Argonne NATIONAL LABORATORY

Data Parallel Deep Learning

Huihuo Zheng
Data science group at ALCF
August 9, 2019

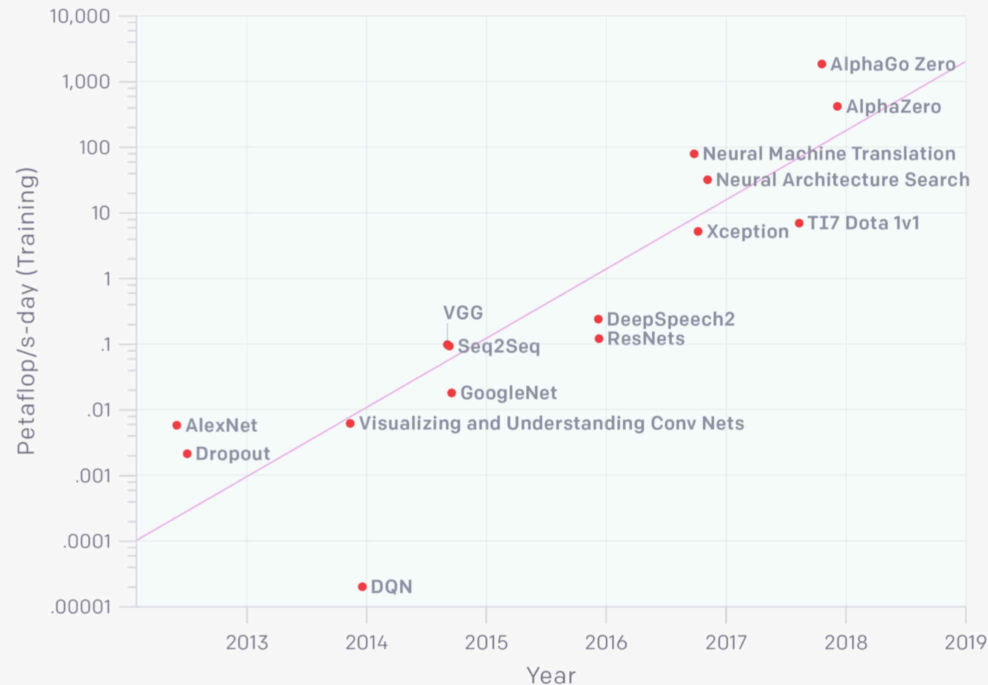
huihuo.zheng@anl.gov

Outline

- Why do we need for distributed / parallel deep learning on HPC
- Distribution schemes: model parallelism vs data parallelism
- Challenges and tips on large batch size data parallel training
- I/O and data management
- Science use cases

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:

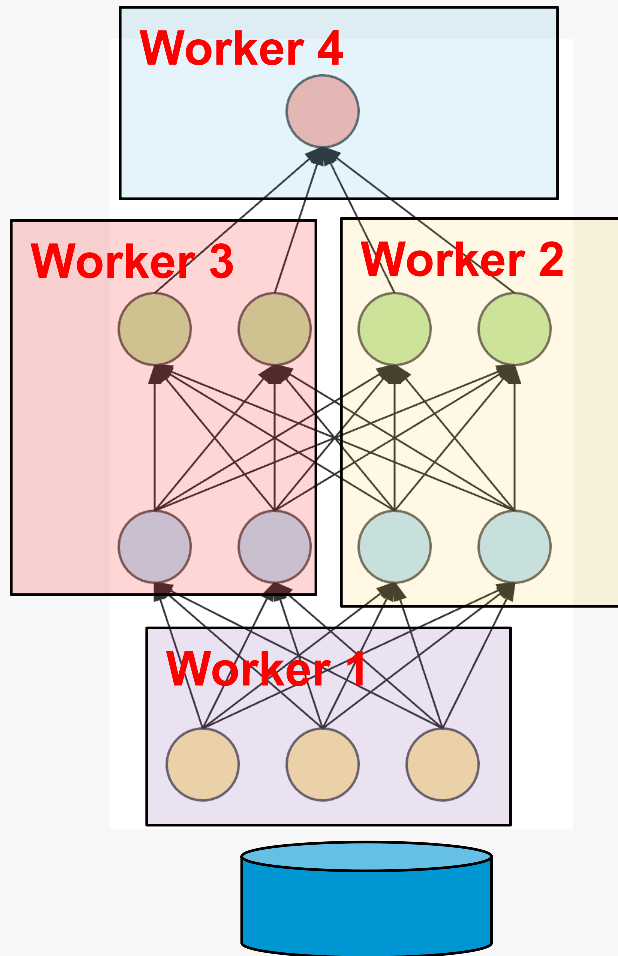
- Before 2012 …
- 2012 – 2014: single to couple GPUs
- 2014 – 2016: 10 – 100 GPUs
- 2016 – 2017: large batch size training, architecture search, special hardware (etc, TPU)

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

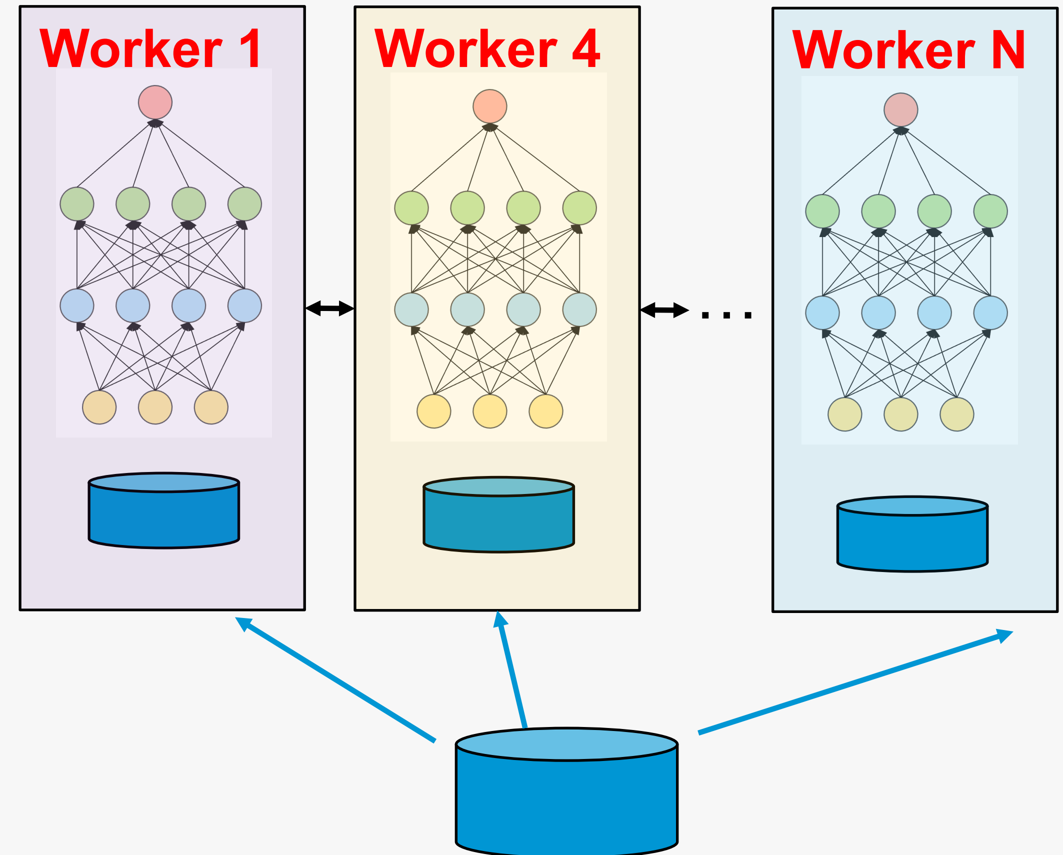
Need for distributed (parallel) training on HPC

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

Parallelization schemes for distributed learning



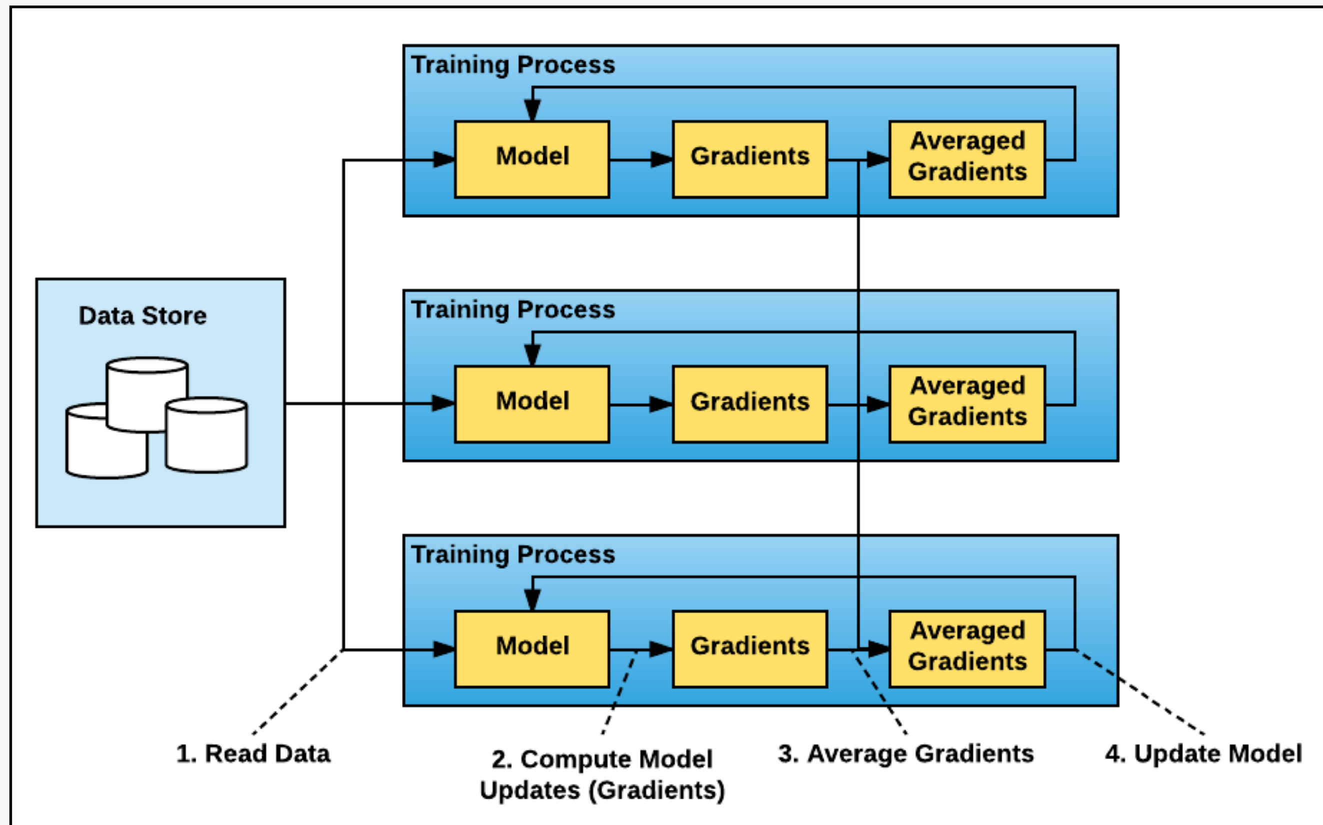
Model parallelism



Data parallelism

Argonne NATIONAL LABORATORY

Model parallelization in Horovod

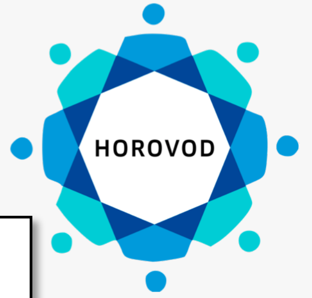


1. Run multiple copies of the model and each copy:
   1) reads a chunk of the data
   2) runs it through the model
   3) computes model updates
2. Average gradients among all the copies
3. Update the model
4. Repeat (from Step 1)

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

Argonne NATIONAL LABORATORY

Deep dive on model parallelism (Horovod)



Minimizing the loss:

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

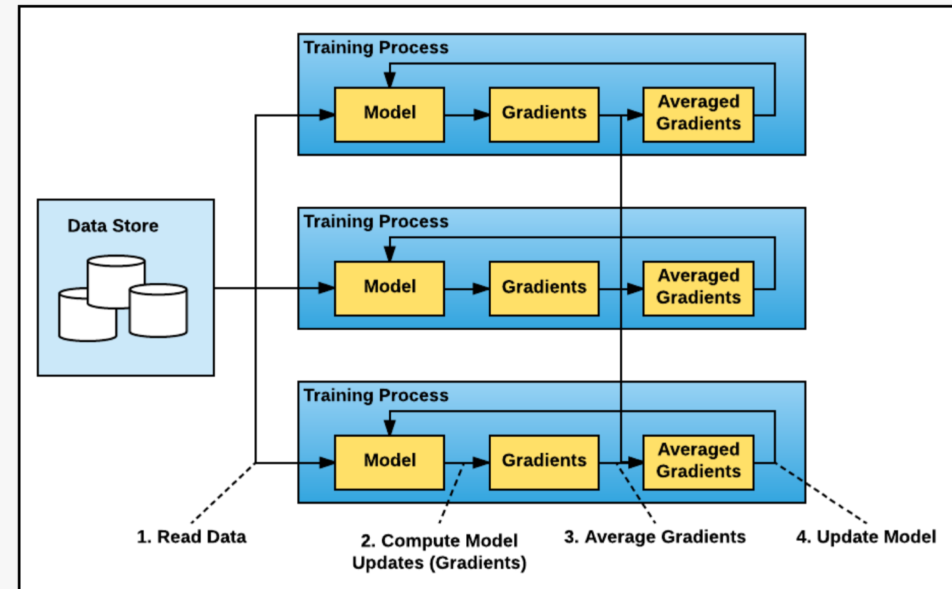
Dataset Weight

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;
- Gradients are averaged at each step (not each epoch)

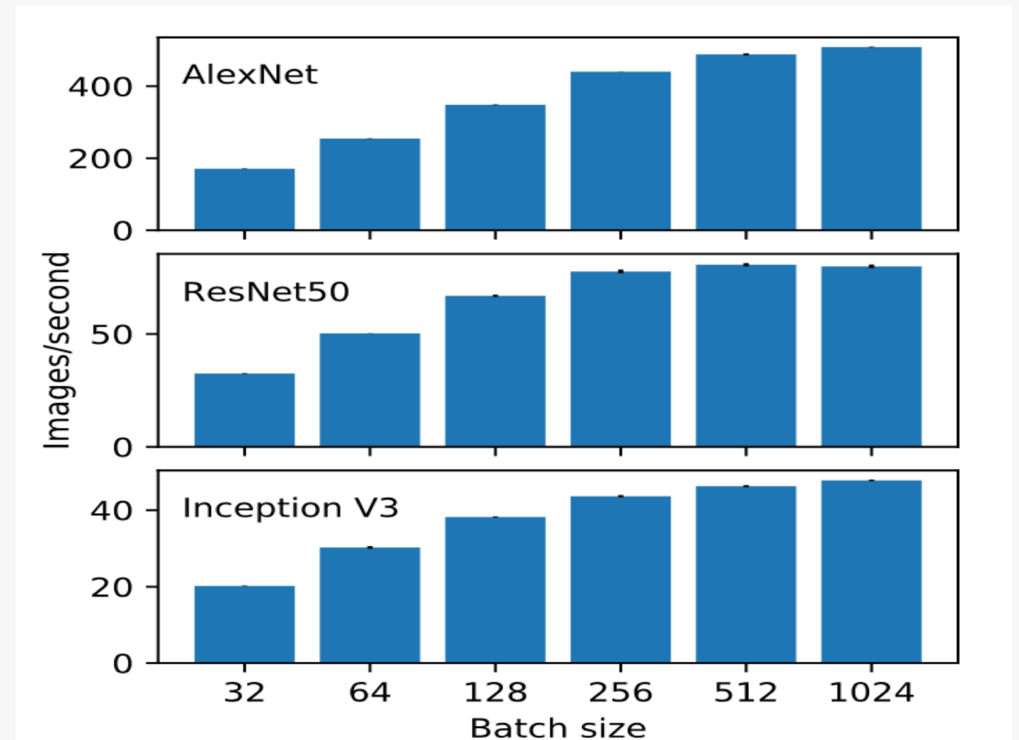
Argonne NATIONAL LABORATORY

Large minibatch training

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

Minibatch

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



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

Argonne NATIONAL LABORATORY

Linear scaling rule

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

- k steps with learning rate η and minibatch size n

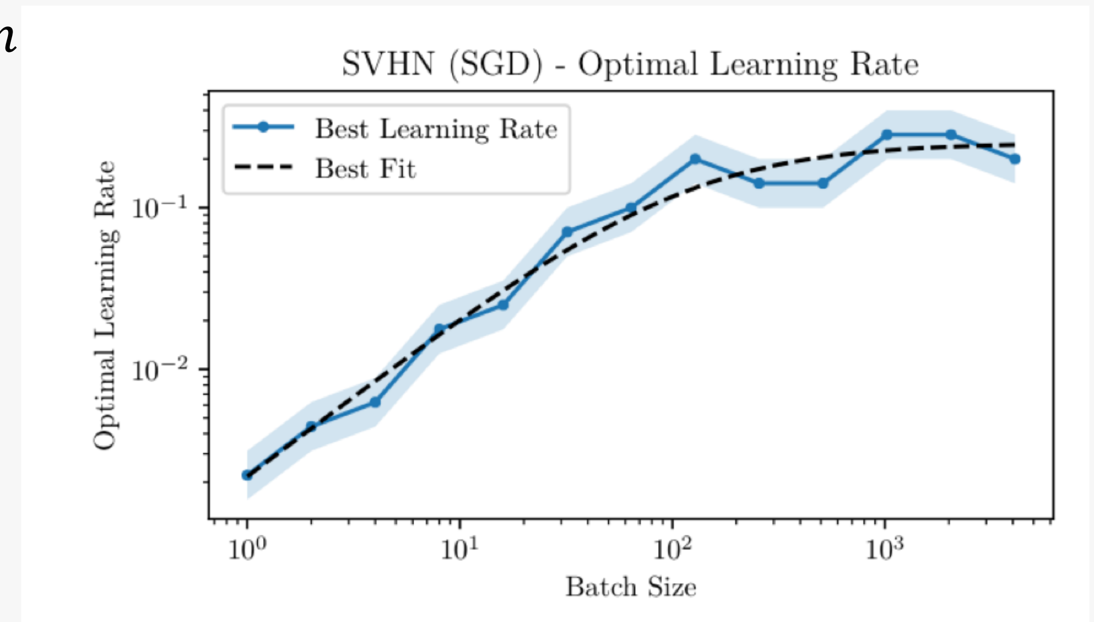
$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_{t+j})$$

- Single step with new learning rate $\hat{\eta}$ and large minibatch $\cup_j B_j$ (batch size kn)

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_t)$$

If $\nabla l(x, \omega_{t+j}) \sim \nabla l(x, \omega_t)$ we have, $\widehat{\omega}_{t+1} \sim \omega_{t+k}$.

Ideally, large batch training with a linear scaled learning rate will reach the similar goal with the same number of epochs (fewer steps per epoch)



The optimal learning for a range of batch sizes, for an SVHN classifier trained with SGD (S. McCandlish, J. Kaplan, D. Amodei, arXiv:1812.06162)

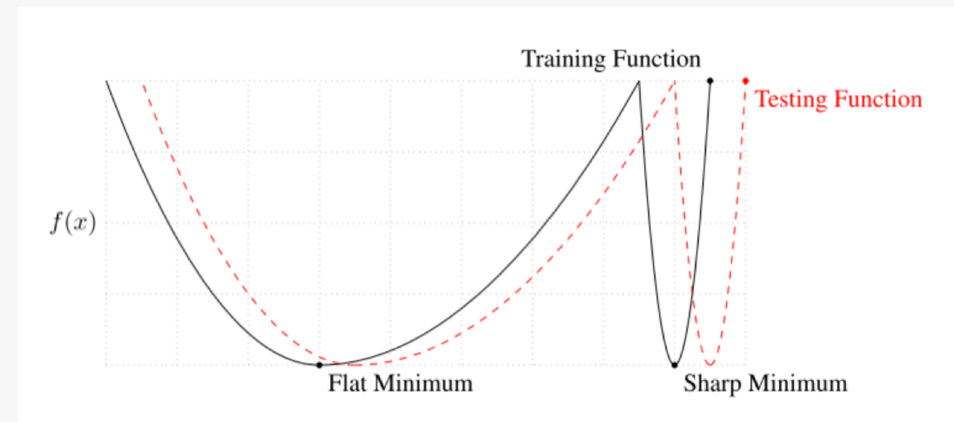
Argonne NATIONAL LABORATORY

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).

Name	Training Accuracy		Testing Accuracy	
	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"

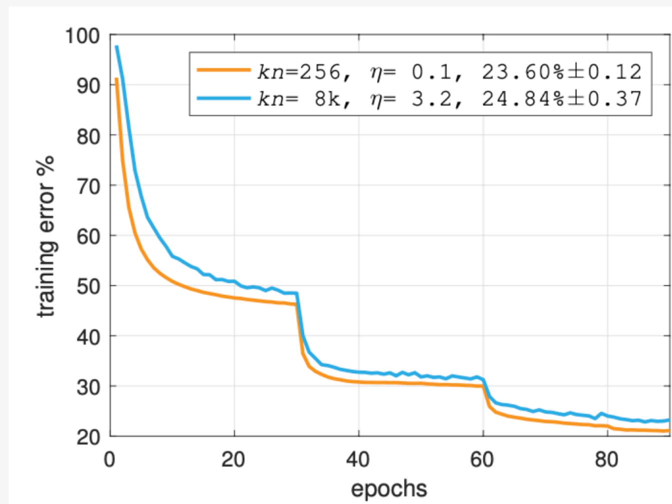
Keskar et al, arXiv:1609.04836

Argonne NATIONAL LABORATORY

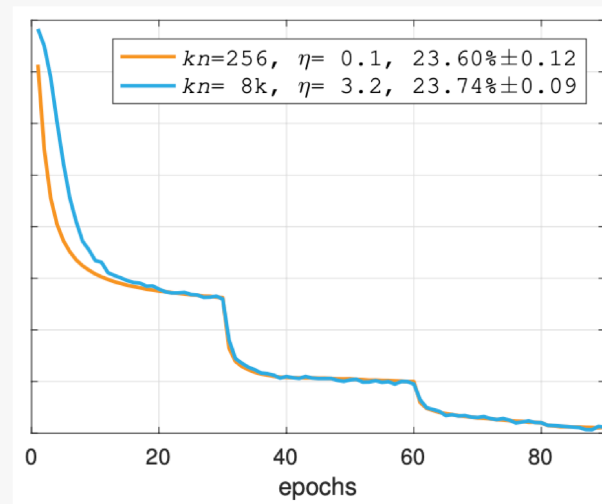
Challenges with large batch training

Solutions: using warm up steps

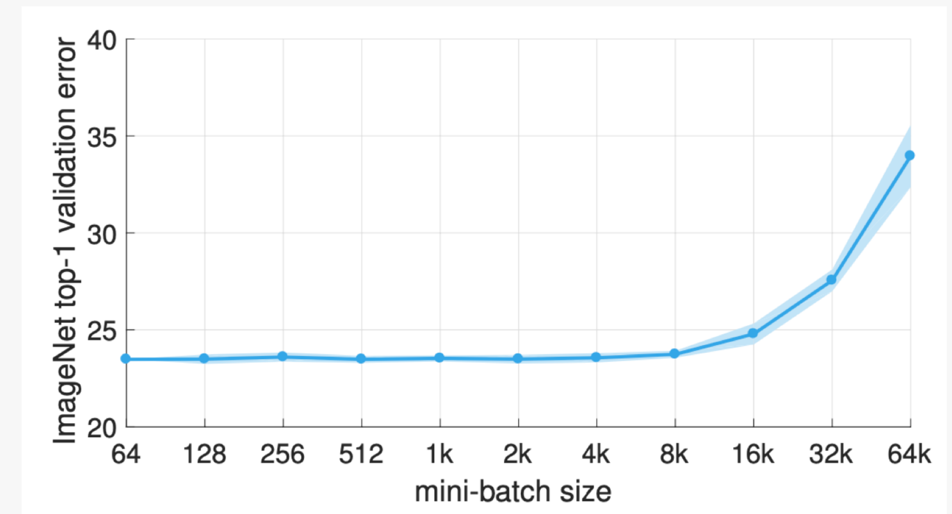
- Using a smaller learning rate at the initial stage of training (couple epochs), and gradually increase to $\hat{\eta} = N\eta$
- Using linear scaling of learning rate ($\hat{\eta} = N\eta$)



No warm up



Gradual warm up

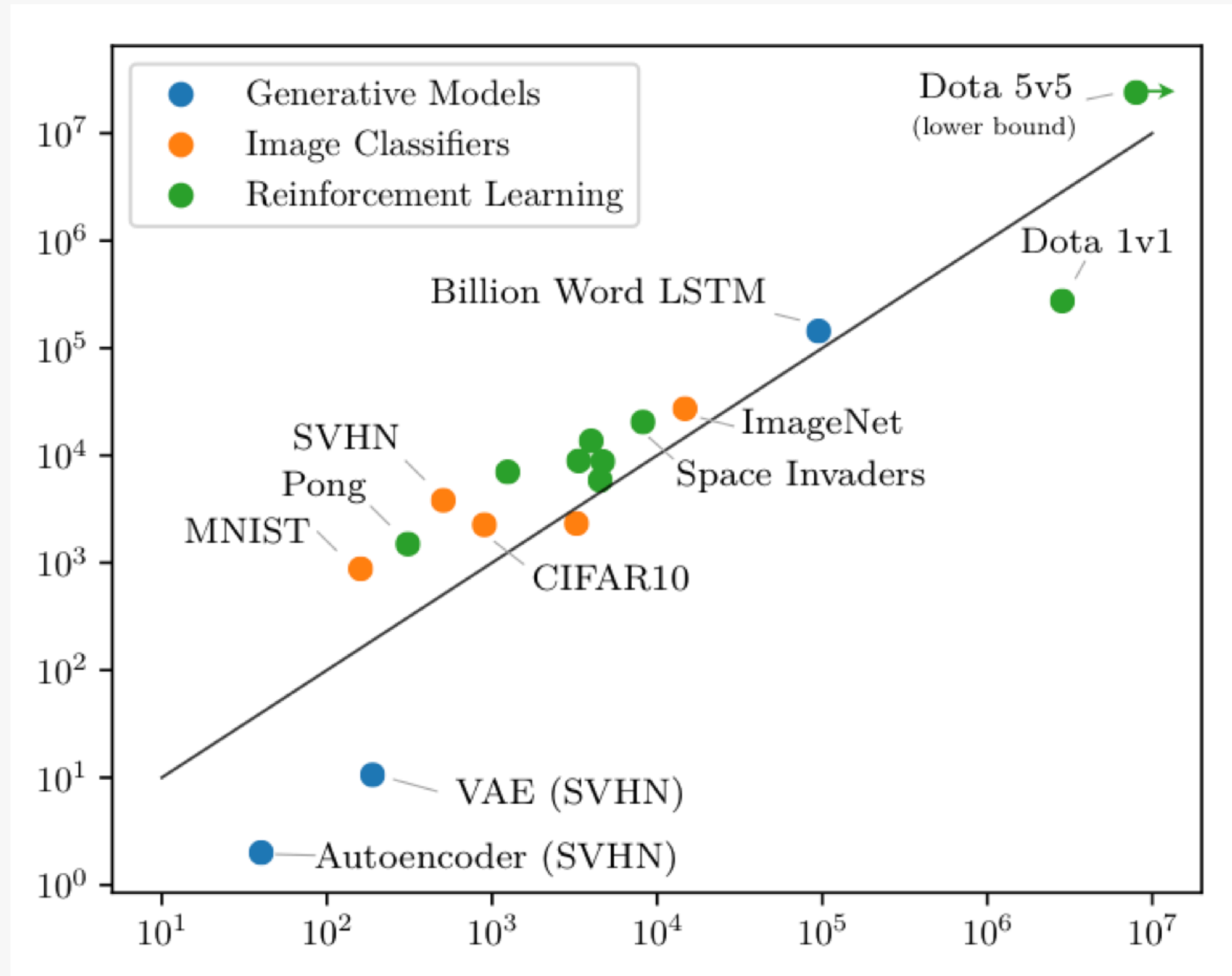


This scheme works up to 8k batch size

P. Goyal et al,arXiv: 1706.02677

Argonne NATIONAL LABORATORY

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

Argonne NATIONAL LABORATORY

Data parallel training with Horovod

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

- Import Horovod modules and initialize horovod
- Wrap optimizer in hvd.DistributedOptimizer
  - Scale the learning rate by number of workers
- Broadcast the weights from worker 0 to all the workers and let worker 0 save check point files
- Divide the dataset and each worker only work on piece of dataset.

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

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())
```

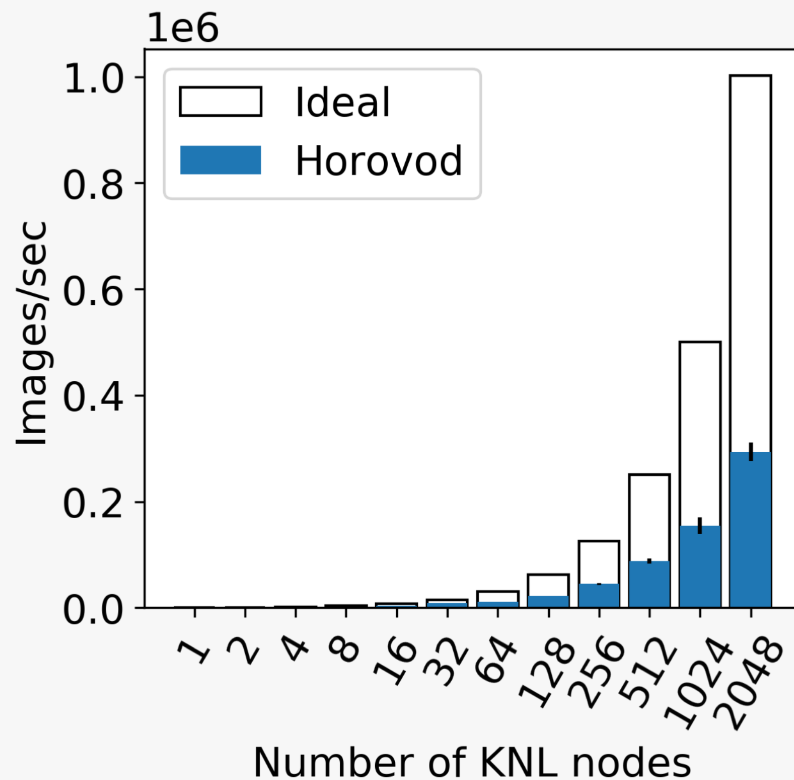
More examples can be found in https://github.com/uber/horovod/blob/master/examples/

Keras with Horovod

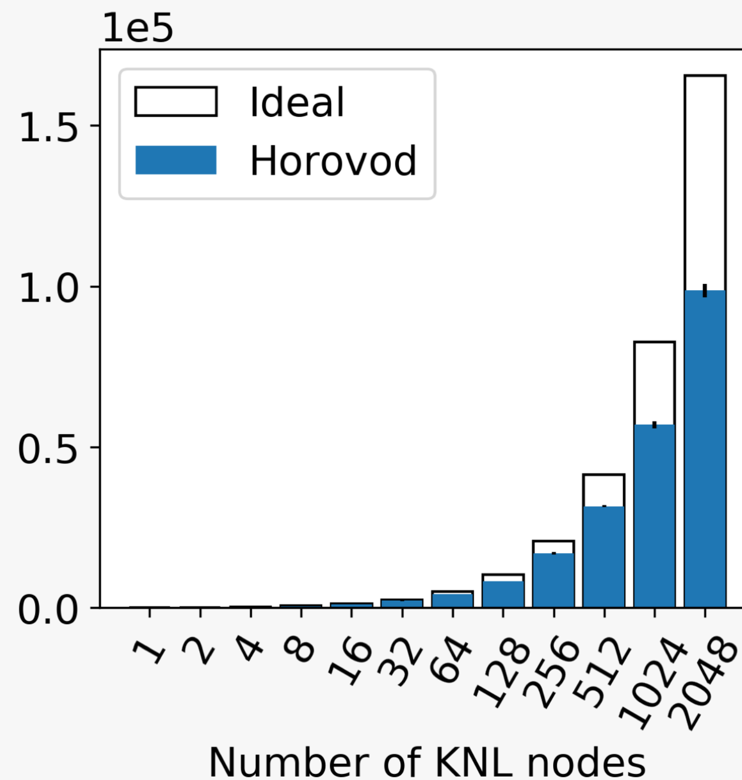
```
import keras
import tensorflow as tf
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)
model.compile(loss=keras.losses.categorical_crossentropy,
              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/

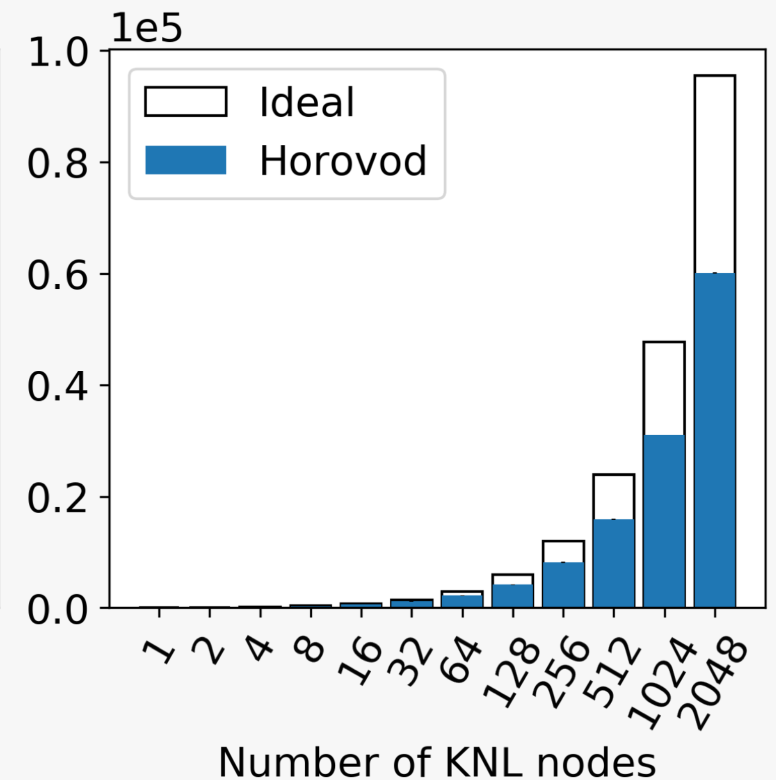
Scaling TensorFlow using Horovod on Theta @ ALCF (Intel Knights Landing): batch size = 512



AlexNet

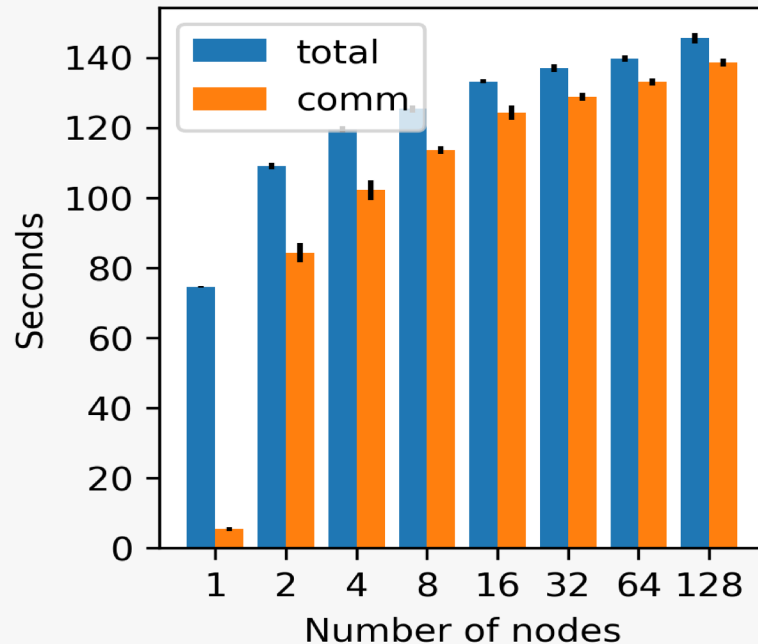


ResNet-50

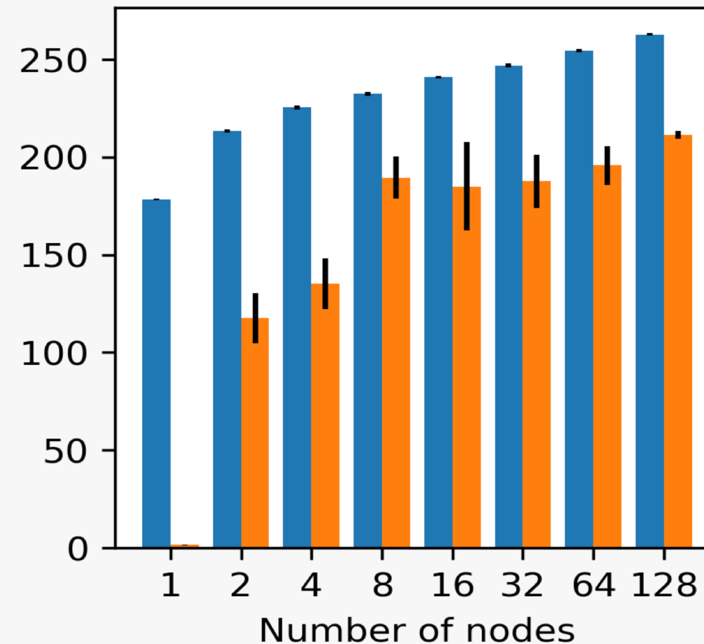


Inception V3

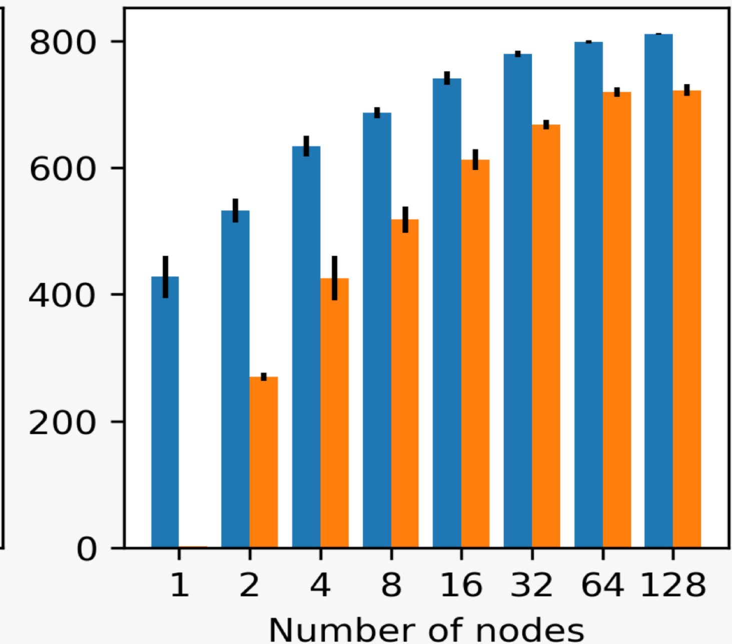
Overlap of communication and compute in Horovod



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
```

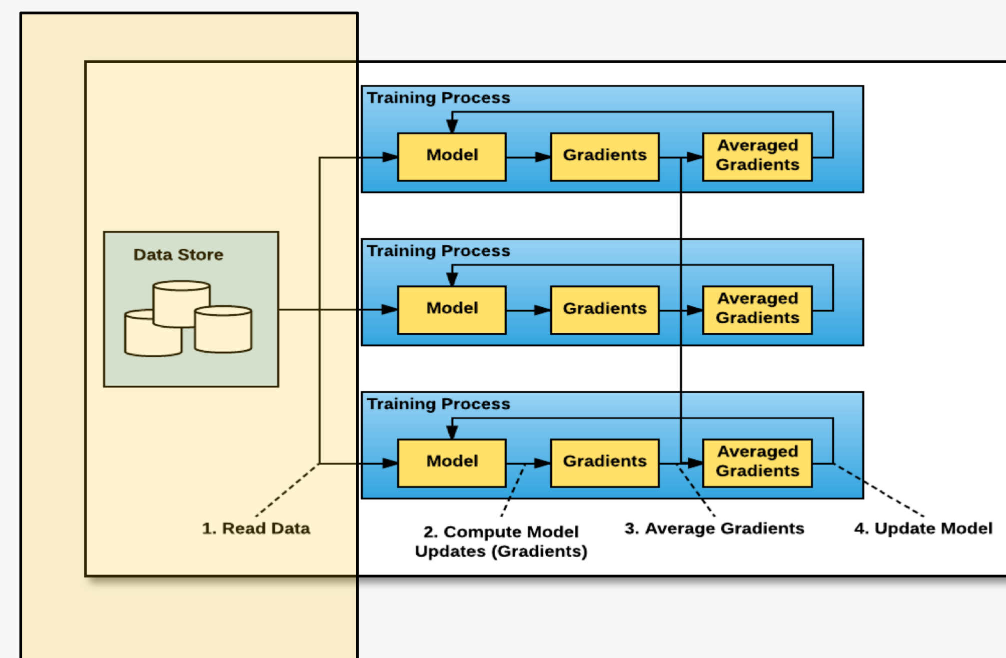
```
MPI_Allreduce          #calls    avg. bytes     time(sec)
                           10        4004.0         1.045
                           21       16384.0         1.269
                           10       32768.0         0.521
                            8     1322752.0         0.263
                            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)

Argonne NATIONAL LABORATORY

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;
- Store the dataset in a reasonable way (avoiding file per sample)
- Prefetch the data (from disk; from host to device)



I/O and data management

Argonne NATIONAL LABORATORY

I/O and data management

File per dataset (stripe Size = 32m, Stripe Count = 48)

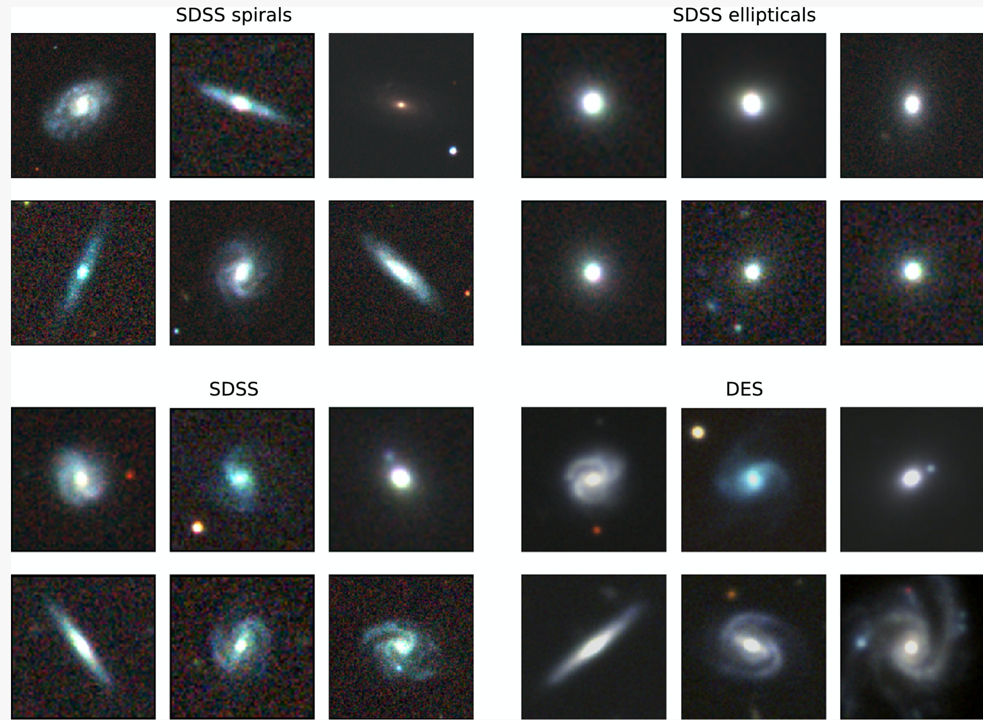
Sergio Servantez

File per batch is the optimal way of setting – scales well and lower overhead from open/ close file.

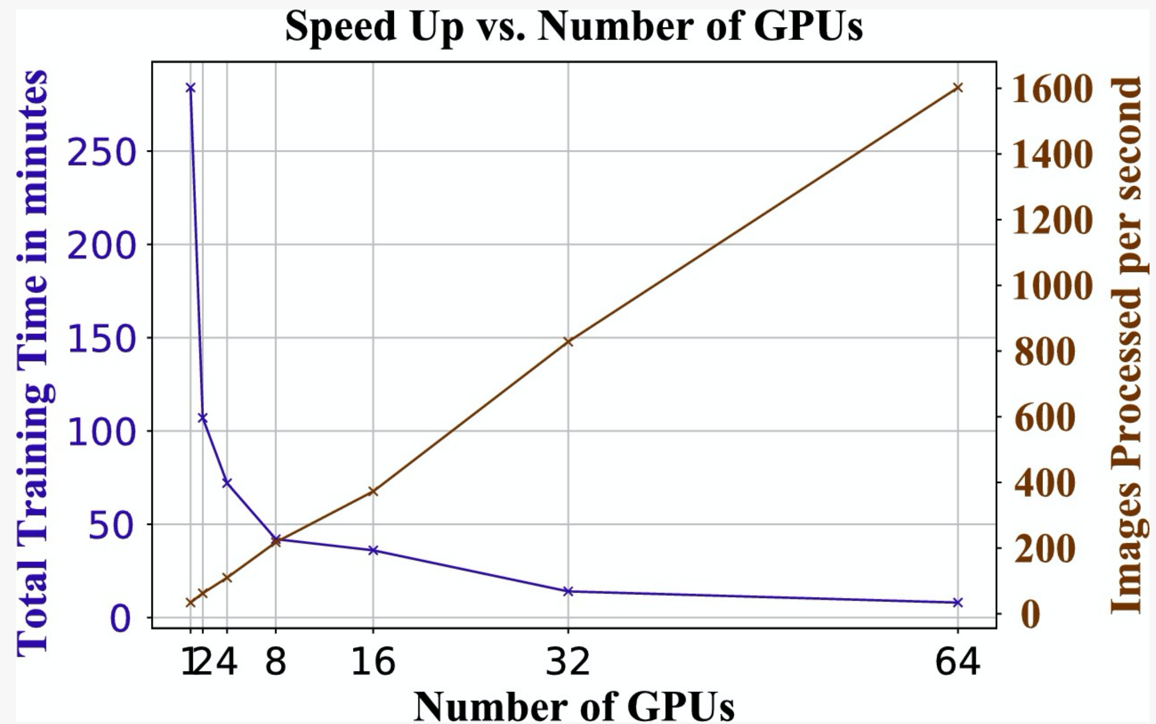
IO benchmarks for ImageNet dataset on Lustre file system on Theta @ ALCF (128 KNL nodes, lustre Stripe Size = 1m and lustre Stripe =1 except the point anointed),

Argonne NATIONAL LABORATORY

Science use case 1 - Galaxy classification using modified Xception model



Galaxy images

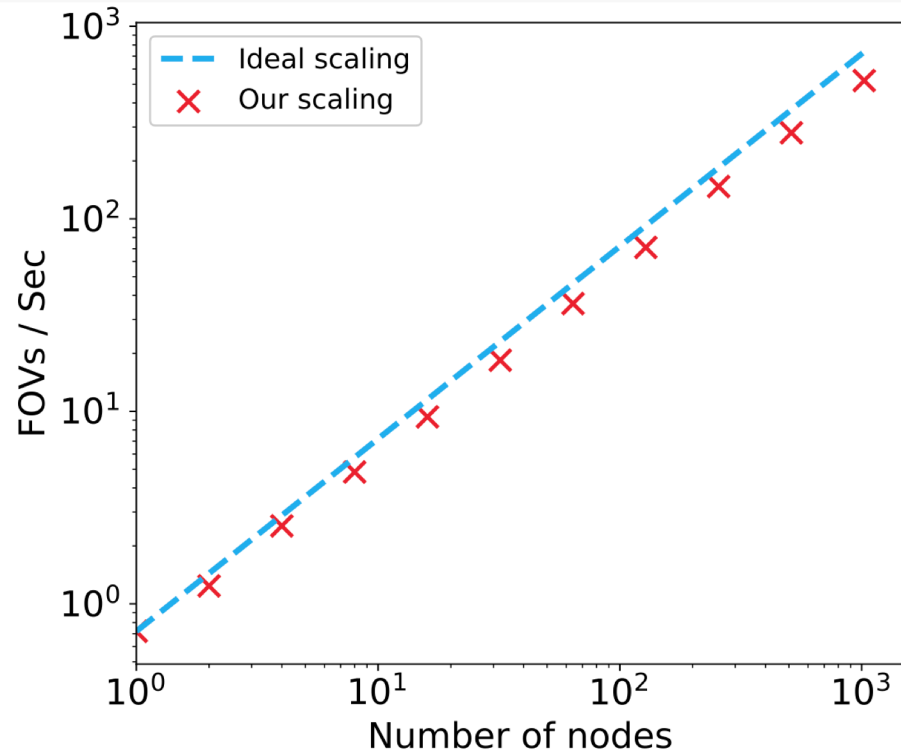


~ 5 Hrs using 1 K80 GPU to 8 mins using 64 K80 GPUs using computing resource from Cooley @ ALCF

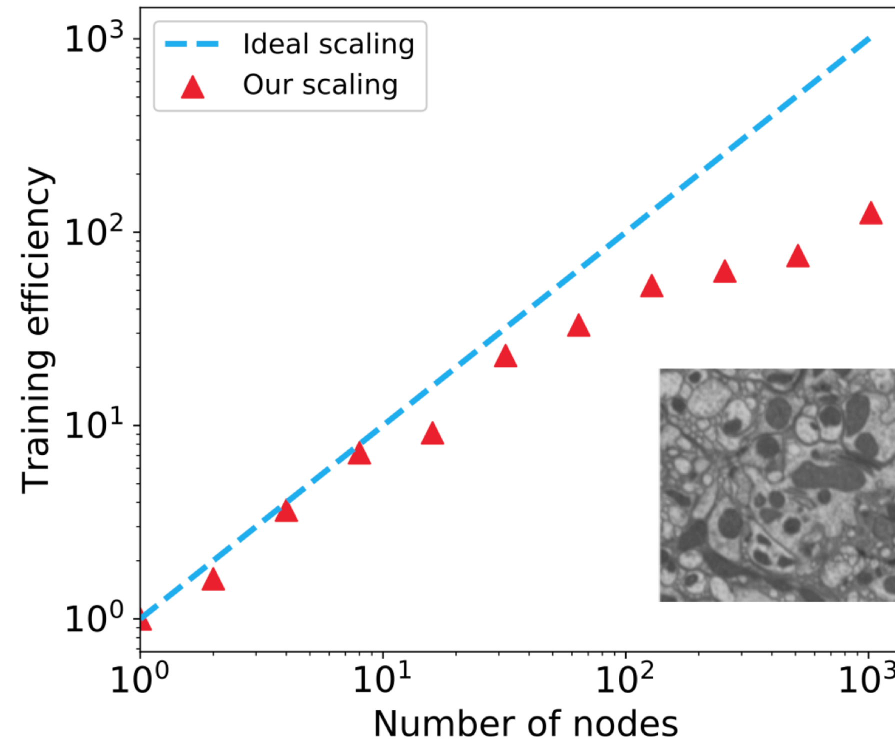
A Khan et al, Physics Letters B, 793, 70-77 (2019)

Argonne NATIONAL LABORATORY

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



Scaling results in terms of throughput

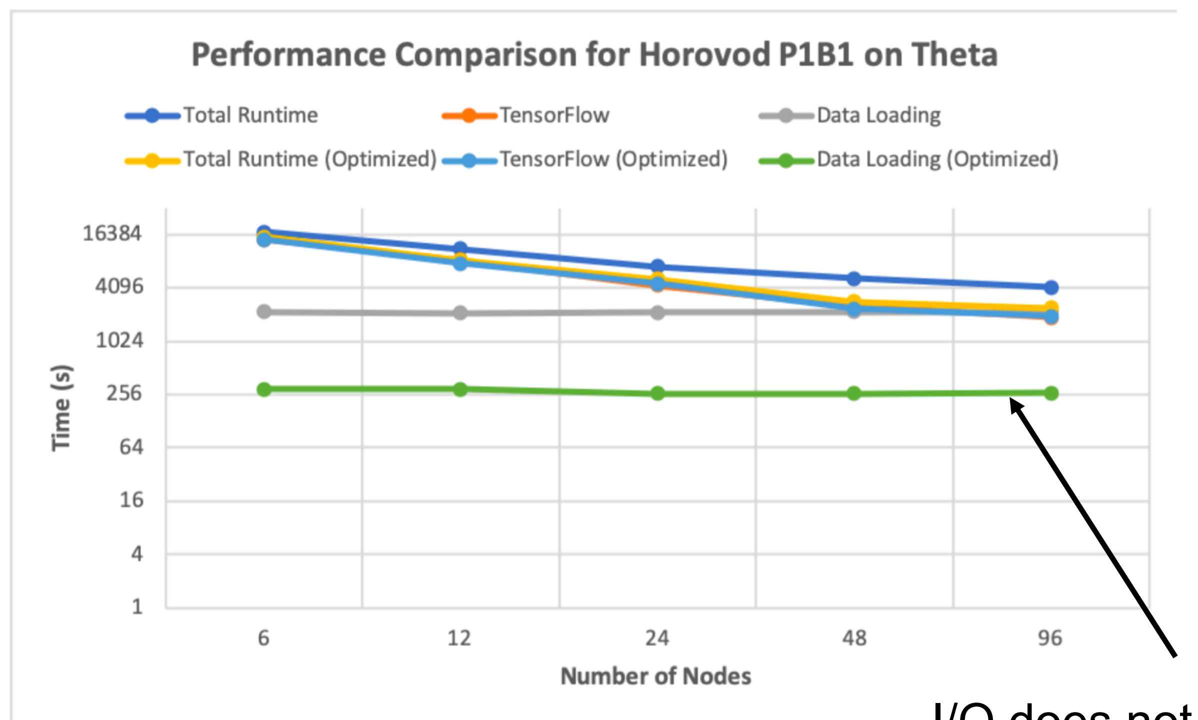


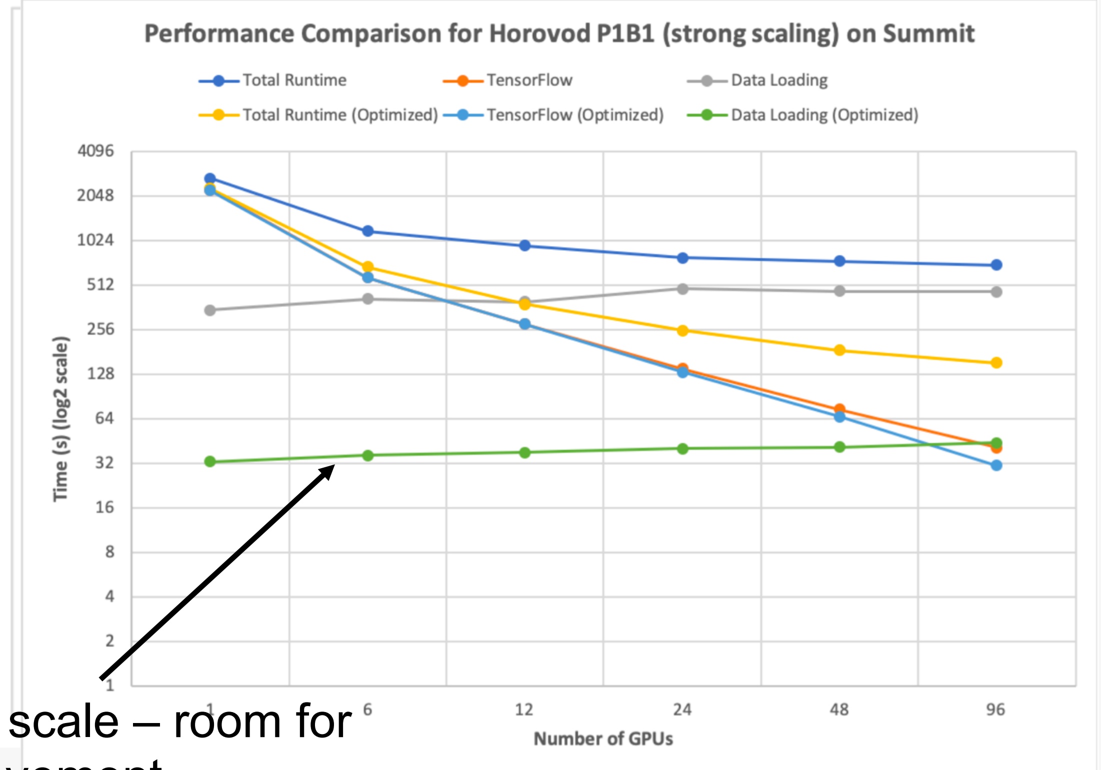
Scaling results in terms of training efficiency (measured by time needed for the training to reach to certain accuracy)

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

Argonne NATIONAL LABORATORY

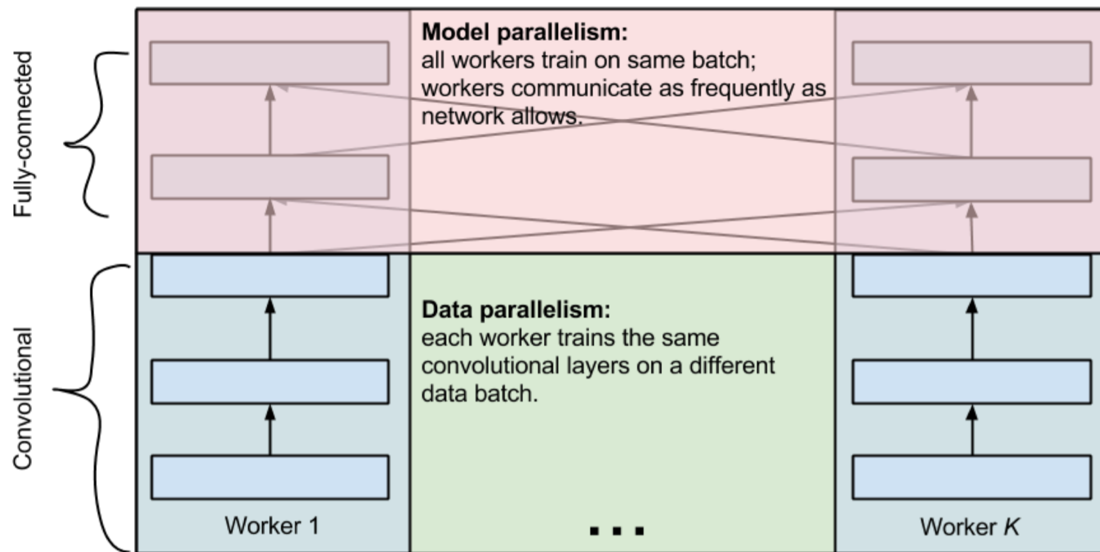
Conclusion

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

Thank you!

huihuo.zheng@anl.gov

Mix data parallelism and model parallelism in CNN

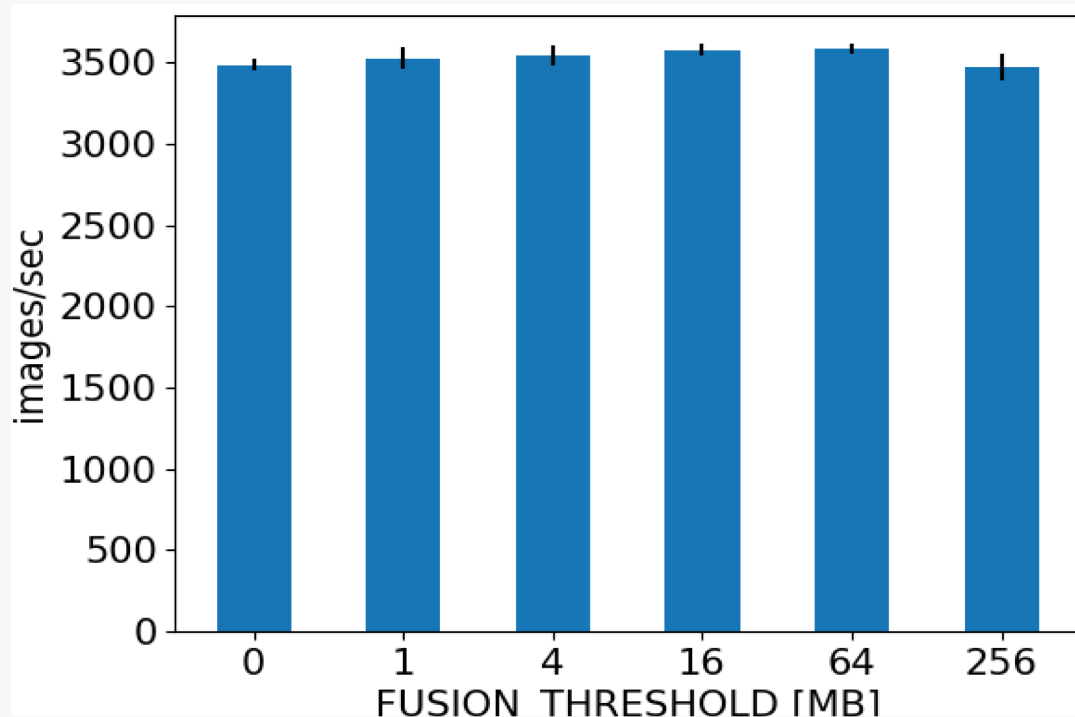


A. Krizhevsky, arXiv:1404.5997 [cs.NE]

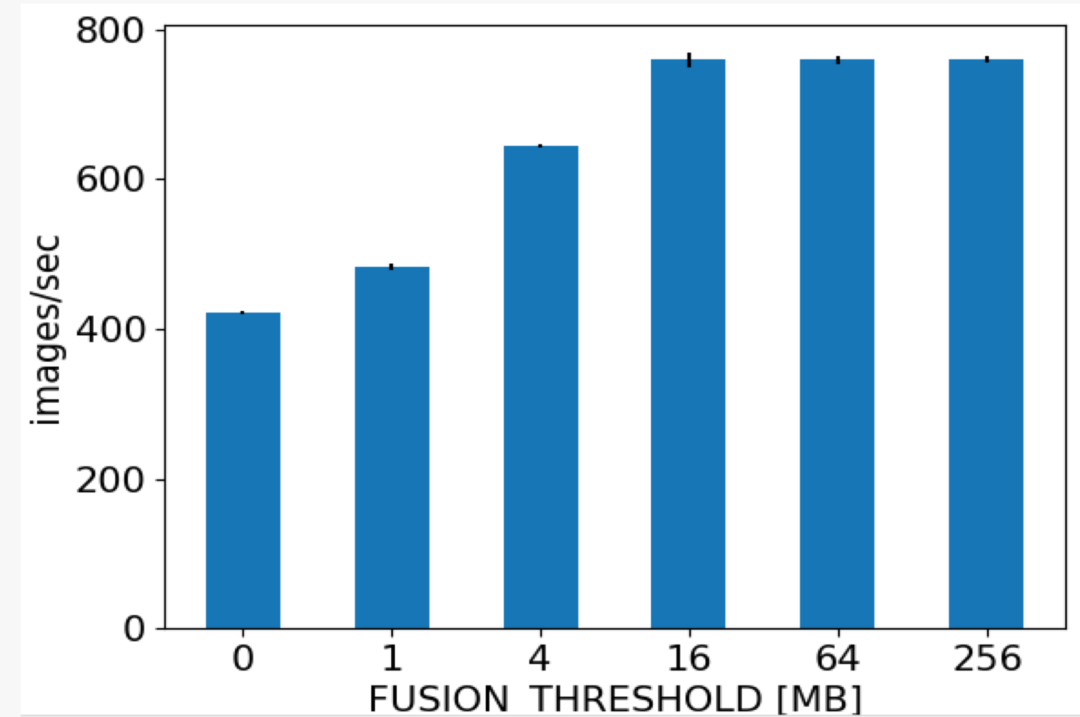
- Convolutional layers cumulatively contain about 90-95% of the computation, about 5% of the parameters, and have large representations.
- Fully-connected layers contain about 5-10% of the computation, about 95% of the parameters, and have small representations.

HOROVOD_FUSION_THRESHOLD (default: 64MB)

Horovod has tensor fusion implemented, which fuses smaller tensor into a big buffer before doing MPI_Allreduce right away.



Alexnet (16 KNL nodes)



Inception3 (16 KNL nodes)

FUSION_THRESHOLD = 64 MB already gets optimal performance.

Argonne NATIONAL LABORATORY

Alexnet (Horovod 0.16.1) on 128 KNL nodes

```
Data for MPI rank 0 of 128:
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                            6617        37476.0          2.174
MPI_Allreduce                         800     15460586.2        168.680
MPI_Gather                           3300            4.0         53.732
MPI_Gatherv                          3300            0.0          0.702
MPI_Allgather                           1            4.0          0.000
-----------------------------------------------------------------
total communication time = 225.288 seconds.
total elapsed time       = 229.604 seconds.
user cpu time            = 5795.724 seconds.
system time              = 260.292 seconds.
max resident set size    = 3428.168 MBytes.
```

```
MPI_Allreduce              #calls     avg. bytes       time(sec)
                               50          256.0           0.056
                              100          896.0           0.191
                              100         1536.0           0.168
                               50         4004.0           0.188
                              100        16384.0           1.421
                               50        92928.0           0.412
                               50      1228800.0           0.541
                              100      3096576.0          26.835
                               50      5308416.0           3.762
                               50     16400384.0           6.612
                               50     67108864.0          30.896
                               50    150994944.0          97.598

MPI_Gather                 #calls     avg. bytes       time(sec)
                             3300            4.0          53.732

MPI_Allgather              #calls     avg. bytes       time(sec)
                                1            4.0           0.000
```

FUSION_THRESHOLD=0

```
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                            7743        31979.6          2.369
MPI_Allreduce                         126     98162452.4        148.776
MPI_Gather                           3863            4.0         57.352
MPI_Gatherv                          3863            0.0          1.006
MPI_Allgather                           1            4.0          0.000
-----------------------------------------------------------------
total communication time = 209.503 seconds.
total elapsed time       = 220.001 seconds.
user cpu time            = 5141.012 seconds.
system time              = 369.980 seconds.
max resident set size    = 5300.309 MBytes.
MPI_Allreduce              #calls     avg. bytes       time(sec)
                               21         4004.0           3.579
                                5        93132.8           0.175
                               31     16405715.2          20.304
                               19     85208441.3          52.643
                               50    204807633.9          72.075

MPI_Gather                 #calls     avg. bytes       time(sec)
                             3863            4.0          57.352

MPI_Allgather              #calls     avg. bytes       time(sec)
                                1            4.0           0.000
```

FUSION_THRESHOLD=256M

# of Allreduce decreases as we increase FUSION_THRESHOLD, and message size increases.