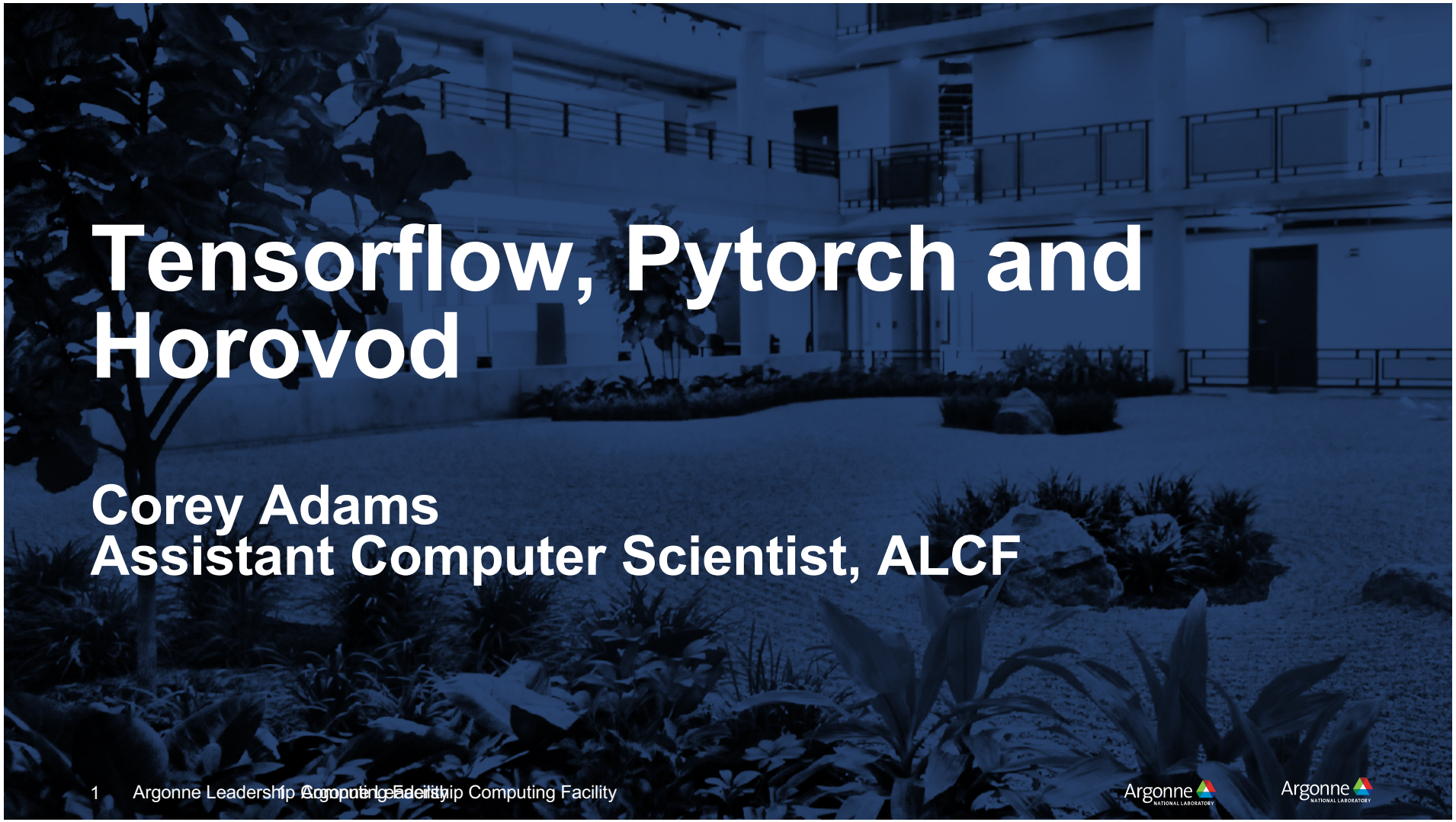


Tensorflow, Pytorch and Horovod

Corey Adams
Assistant Computer Scientist, ALCF

Argonne Leadership Computing Facility

datascience@alcf.anl.gov


Corey Adams

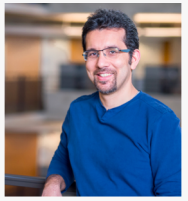

Prasanna Balaprakash


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

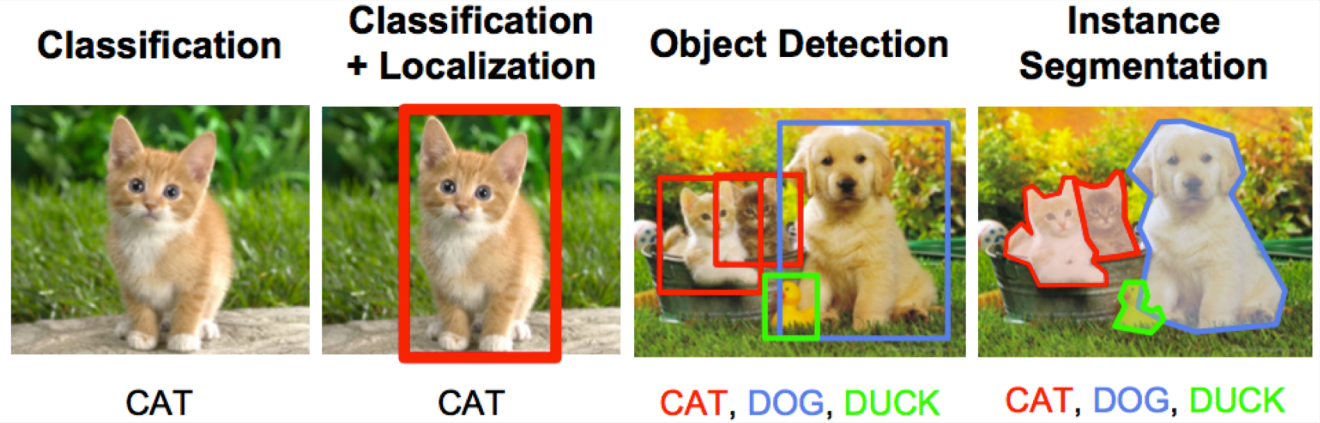

Huihuo Zheng

ALCF 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.

What is Deep Learning?

Photo from Analytics Vidhya



https://thispersondoesnotexist.com

Deep learning is …

- an ~~emerging~~ exploding field of research that is transforming how we do science.
- able to outperform humans on many complex tasks such as classification, segmentation, regression
- able to replace data and simulation with hyper realistic generated data
- expensive and time consuming to train

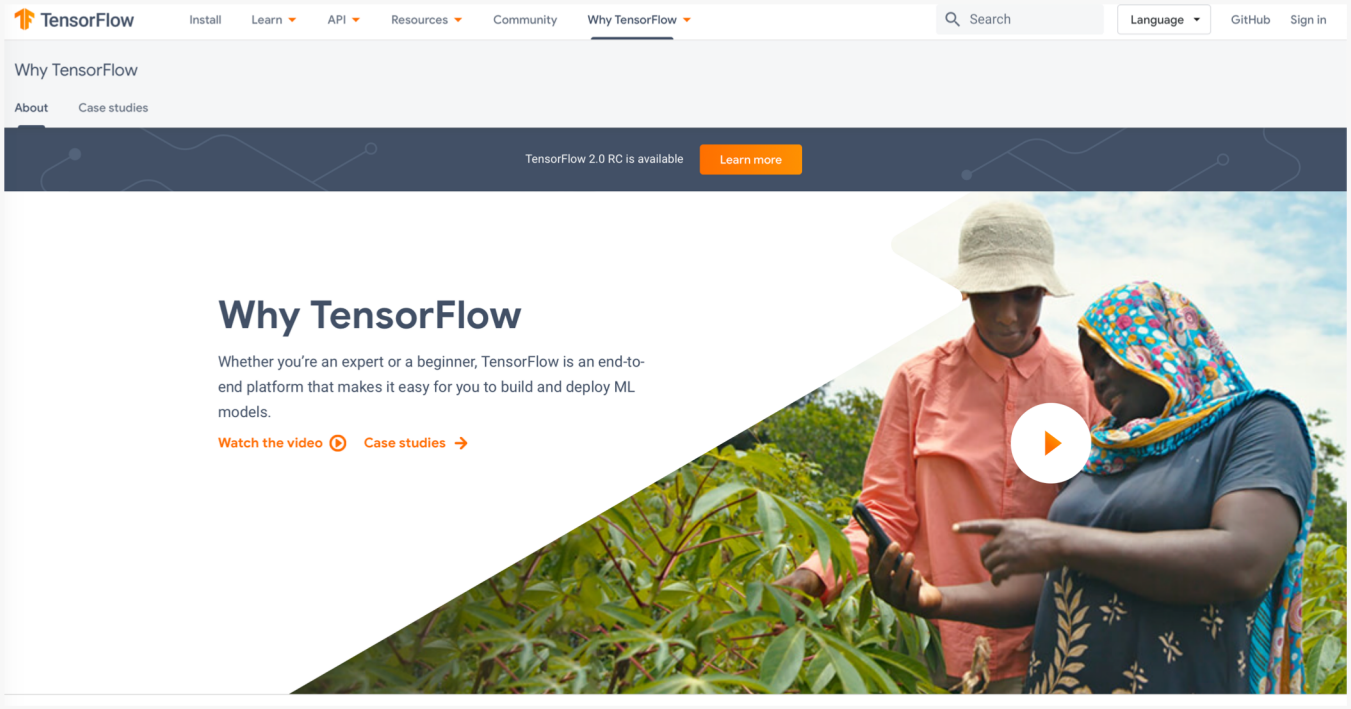
Deep Learning and Machine Learning on Aurora

Aurora will be an Exascale system highly optimized for Deep Learning

- Intel's discrete GPUs (X^e) will drive accelerated single node performance
- Powerful CPUs will keep your GPUs fed with data and your python script moving along quickly
- High performance interconnect will let you scale your model training and inference to large scales.
- Optimized IO systems will ensure you can keep a distributed training fed with data at scale.

This talk: deep learning frameworks are already run on supercomputers. We'll cover all of the fundamentals of these frameworks.

Tensorflow



https://www.tensorflow.org

Tensorflow

Tensorflow is high level framework for gluing together math operations in a way that is useful for machine learning. Tensorflow supports predominantly math operations relevant to neural networks, such as:

- Convolutions
- Dense Layers
- Activations
- Normalization Layers
- Pooling and Un-Pooling layers
- Reshaping, concatenation, splitting, and other tensor manipulation functions
- Loss functions and core math functions
- … many others

Tensorflow abstracts away the details of applying these operations to allow users to spend most of their time on what matters: developing their models and applications

Tensorflow comes with several “backends” that have highly optimized kernels for executing individual operations. Different back ends for CPU, GPU(nvidia), and there will be a performant backend for A21 GPUs.

- Not all operations are as optimized as others, so your mileage may vary if you need performance but use non-standard ops.

Tensorflow Example 1 - Keras

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

# Load the dataset and cast to a normalized floating point image:
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([      tf.keras.layers.Flatten(input_shape=(28, 28)),
          tf.keras.layers.Dense(128, activation='relu'),
          tf.keras.layers.Dropout(0.2),
          tf.keras.layers.Dense(10, activation='softmax') ])
model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=64) model.evaluate(x_test, y_test)
```

Tensorflow – Example 2 (Part 1/3)

```
import tensorflow as tf
import numpy
import time
# Enable eager execution
tf.enable_eager_execution()

mnist = tf.keras.datasets.mnist

# Load the dataset and cast to the right formats:
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

x_train, x_test = x_train.reshape([60000,28,28,1]), x_test.reshape([10000,28,28,1])
y_train, y_test = y_train.astype(numpy.int32), y_test.astype(numpy.int32)
```

Tensorflow – Example 2 (Part 2/3)

```
class MyModel(tf.keras.Model):
        def __init__(self):
                super(MyModel, self).__init__()
                self.conv1 = tf.keras.layers.Conv2D(32, 3, activation='relu')
                self.flatten = tf.keras.layers.Flatten()
                self.d1 = tf.keras.layers.Dense(128, activation='relu')
                self.d2 = tf.keras.layers.Dense(10, activation='softmax')

        def call(self, x):
                x = self.conv1(x)
                x = self.flatten(x)
                x = self.d1(x)
                return self.d2(x)

# Create an instance of the model
model = MyModel()
# Use a list of indexes to shuffle the dataset each epoch
indexes = numpy.arange(len(x_train))
epochs = 5;  batch_size = 64
# Create an instance of an optimizer:
optimizer=tf.train.AdamOptimizer()
```

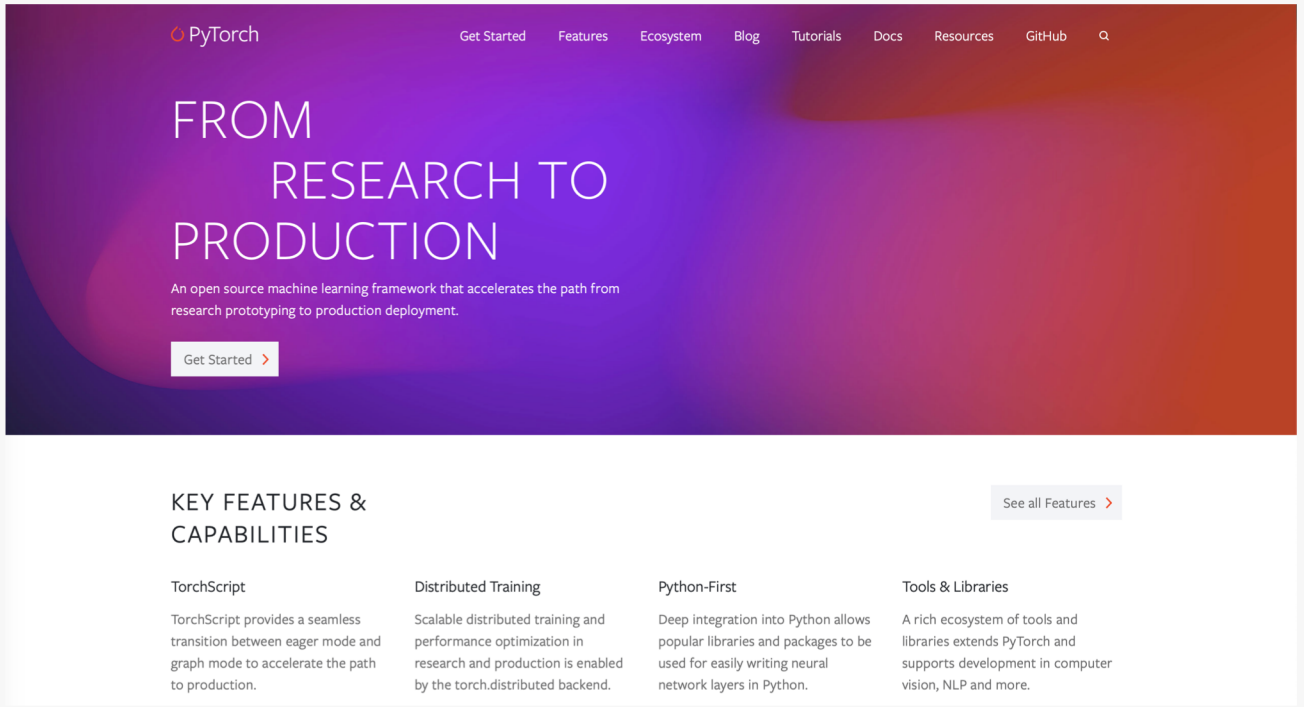
Tensorflow – Example 2 (Part 3/3)

```
for epoch in range(5):
        # Shuffle the indexes:
        numpy.random.shuffle(indexes)
        for batch in range(batch_size):
                batch_indexes = indexes[batch*batch_size:(batch+1)*batch_size]
                images, labels = x_train[batch_indexes], y_train[batch_indexes].reshape([batch_size,])

                # Gradient tape indicates to TF to build a graph on the fly.
                with tf.GradientTape() as tape:
                        # This line is the forward pass of the network:
                        # (The first call to model will initialize the weights)
                        logits = model(images)
                        # Loss value is computed imperatively
                        loss_value = tf.losses.sparse_softmax_cross_entropy(logits=logits, labels=labels)

                # Compute the backward pass with the gradient tape:
                grads = tape.gradient(loss_value, model.trainable_variables)
                # Use the optimizer to update the gradients:
                optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Thoughts on Tensorflow for HPC

- Tensorflow is heavily optimized, but python is not.
  - As much as possible, do preprocessing and IO with optimized libraries (numpy, hdf5, tensorflow itself)
  - Use the latest versions of tensorflow
    - They often include performance improvements, sometimes very dramatic.
- Tensorflow (standard) builds and compiles a computation graph
  - Can do operation merging such as BatchNorm + ReLU
    - Still possible with eager execution
- Eager Execution is already available but may be the default by Aurora
  - Conceptually simpler, but can still get full acceleration
- Can be challenging to get full utilization of an accelerator
- IO can easily become a bottleneck – more later..

PyTorch



https://pytorch.org

PyTorch Details

- Pytorch is fully pythonic: no Sessions, no graphs, no fit functions, etc.
- **With great power comes great responsibility:**
  - Pro: you can do nearly anything in the ML/DL space
  - Con: you have to do many things deliberately

"At its core, PyTorch provides two main features:
An n-dimensional Tensor, similar to numpy but can run on GPUs
Automatic differentiation for building and training neural networks"

* There are also a lot of really useful predefined tools for building and training models, getting fine grained control flow and conditionals (Useful for RNNs). Also supports optimized static models.

PyTorch Example (1/4)

```
import torch
import torchvision
import numpy

train_set = torchvision.datasets.MNIST('./', train=True, download=True, transform
= torchvision.transforms.ToTensor())
test_set = torchvision.datasets.MNIST('./', train=False, download=True, transform
= torchvision.transforms.ToTensor())
# We can get directly at the tensors:
x_test = test_set.data.reshape([10000,1,28,28]).type(torch.FloatTensor)
y_test = test_set.targets
x_train = train_set.data.reshape([60000,1,28,28]).type(torch.FloatTensor)
y_train = train_set.targets
```

PyTorch Example (2/4)

```
class MyModel(torch.nn.Module):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)
    self.pool1 = torch.nn.MaxPool2d(kernel_size=2)
    self.conv2 = torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
    self.pool2 = torch.nn.MaxPool2d(kernel_size=2)
    self.d1 = torch.nn.Linear(in_features=64*5*5, out_features=128)
    self.d2 = torch.nn.Linear(in_features=128, out_features=10)

  def forward(self, x):
    x = self.conv1(x)
    x = torch.relu(x)
    x = self.pool1(x)
    x = self.conv2(x)
    x = torch.relu(x)
    x = self.pool2(x)
    x = torch.flatten(x, 1, -1)
    x = self.d1(x)
    x = torch.relu(x)
    return self.d2(x)
```

PyTorch Example (3/4)

```
# Create an instance of the model
model = MyModel()

# Use a list of indexes to shuffle the dataset each epoch
indexes = numpy.arange(len(train_set))

epochs = 5
batch_size = 128

# Create an instance of an optimizer:
optimizer=torch.optim.Adam(model.parameters())

loss_operation = torch.nn.CrossEntropyLoss()
```

PyTorch Example (4/4)

```
for epoch in range(epocs):
  # Shuffle the indexes:
  numpy.random.shuffle(indexes)
  for batch in range(len(indexes/batch_size)):
    if (batch+1)*batch_size > 60000:
      continue

    batch_indexes = indexes[batch*batch_size:(batch+1)*batch_size]
    images = x_train[batch_indexes]
    labels = y_train[batch_indexes].reshape([batch_size,])

    # Set the model to training mode:
    model.train()
    # Reset the gradient values for this step:
    optimizer.zero_grad()
    # Compute the logits:
    logits = model(images)
    # Loss value is computed imperatively
    loss = loss_operation(input=logits, target=labels)
    # This call performs the back prop:
    loss.backward()
    # This call updates the weights using the optimizer
    optimizer.step()
```

Tensorflow or Pytorch – Which should I use?

It depends on your application and preference.

1. **Performance:** Each can have slightly better performance on different model types, in particular pytorch is more suited for RNNs because of it's dynamic graph building.
2. **Performance:** With big models on powerful accelerators, both frameworks perform very well with little difference on the same models.
    1. If using a custom model, mileage may vary…
3. **User Experience:** Pytorch is a bit easier to learn from numpy/python skills already acquired, but tensorflow has better documentation.
4. **User Experience:** Tensorflow is harder to debug a graph, but has tensorboard built in. Pytorch can use tensorboard with tensorboardX package.
5. **Ease of Use:** If you like to do everything yourself, pytorch is easier. If you like to have most pieces filled in for you, tensorflow (particularly keras) is easier.

Tensorflow, Pytorch but without python?

- Tensorflow has a C and C++ API for execution directly from lower level software
  - https://www.tensorflow.org/guide/extend/cc
  - (You can also extend tensorflow from C++ with new ops)

- Pytorch also has a C++ API:
  - https://pytorch.org/cppdocs/
  - You can similarly build new ops, leverage the tensors + autograd feature
  - You can also train a model in python, and run it directly at inference in C++

These frameworks have a host of optimized operations and can be very useful outside of python.

What happened to numpy?

(Nothing - only getting better!)

Numpy has been the reigning champion of performance in python - often has more diverse operations implemented, sklearn, etc.

- Numpy is not ignored in the machine learning space: https://github.com/google/jax
- **JAX** (formerly autograd) enables automatic differentiation of numpy operations

Numba accelerates python code in general (Including GPU optimizations for today's top GPUs)

- Does just-in-time compilation on python code, full integration with numpy, GPU offloading ...

What happened to numpy?

(Nothing - only getting better!)

Numpy has been the reigning champion of performance in python - often has more diverse operations implemented, sklearn, etc.

- Numpy is not
  https://github.
- JAX (formerly
  operations

Numpy + numba + JAX = core of TF, torch

numpy

Numba accelerates python code in general (Including GPU optimizations for today's top GPUs)

- Does just-in-time compilation on python code, full integration with numpy, GPU offloading ...

MKL and ~~MKL-DNN~~ DNNL

If you have heard of CUDA and CUDNN, this is not a surprise.

"**One Solution for Multiple Environments**

Intel® Math Kernel Library (Intel® MKL) optimizes code with minimal effort for future generations of Intel® processors. It is compatible with your choice of compilers, languages, operating systems, and linking and threading models"

"Deep Neural Network Library (DNNL) is an open-source performance library for deep learning applications. The library includes basic building blocks for neural networks optimized for Intel Architecture Processors and Intel Processor Graphics."

Intel is committed to delivering high performance versions of MKL~~, MKL-DNN~~ DNNL for the X^e accelerators with full integration into pytorch and tensorflow.

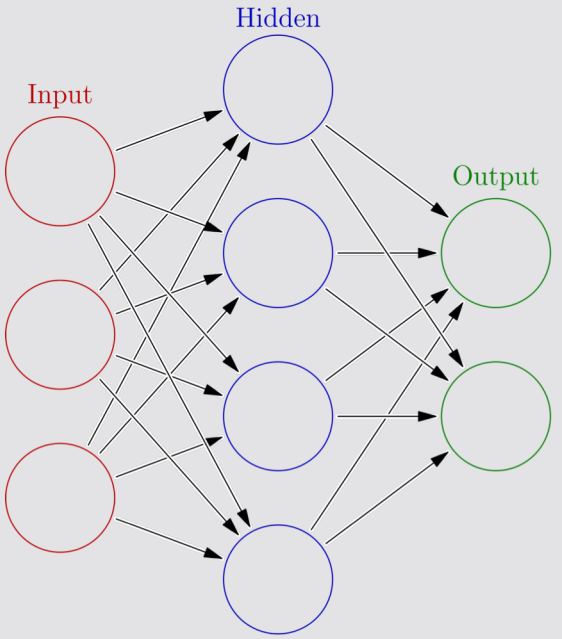
Machine Learning and HPC

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.

Distributed Learning with Horovod

Machine learning is a very important workflow for current and future supercomputing systems.

How can you accelerate learning with more computing power?

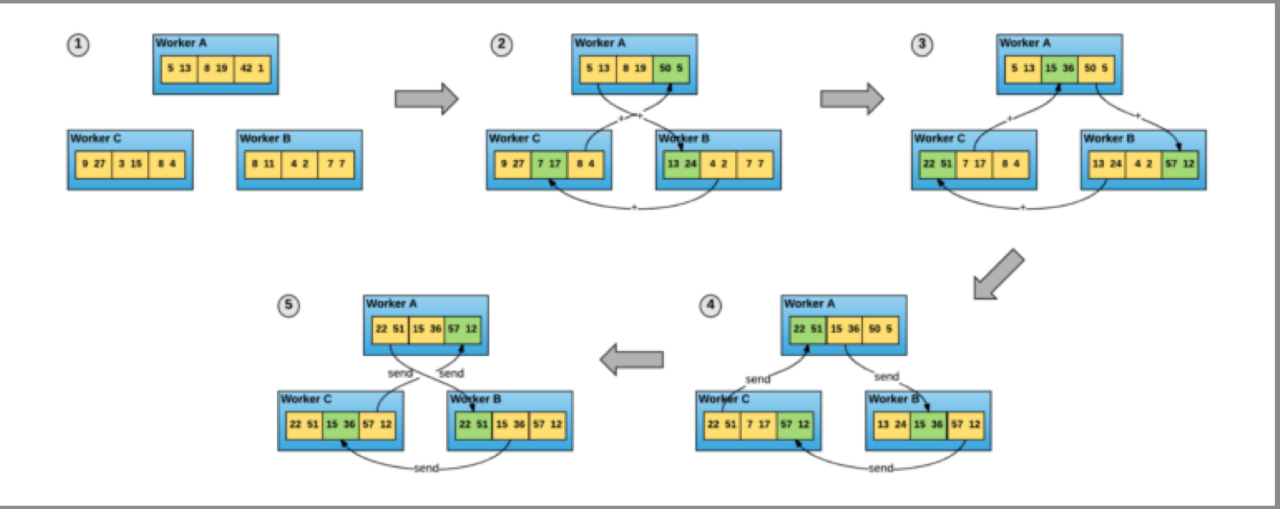


Image from Uber's Horovod: https://eng.uber.com/horovod/

What is Distributed Learning?

The backpropagation algorithm is unchanged at it's 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

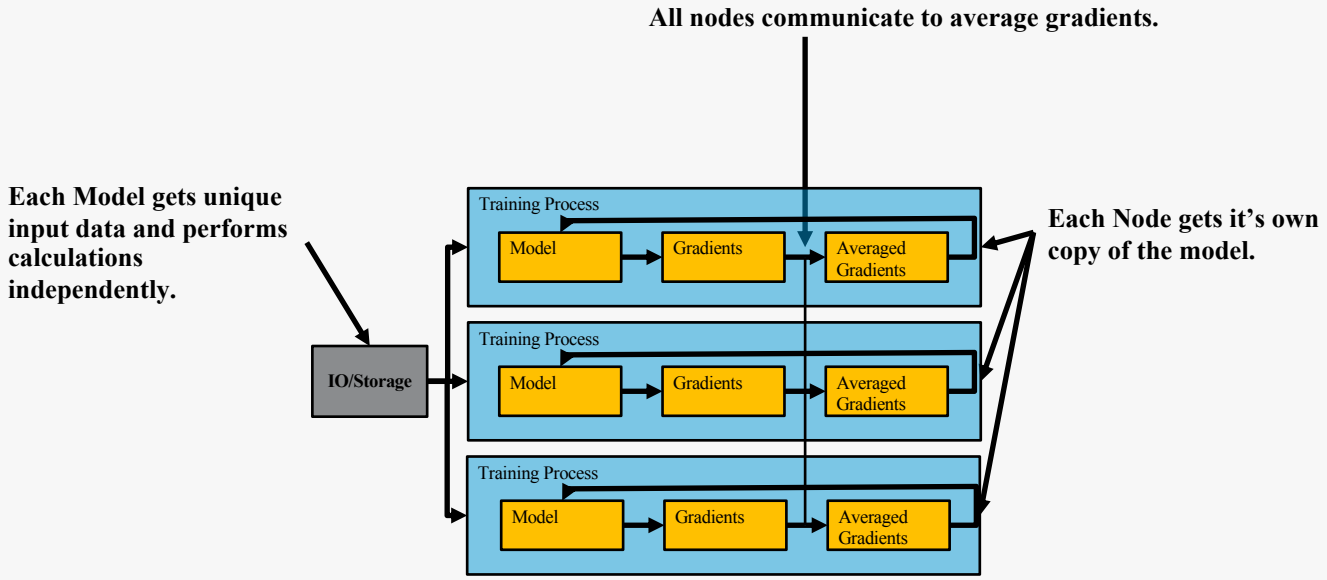


Image from Uber's Horovod: https://eng.uber.com/horovod/

Data Parallel Learning

All nodes communicate to average gradients.

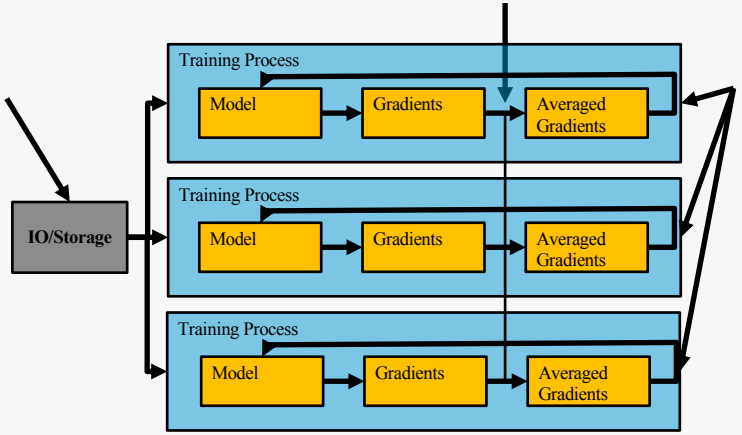
Computation stalls during communication: keeping the communication to computation ratio small is important for effective scaling.

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

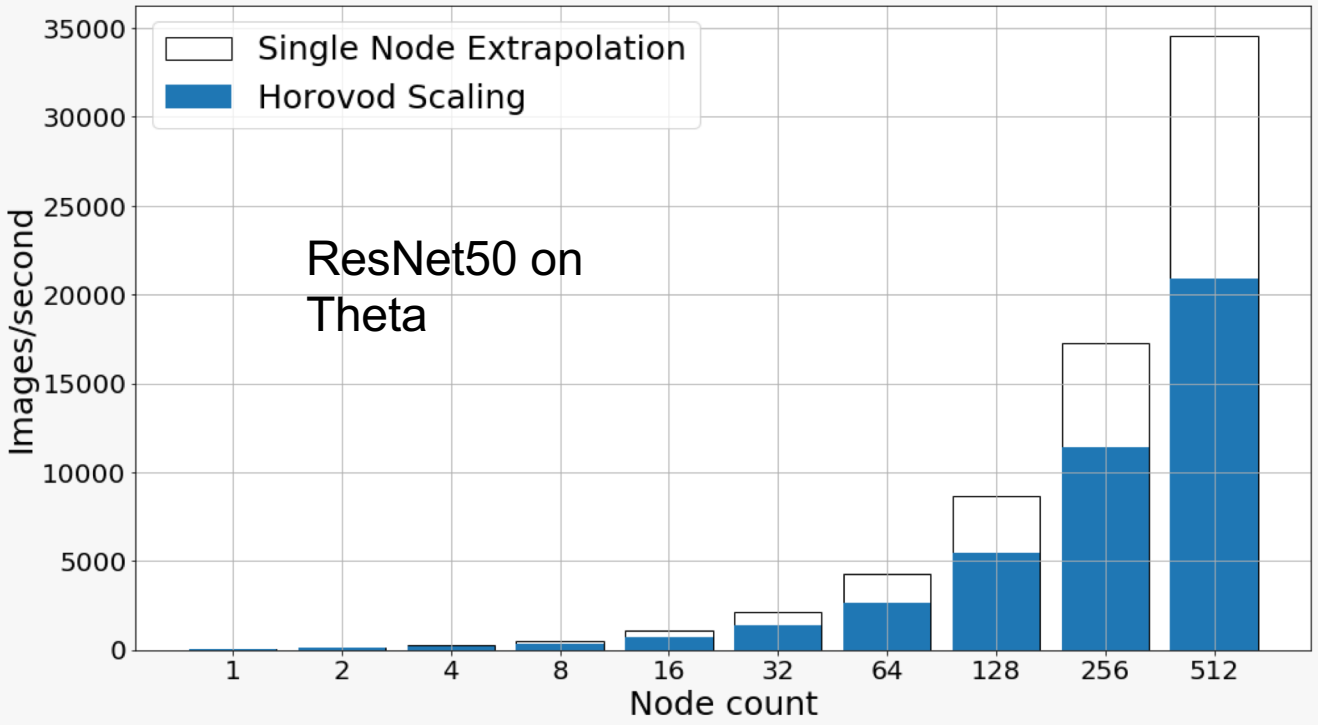


Each Node gets it's own copy of the model.

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? 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…

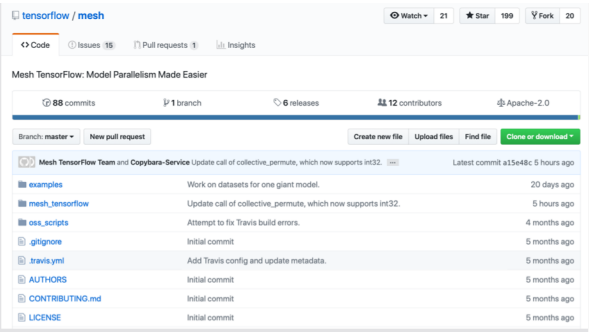
Why might you need a Mesh?

- 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

https://github.com/tensorflow/mesh

IO for Machine Learning

With optimized models for training and inference, keeping your network fed with data can become the biggest bottleneck in training.

Some general good practices that will be important on big, powerful nodes on Aurora:

1. Use parallel IO whenever possible
    1. Could feed each rank from different files, or
    2. Use mpi IO to have each rank read it's own batch from a file, or
    3. Use several ranks to read data and MPI to scatter data to remaining ranks
        1. This is most practical in big, at-scale trainings
2. Take advantage of the data storage
    1. Using striping on lustre
    2. Use the right optimizations for Aurora
3. Preload data when possible
    1. Offloading to a GPU frees CPU cycles for loading the next batch of data – you can minimize IO latency this way.

mpi4py and h5py

- **Loading and preprocessing of data can be done efficiently in python with mpi4py and h5py.**
- 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
- h5py is the hdf5 python wrapper and also supports parallel hdf5, using mpi4py
  - Need parallel hdf5 libraries to use this

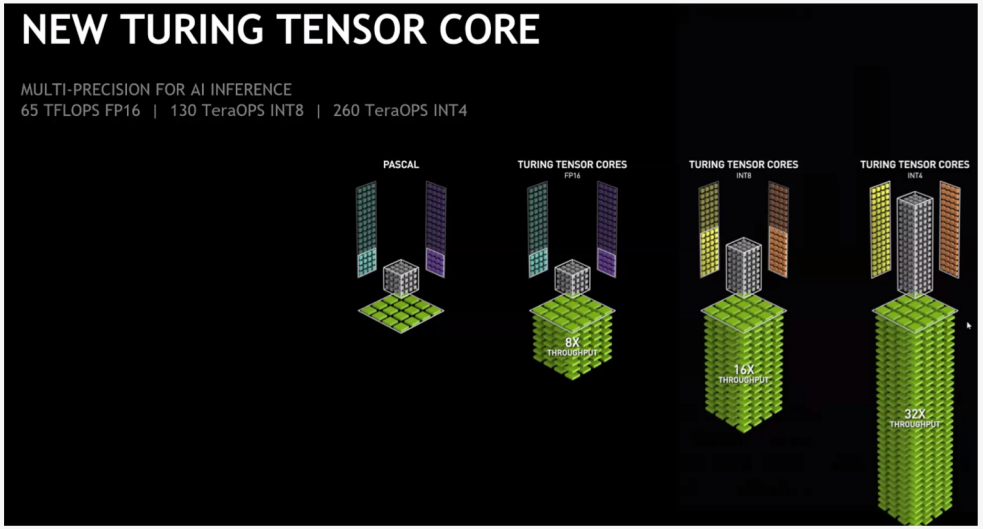
Hyperparameter Searches

https://deephyper.readthedocs.io/en/latest/

Hyperparameter optimization is the fine tuning of your network parameters to optimize your network's performance.

- Search space is combinatorically large
- Search space is "awkward" – some continuous values, some discrete values, some values are just a few choices
- Algorithms to search over hyper parameters are challenging
  - Random search? Surrogate Model?
- **This is a challenging but important workflow**. Aurora will have enormous compute power, allowing you to scale out a hyper parameter search to very large searches.

Lower Precision Deep Learning

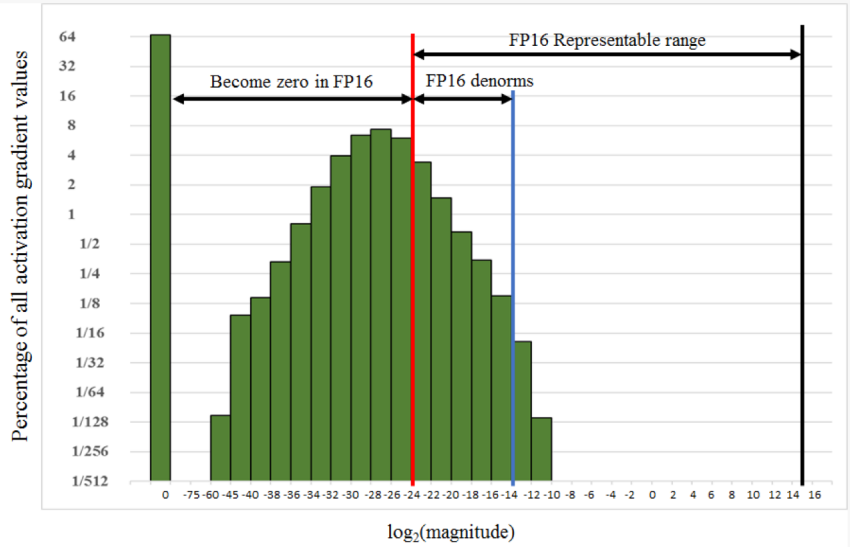


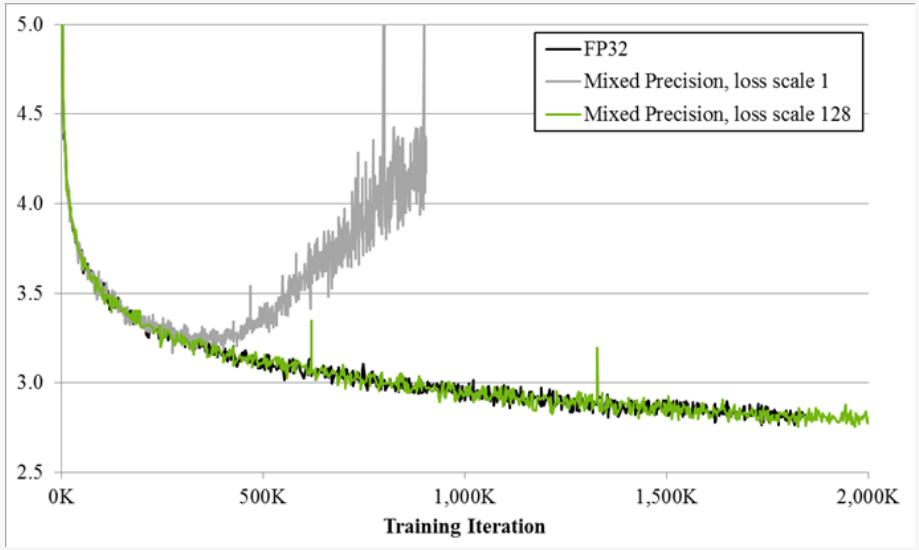
- Nvidia int8 inference
- Intel int8 inference

GPU accelerators are capable of faster computation on smaller data types – this will be an important technique on Aurora.

- Training can be done in half precision
- Inference can be done in integer precision.

Lower Precision Deep Learning - Training

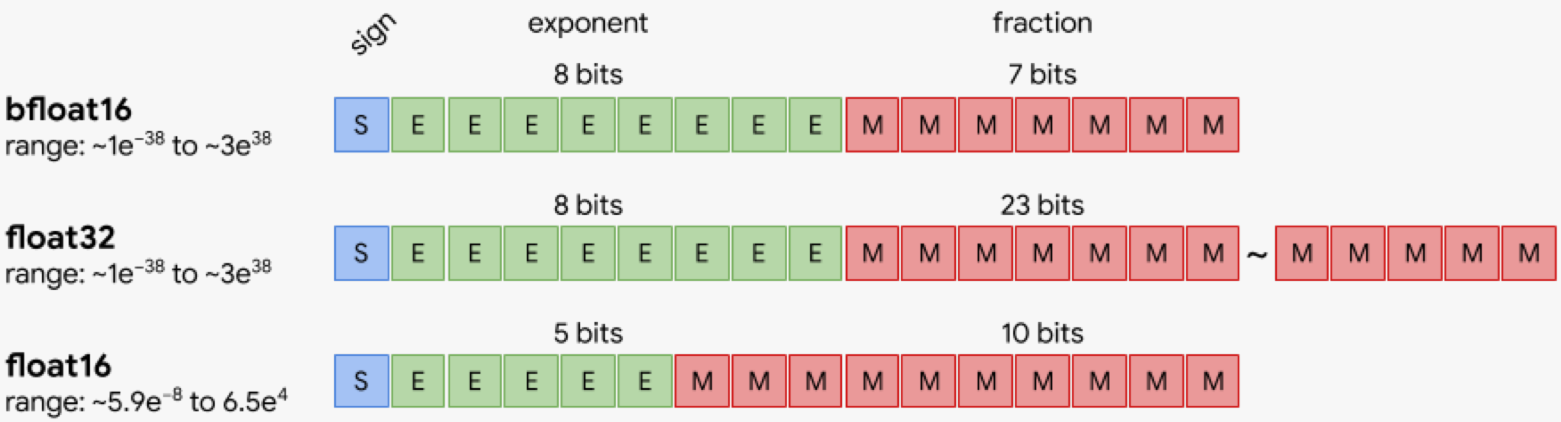




https://devblogs.nvidia.com/mixed-precision-training-deep-neural-networks/

With lower precision, you can easily have underflow or overflow of weights and activations. Can address with loss scaling, keeping a higher precision (float32) copy of weights, accumulating matrix multiplies into float32…

Lower Precision Deep Learning - Bfloat16



Bfloat 16 is a new data format that keeps as much precision as float32 (single precision) in the exponent, but truncates the fraction.

- Underflow/overflow in gradients is no longer an issue if a model converges with float32
- Hardware acceleration is at the same power as float16 rather than float32
- Tensorflow already supports bfloat16 on TPUs (Google first proposed it), Intel has full support behind bfloat16

Expect to accelerate your app even further on Aurora with half precision.

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 stop 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

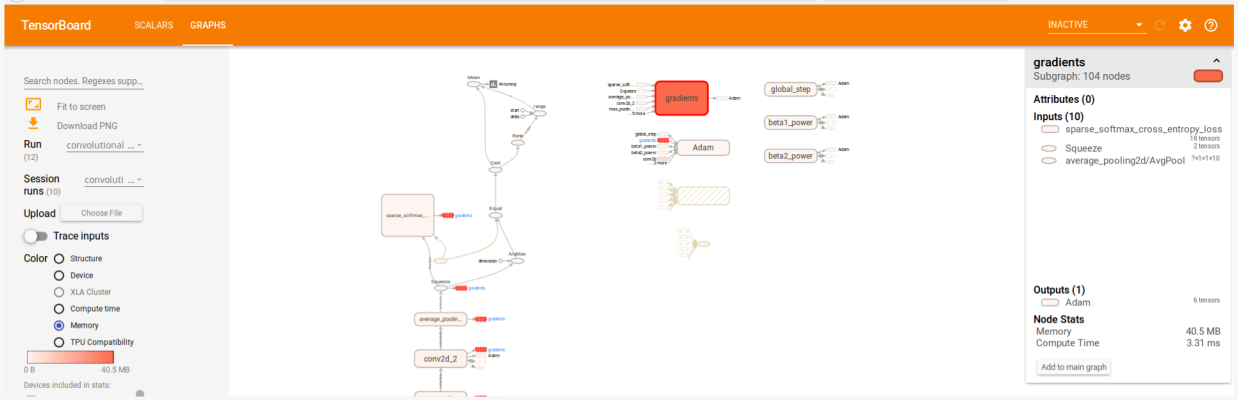
- Very easy using **`datetime.datetime.now()`**
- Trivial to analyze in the log files
- Can give a good top-level, cross software/system comparison using **images/second** or **global-step/second** for the entire application
- System Independent (laptop vs. HPC node, etc)

Cons

- Not useful for finding hotspots, only for monitoring know 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 line_profiler

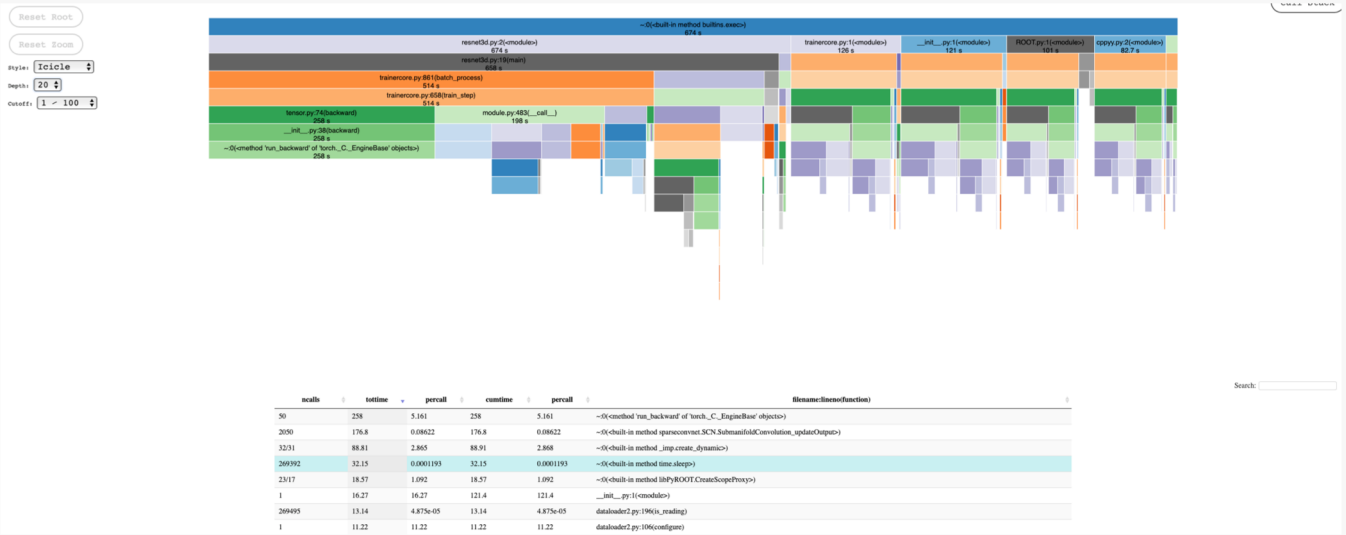
Python also has a line_profiler tool which is useful for measuring expensive functions that you write (like a training loop)

```
kernprof –l script.py
```

Gives you a line-by-line measurement of your functions:

```
Line #      Hits         Time  Per Hit   % Time  Line Contents
.......
    66    300005     417866.0      1.4      0.8      for batch in range(len(indexes/batch_size)):
    67    300000     447185.0      1.5      0.8        if (batch+1)*batch_size > 10000:
    68    299610     403266.0      1.3      0.8          continue
    69
    70       390       3375.0      8.7      0.0        batch_indexes = indexes[batch*batch_size:(batch+1)*batch_size]
    71       390      54922.0    140.8      0.1        images = x_train[batch_indexes]
    72       390      15930.0     40.8      0.0        labels = y_train[batch_indexes].reshape([batch_size,])
    73
    74                                                  # Set the model to training mode:
    75       390      44727.0    114.7      0.1        model.train()
    76                                                  # Reset the gradient values for this step:
    77       390      30214.0     77.5      0.1        optimizer.zero_grad()
    78                                                  # Compute the logits:
    79       390   11985557.0  30732.2     22.5        logits = model(images)
    80
    81
    82                                                  # Loss value is computed imperatively
    83       390      44049.0    112.9      0.1        loss = loss_operation(input=logits, target=labels)
    84                                                  # This call performs the back prop:
    85       390   17814337.0  45677.8     33.5        loss.backward()
    86                                                  # This call updates the weights using the optimizer
    87       390     671574.0   1722.0      1.3        optimizer.step()
```

Python cProfile



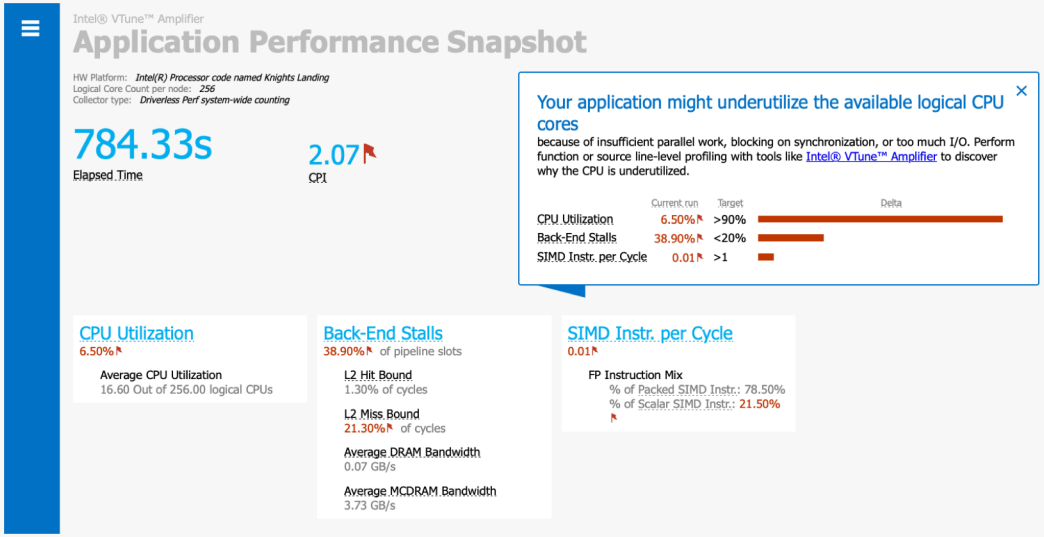
Pros

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

Cons

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

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.

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)
```

Intel Vtune – Advanced Hotspots

Vtune advanced hotspots can give a very useful report of what your CPUs are doing, how effectively the 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
```

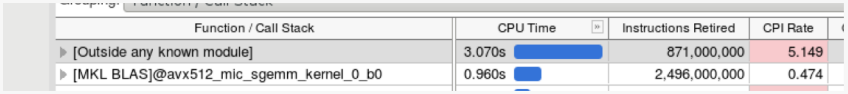
You don't have to, but **should** run the finalization after the run completes (do this from the **login nodes**):

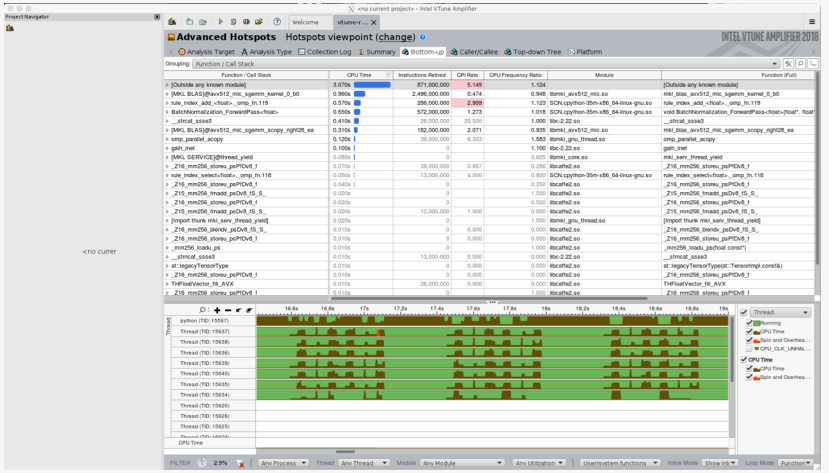
```
source /opt/intel/vtune_amplifier/apsvars.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64/
export PMI_NO_FORK=1

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
```





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/
```

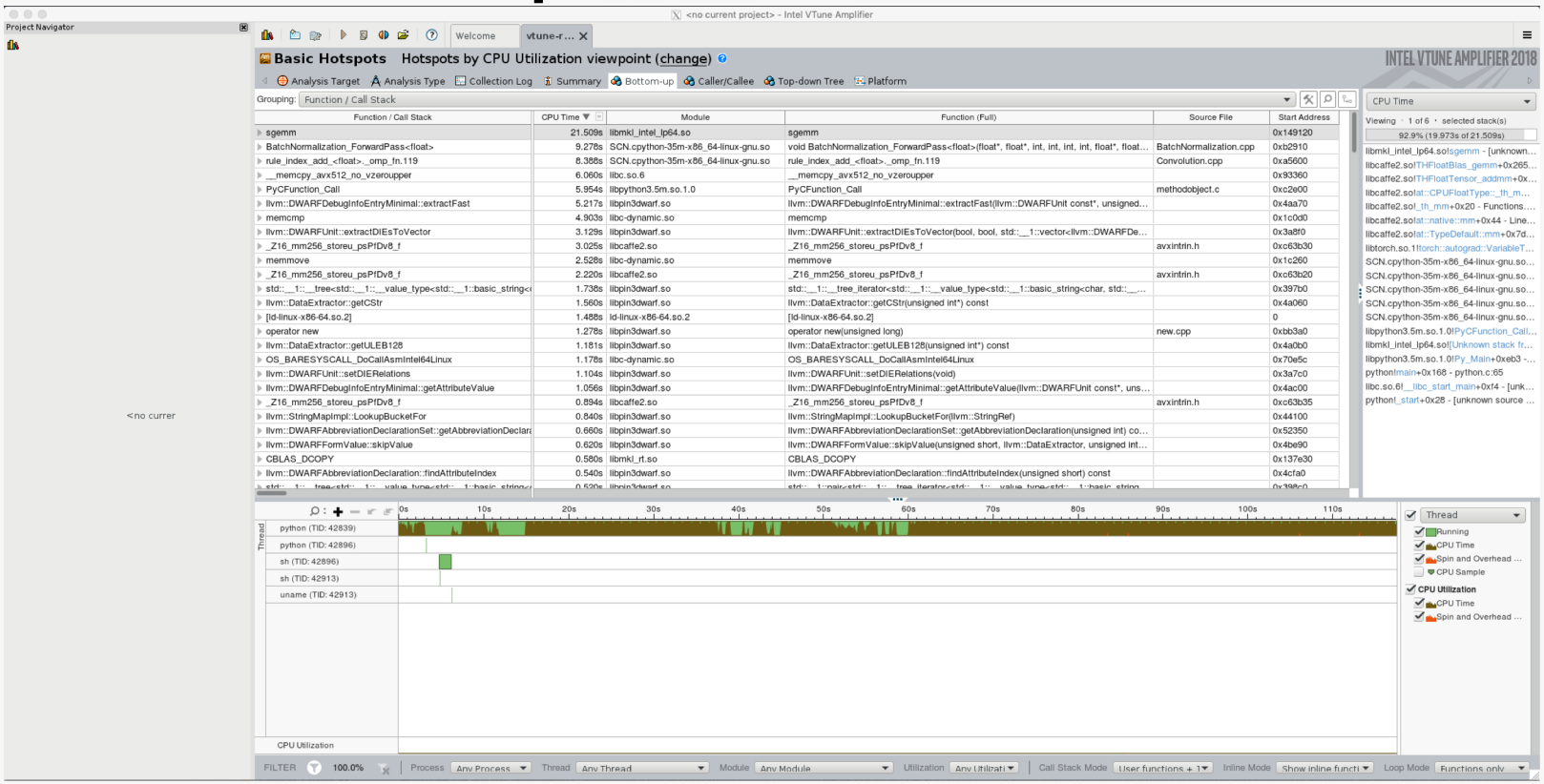
Pros

- 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



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):

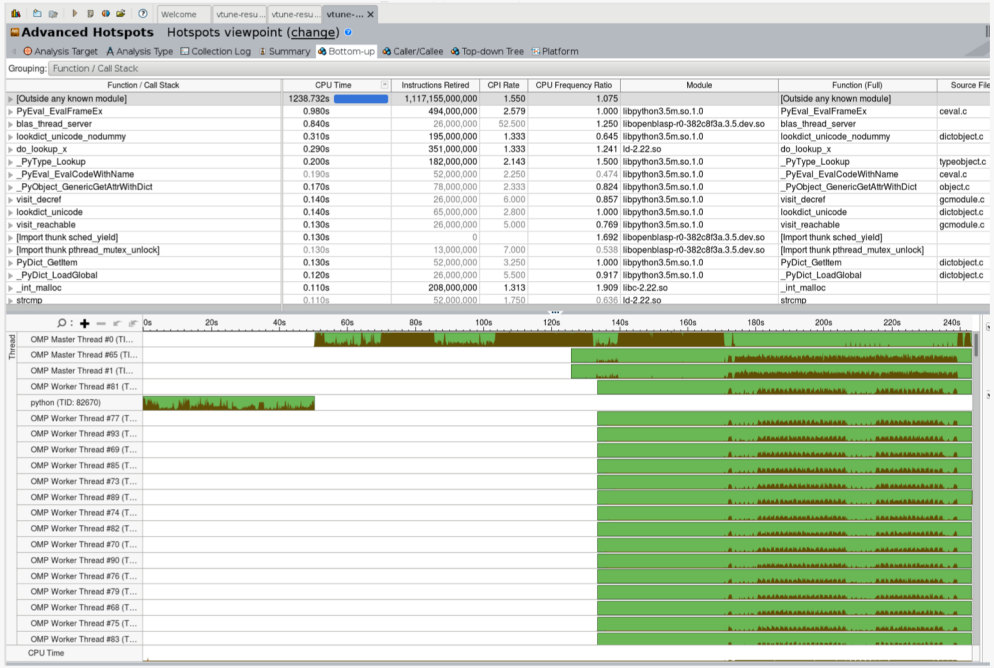
- 31% of the application time was spent doing FFTs with tensorflow
- 10% was spent creating tensorflow traces
- 8% was computing loss functions.
- 25% was spent creating and optimizing the tensorflow graph (measured for a short run, this is a smaller fraction for production runs)

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

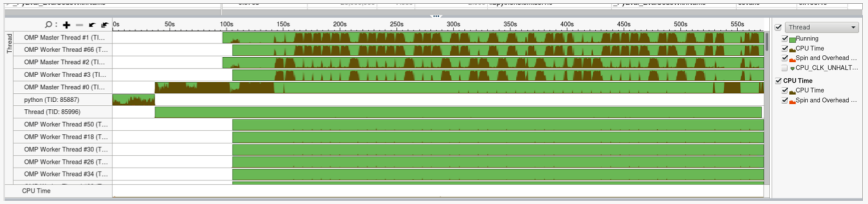
A user reported seeing a significant degradation in performance with tensorflow when going from single image to multi-image batches.



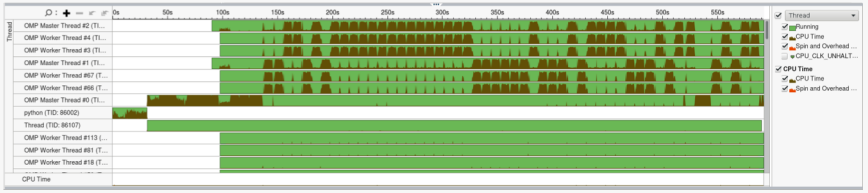
Batch Size 1 showed decent balance between threads, even if utilization was lower than ideal.

Profiling Example – Tensorflow CNN

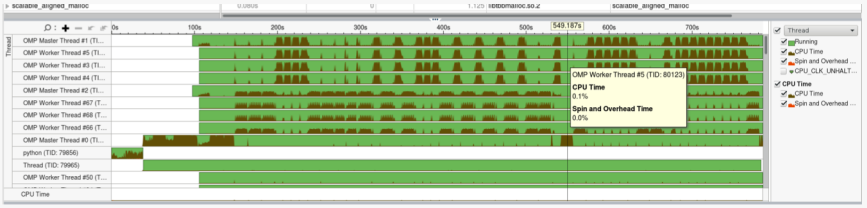
A user reported seeing a significant degradation in performance with tensorflow when going from single image to multi-image batches.



Batch Size 2



Batch Size 3

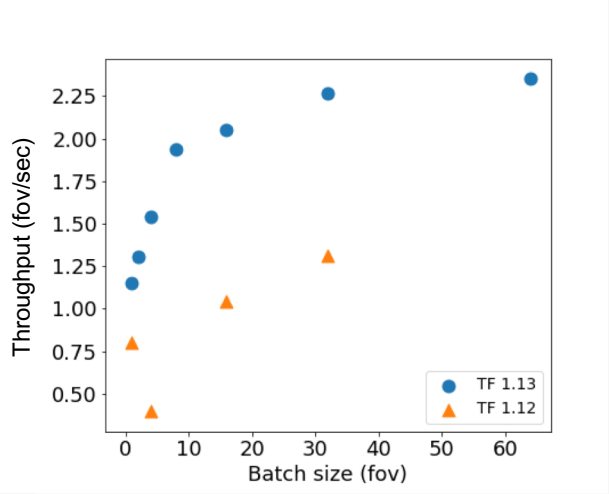


Batch Size 4

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

59 Argonne Leadership Computing Facility