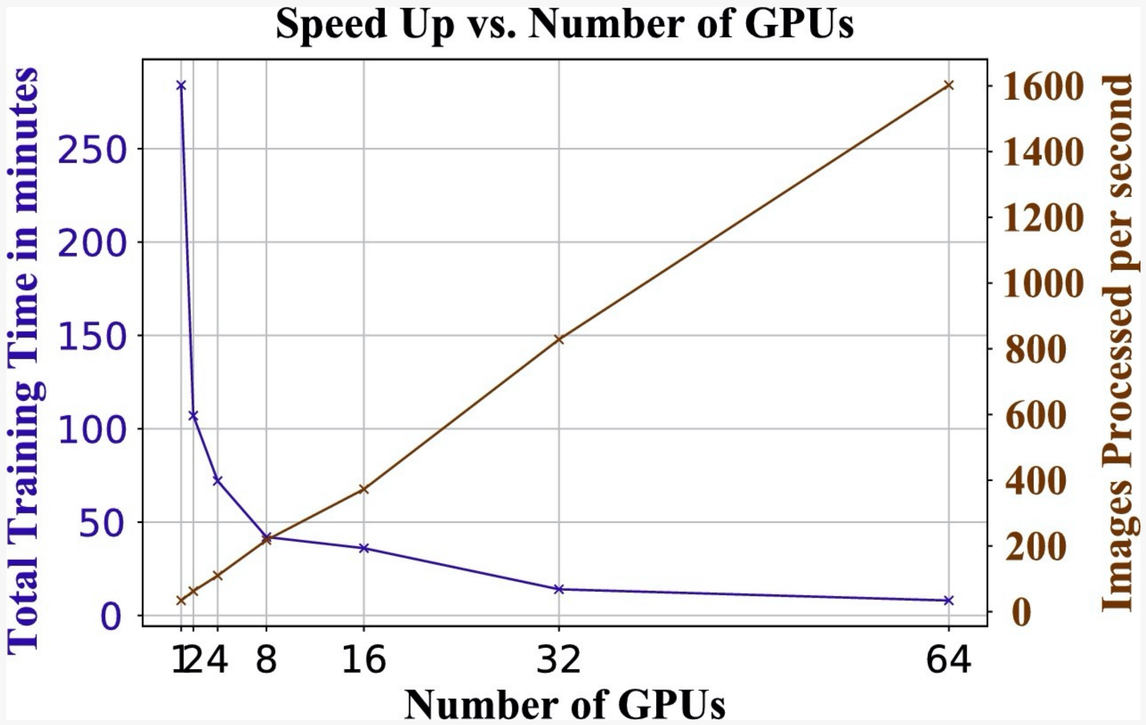


I/O and data management

Science use case 1 - Galaxy classification using modified Xception model



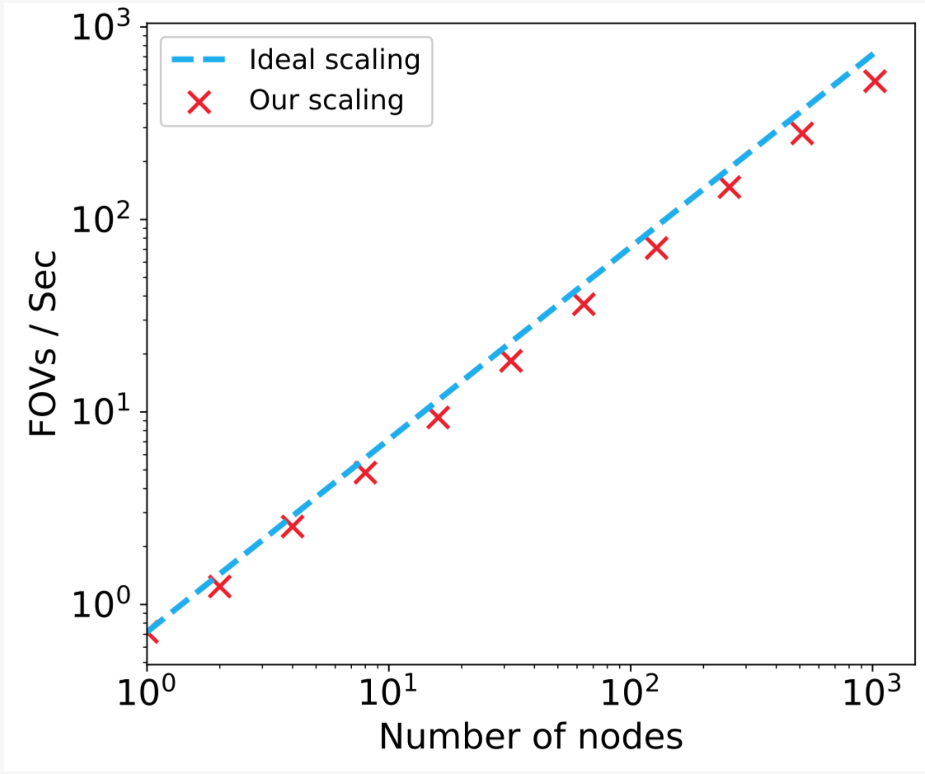
Galaxy images



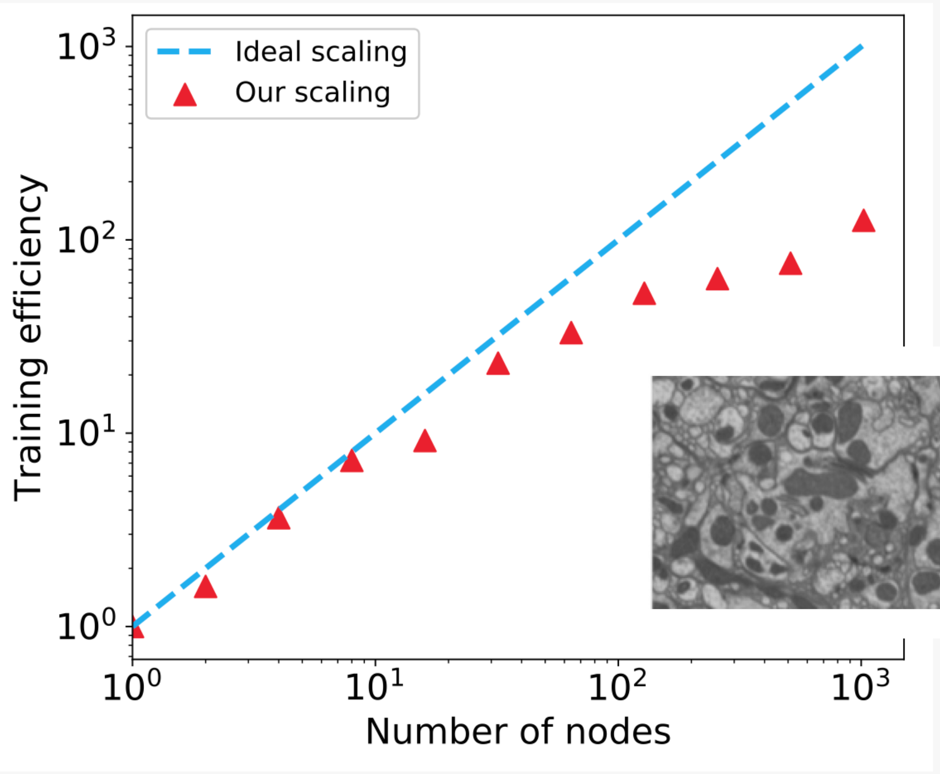
~ 5 Hrs using 1 K80 GPU to 8 mins using 64 K80 GPUs using computing resource from Cooley @ ALCF

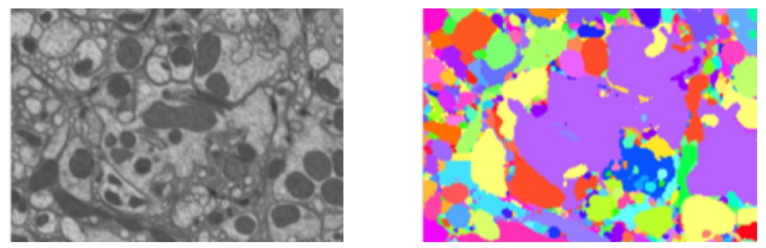
A Khan et al, Physics Letters B, 793, 70-77 (2019)

Science use case 2 - Brain Mapping: reconstruction of brain cells from volume electron microscopy data



Scaling results in terms of throughput



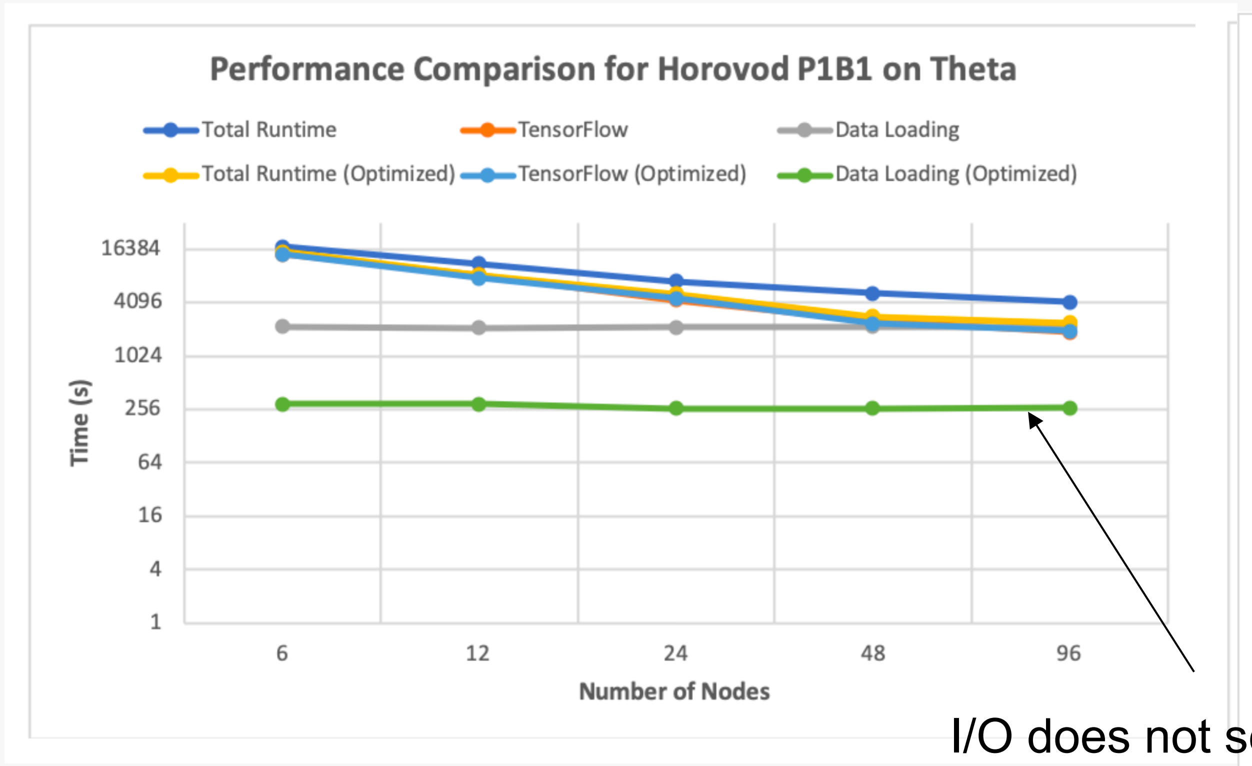


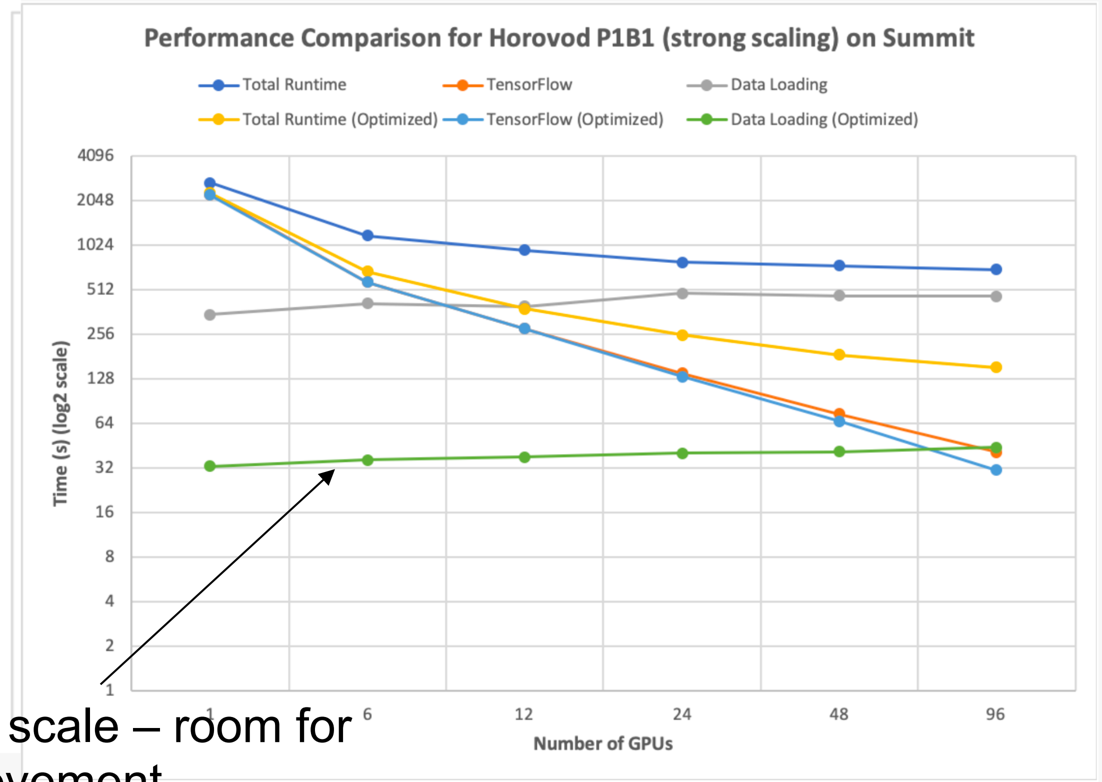
Scaling results in terms of training efficiency (measured by time needed for the training to reach to certain accuracy)

Work done on Theta @ ALCF

W. Dong et al, arXiv:1905.06236 [cs.DC]

Science use case 3 - CANDLE benchmarks: deep learning for cancer problems





I/O does not scale – room for further improvement.

Strong scaling study of CANDLE P1B1 on Theta and Summit

X. Wu et al SC18 Workshop on Python for High-Performance and Scientific Computing

Conclusion

- Increase of model complexity and the amount of dataset
- Data parallelism can scale efficiently in HPC supercomputers
- Warm up steps might be needed to stabilize the initial stage of training and to avoid the generation gap for large batch size training
- Distributed learning requires efficient and scalable I/O and data management.

References

- https://horovod.readthedocs.io/en/stable/
- Sergeev, A., Del Balso, M. (2017) Meet Horovod: Uber’s Open Source Distributed Deep Learning Framework for TensorFlow. Retrieved from https://eng.uber.com/horovod/
- Sergeev, A. (2017) Horovod - Distributed TensorFlow Made Easy. Retrieved from https://www.slideshare.net/AlexanderSergeev4/horovod-distributed-tensorflow-made-easy

Hands on Exercise

```
ssh <username>@polaris.alcf.anl.gov

ssh <username>@ theta.alcf.anl.gov
ssh thetagpusn1

module load datascience

/lus/grand/projects/ATPESC2022/EXAMPLES/track-8-ML/Horovod_Examples_atpesc22 [Polaris]
/grand/projects/ATPESC2022/EXAMPLES/track-8-ML [Theta]

cd /lus//grand/projects/ATPESC2022/usr/<username> [Polaris]
cd /grand/projects/ATPESC2022/usr/<username>  [Theta]

qsub -l select=16:system=polaris -l walltime=01:00:00 -A ATPESC2022 -q R313446 ./qsub_polaris.sc
qsub -A ATPESC2022 -q ATPESC2022 -n 16 -t 60 --attrs filesystems='home,grand,theta-fs0' ./qsub_theta.sc
qsub -A ATPESC2022 -q training-gpu -n 16 -t 60 --attrs filesystems='home,grand,theta-fs0' ./qsub_thetagpu.sc
```

1. https://status.alcf.anl.gov/theta/activity
2. https://github.com/argonne-lcf/ATPESC_MachineLearning
3. https://github.com/argonne-lcf/sdl_ai_workshop 4.
4. https://github.com/argonne-lcf/ai-science-training-series

Thank you!