

# ALCF Data and Learning Frameworks

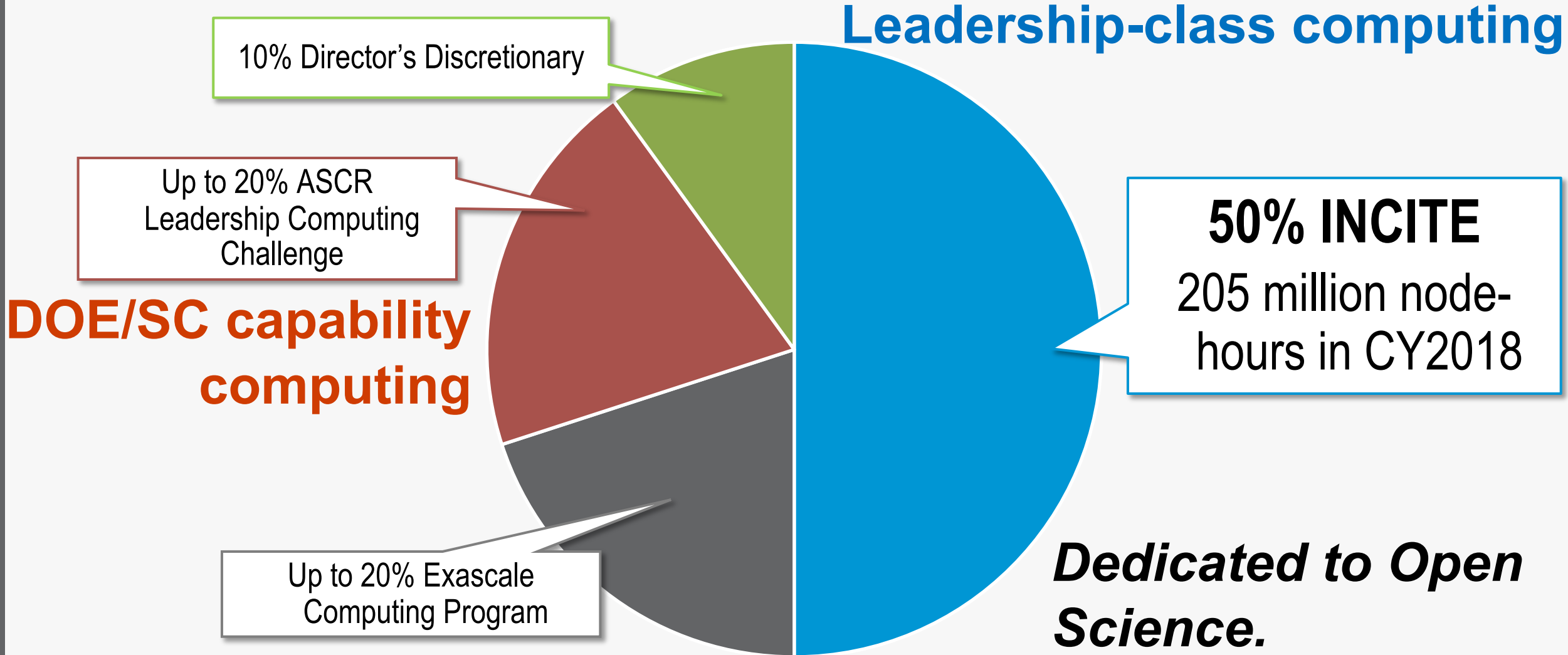
Corey Adams, on behalf of the Datascience group  
@ ALCF



# Data Science at ALCF

# Who Uses ALCF?

## Allocation Distributions





# [datascience@alcf.anl.gov](mailto:datascience@alcf.anl.gov)



Corey Adams



Prasanna  
Balaprakash



Taylor Childers



Murali Emani



Elise Jennings



Xiao-Yong Jin



Murat Keceli



Bethany Lusch



Alvaro Vazquez



Adrian Pope



Misha Salim



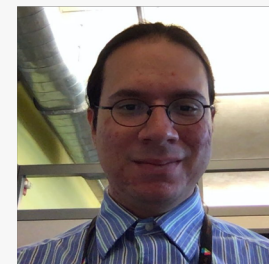
Himanshu Sharma



Ganesh Sivaraman



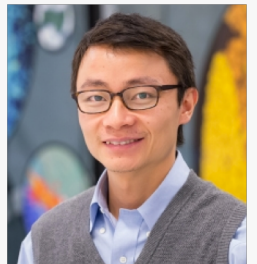
Tom Uram



Antonio Villarreal



Venkat Vishwanath



Huihuo Zheng



# ALCF Data Science Program (ADSP) Targets Data & Learning Pillars



## Data

- Experimental/observational data
  - Image analysis
  - Multidimensional structure discovery
- Complex and interactive workflows
- On-demand HPC
- Persistent data techniques
  - Object store
  - Databases
- Streaming/real-time data
- Uncertainty quantification
- Statistical methods
- Graph analytics

## Learning

- Deep learning
- Machine learning steering simulations
  - Parameter scans
  - Materials design
  - Observational signatures
- Data-driven models and refinement for science using ML/DL
- Hyperparameter optimization
- Pattern recognition
- Bridging gaps in theory

**Big Data**

# ADSP Allocation Program

- Allocation program for projects that push the limits for data-centric and data-intensive computing, emphasizing techniques using machine learning, deep learning, and AI at scale.
- Current projects include science from :
  - Large Hadron Collider
  - Material Sciences
  - Cosmology
  - Industry research
  - Brain Connectomes and Brain Modelling
  - Hyperparameter Searches
  - Large Synoptic Survey Telescope
  - APS Xray Sciences
- More information online: <https://www.alcf.anl.gov/alcf-data-science-program>
- Call is out now! [ADSP Announcement](#)



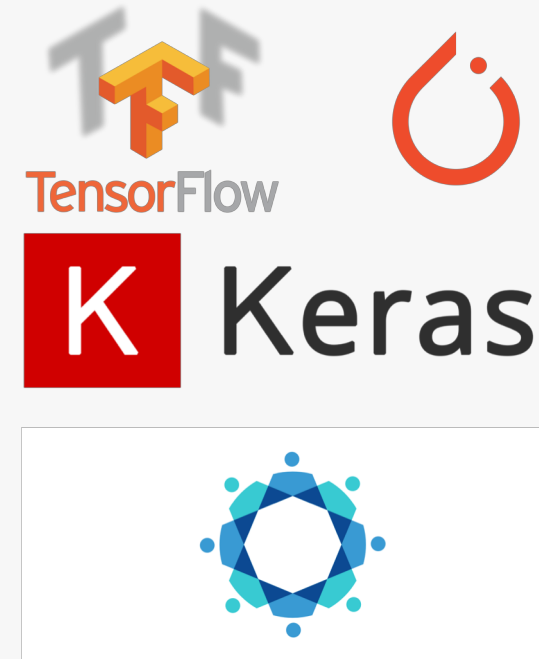
# ALCF Datascience Group Supports ...

- **Software:** optimized builds of important ML and DL software (tensorflow, pytorch, horovod)
- **Projects:** datascience members work with ADSP projects, AESP projects, and other user projects to help users deploy their science on ALCF systems
- **Users:** we are always interested to get feedback and help the users of big data and learning projects, whether you're reporting a bug or telling us you got great performance.

# Datascience software

Specifically optimized for KNL(*CPU*), with AVX512

- Using intel python 3.5 (based on intelpython35 module)
- GCC/7.3.0
- With `-g`, could be used for profiling
- Linked to MKL and MKL-DNN (home build)
- Dynamically linked to external libraries (be careful of your `LD_LIBRARY_PATH`, `PYTHONPATH`)
- With MPI through Horovod



```
$ module avail datascience
----- /soft/environment/modules/modulefiles -----
datascience/craype-ml-1.1.2      datascience/pytorch-0.5.0-mkl-dnn
datascience/gcc-8.2.0          datascience/pytorch-1.0
datascience/horovod-0.13.11    datascience/tensorboard
datascience/horovod-0.14.1    datascience/tensorflow-1.4
datascience/horovod-0.15.2    datascience/tensorflow-1.6
datascience/keras-2.2.2       datascience/tensorflow-1.8
datascience/keras-2.2.4       datascience/tensorflow-1.10
datascience/mpi4py            datascience/tensorflow-1.12
```



# How to run datascience modules

```
user@thetalogin6 : ~
$ module load datascience/tensorflow-1.12
Intel Python 3.5 version 2017.0.035 loaded
-----
Tensorflow version 1.12.0 loaded

$ which python
/opt/intel/python/2017.0.035/intelpython35/bin/python

$ python -c "import tensorflow as tf"
2019-04-18 19:53:59.152614: 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)
```

Datascience modules are compiled for AVX512 vectorization and **do not** run on the login nodes. Use qsub instead:

```
#!/bin/bash

#!/bin/bash
#COBALT -A <allocation>
#COBALT -n 128
#COBALT -q default --attrs mcdram=cache:numa=quad

# Load the datascience modules:
module load datascience/tensorflow-1.12 datascience/horovod-0.15.2

# Run your application:
aprun -n nproc -N nproc_per_node -cc depth -j 2 python script
```

# Monitoring your training: Tensorboard

Monitoring neural network training is important for iterating on your models and networks. Tensorboard is the widely adopted solution. Two pieces:

1. During training, store scalars, images, histograms, etc into a tensorboard file
  1. It's built into tensorflow: [tensorboard](#)
  2. You can use [tensorboardX](#) to generate compatible files from non-tensorflow frameworks – you will need to pip install this into your own python area.
2. In a different session, use tensorboard to generate an interactive web page to monitor your training.

```
[on theta login node:]
$ module load datascience/tensorboard

$ tensorboard --logdir /path/to/your/tensorboard/directory/
=====
* ALCF Data and Learning Frameworks -- Tensorboard
* Please go to your browser http://localhost:PORTNUMBER
* if you have used ssh -XY PORTNUMBER:127.0.0.1:6006 user@theta.alcf.anl.gov
* Contact: Huihuo Zheng <huihuo.zheng@anl.gov> for any issues
=====

TensorBoard 1.13.1 at http://thetalogin4:6006 (Press CTRL+C to quit)

[from a fresh shell on your laptop:]
$ ssh -L 6006:127.0.0.1:6006 [user]@theta.alcf.anl.gov
```

Then, point your laptop browser here: <http://127.0.0.1:6006>



# We also support ... pip, conda

If you need more than just tensorflow/pytorch/numpy, you can usually meet your requirements by using pip or conda. Some notes:

- Intel distributes intel optimized python, tensorflow, and pytorch via conda. **If you use conda it's highly recommended you use the Intel channel for these packages.**

```
conda install tensorflow -c intel
```

```
conda create -n IDP intelpython3_full -c intel
```

```
pip install intel-tensorflow
```

- **Conda is a good option for getting up-to-date builds of python/tensorflow/pytorch on Theta from Intel.**
- How do you build a conda environment? Full documentation for [Theta](#) and in [general](#).

# We also support ... Singularity

**Singularity is a standalone runtime environment for containerized, self-contained operating systems.** You can set up your own software at all levels (compiler, python, tensorflow) and use optimized MPI libraries on the host system.

For some workflows, it is easiest to package all dependent software in an isolated container. ALCF supports Singularity (not docker), documentation for Theta can be found here:  
<https://www.alcf.anl.gov/user-guides/singularity>

Singularity can be a very useful tool if you need it, but there are some important notes:

- For optimal communication performance, you must use the right MPI libraries inside of your container.
- Launch singularity from within your aprun command, not the other way around.

See the documentation for a complete walkthrough of how to build containers with all the best practices.

# Machine Learning on Theta

# Machine Learning and HPC

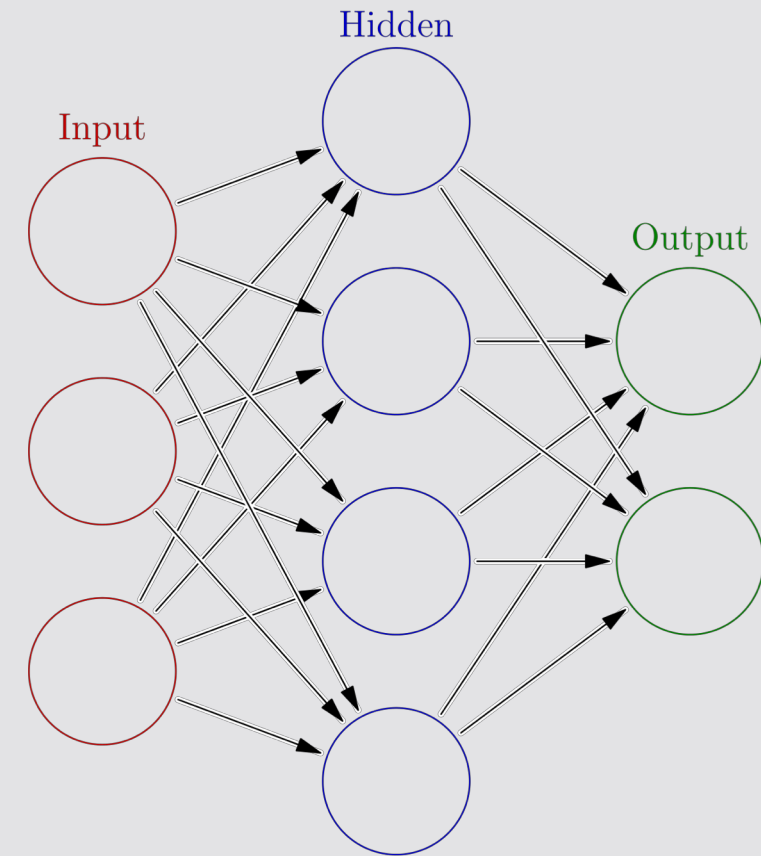
Accelerate and improve an application's:

---

**Time to Solution (Training)** – with scalable learning techniques, you can process more images per second, reduce the time per epoch, and reach a trained network faster.

**Quality of Solution** – with more compute resources available, you can perform hyperparameter searches to optimize network designs and training schemes. With powerful accelerators, you can train bigger and more computationally intense networks.

**Inference Throughput** – with high bandwidth IO, it is easy to scale up the throughput of inference techniques for deep learning.



High Performance Computing can improve all aspects of training and inference in machine learning.



# Machine Learning and HPC – KNL Nodes

## Single Node Performance Matters!

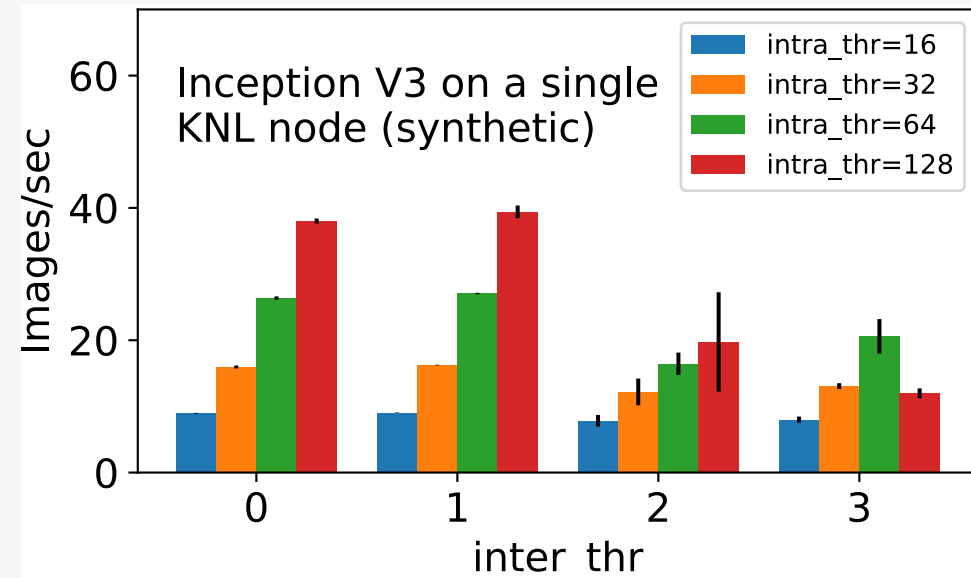
Running a model on an HPC node is often not like a standard GPU – **many configuration parameters matter, not all models have the same optimum parameters.**

Intel KNL != Nvidia GPU, so there are some configurations that are new/different for good performance.

**intra\_op\_parallelism\_threads:** Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool.

**inter\_op\_parallelism\_threads:** All ready nodes are scheduled in this pool.

```
config = tf.ConfigProto()  
config.intra_op_parallelism_threads = num_intra_threads  
config.inter_op_parallelism_threads = num_inter_threads  
tf.Session(config=config)
```



<https://www.tensorflow.org/guide/performance/overview>

# Performance Setting Guidelines

Performance with Tensorflow on KNLs requires management of many parameters at both build and run time.

**Intel Performance Guidelines:** <https://software.intel.com/en-us/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference>

**ALCF Performance Guidelines:** <https://www.alcf.anl.gov/user-guides/machine-learning-tools>

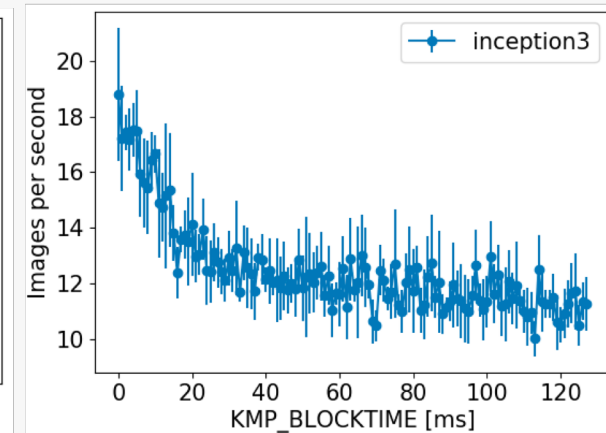
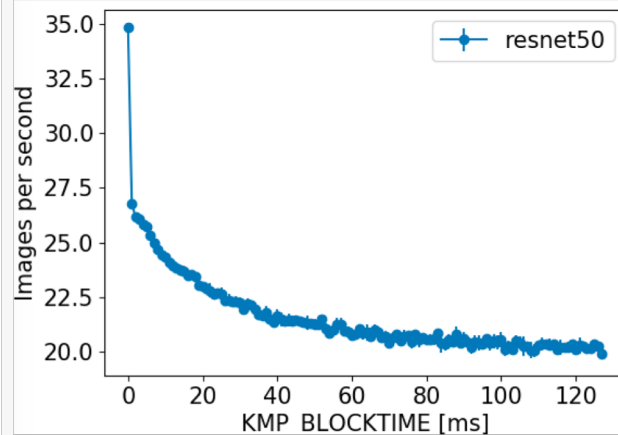
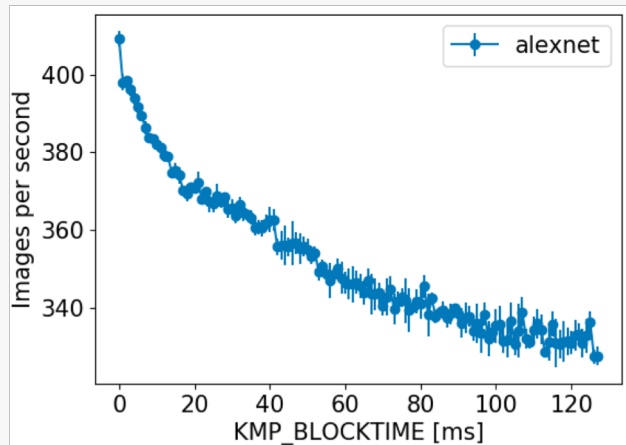
## Key Takeaways:

- Set **OMP\_NUM\_THREADS**=[number of physical cores = 64 on Theta]
- Set **KMP\_BLOCKTIME**=0 (sometimes =1 can be better for non-CNN)
- (tensorflow only) Set `intra_op_parallelism_threads == OMP_NUM_THREADS == number of physical cores == 64`
- (tensorflow only) Set `inter_op_parallelism_threads` for your application. 0 will default to the number of cores, the optimal value can be different for different applications. Run some tests!

# PyTorch on Theta

- [PyTorch](#) is an open source deep neural network framework similar to Tensorflow
- Available on theta via the datascience modules or by intel's conda channel
- Compatible with horovod, though also has distributed tools built in
  - Torch native distributed learning tools are untested on Theta, we suggest horovod.
- Intel has worked to optimize pytorch, the optimizations are in the main branch of torch but require building from source (datascience module does this) or using intel conda channels.
- Intel has some suggestions for pytorch performance:
  - <https://software.intel.com/en-us/articles/intel-and-facebook-collaborate-to-boost-pytorch-cpu-performance>
  - <https://software.intel.com/en-us/articles/how-to-get-better-performance-on-pytorchcaffe2-with-intel-acceleration>

# KMP\_BLOCKTIME



KMP\_BLOCKTIME=0 is optimal for KNL Nodes

**KMP\_AFFINITY=granularity=fine,verbose,compact,1,0**

Intel Affinity Guidelines

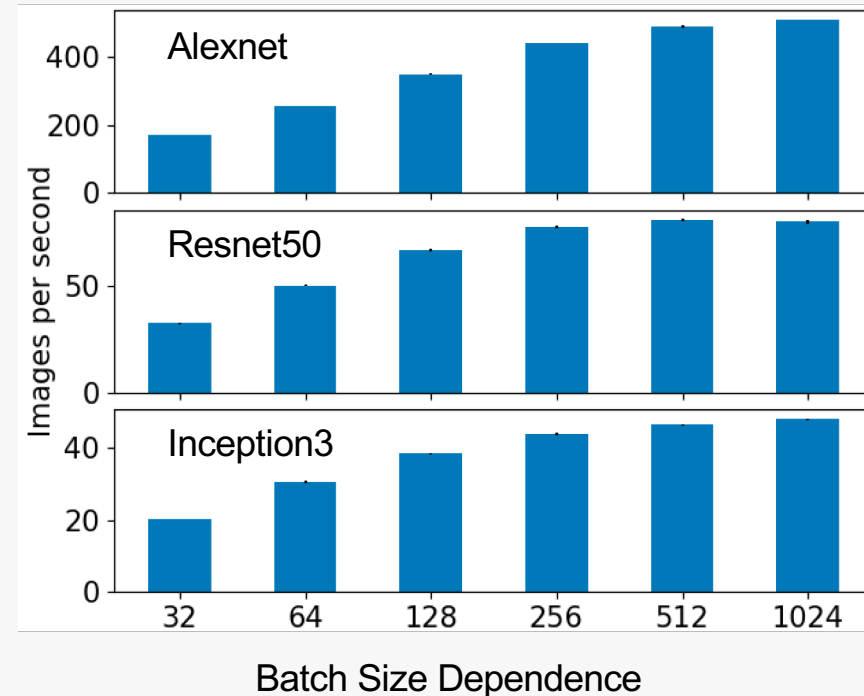


# Machine Learning and HPC – KNL Nodes

## Batch Size can be important

Bigger batch size often yields more images/second throughput (though not always), but the downside is always more seconds/global step at a large batch size.

Have to balance throughput vs. number of iterations in a fixed wallclock time.



# mpi4py and h5py on Theta

- [mpi4py](#) is python wrapper for mpi, with compatibility for general python objects (slow) and numpy objects (fast)
  - Support for many mpi operations:
    - Point to point communication
    - Collectives
    - Scatter/gather
  - Compatible with horovod
  - Use functions with Uppercase syntax (Send, Receive, Scatterv, Gatherv) for numpy objects
  - Use functions with lowercase syntax (send, receive, scatter, gather) for generic python objects
  - Available with modules (`module load datascience/mpi4py`)
- [h5py](#) is the hdf5 python wrapper and also supports [parallel hdf5](#), using mpi4py
  - Need parallel hdf5 libraries to use this
  - `module load miniconda-2.7/conda-4.4.10-h5py-parallel`

# Distributed Learning

Machine learning is a very important workflow for current and future supercomputing systems.  
How can you accelerate learning with more computing power?

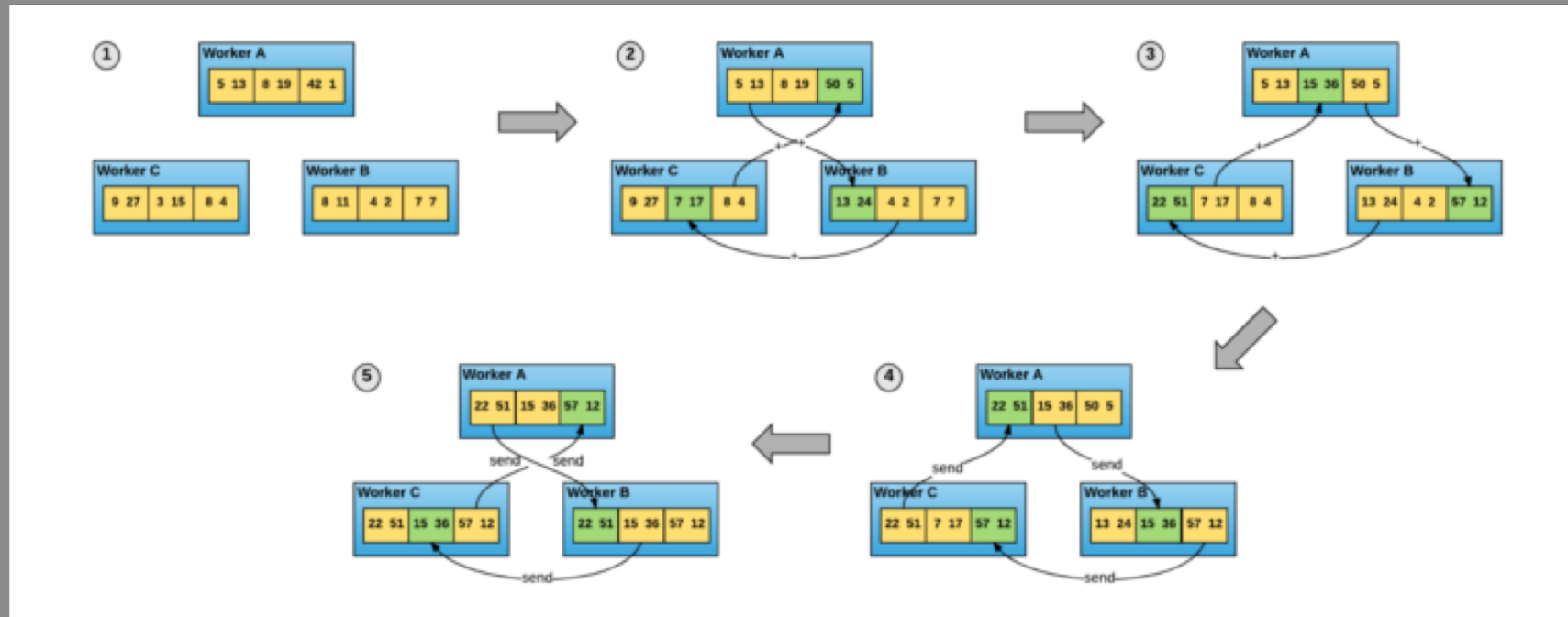


Image from Uber's Horovod: <https://eng.uber.com/horovod/>

# What is Distributed Learning?

The backpropagation algorithm is unchanged at its heart.

---

**Data Parallel learning** – with  $N$  nodes, replicate your model on each node. After the forward and backward computations, average the gradients across all nodes and use the averaged gradients to update the weights. Conceptually, this multiplies the minibatch size by  $N$ .

**Model Parallel Learning** – for models that don't fit on a single node, you can divide a single model across multiple locations. The design of distributing a model is not trivial, but tools are emerging.

**Both (“Mesh” training)** – Using  $n$  nodes for a single model, and  $N = k*n$  nodes for distributed training, you can achieve accelerated training of extremely large or expensive models.



# Data Parallel Learning

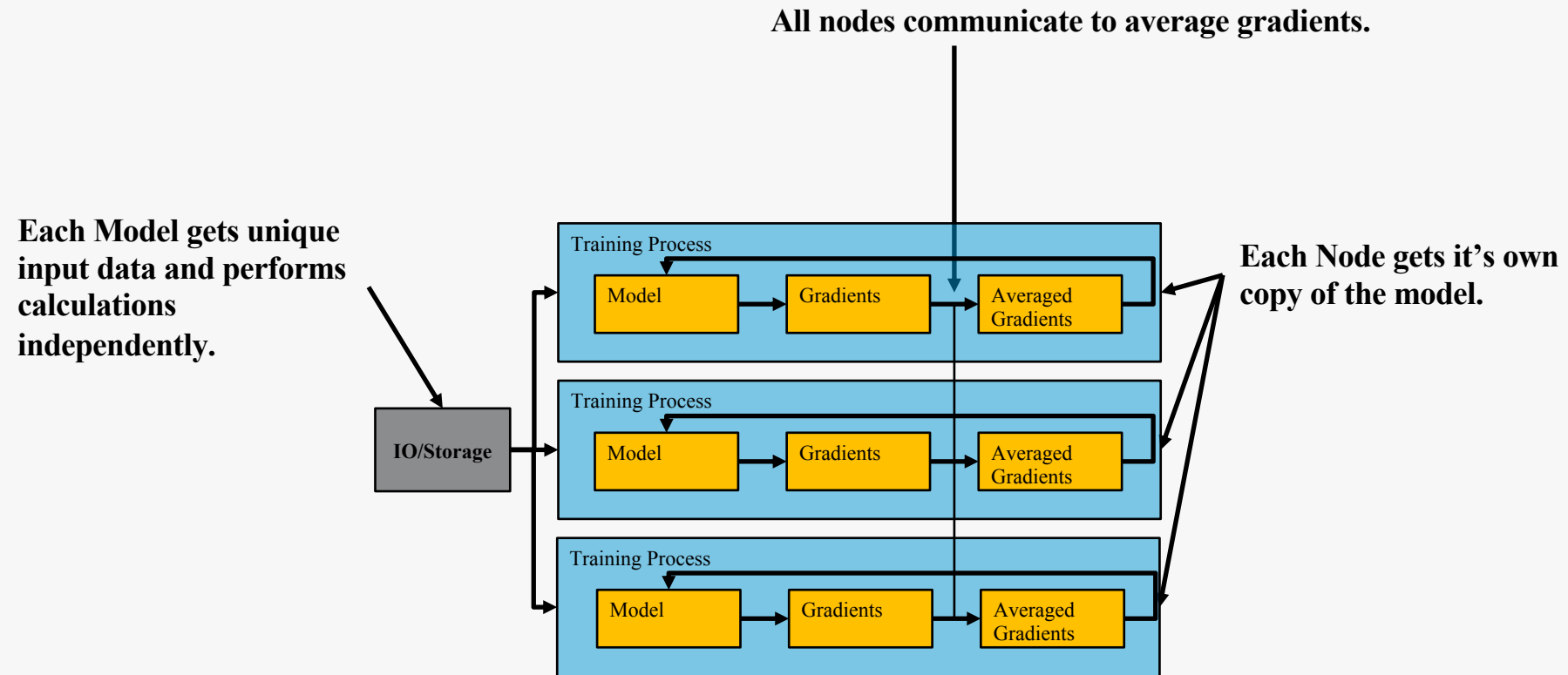


Image from Uber's Horovod: <https://eng.uber.com/horovod/>

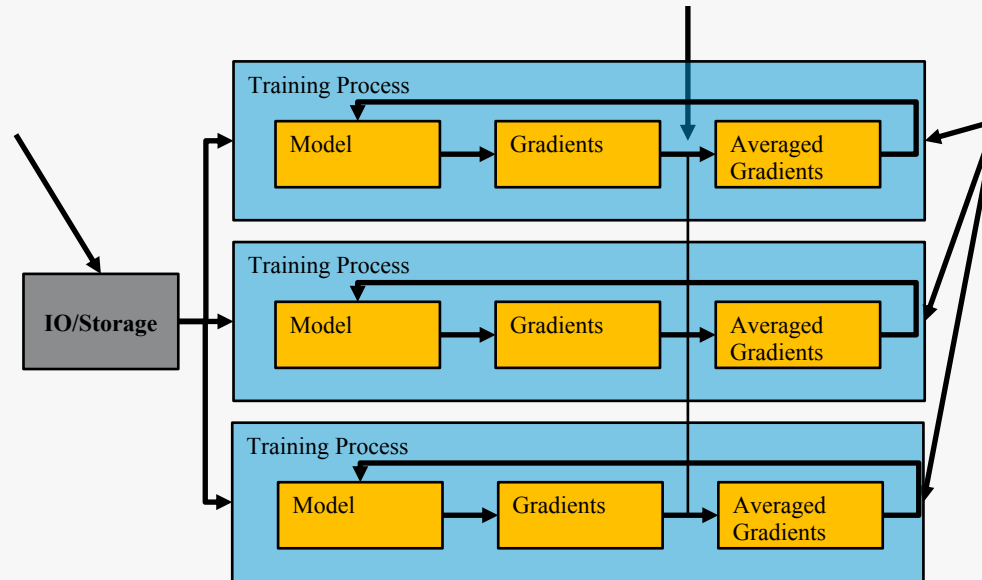
# Data Parallel Learning

## Scaling Challenges

Each Model gets unique input data and performs calculations independently.

IO requires organization to ensure unique batches.

IO contention with many nodes requires parallel IO solutions



All nodes communicate to average gradients.

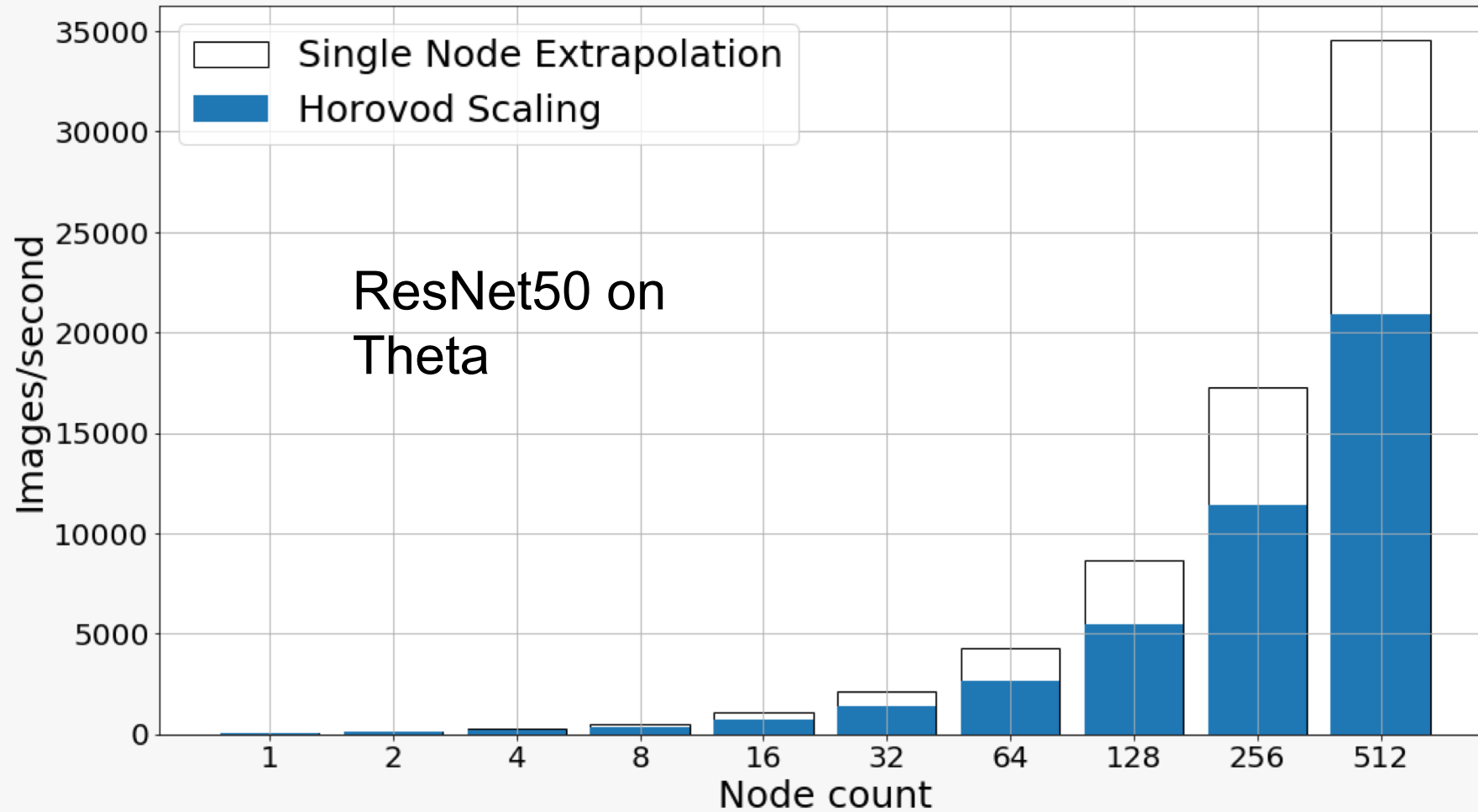
Computation stalls during communication: keeping the communication to computation ratio small is important for effective scaling.

Each Node gets it's own copy of the model.

Initialization must be identical or synchronized, and checkpointing/summary information must be managed with just one node.

Image from Uber's Horovod: <https://eng.uber.com/horovod/>

# Data Parallel Learning



# Data Parallel Learning

## Horovod

The simplest technique for data parallel learning

---

Initialize horovod ( `hvd.init()` ).

Wrap the optimizer in `hvd.DistributedOptimizer`.

- This uses the underlying optimizer for gradient calculations, and performs an averaging of all gradients before updating.
- Can adjust the learning rate to account for a bigger batch size.

Initialize the networks identically, or broadcast one network's weights to all others.

Ensure snapshots and summaries are only produced by one rank.

Horovod focuses on handling collective communication so you don't have to, but lets you use all of the tools of your favorite framework. Compatible with mpi4py.



**Horovod is an open source data parallel training software compatible with many common deep learning frameworks.**

**[Meet Horovod](#)**  
**[Github](#)**

# Horovod Example Code

## Tensorflow

```
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
```



# Horovod Example Code

## Keras

```
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))
```

# Horovod Example Code

## Pytorch

```
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())
```

# Effects of Distributed Learning

Increased Batch size means improved estimate of gradients.

- Scale by N nodes?  $\text{Sqrt}(N)$ ?
- Scale in a layerwise way? See paper: [Layerwise Adaptive Rate Scaling \(LARS\)](#)

Increased learning rate can require warm up iterations.

- See paper: [Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour](#)

Bigger minibatch means less iterations for the same number of epochs.

- May need to train for more epochs if another change is not made like boosting the learning rate.

# Mesh Learning

When data-parallel isn't enough...

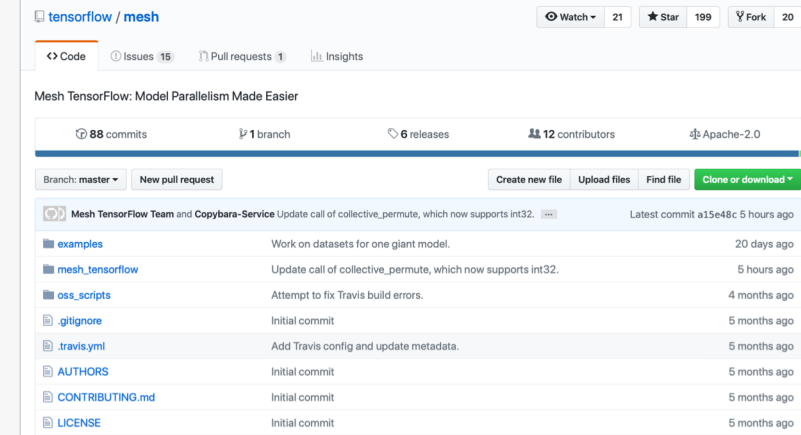
## Why might you need a Mesh?

- Memory limitations due to CNN size (number of parameters)
- Memory limitations due to input size (massive images, 3D volumes, etc)

## Mesh Scaling is not trivial:

- Computations need to be distributed in an intelligent way to prevent idle nodes
- Communication needs to happen frequently during both the forward/backward pass
- Message passing organization details arise from forward/backward small-group communications and multi-group communications

**Expect mesh scaling to get easier over the next few years (or wait for bigger, more powerful nodes?)**



tensorflow / mesh

Code Issues 15 Pull requests 1 Insights

Mesh TensorFlow: Model Parallelism Made Easier

88 commits 1 branch 6 releases 12 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Commit Message	Latest commit
examples	Work on datasets for one giant model.	20 days ago
mesh_tensorflow	Update call of collective_permute, which now supports int32.	5 hours ago
oss_scripts	Attempt to fix Travis build errors.	4 months ago
.gitignore	Initial commit	5 months ago
.travis.yml	Add Travis config and update metadata.	5 months ago
AUTHORS	Initial commit	5 months ago
CONTRIBUTING.md	Initial commit	5 months ago
LICENSE	Initial commit	5 months ago

## Tensorflow Mesh

<https://github.com/tensorflow/mesh>



# Measuring Performance for Machine Learning Workflows

# Performance Measurements – Deep Learning

How to measure performance for tensorflow/pytorch?

---

**Deep Learning workflows typically are diverse in requirements:**

- Start in python
- Call upon IO libraries to read all or part of a dataset
- Feed data into an optimized (compared to python) library for ML/DL algorithms
- Use Horovod to communicate between nodes and average gradients

**Many different pieces benefit from different profiling techniques:**

- Timing based profiling (global\_step/second, images/second)
- Python line based profiling (cProfile)
- Advanced Profiling Tools (Vtune, Advisor)



# Time Stamp “Profiling”

Timing printouts are the first step for understanding performance of training algorithms for deep learning. From one of my own applications, I catch time stamps for:

- forward/backward pass of the network
- time required for IO
- time required to synchronize gradients across nodes:

```
train Step 363 metrics: loss: 2.21, accuracy: 0.961 (1.5e+01s / 0.066 IOs / 3.0)
train Step 364 metrics: loss: 2.14, accuracy: 0.962 (1.6e+01s / 0.053 IOs / 3.2)
train Step 365 metrics: loss: 2.09, accuracy: 0.96 (1.5e+01s / 0.053 IOs / 3.0)
train Step 366 metrics: loss: 2.1, accuracy: 0.963 (1.5e+01s / 0.06 IOs / 3.0)
```

## Pros

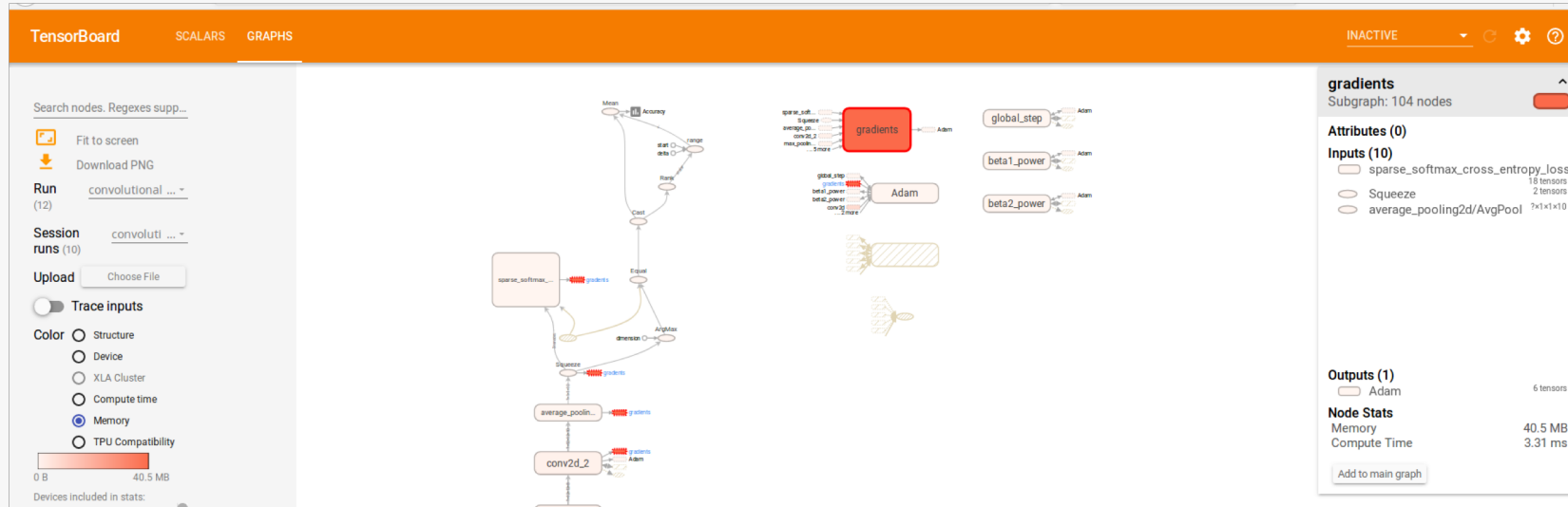
- Very easy using `datetime.datetime.now()`
- Trivial to analyze in the log files
- Can give a good top-level, cross software/system comparison using **images/second** or **global-step/second** for the entire application
- System Independent (laptop vs. HPC node, etc)

## Cons

- Not useful for finding hotspots, only for monitoring known blocks of easily separable code
- Overly coarse and useless for optimizations, only for monitoring for problems

# Tensorboard Profiling

For Tensorflow applications, you can visualize tensorflow application performance for each node of your graph using tensorboard, as well chrome traces.



## Pros

- Gives a good idea of what nodes in your graph are most resource intensive (memory usage, computation time)
- Pretty easy to setup and use via `tf.train.ProfilerHook`

## Cons

- Can be difficult to analyze graphical form in tensorboard, compared to sorted lists of operations in other profilers
- Doesn't reveal hardware utilization metrics or performance.
- Profiling only available for tensorflow

# Python cProfile

For the diverse set of workflows you need to stitch together with python, it can be very useful to use python's built in profiling module **cProfile**:

```
python -m cProfile -o cprofile_data.prof script.py
```

Generates a list of function calls, time spent, number of calls, etc. Lots of open source tools for interpreting and analyzing the results, such as [here](#), [here](#), and [here](#)

```
>>> import pstats
>>> p = pstats.Stats("cprofile_data.prof")
>>> p.sort_stats("time").print_stats(3)
Fri Apr 5 20:13:02 2019      cprofile_data

      1431679 function calls (1401373 primitive calls) in 673.782 seconds

Ordered by: internal time
List reduced from 3212 to 3 due to restriction <3>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     50   258.029    5.161   258.029    5.161 {method 'run_backward' of 'torch._C._EngineBase' objects}
    2050   176.755    0.086   176.755    0.086 {built-in method
sparseconvnet.SCN.SubmanifoldConvolution_updateOutput}
    32/31    88.808    2.775    88.909    2.868 {built-in method _imp.create_dynamic}
```

# Python cProfile



## Pros

- It's open source, native python, extremely easy to use.
- A lot of tools available for results interpretation.

## Cons

- Doesn't go beyond python lines.
- Despite available tools, relatively high effort required to make sense of the results.

# Application Performance Snapshot

APS generates a highlevel performance snapshot of your application. Easy to run:

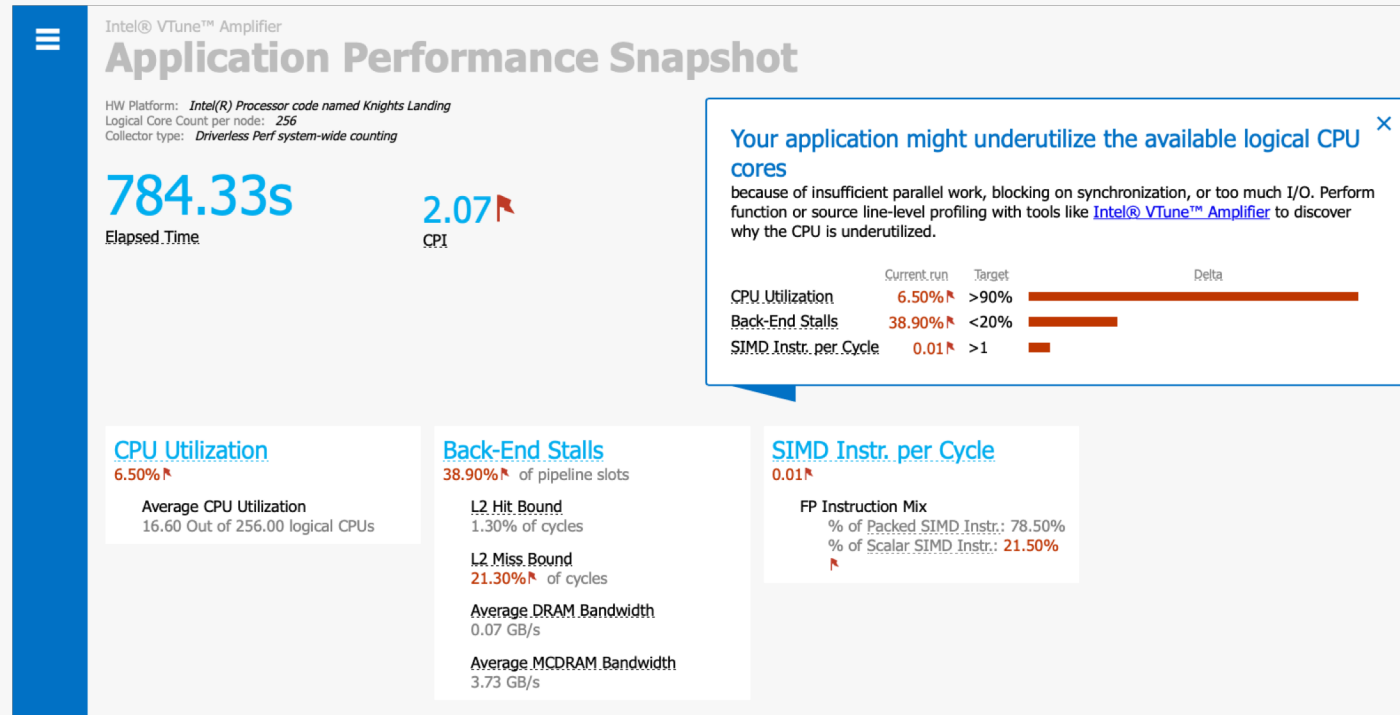
```
source /opt/intel/vtune_amplifier/apsvars.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64/
export PMI_NO_FORK=1
```

```
aps --result-dir=aps_results/ python /full/path/to/script.py
```

Results can be viewed in a single html file, or via command line:

```
| Summary information
|-----
| HW Platform           : Intel(R) Processor code named Knights Landing
| Logical core count per node: 256
| Collector type       : Driverless Perf system-wide counting
| Used statistics      : aps_results
|
| Your application might underutilize the available logical CPU cores
| because of insufficient parallel work, blocking on synchronization, or too much I/O. Perform function or source
| line-level profiling with tools like Intel(R) VTune(TM) Amplifier to discover why the CPU is underutilized.
| CPU Utilization:           6.50%
| Your application might underutilize the available logical CPU cores because of
| insufficient parallel work, blocking on synchronization, or too much I/O.
| Perform function or source line-level profiling with tools like Intel(R)
```

# Application Performance Snapshot



## Pros

- Very easy to use
- Tracks important hardware metrics:
  - Thread Load Balancing
  - Vectorization
  - CPU Usage

## Cons

- Only high level information – but then again, that is the design of this tool.



# Intel Vtune – Advanced Hotspots

Vtune advanced hotspots can give a very useful report of what your CPUs are doing, how effectively they are running, etc. Slightly more involved to use:

```
source /opt/intel/vtune_amplifier/apsvars.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64/
export PMI_NO_FORK=1

amplxe-cl -collect advanced-hotspots -finalization-mode=none -r vtune-result-dir_advancedhotspots/ python /full/path/to/script.py
```

You don't have to, but **should** run the finalization after the run completes (do this from the **login nodes**):

```
source /opt/intel/vtune_amplifier/apsvars.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64/
export PMI_NO_FORK=1

amplxe-cl -finalize -search-dir / -r vtune-result-dir_advancedhotspots

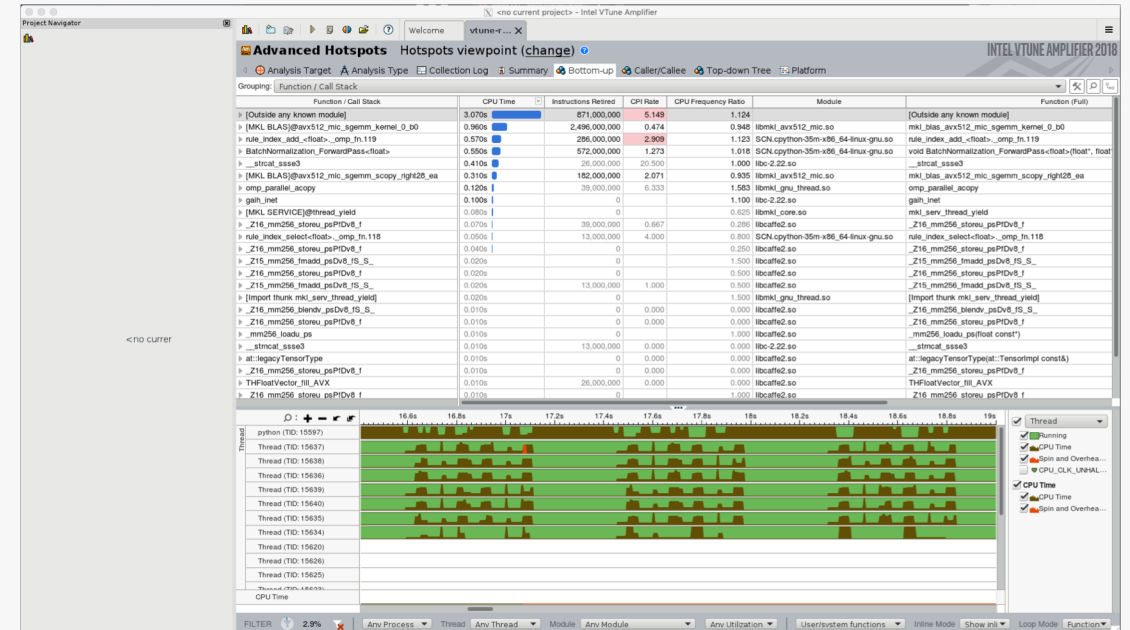
source /opt/intel/vtune_amplifier/apsvars.sh
amplxe-gui vtune-result-dir_advancedhotspots
```

# Intel Vtune – Advanced Hotspots

Run the GUI to view your results:

```
source /opt/intel/vtune_amplifier/apsvars.sh  
amplxe-gui vtune-result-dir_advancedhotspots
```

Function / Call Stack	CPU Time	Instructions Retired	CPI Rate
[Outside any known module]	3.070s	871,000,000	5.149
[MKL BLAS]@avx512_mic_sgemm_kernel_0_b0	0.960s	2,496,000,000	0.474



## Pros

- You can see the activity of each thread, and the functions that cause it.
- Give a bottom up and top down view, very useful for seeing which functions are hotspots and which parts of your workflow are dominant.
- **Allows line by line analysis of source code.**

## Cons

- **Doesn't keep information at python level.**
- If your workflow uses JIT, you can lose almost all useful information.
- Understanding the information present takes some practice.

# Intel Vtune – Hotspots

Vtune hotspots is similar to advanced hotspots but keeps python information – very very useful for profiling.

```
source /opt/intel/vtune_amplifier/apsvars.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64/
export PMI_NO_FORK=1

amplxe-cl -collect hotspots -finalization-mode=none -r vtune-result-dir_hotspots/
```

## Pros

- Similar benefits as hotspots
- **Additionally, allows you to track activity from python code**
- Same finalization techniques and gui as advanced hotspots

## Cons

- Will **not** run with more than a few threads, making it impossible to profile the “real” application.

# Intel Vtune – Hotspots

**Basic Hotspots** Hotspots by CPU Utilization viewpoint (change)

Analysis Target: vtune-r... | Analysis Type: Welcome | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
sgemm	21.509s	libmkl_intel_ip64.so	sgemm		0x149120
BatchNormalization_ForwardPass<float>	9.278s	SCN.cpython-35m-x86_64-linux-gnu.so	void BatchNormalization_ForwardPass<float>(float*, float*, int, int, int, float*, float...	BatchNormalization.cpp	0xb2910
rule_index_add<float>._omp_fn.119	8.388s	SCN.cpython-35m-x86_64-linux-gnu.so	rule_index_add<float>._omp_fn.119	Convolution.cpp	0xa5600
__memcpy_avx512_no_vzeroupper	6.060s	libc.so.6	__memcpy_avx512_no_vzeroupper		0x93360
PyCFunction_Call	5.954s	libpython3.5m.so.1.0	PyCFunction_Call	methodobject.c	0xc2e00
llvm::DWARFDebugInfoEntryMinimal::extractFast	5.217s	libpin3dwarf.so	llvm::DWARFDebugInfoEntryMinimal::extractFast(llvm::DWARFUnit const*, unsigned...		0x4aa70
memcmp	4.903s	libc-dynamic.so	memcmp		0x1c0d0
llvm::DWARFUnit::extractDIEsToVector	3.129s	libpin3dwarf.so	llvm::DWARFUnit::extractDIEsToVector(bool, bool, std::...1::vector<llvm::DWARFDe...		0x3a8f0
_Z16_mm256_storeu_psPIDv8_f	3.025s	libcaffe2.so	_Z16_mm256_storeu_psPIDv8_f	avxintrin.h	0xc63b30
memmove	2.528s	libc-dynamic.so	memmove		0x1c260
_Z16_mm256_storeu_psPIDv8_f	2.220s	libcaffe2.so	_Z16_mm256_storeu_psPIDv8_f	avxintrin.h	0xc63b20
std::_1::tree<std::_1::value_type<std::_1::basic_string<...	1.738s	libpin3dwarf.so	std::_1::tree_iterator<std::_1::value_type<std::_1::basic_string<char, std::...		0x397b0
llvm::DataExtractor::getCStr	1.560s	libpin3dwarf.so	llvm::DataExtractor::getCStr(unsigned int*) const		0x4a060
[ld-linux-x86-64.so.2]	1.488s	ld-linux-x86-64.so.2	[ld-linux-x86-64.so.2]		0
operator new	1.278s	libpin3dwarf.so	operator new(unsigned long)	new.cpp	0xbb3a0
llvm::DataExtractor::getULEB128	1.181s	libpin3dwarf.so	llvm::DataExtractor::getULEB128(unsigned int*) const		0x4a0b0
OS_BARESYS CALL_DoCallAsmIntel64Linux	1.178s	libc-dynamic.so	OS_BARESYS CALL_DoCallAsmIntel64Linux		0x70e5c
llvm::DWARFUnit::setDIERelations	1.104s	libpin3dwarf.so	llvm::DWARFUnit::setDIERelations(void)		0x3a7c0
llvm::DWARFDebugInfoEntryMinimal::getAttributeValue	1.056s	libpin3dwarf.so	llvm::DWARFDebugInfoEntryMinimal::getAttributeValue(llvm::DWARFUnit const*, uns...		0x4ac00
_Z16_mm256_storeu_psPIDv8_f	0.894s	libcaffe2.so	_Z16_mm256_storeu_psPIDv8_f	avxintrin.h	0xc63b35
llvm::StringMapImpl::LookupBucketFor	0.840s	libpin3dwarf.so	llvm::StringMapImpl::LookupBucketFor(llvm::StringRef)		0x44100
llvm::DWARFAbbreviationDeclarationSet::getAbbreviationDeclar	0.660s	libpin3dwarf.so	llvm::DWARFAbbreviationDeclarationSet::getAbbreviationDeclaration(unsigned int) co...		0x52350
llvm::DWARFFormValue::skipValue	0.620s	libpin3dwarf.so	llvm::DWARFFormValue::skipValue(unsigned short, llvm::DataExtractor, unsigned int...		0x4be90
CBLAS_DCOPIY	0.580s	libmkl_rt.so	CBLAS_DCOPIY		0x137e30
llvm::DWARFAbbreviationDeclaration::findAttributeIndex	0.540s	libpin3dwarf.so	llvm::DWARFAbbreviationDeclaration::findAttributeIndex(unsigned short) const		0x4cfa0
std::_1::tree<std::_1::value_type<std::_1::basic_string<...	0.520s	libpin3dwarf.so	std::_1::tree_iterator<std::_1::value_type<std::_1::basic_string<char, std::...		0x398c0

Timeline: Thread (python (TID: 42839), python (TID: 42896), sh (TID: 42896), sh (TID: 42913), uname (TID: 42913)) CPU Utilization (100.0%)

# Profiling Example – Tensorflow FFTs

One user reported very very slow performance with tensorflow on Theta, even though they were using all of the optimized settings. Using vtune hotspots and advanced hotspots, we discovered (for a shortened run):

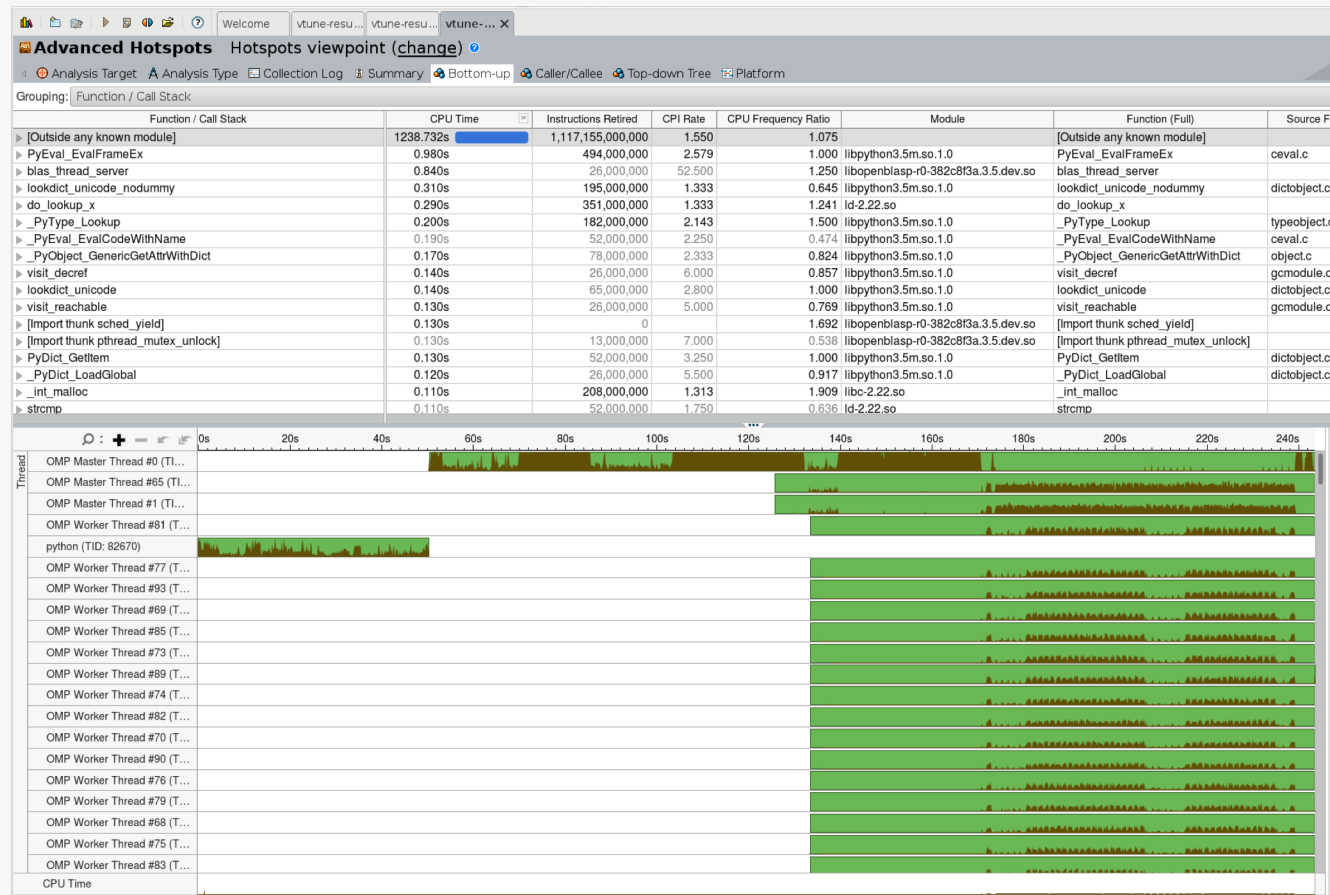
- 31% of the application time was spent doing FFTs with tensorflow
- 10% was spent creating tensorflow traces
- 8% was computing loss functions.
- 25% was spent creating and optimizing the tensorflow graph (measured for a short run, this is a smaller fraction for production runs)

Talking with Intel engineers revealed that the most important hotspot (**FFT**) **was underperforming on Theta by up to 50x compared with the optimized FFT in Numpy.**

For this workflow, replacing tensorflow with numpy FFT + autograd for gradient calculations made a huge impact in their performance.

# Profiling Example – Tensorflow CNN

A user reported seeing a significant degradation in performance with tensorflow when going from single image to multi-image batches.

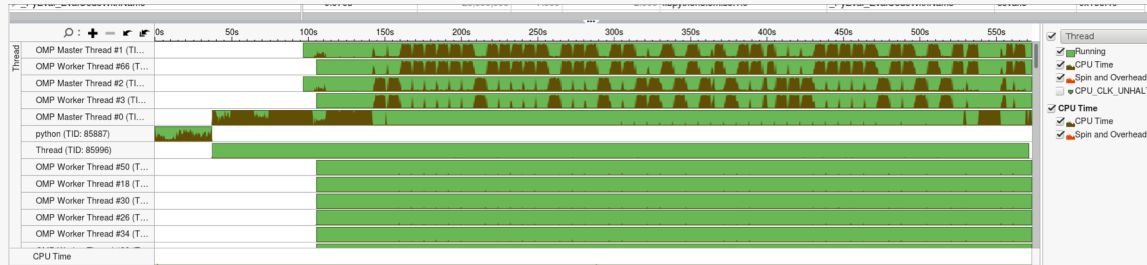


Batch Size 1 showed decent balance between threads, even if utilization was lower than ideal.

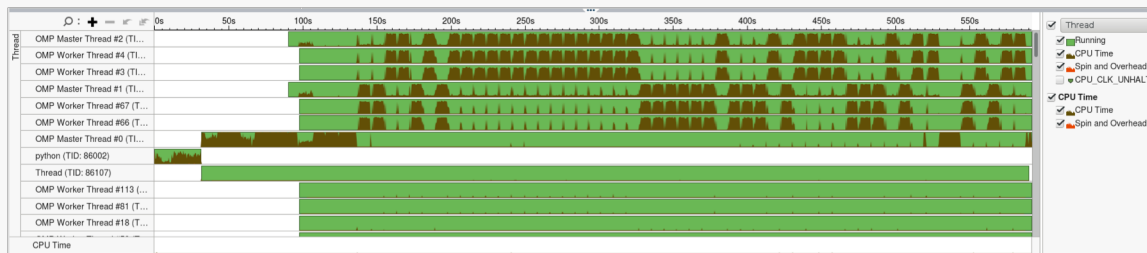


# Profiling Example – Tensorflow CNN

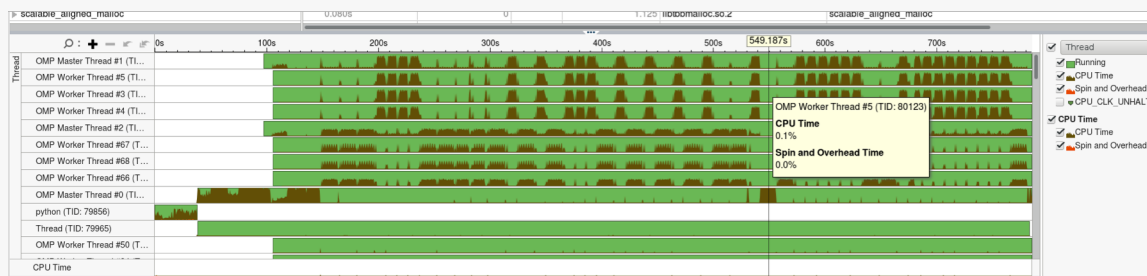
A user reported seeing a significant degradation in performance with tensorflow when going from single image to multi-image batches.



Batch Size 2



Batch Size 3

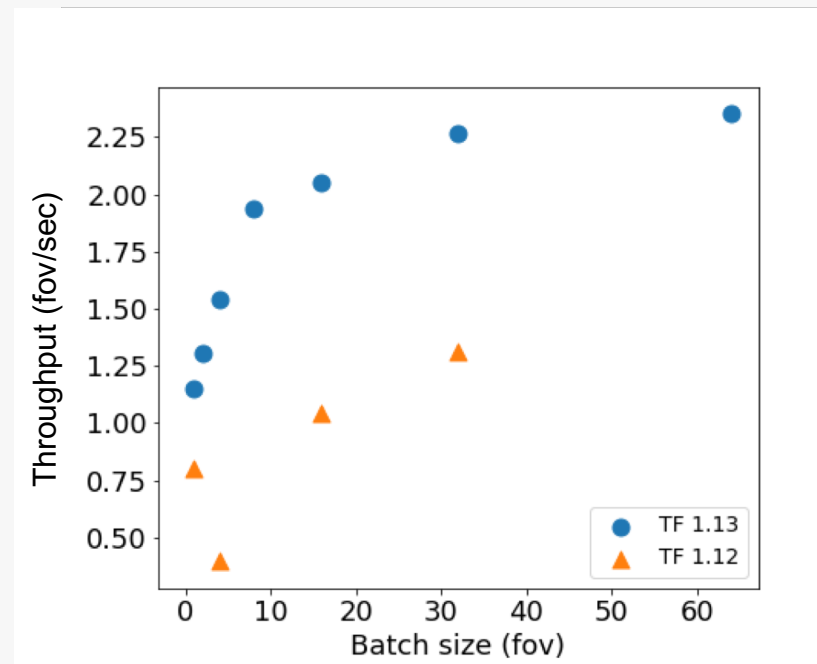


Batch Size 4

# Profiling Example – Tensorflow CNN

As seen above, the parallelization of operations broke when batch size was increased beyond 1.

Appeared to be a bug in tf1.12 on CPUs, but resolved in tf1.13:



# Conclusions

- The ALCF Datascience group supports a variety of machine learning and deep learning workflows for performant computing.
- You should tell us if your workflow doesn't fit into our supported tools, we want to know!
- Machine Learning and Deep Learning is an extremely important workflow for current and future HPC.
- Tools for parallelization are good and getting better.
- Modern frameworks (TF, torch) are big and difficult to squeeze for peak performance without systematic study and profiling.
- Profiling of complex workflows glued together with python are becoming more common, expect profiling tools to adapt to this as well.

**Thank you!**

**Questions?**

**Reach out to me:  
corey.adams@anl.gov**