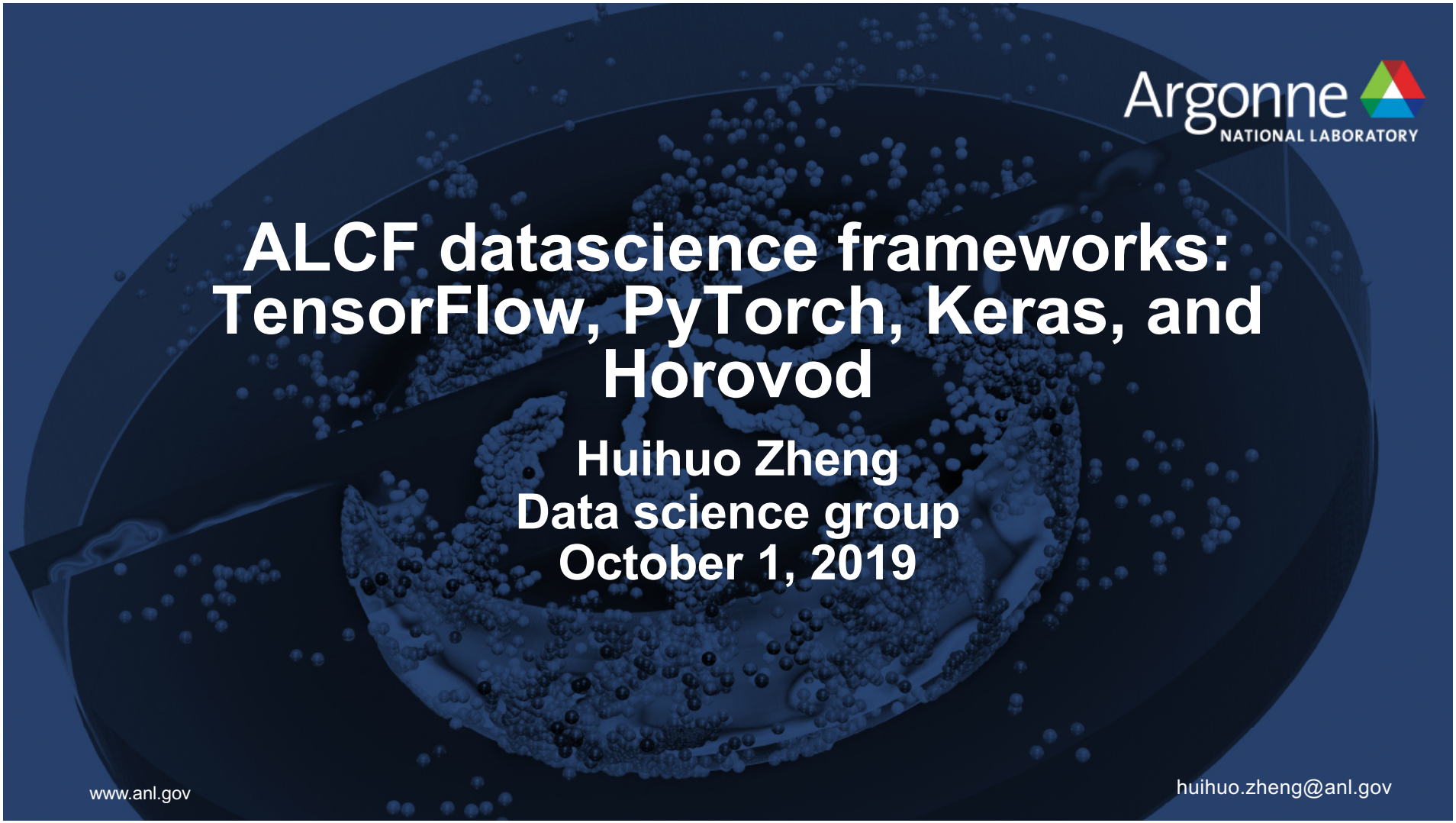


ALCF datascience frameworks: TensorFlow, PyTorch, Keras, and Horovod

Huihuo Zheng
Data science group

October 1, 2019

www.anl.gov

huihuo.zheng@anl.gov

Outline

- Using the datascience modules on Theta
  - How we built TensorFlow, PyTorch, Keras, and Horovod
  - Best practices for using the modules / installing other python packages
- Optimal setup for efficient single node performance (TensorFlow)
- Data parallel training with Horovod
- Visualization through Tensorboard remotely
- Profiling: timeline trace, Vtune, MKL-DNN verbose output

“datascience” modules on Theta

```
[xxx@thetalogin4 ~]$ module avail datascience

--------------- /soft/environment/modules/modulefiles ---------------
datascience/horovod-0.16.1       datascience/tensorflow-1.13
datascience/keras-2.3.0          datascience/tensorflow-1.14
datascience/pytorch-1.1          datascience/tensorflow-2.0-rc2
datascience/tensorboard          datascience/mpi4py-3.0.2
Datascience/h5py-2.9.0
```

- Compiled using GCC/8.2.0 with “–g” flag and AVX512, specifically optimized for KNL
- Based on intelpython 3.6, not compatible with other python modules, such as miniconda, cray-python, alcf-python (we still have old datascience/intelpython35/* modules available; but we will migrate to intelpython36)
- PyTorch and TensorFlow are linked to MKL and MKL-DNN
- Horovod, mpi4py, and h5py are linked to Cray MPI
- H5py is with parallel HDF5 support;

All the libraries are dynamically linked, be careful of your LD_LIBRARY_PATH, PYTHONPATH.

How to use the “datascience” modules

- *module load datascience/tensorflow-1.14* (horovod and intelpython36 are loaded automatically)
- The packages do NOT run directly on login nodes or mom nodes.

```
>>> import tensorflow as tf
2018-09-23 20:08:54.289480: F tensorflow/core/platform/cpu_feature_guard.cc:37]
The TensorFlow library was compiled to use AVX512F instructions, but these aren'
t available on your machine.
Aborted (core dumped)
```

Illegal instruction!

- Run with qsub script.sh, or on mom node interactively through aprun.

```
#!/bin/bash
#COBALT -A project
#COBALT -n node
#COBALT -q default --attrs mcdram=cache:numa=quad
module load datascience/tensorflow-1.14 datascience/keras-2.3.0
aprun -n nproc -N nproc_per_node -cc depth -j 2 python script
```

- Download/transfer the datasets to Theta first; the compute nodes do not have access to external internet.

How to use the “datascience” modules

- If your applications need other custom python packages**, pip install the package** to a separate directory and add the path to your PYTHONPATH:

```
> module load intelpython36 gcc/8.2.0
> pip install package_name –target=/path_to_install
> export PYTHONPATH=/path_to_install/:$PYTHONPATH
```

- Or you could try to **build your own package as follows:**

```
> module load intelpython36 gcc/8.2.0
> python setup.py build
> python setup.py install --prefix=/path_to_install/
> export PYTHONPATH=/path_to_install/lib/python3.6/site-packages/:$PYTHONPATH
```

Remember to add “export PYTHONPATH=extra_python_paths:$PYTHONPATH” **in your submission scripts.**

- You don’t need to use virtual environment. If you use that, be careful not to override the TensorFlow, PyTorch and Horovod packages.

How to use the “datascience modules”

- If you use mpi4py or h5py, load other modules first, and load h5py/mpi4py **AT THE END** to override the h5py and mpi4py packages in other modules

```
module load intelpython36
…
module load XXX
module load h5py
```

- Avoid installing packages to .local/lib/python3.6/site-packages (never do *pip install XXX --user*)
- If you want to link python functions to your simulation code, remember to use dynamic linking when compiling your application by adding ”-dynamic” to the compiler flag; by default, Theta uses static linking.
- Let us know if you encounter any issues support@alcf.anl.gov.

Optimal setups for best performance on KNL

TensorFlow CNN benchmarks
https://github.com/tensorflow/benchmarks
ImageNet models: AlexNet, ResNet50, Inception V3

H. Zheng, E. Jennings, W. Scullin, V. Mrovozov, and V. Vishwanath, *Performance evaluation and analysis of TensorFlow on Cray XC40*, submitted to SC19 Deep Learning on HPC workshop

TensorFlow internal threading setup (For CPUs)

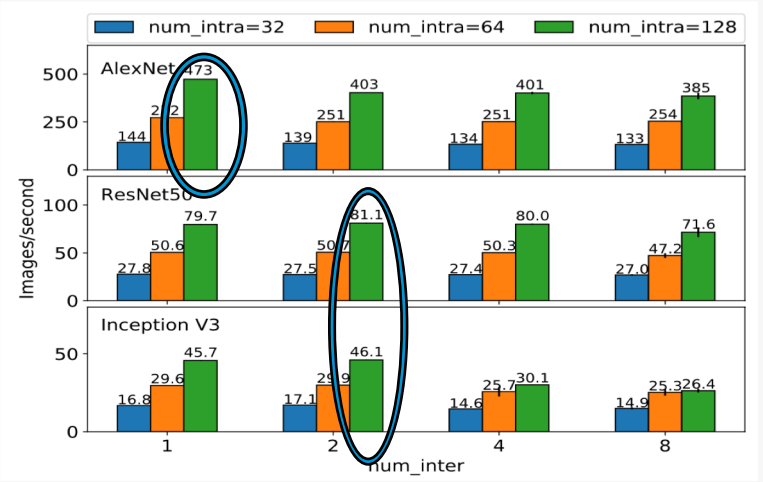
- **intra_op_parallelism_threads**: this sets the number of threads available in the pool. The thread pool is divided to different thread teams at runtime for executing different operations concurrently;
- **inter_op_parallelism_threads**: this sets the maximum number of thread teams which perform different TensorFlow operations concurrently.

Threading setup in the python script

```
config = tf.ConfigProto()
config.intra_op_parallelism_threads = num_intra
config.inter_op_parallelism_threads = num_inter
tf.Session(config=config)
```

Optimal setup on Theta based on benchmarks

- inter_op_parallelism_threads = 1, 2
- Intra_op_parallelism_threads = OMP_NUM_THREADS
- Use aprun -e OMP_NUM_THREADS=.. to setup threads

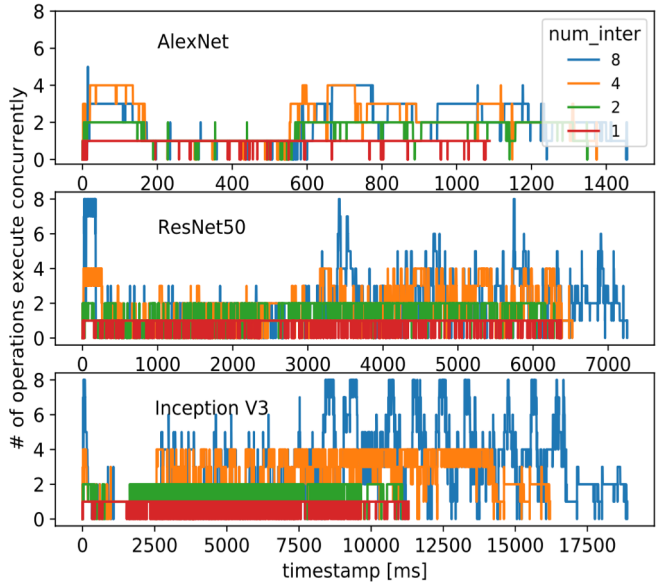


Training throughput of AlexNet, ResNet50 and Inception V3 for different inter and intra threading setups (batch size = 512). In all the cases, we set OMP NUM THREADS = *num intra*.

TensorFlow internal threading setup

Timeline tracing of a single training steps

- Blocks of different colors correspond to execution of different operations.
- Different rows correspond to operations executed on different thread teams.
- At specific time slice, the number of operations executed concurrently is less than *num_inter*



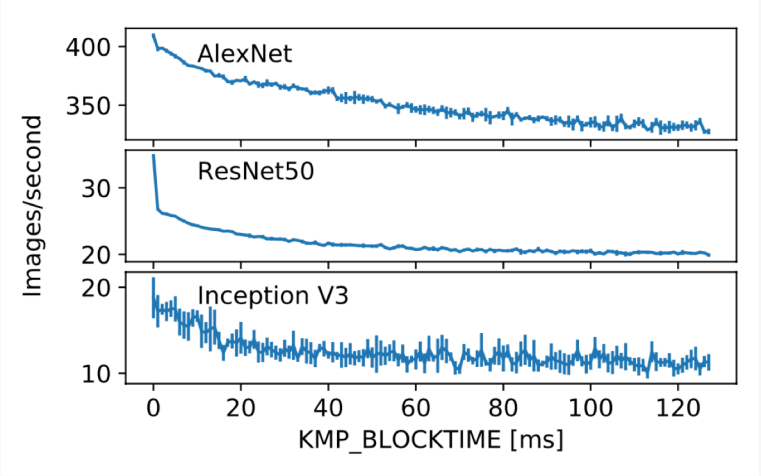
Number of operators executed at different timestamps at a training step for AlexNet, ResNet50, and Inception V3.

Argonne NATIONAL LABORATORY

OpenMP environmental variables

KMP_BLOCKTIME

KMP_BLOCKTIME sets the time (in ms) that a thread shall wait before sleeping, after completing the execution of a parallel region.



The default value in MKL is 200ms, which was not optimal in our testing. The optimal is 0.
You could set KMP_BLOCKTIME in two ways:

- Parse through aprun:
  aprun ... -e KMP_BLOCKTIME=0 …
- Set inside your python script:
  os.environ['KMP_BLOCKTIME']=0

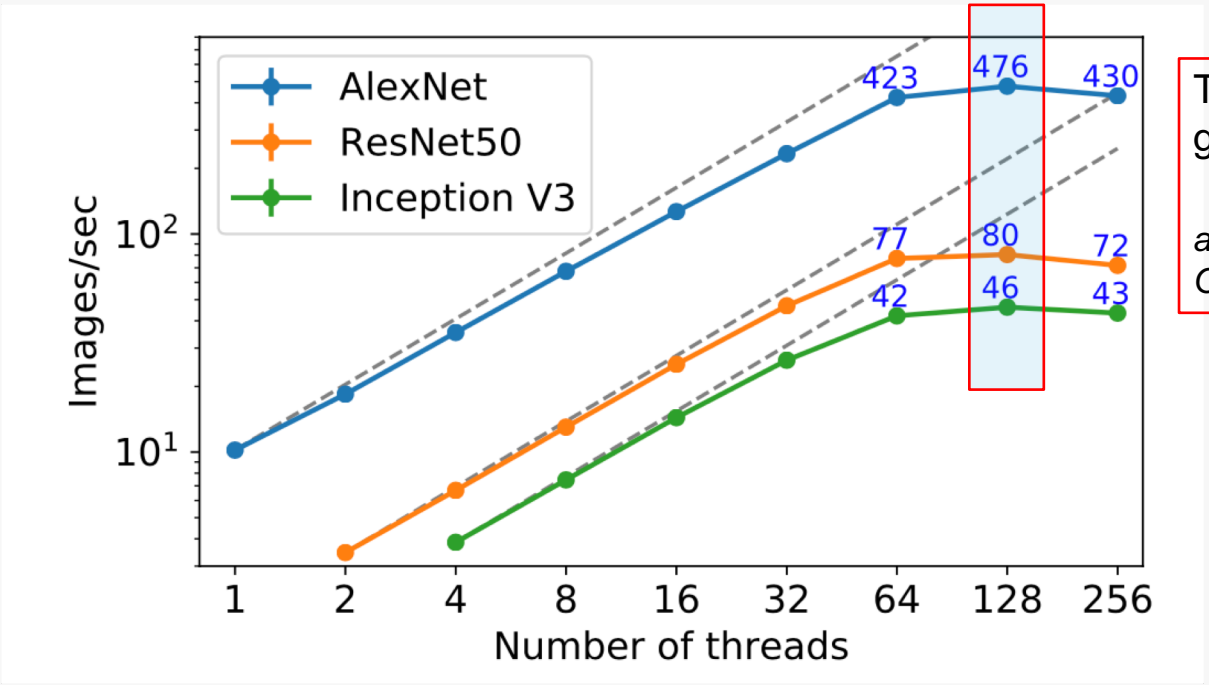
KMP_AFFINITY

Optimal value: granularity=fine,verbose,compact,1,0

Always specify "aprun … -cc depth"

Argonne NATIONAL LABORATORY

OpenMP thread scaling

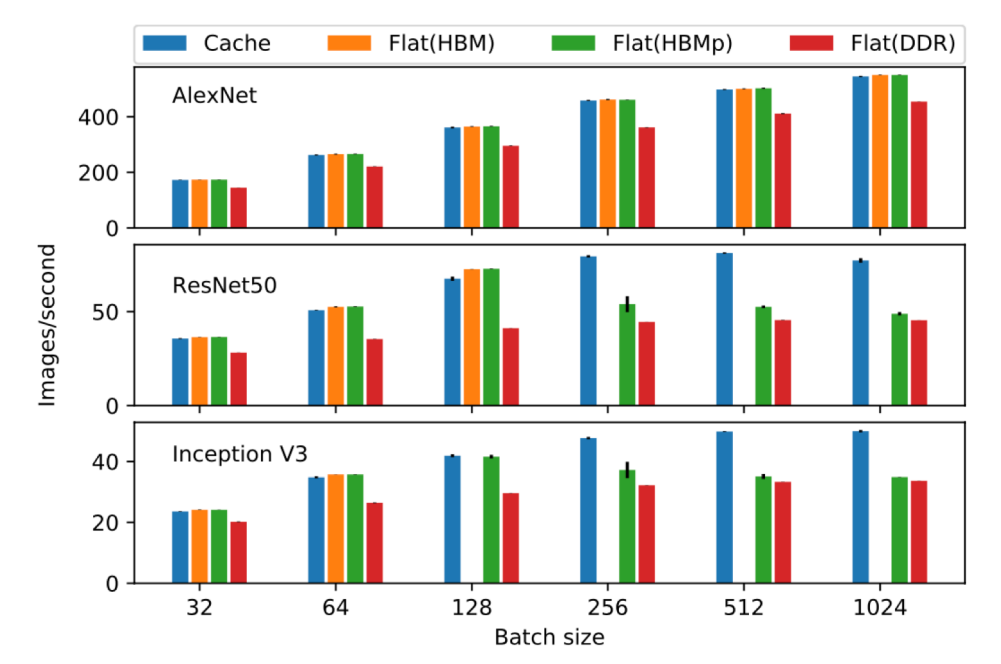


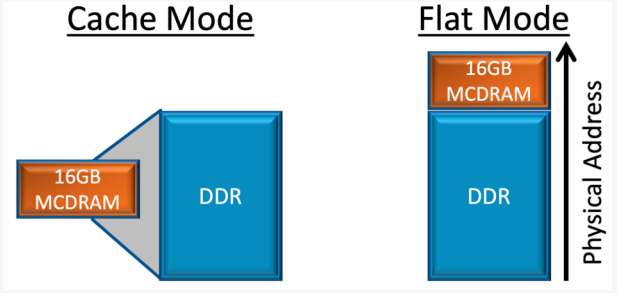
Two hyper threads per core gives optimal performance.

aprun … -j 2 –e OMP_NUM_THREADS=128 …

Throughput scaling of TensorFlow on a single node through OpenMP. For the cases of 1-64 threads, one thread per core was set; for the cases of 128 and 256 threads, 2 and 4 hyper threads per core were set respectively. Dash curves indicate ideal scaling.

Memory mode and batch size





Cache – MCDRAM serves as a cache to DDR4
Flat(HBM) – memory is allocated in MCDRAM
Flat(DDR) – memory is allocated in in DDR4
Flat(HBMp) – memory is allocated in MCDRAM first and then spread to DDR4

- Always use **Cache mode** to take advantage of high bandwidth of MCDRAM (16 GB) and high memory capacity of DDR4 (192 GB).

Training throughput on different memory modes for ImageNet models: AlexNet, ResNet50, and Inception V3. Some Flat(HBM) bars are omitted as the memory footprint is larger than 16GB - the capacity of MCDRAM.

Flat profiling from timeline trace

AexNet

Operation	Time[ms]	Percent.	# calls
_MklConv2DBackpropFilterWithBias	246.04	22.74	5
_MklConv2DWithBias	197.31	18.23	5
_MklConv2DBackpropInput	181.26	16.75	4
Transpose	161.47	14.92	1
MatMul	107.57	9.94	9
BiasAddGrad	86.57	8.00	3
_MklReluGrad	16.11	1.49	7
_MklMaxPoolGrad	15.21	1.41	3
_MklRelu	15.05	1.39	7
_MklAddN	14.91	1.38	16
_Mkl total	696.82	64.40	81
Total	1082.06	100	291

Inception V3

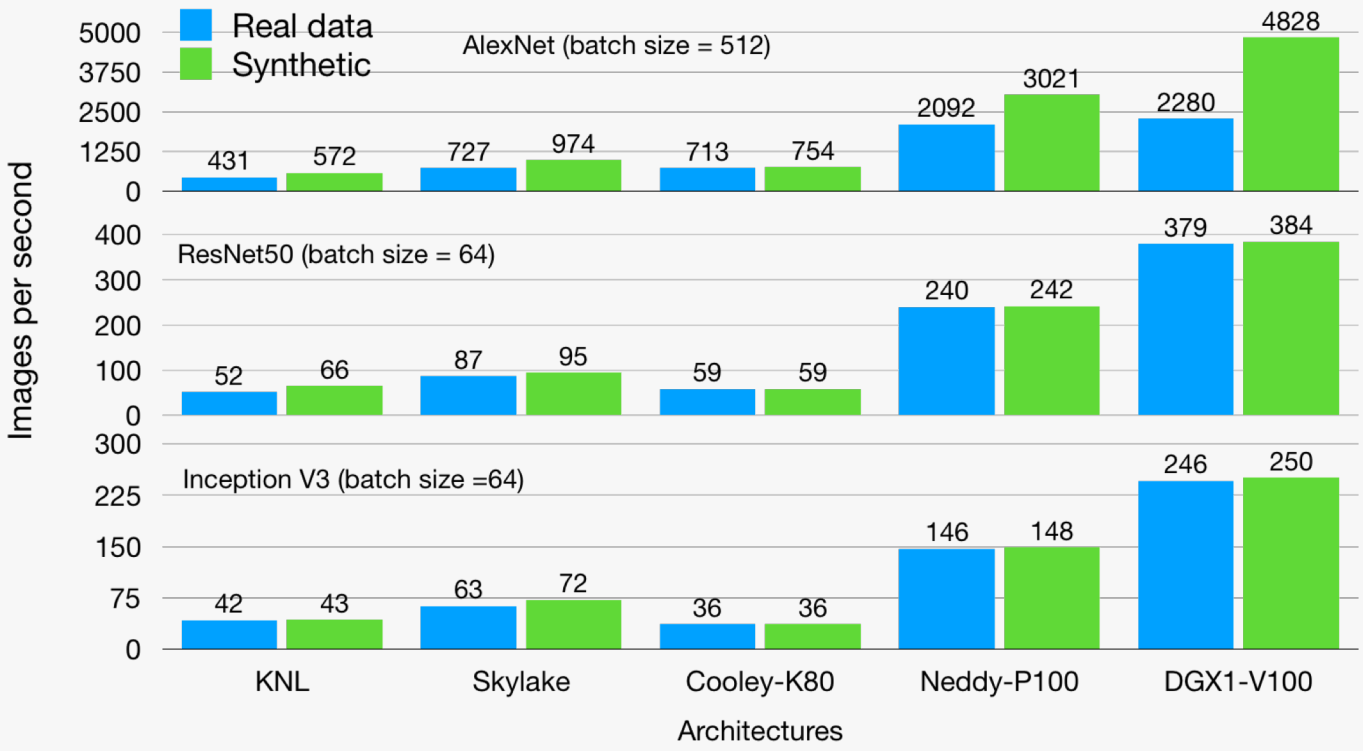
Operation	Time[ms]	Percent.	# calls
_MklConv2D	2356.72	21.05	94
_MklConv2DBackpropFilter	2195.22	19.61	94
_MklConv2DBackpropInput	1787.86	15.97	93
_MklReluGrad	1674.00	14.95	94
_MklFusedBatchNormGrad	718.60	6.42	94
_MklFusedBatchNorm	541.82	4.84	94
Transpose	320.75	2.86	1
_MklMaxPoolGrad	281.39	2.51	5
_MklToTf	243.69	2.18	422
_MklRelu	183.90	1.64	94
_MklAddN	141.66	1.27	194
Slice	131.05	1.17	46
_MklConcatV2	122.35	1.09	11
_Mkl total	10500.12	93.79	2066
Total	11195.56	100	5303

ResNet50

Operation	Time[ms]	Percent.	# calls
_MklConv2DBackpropFilter	1443.36	22.86	53
_MklConv2DBackpropInput	1132.79	17.94	52
_MklConv2D	1130.47	17.90	53
_MklFusedBatchNormGrad	657.15	10.41	53
_MklFusedBatchNorm	520.93	8.25	53
_MklAdd	276.49	4.38	16
_MklReluGrad	271.10	4.29	49
_MklAddN	233.61	3.70	177
Tile	193.32	3.06	1
_MklRelu	149.87	2.37	49
Transpose	131.17	2.08	1
_Mkl total	5872.48	92.99	1267
Total	6314.84	100	3398

- A large portion of the execution time spent on the two math libraries, MKL-DNN and Eigen.
- Mkl* operations take 64%, 94% and 93% of the total time in AlexNet, ResNet50 and Inception V3 respectively.

Performance of CPUs and GPUs for deep learning



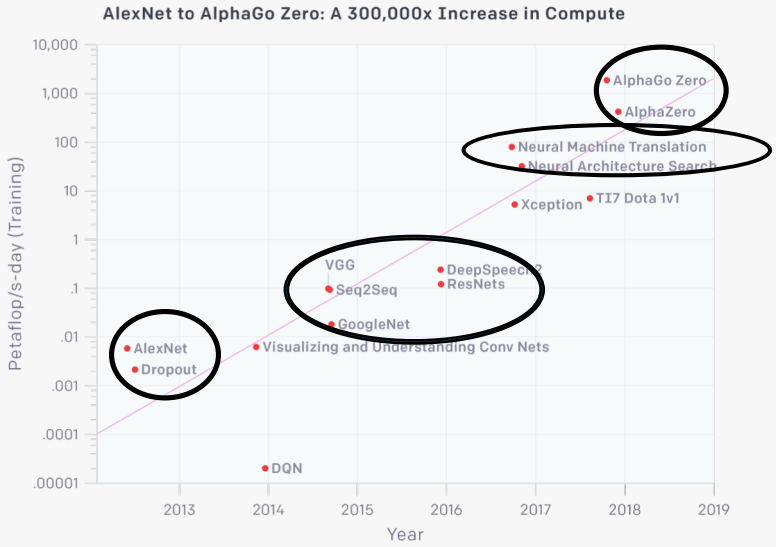
The performance of KNL is about 1/10 – 1/5 of the performance of V100.

Data parallel training through Horovod

Examples
/projects/SDL_Workshop/DeepLearningFrameworks

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:

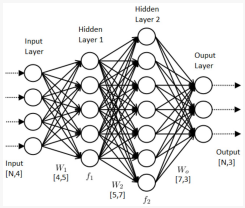
- Before 2012 …
- 2012 – 2014: single to couple GPUs
- 2014 – 2016: 10 – 100 GPUs
- 2016 – 2017: large batch size training, architecture search, special hardware (etc, TPU)

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

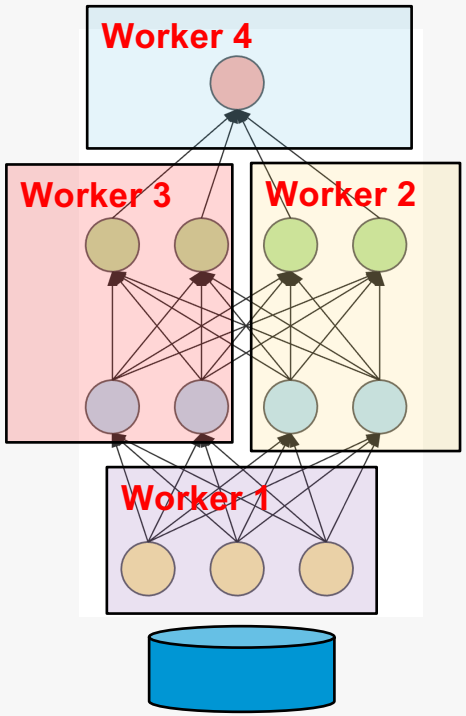
Argonne NATIONAL LABORATORY

Need for distributed (parallel) training on HPC

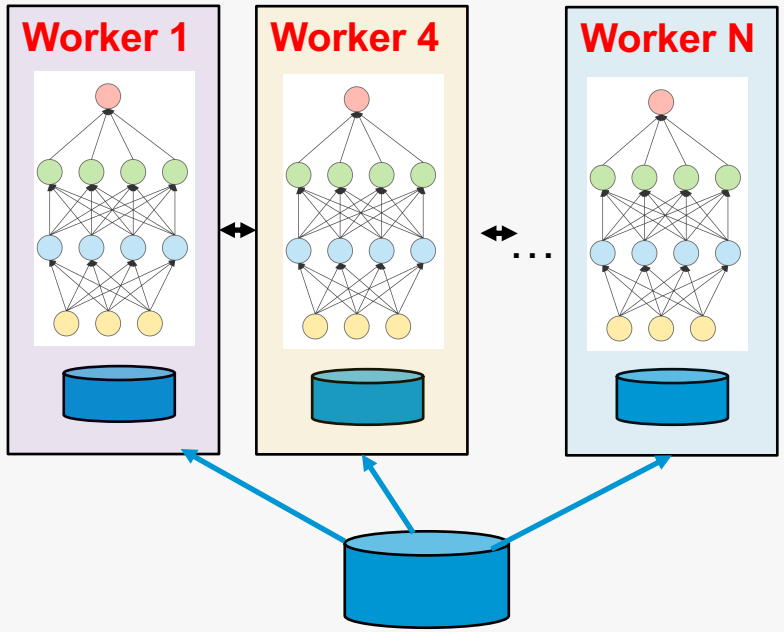
- Increase of model complexity leads to dramatic increase of computation;
- Increase of the amount of dataset makes sequentially scanning the whole dataset increasingly impossible;
- Coupling of deep learning to traditional HPC simulations might require distributed inference.



Parallelization schemes for distributed learning



Model parallelism



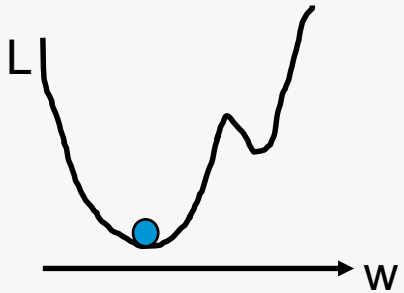
Data parallelism

Deep dive on model parallelism (Horovod)

Minimizing the loss:

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

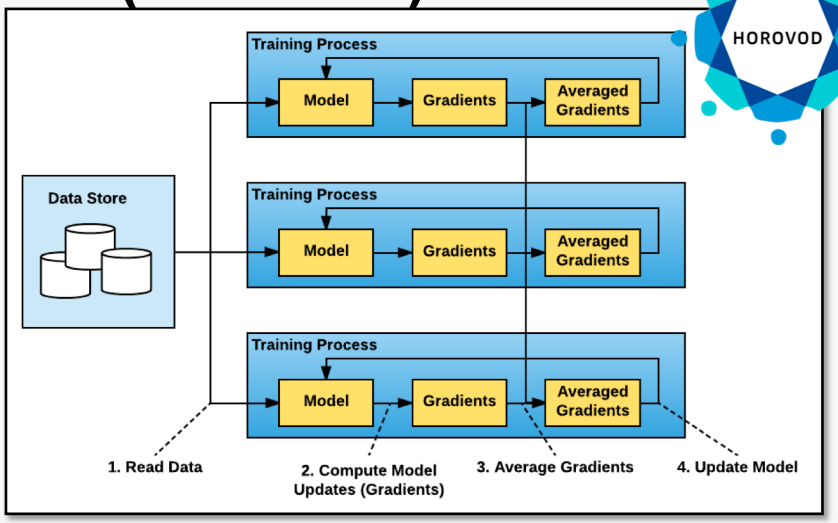
Dataset Weight



Stochastic Gradient Descent (SGD) update

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

Batch



- Model is replicated on all the workers;
- One batch B is divided into many sub batches b_w feed to different workers
- Gradients are averaged at each training step.

$$\frac{\partial L}{\partial P} = \frac{1}{B} \sum_{i=1}^{B} \frac{\partial L_i}{\partial P} = \frac{1}{Nb} \sum_{w=1}^{N} \sum_{i \in b_w} \frac{\partial L_i}{\partial P}.$$

Model is updated at each step. It scans through the entire dataset once at one epoch.

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

Argonne NATIONAL LABORATORY

Scaling the batch size

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

Different ways of scaling in data parallel training (suppose the original batch size is B_0)

Scaling	Global batch size (B)	Local batch size (b_w)
Strong scaling	B_0	B_0/N
Weak scaling	$N*B_0$	B_0
Square root scaling	$\sqrt{N} * B_0$	$B_0/\sqrt{N}$

We suggest to use **weak scaling** to maintain high throughput per node.

Argonne NATIONAL LABORATORY

Linear scaling rule

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

- k steps with learning rate η and batch size n

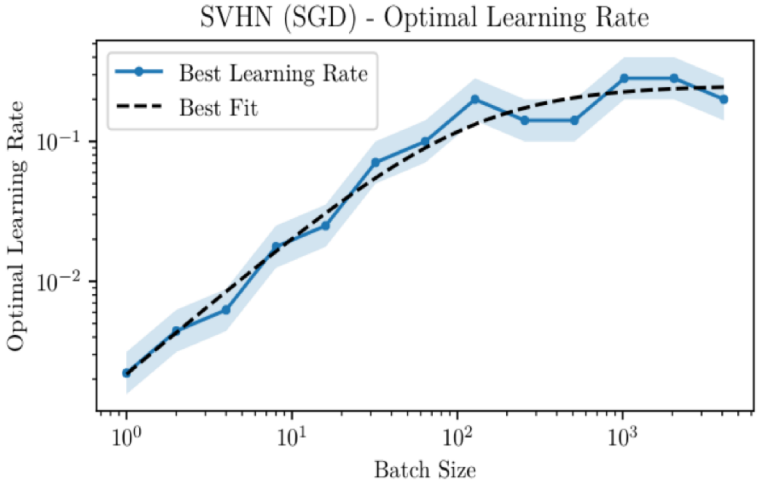
$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_{t+j})$$

- Single step with new learning rate $\hat{\eta}$ and large batch $\cup_j B_j$ (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, $\hat{\omega}_{t+1} \sim \omega_{t+k}$.

Ideally, large batch training with a scaled learning rate will reach the similar minimum with fewer steps.



The optimal learning for a range of batch sizes, for an SVHN classifier trained with SGD

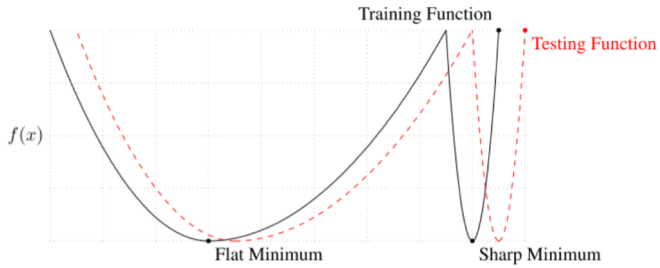
S. McCandlish, J. Kaplan, D. Amodei, arXiv:1812.06162

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



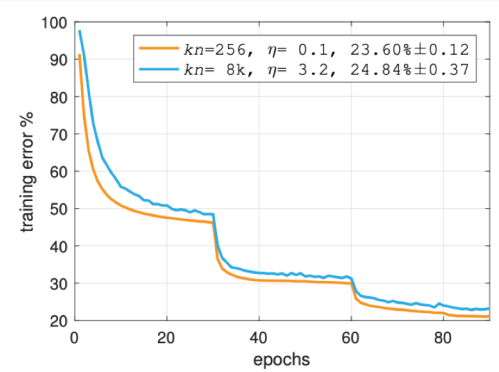
"*... large-batch ... converge to sharp minimizers of the training function ... In contrast, small-batch methods converge to flat minimizers*"

Keskar et al, arXiv:1609.04836

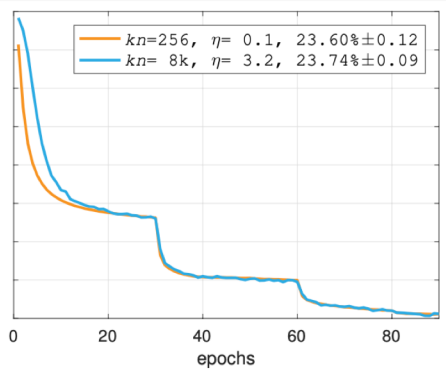
Challenges with large batch training

Solution: using warm up steps

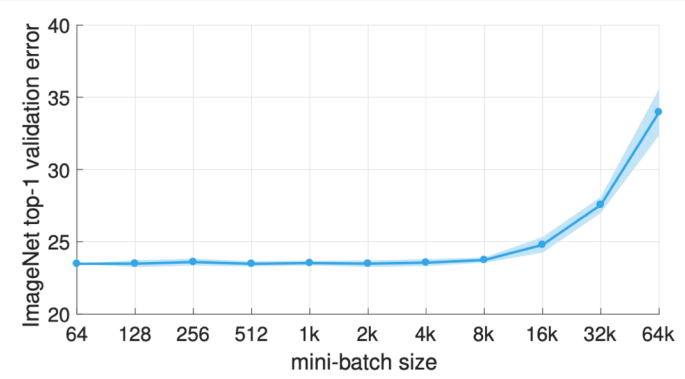
- Use a smaller learning rate at the initial stage of training (couple epochs), and gradually increase to $\hat{\eta} = N\eta$
- Use linear scaling of learning rate ($\hat{\eta} = N\eta$)



No warm up



Gradual warm up

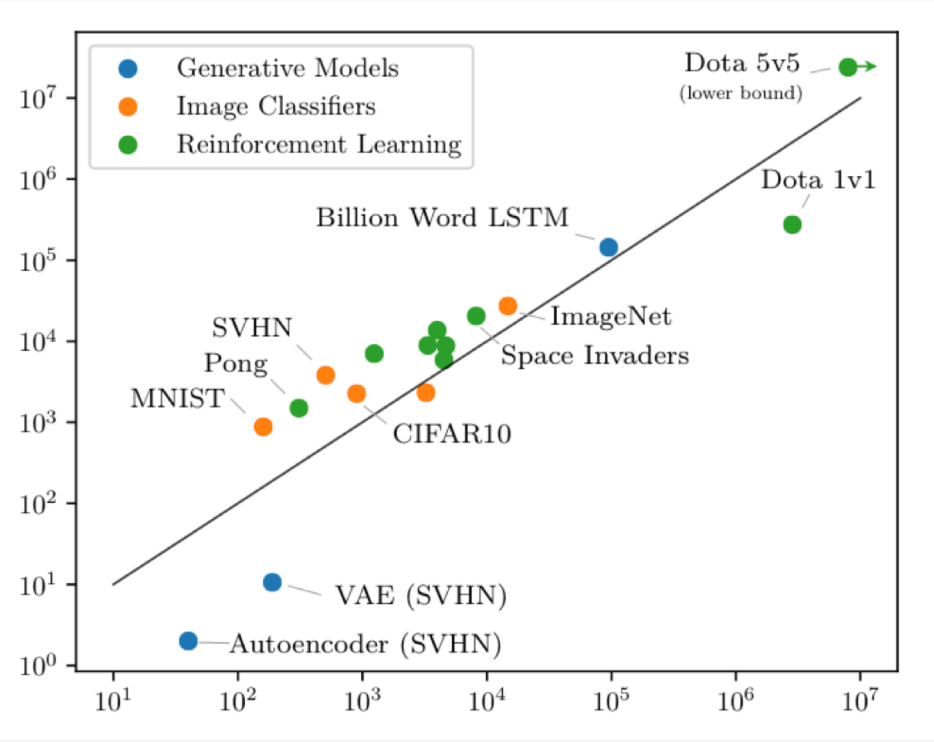


This scheme works up to 8k batch size

ResNet-50 training

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 places a strong scaling limit to data parallel:

# of workers < maximum batch size

Data parallel training with Horovod

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

- Import Horovod modules and initialize horovod;
- Wrap optimizer using hvd.DistributedOptimizer: scale the learning rate by number of workers;
- Broadcast the weights from worker 0 to all the workers;
- Let worker 0 write check point files
- Data loading:
  - Option 1. All the workers scan through the whole dataset in a random way (make sure that different worker have different random seeds);
  - Option 2. Divide the dataset and each worker only scans through a subset 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
```

https://github.com/uber/horovod/blob/master/examples/tensorflow_mnist.py

Argonne NATIONAL LABORATORY

PyTorch with Horovod

```
#…
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())
```

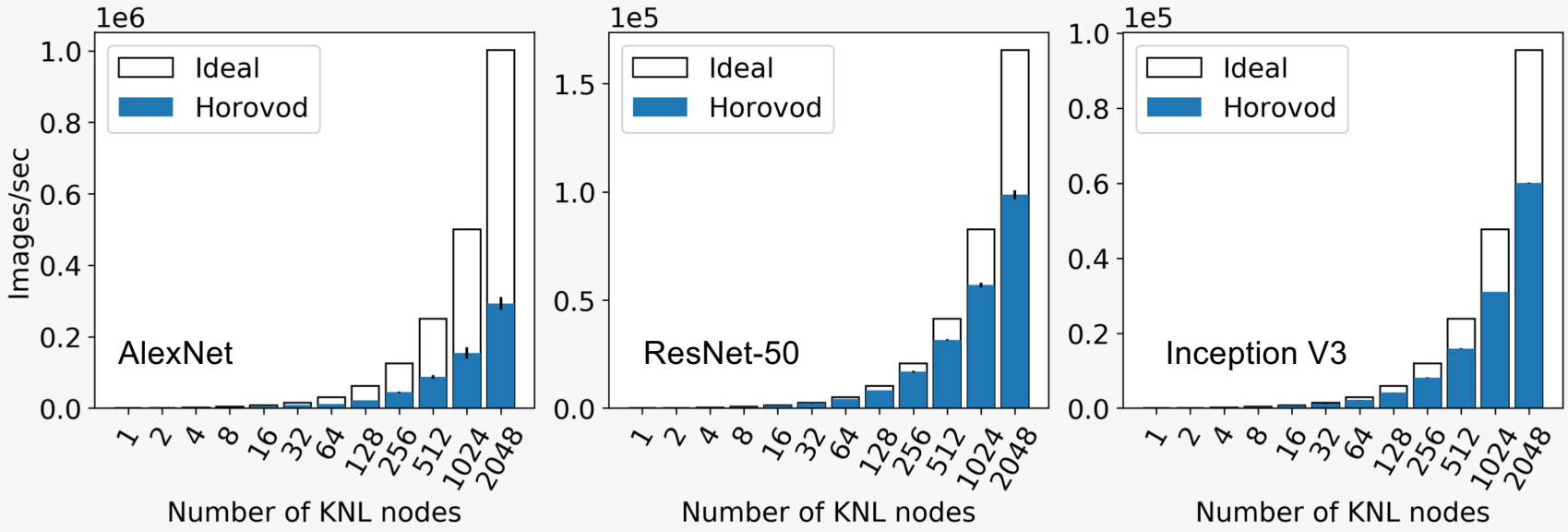
https://github.com/uber/horovod/blob/master/examples/pytorch_mnist.py

Keras with Horovod

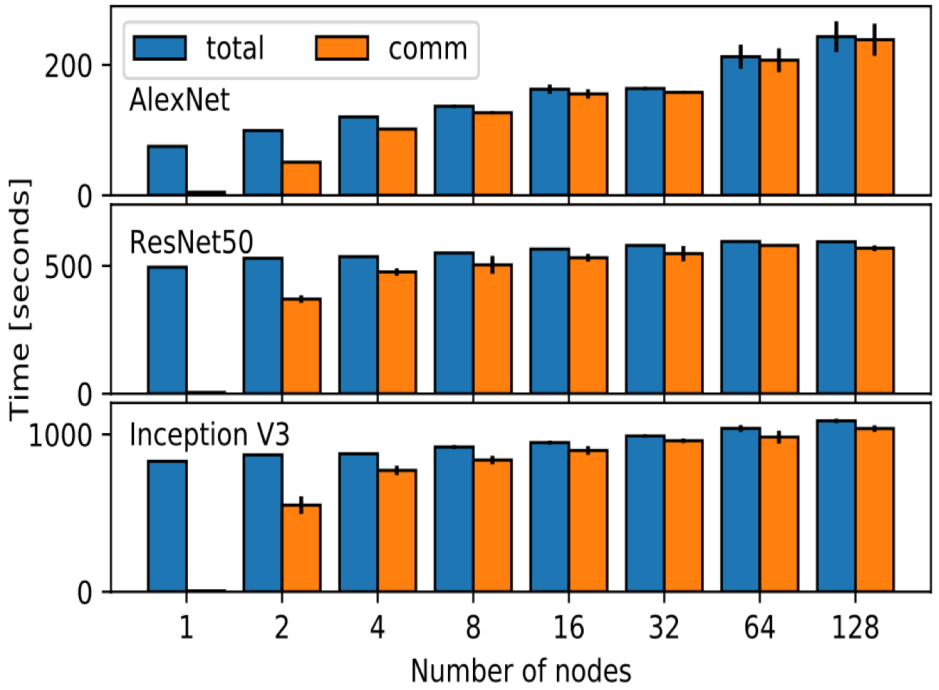
```
import keras
import tensorflow as tf
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)
model.compile(loss=keras.losses.categorical_crossentropy,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

https://github.com/uber/horovod/blob/master/examples/keras_mnist.py

Scaling TensorFlow using Horovod on Theta: One worker per node, batch size = 512



Communication overhead in Horovod



Total wall time and communication time for training of 60 steps

- The communication time increases as the number of nodes increases and becomes nearly identical to the total elapse time at large scale
- The compute operations and reduction operations on different layers are independent from each other,
- Reduction operations and compute operations are executed concurrently through creating communication helper threads.

Communication overhead in Horovod

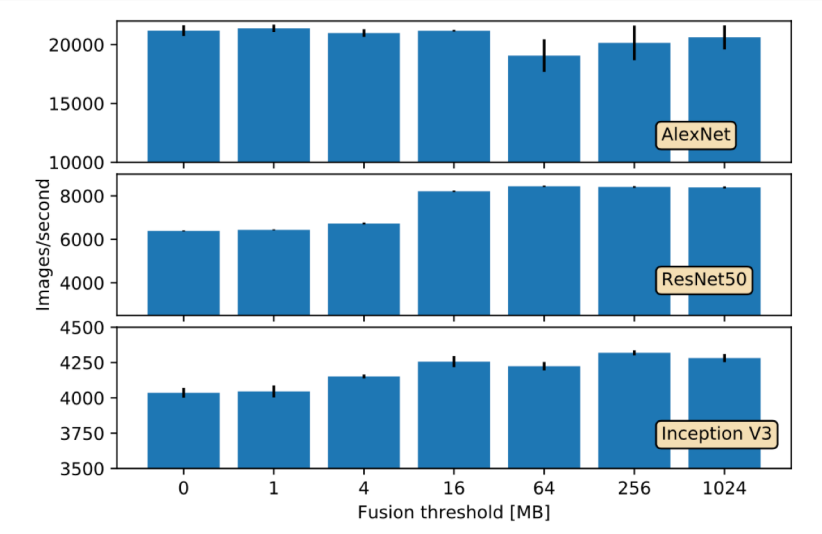
Model	AlexNet			ResNet50			Inception V3		
MPI Routine	#calls	avg. bytes	time(sec)	#calls	avg. bytes	time(sec)	#calls	avg. bytes	time(sec)
MPI_Bcast	5323	46534.7	1.274	41690	2494.8	4.326	79585	1231.2	6.664
MPI_Allreduce	324	45809144.4	182.468	231	26554889.4	229.017	705	8108732.3	343.818
MPI_Gather	2653	4.0	18.622	20711	4.0	54.865	39603	4.0	114.052
MPI_Gatherv	2653	0.0	18.055	20711	0.0	81.816	39603	0.0	165.714
Sync.			59.220			189.875			420.562
Comm			279.640			559.898			1050.810
Elapsed time			283.944			596.536			1099.880
MPI_Allreduce	#calls	bytes	time(sec)	#calls	bytes	time(sec)	#calls	bytes	time(sec)
	118	12973041.2	29.986	46	7708210.0	123.890	50	7353762.7	44.238
	60	67108864.0	38.266	9	12367668.4	19.958	64	12536660.4	144.036
	60	150994944.0	110.314	55	27871865.0	42.138	23	23015552.0	29.768
				65	63484380.6	39.437	60	64974078.9	111.199

- The dominant collective MPI routines are *MPI Allreduce*, *MPI Gatherv*, *MPI Gather.*
- The majority of the communication is spent on large message (10-100MB) reduction.

Argonne NATIONAL LABORATORY

Horovod tensor fusion

To avoid the latency for small message size reduction, Horovod adapts a tensor fusion approach, where several small tensors are fused into a bigger buffer before executing reduction. Fusion threshold sets the buffer size.

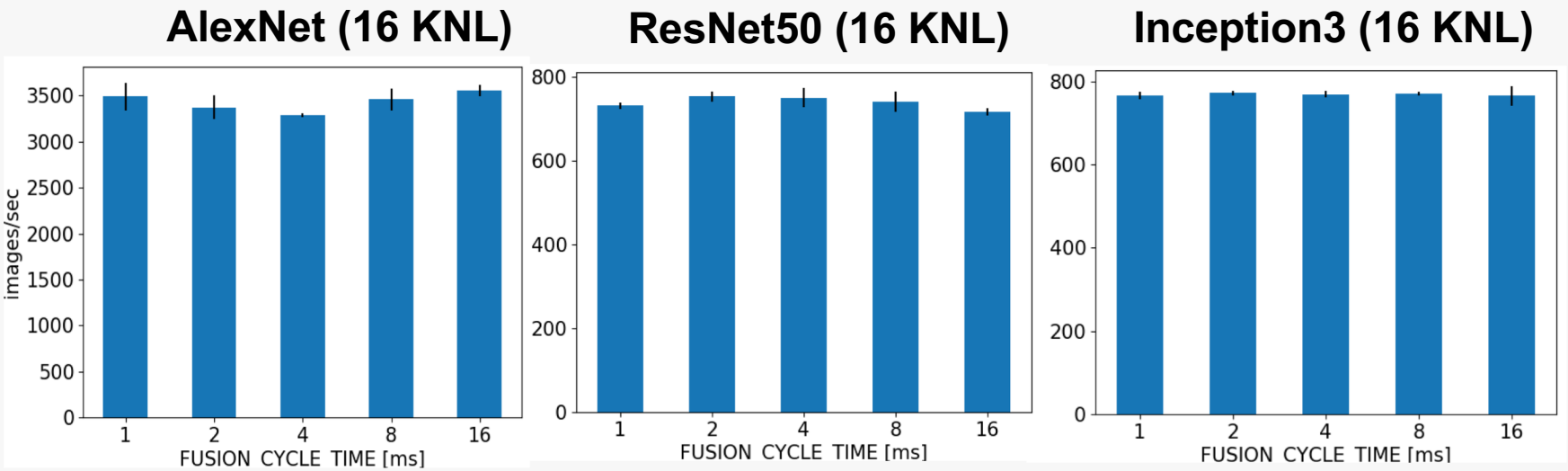


Aggregated training throughput of TensorFlow for different fusion thresholds on 128 KNL nodes.

fusion threshold=0			fusion threshold=256MB		
#calls	avg. bytes	time(sec)	#calls	avg. bytes	time(sec)
900	4089.9	11.220	57	4004.0	8.468
900	926242.1	19.345	58	8202659.0	230.191
120	2097152.0	6.429	2	12412928.0	7.345
660	3193390.5	72.309	60	93785642.7	42.493
120	8294400.0	260.100			
180	9437184.0	44.554			
total		425.696			289.092

MPI Allreduce messages size increases as fusion threshold increases (ResNet50, batch size=512, 60 batches): only those taking more than 1% of the total allreduce time are listed

Horovod tensor fusion
FUSION_CYCLE_TIME (default: 3.5ms)



The runtime is not sensitive with respect to the changing of FUSION_CYCLE_TIME

Tips for data management in data parallel training

- Preprocess the raw data before the training and perform training on preprocessed data.
- Save the dataset into binary format (e.g., HDF5) instead of text format such as CSV files.
- Organize the dataset in a reasonable way:
  - Avoid using one sample per file which may introduce large overhead from opening/closing small files;
  - Consider saving several batches of samples into one file (like TF records in TensorFlow) and let different workers read different files independently.
- Make sure different workers read different subsets of the dataset, avoiding redundant I/O access.
- If the dataset is small which fits into the memory, load the entire dataset at once rather than reading them at each epoch on the fly; notice that each node on Theta has 192 GB DDR4 and 16 GB MCDRAM.
- If the dataset is large, consider prefetching the data from the disk using asynchronous I/O to overlap I/O and compute.
- Consider use node local SSD storage.

Examples /projects/SDL_Workshop/training/DeepLearningFrameworks

MNIST examples:

- keras_mnist.py
- tensorflow_mnist.py
- pytorch_mnist.py

ImageNet examples:

- keras_imagenet_resnet50.py
- pytorch_imagenet_resnet50.py
- tensorflow_synthetic_benchmark.py

These examples were created based on https://github.com/horovod/horovod/tree/master/examples.

The original examples from Horovod are assumed to be run on GPU. I modified them to be able to run on CPUs.

Submission scripts:

The submission scripts are for Theta.Cooley @ ALCF. submissions/theta/qsub_*.sh. submissions/cooley/qsub_*.sh

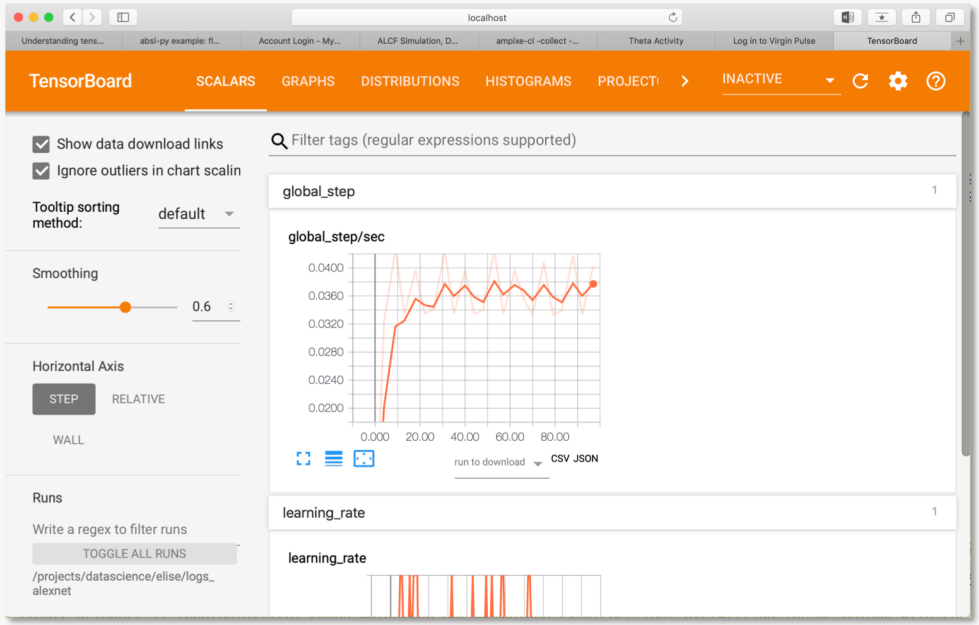
Running the examples

qsub -A SDL_Workshop -q training -t 1:00:00 -n 4 submissions/theta/qsub_keras_mnist.py

modify -n and -t to run on different number of nodes with different walltime.

We assume one worker per node in all the cases. Modify PROC_PER_NODE and num_intra, num_threads accordingly if you want to set more than one workers per node.

Visualization with Tensorboard



Read log files through ssh tunneling

(1) SSH tunnel to Theta

ssh -XL 16006:127.0.0.1:6006 user@theta.alcf.anl.gov

(2) Run tensorboard on Theta

> module load tensorboard

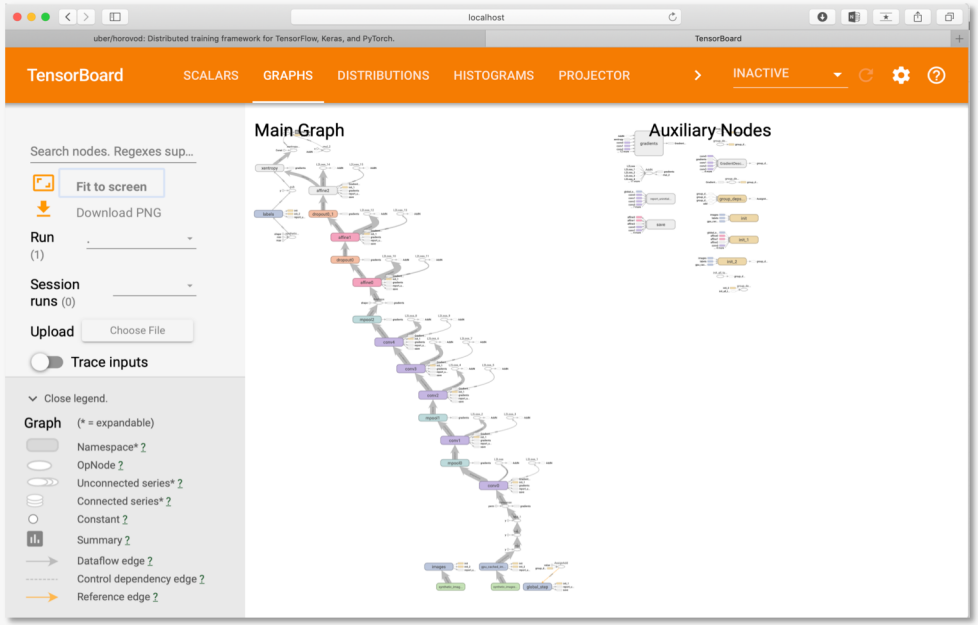
> tensorboard --logdir DIR

(3) Open browser from local machine: https://localhost:16006

Interactive job controlling through Tensorboard is not supported on Theta yet.

https://www.datacamp.com/community/tutorials/tensorboard-tutorial

Visualization with Tensorboard



Read log files through ssh tunneling

(1) SSH tunnel to Theta

ssh -XL 16006:127.0.0.1:6006 user@theta.alcf.anl.gov

(2) Run tensorboard on Theta

> module load tensorboard

> tensorboard --logdir DIR

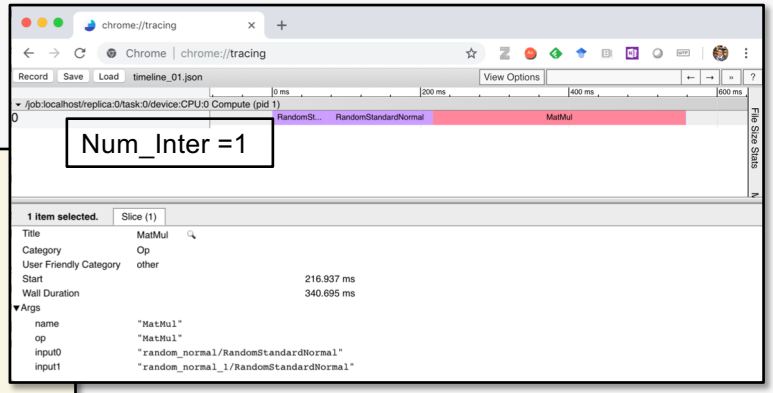
(3) Open browser from local machine: https://localhost:16006

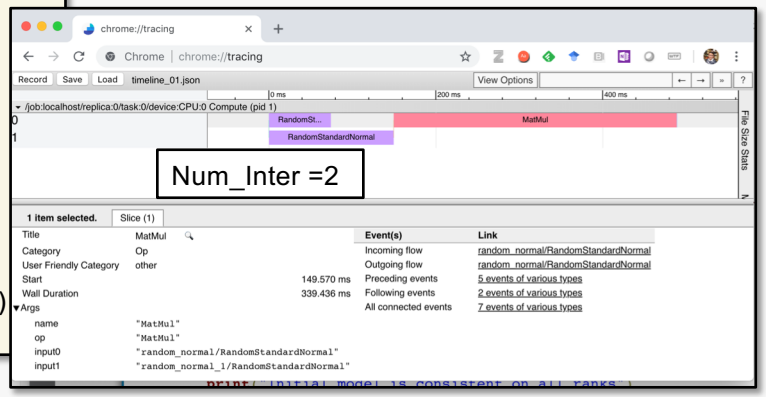
Interactive job controlling through Tensorboard is not supported on Theta yet.

https://www.datacamp.com/community/tutorials/tensorboard-tutorial

Tracing profile

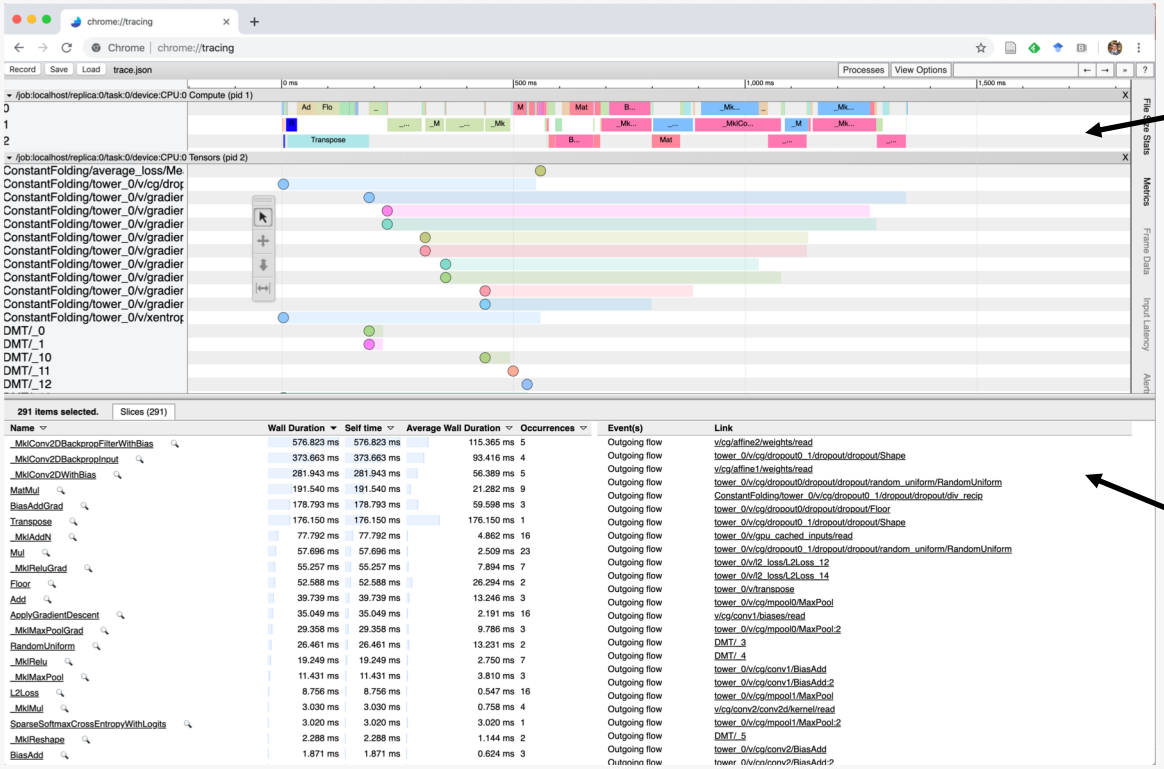
```
import tensorflow as tf
from tensorflow.python.client import timeline
import sys
a = tf.random_normal([2000, 5000])
b = tf.random_normal([5000, 1000])
res = tf.matmul(a, b)
sess = tf.Session(config=tf.ConfigProto(\
        inter_op_parallelism_threads=1,\
            intra_op_parallelism_threads=1 ))
# add additional options to trace the session execution
options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
run_metadata = tf.RunMetadata()
sess.run(res, options=options, run_metadata=run_metadata)
# Create the Timeline object, and write it to a json file
fetched_timeline = timeline.Timeline(run_metadata.step_stats)
chrome_trace = fetched_timeline.generate_chrome_trace_format()
f=open('timeline_01.json', 'w'); f.write(chrome_trace);f.close()
```





Open timeline_01.json using Chrome.
Go to the page chrome://tracing. ”Load” the JSON file.

Tracing profile (AlexNet)



Concurrent execution of TF operations

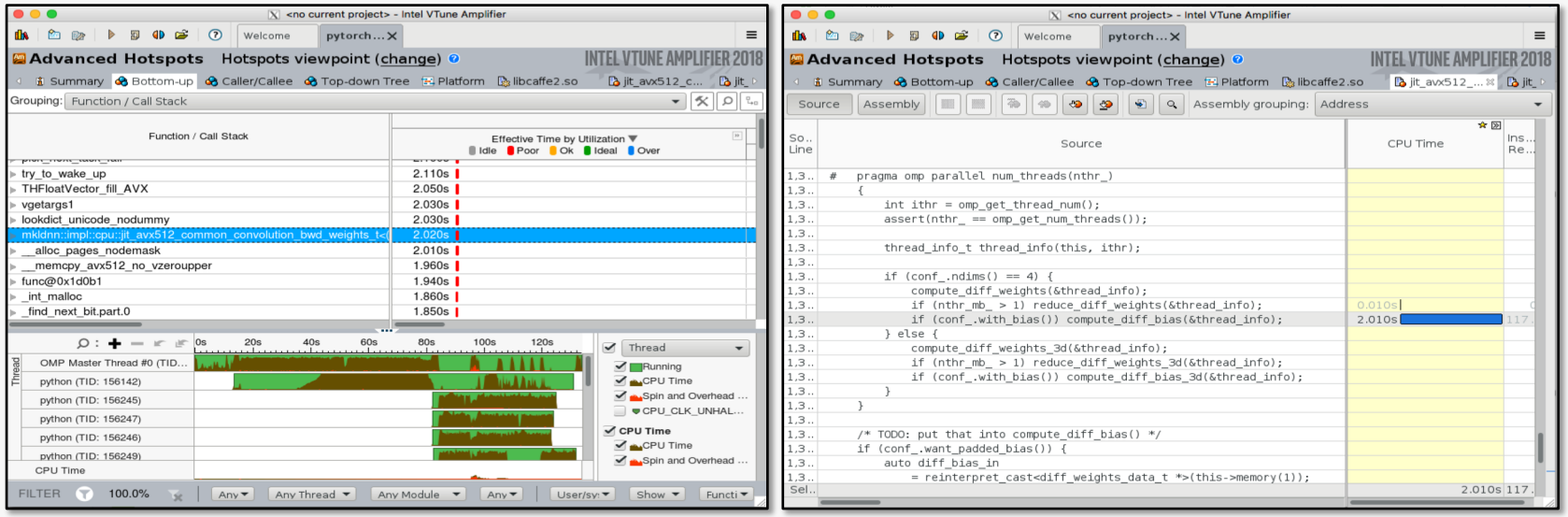
Flat profiling of TF operations

Argonne NATIONAL LABORATORY

VTune profiling

```
source /opt/intel/vtune_amplifier/amplxe-vars.sh
aprun -n … -e OMP_NUM_THREADS=128 \
          -e LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64 \
          ampxle-cl -collect advanced-hotspots -r output_dir python script.py
```

Remember to set LD_LIBRARY_PATH,
Put vtune library at the end!! Otherwise, it might complaint about the GLIBCXX version.



The python modules are compiled using -g flag. Therefore, the user could trace the source file in Vtune.

MKL-DNN VERBOSE output

aprun … -e MKLDNN_VERBOSE=2

```
#verbose,stage,primitive-kind,primitive-implementation,propagation-kind,input/output,auxiliary information,time
mkldnn_verbose,create,convolution,jit:avx512_common,forward_training,fsrc:nchw fwei:Ohwi16o fbia:x fdst:nChw16c,a\
lg:convolution_direct,mb256_g1ic3oc64_ih227oh55kh11sh4dh0ph0_iw227ow55kw11sw4dw0pw0,3.73804
mkldnn_verbose,create,reorder,jit:uni,undef,in:f32_hwio out:f32_Ohwi16o,num:1,64x3x11x11,9.52808
mkldnn_verbose,exec,reorder,jit:uni,undef,in:f32_hwio out:f32_Ohwi16o,num:1,64x3x11x11,107.507
```

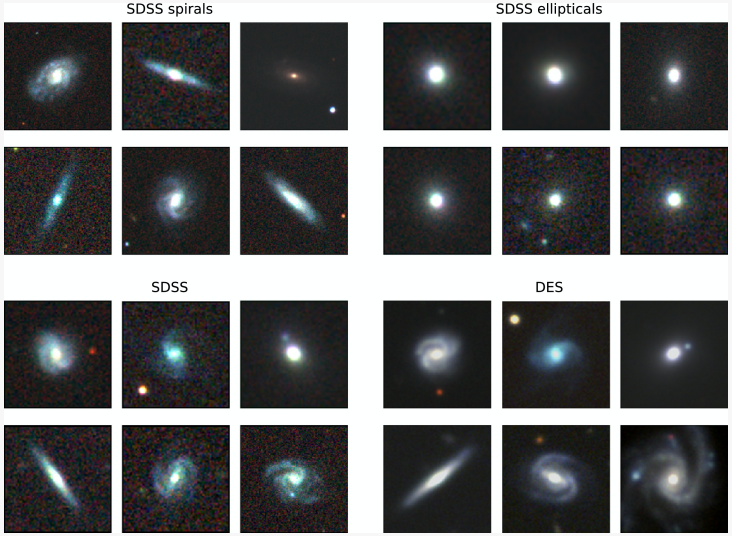
AlexNet	batch size = 512	
primitive-kind	Cache	Flat(DDR)
reorder	15.82	16.12
convolution	615.49	678.69
eltwise	26.68	71.00
pooling	14.42	44.20
sum	10.71	19.43
mkldnn	683.12	829.43
total	1087.49	1287.53

ResNet50	batch size = 512	
primitive-kind	Cache	Flat(DDR)
reorder	140.06	149.03
convolution	3632.35	5814.90
eltwise	347.13	1159.54
pooling	36.36	118.47
sum	214.51	611.65
batch_norm.	1186.56	2447.31
mkldnn	5556.97	10300.90
total	6512.34	11508.20

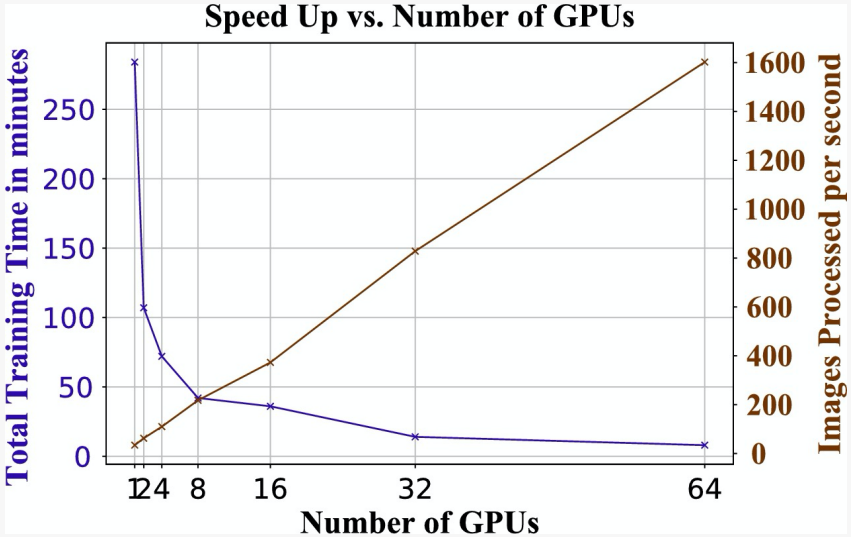
Inception V3	batch size = 512	
primitive-kind	Cache	Flat(DDR)
reorder	1230.60	1753.43
convolution	6065.22	8139.36
eltwise	453.38	1215.40
pooling	301.38	739.56
sum	92.32	98.41
batch_norm.	1144.74	2142.06
mkldnn	9410.67	14278.62
total	11196.15	16326.53

Time spent on different types of MKL-DNN functions. Convolution is compute bound, others are memory bandwidth bound.

Science use case 1 - Galaxy classification using modified Xception model



Galaxy images

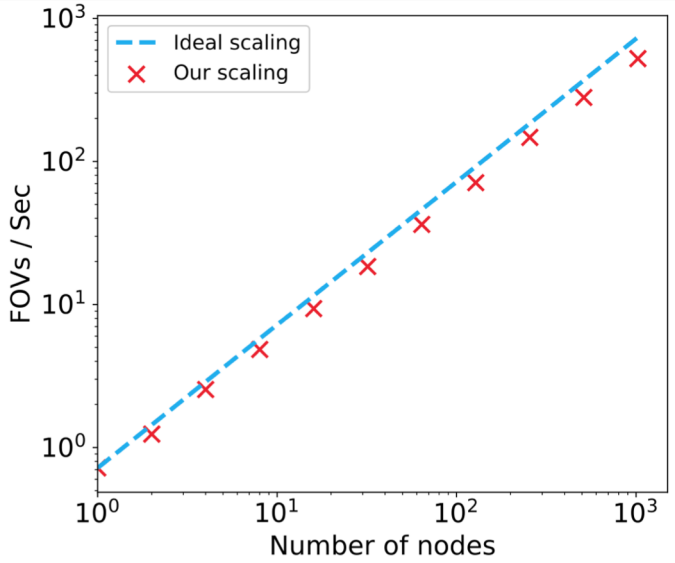


~ 5 Hrs using 1 K80 GPU to 8 mins using 64 K80 GPUs using computing resource from Cooley @ ALCF

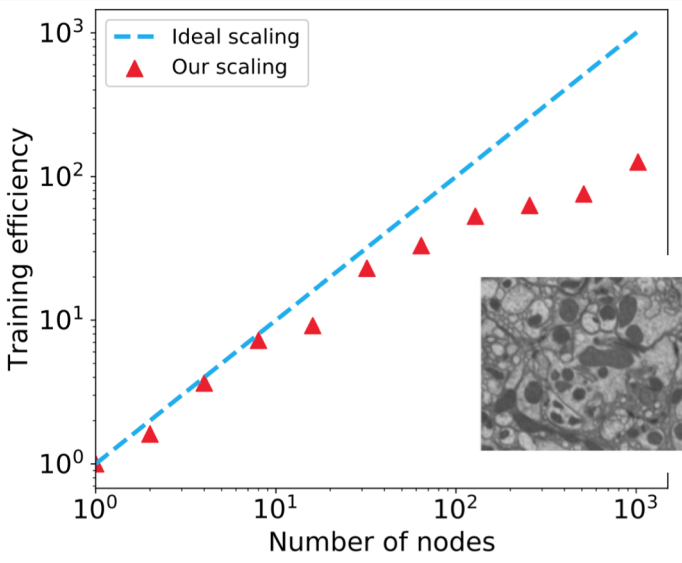
See Elise Jennings talk for more details.

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

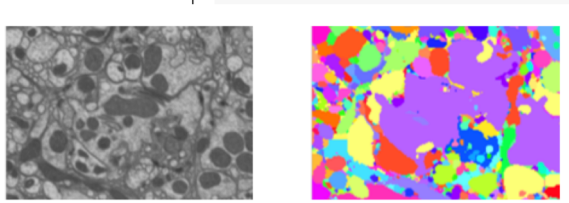


Scaling results in terms of throughput



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]


Thank you!
huihuo.zheng@anl.gov