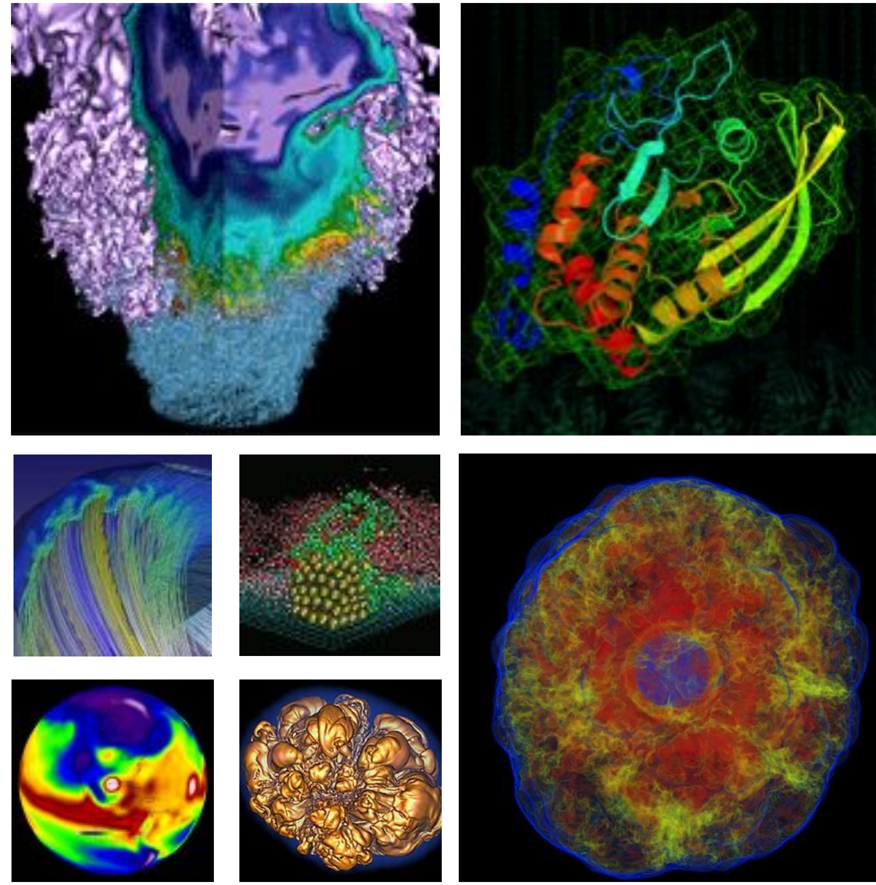
Overview of Deep Learning Stack at NERSC



NERSC

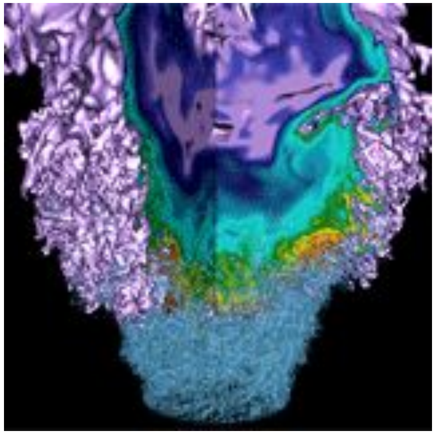
Wahid Bhimji, Mustafa Mustafa

User Training Jan/2019

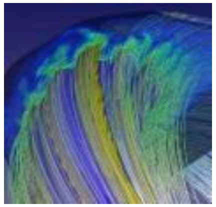
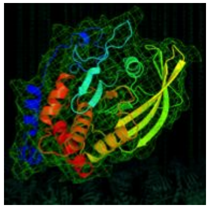
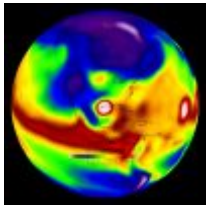
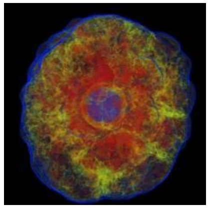
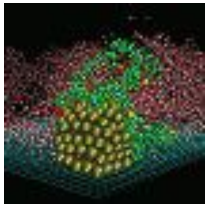
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Deep learning Stack



NERSC

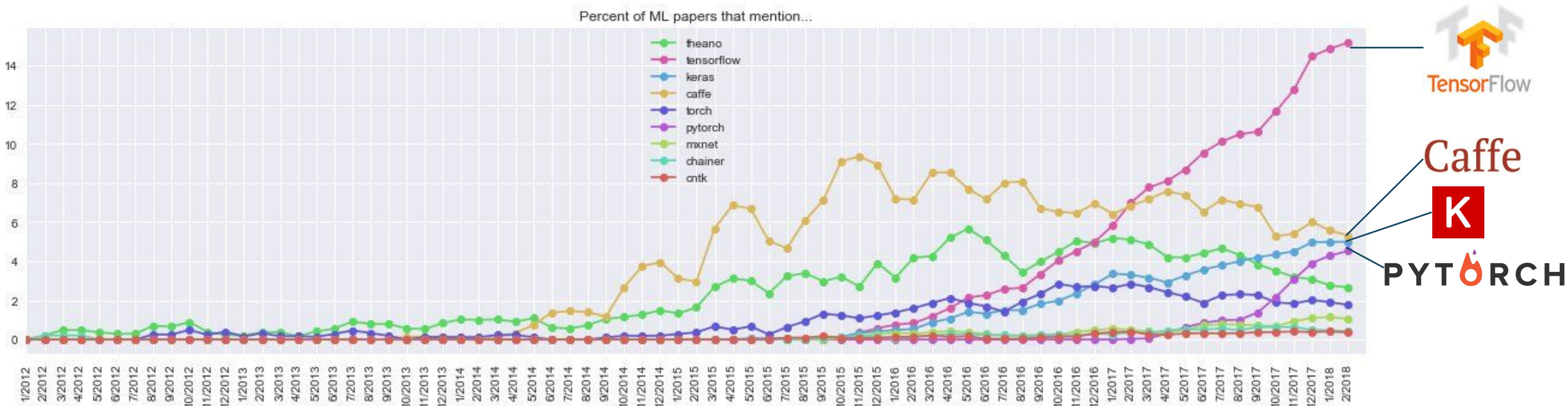






Deep Learning Stack on HPC

	Technologies
Deep Learning Frameworks	K, TensorFlow, PYTORCH, Caffe, Neon, CNTK, MXNet, …
Multi Node libraries	Cray ML PE Plugin, Horovod, MLSL, MPI, GRPC
Single Node libraries	MKL-DNN, CuDNN
Hardware	CPUs (KNL), GPUs, FPGAs, Accelerators

Software Frameworks

- Different frameworks popularity has evolved rapidly
- Percentage of ML Papers that mention a particular framework:



Source: https://twitter.com/karpathy/status/972295865187512320?lang=en

- Caffe and Theano most popular 3-4 years ago
- Then Google released TensorFlow which now dominates
- PyTorch is recently rising rapidly in popularity

Framework overview (IMHO)

- **TensorFlow:**
  – Reasonably easy to use directly within python (not as easy as with Keras)
  – Very nice tools for development like TensorBoard
  – Active development for features (e.g. dynamic graphs) and performance (e.g. for CPU/KNL) and ease (e.g. estimators)
- **Keras:**
  – High-level framework sits on top of tensorflow (or theano) (and now part of TensorFlow).
  – Very easy to create standard and even advanced deep networks with a lot of templates/ examples

Pytorch and Caffe (IMHO)

NERSC

- **PyTorch** PYTORCH
  - Relatively recent python adaption of 'torch' framework - heavily contributed to by FaceBook
  - More pythonic than tensorflow/keras
  - Dynamic graphs from the start - very flexible
    - Popular with (some) ML researchers
  - Experimental, some undocumented quirks
    - Version 1.0 coming soon! rc1 is out already.
- **Caffe** Caffe
  - Optimised performance (still best for certain networks on CPUs)
  - Relatively difficult to develop new architectures

TensorFlow (and Keras) @NERSC

http://www.nersc.gov/users/data-analytics/data-analytics-2/deep-learning/using-tensorflow-at-nersc/

- **Easiest is to use default anaconda python:**

```
module load python
python
>>> import tensorflow as tf
```

- **Active work by intel to optimize for CPU:**
  - Available in anaconda. Modules on Cori:

```
module avail tensorflow  #[list options]
module load tensorflow/intel-1.12.0-py36
```

- **Can also tune variables for performance (e.g. see intel blog)**
  - E.g Inter-op and Intra-op

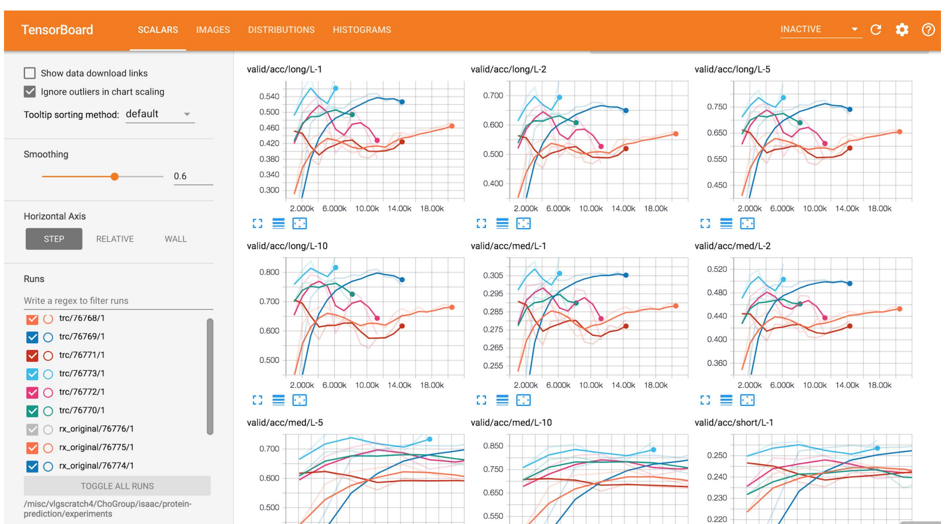
TensorBoard

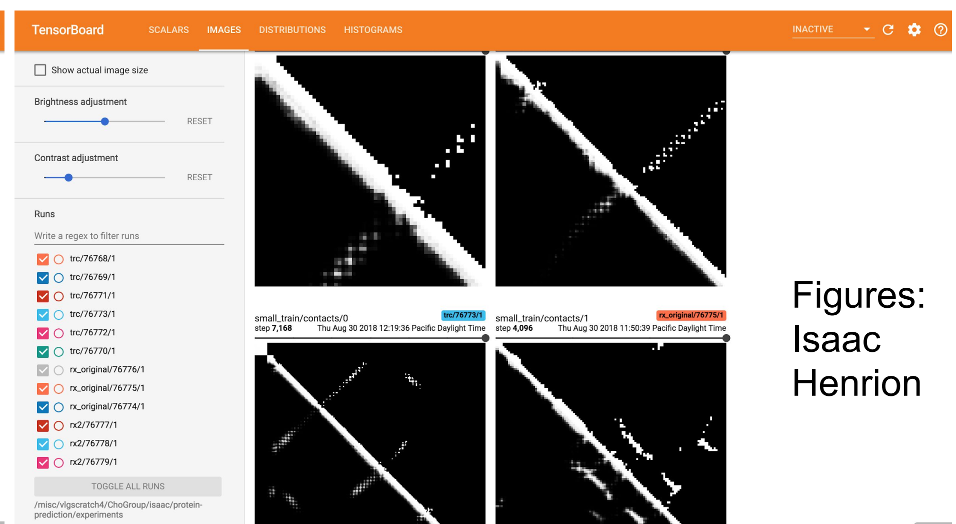
- **Easy, customisable, visualization of training in progress**
- **At NERSC run TensorBoard on login node; point to logs made by jobs on compute node (chose an unused port)**

```
cori05 > tensorboard --logdir=path/to/logs --port 9998
```

- **Use a ssh tunnel from your laptop to connect then open localhost:9998 in your browser (note: others will also be able to see your experiments if they connect to that port)**

```
YourLaptop > ssh -L 9998:localhost:9998 cori.nersc.gov
```





Figures: Isaac Henrion

PyTorch @NERSC

https://www.nersc.gov/users/data-analytics/data-analytics-2/deep-learning/pytorch/

- **Again easiest is to use default anaconda python:**

```
module load python
python
>>> import torch
```

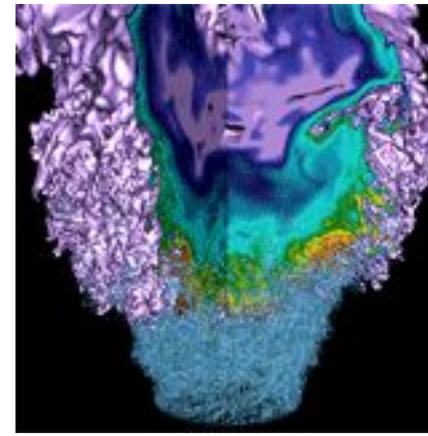
- **Note however the anaconda version isn't built with the pytorch MPI (for multi-node) - so we provide a build**

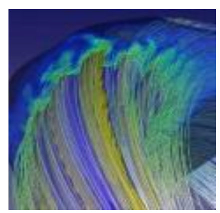
```
module load pytorch-mpi/v0.4.1
```

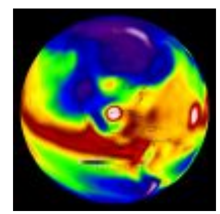
- **And again we are looped into intel optimizations:**

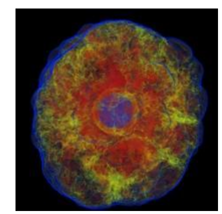
```
module load pytorch/v1.0.0-intel
```

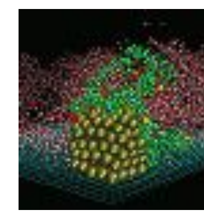
Deep learning is EASY











Keras: CNN Classification

```python
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import utils as k_utils

# Load MNIST data and add channel
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = np.expand_dims(x_train.astype('float32'), axis=-1)
x_test = np.expand_dims(x_test.astype('float32'), axis=-1)

# normalize data
x_train /= 255
x_test /= 255

# convert class vectors to binary class matrices
y_train = k_utils.to_categorical(y_train, num_classes)
y_test = k_utils.to_categorical(y_test, num_classes)
```

Keras is TF’s official high-level API

Example source: modified version of github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

Keras: CNN Classification

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=[28, 28, 1]))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
# compile model
model.compile(loss='categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
# check model architecture summary
model.summary()

# train model
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
          verbose=1, validation_data=(x_test, y_test))
# evaluate model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0], 'Test accuracy:', score[1])
```

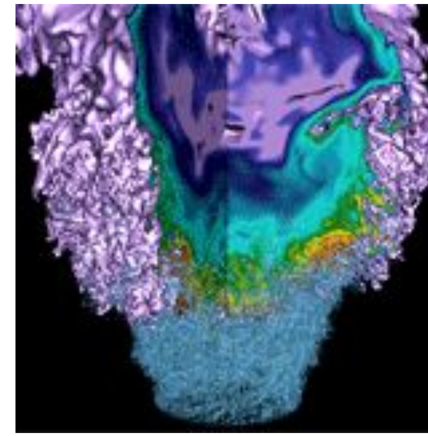
```
Layer (type)                 Output Shape          Param #
==========================================================
conv2d_0 (Conv2D)            (None, 26, 26, 32)    320
__________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)    18496
__________________________________________________________
max_pooling2d_0 (MaxPooling2 (None, 12, 12, 64)    0
__________________________________________________________
dropout_0 (Dropout)          (None, 12, 12, 64)    0
__________________________________________________________
flatten_0 (Flatten)          (None, 9216)          0
__________________________________________________________
dense_0 (Dense)              (None, 128)           1179776
__________________________________________________________
dropout_1 (Dropout)          (None, 128)           0
__________________________________________________________
dense_1 (Dense)              (None, 10)            1290
==========================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
```

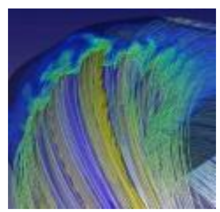
Test loss: 0.0284083929521
Test accuracy: 0.9922

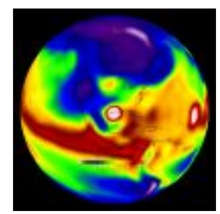
Keras LHC-CNN in Jupyter

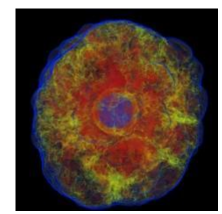
- **Restricted to single core on jupyter-dev**
  - Submit to batch (or interactive queue) for bigger problems
  - Can use ipyparallel
    https://github.com/sparticlesteve/cori-intml-examples

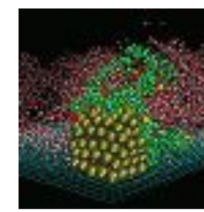
CPU Optimizations and Multi-node training





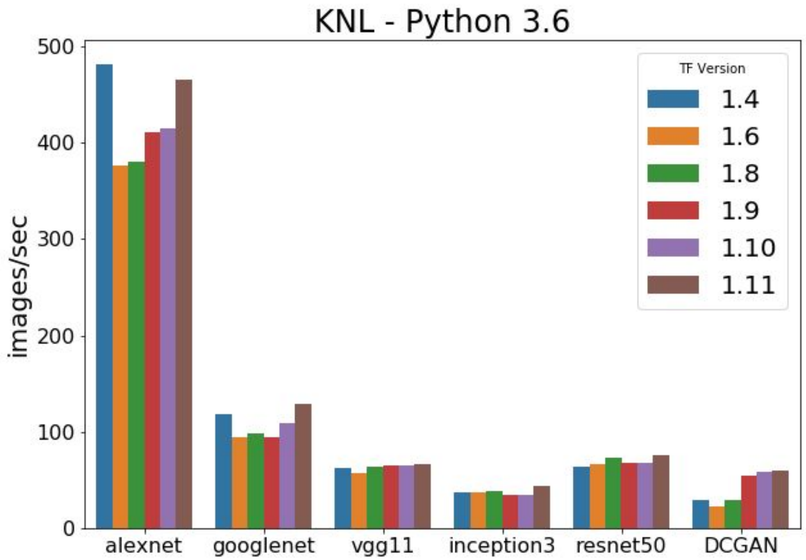


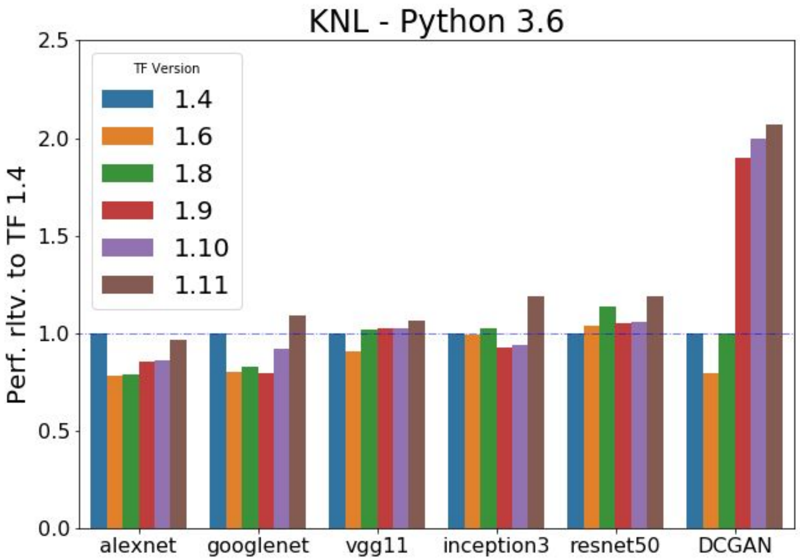




(TensorFlow) MKL Optimizations

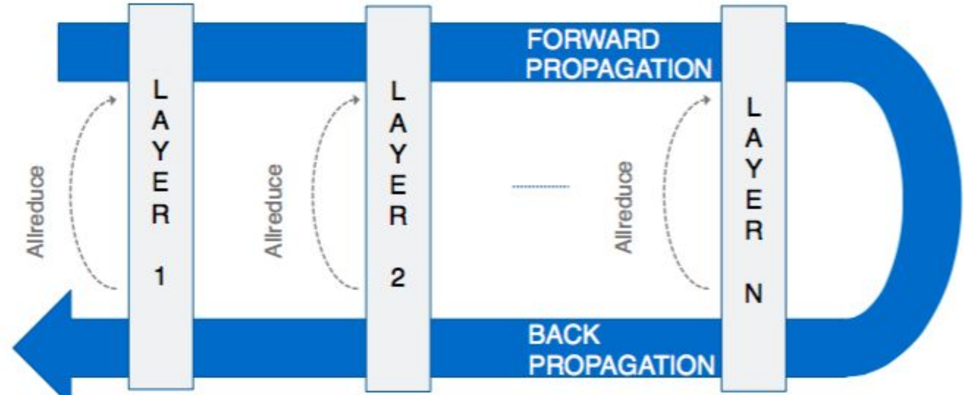
- **Python frameworks rely on optimized backends to perform**
- **For CPU like Cori KNL this is Intel Math Kernel Library (MKL) (e.g. MKL-DNN)**
- **Blog posts on Intel optimisations:**
  https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture
  https://ai.intel.com/tensorflow-optimizations-intel-xeon-scalable-processor/
  https://software.intel.com/en-us/articles/using-intel-xeon-processors-for-multi-node-scaling-of-tensorflow-with-horovod



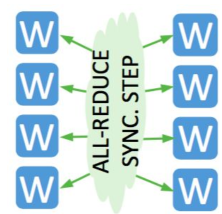


Multi-node training

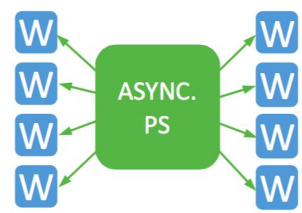
- **Data parallel training for SGD**
  - Each node processes data independently then a global update
  - Synchronous; Asynchronous;hybrid gradient lag approaches
- **Challenges to HPC scaling include convergence and performant libraries**



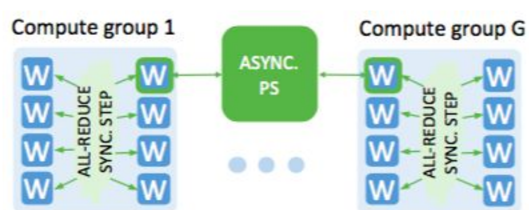
From Pradeep Dubey, "Scaling to Meet the Growing Needs of Artificial Intelligence (AI), IDF 2016
https://software.intel.com/en-us/articles/scaling-to-meet-the-growing-needs-of-ai



SYNCHRONOUS



ASYNCHRONOUS



HYBRID

From Kurth et al. SC17 [arXiv:1708.05256](https://arxiv.org/abs/1708.05256)

Some experiences of DL at scale

Convergence at scale is an active area of research. Some current experiences from multiple projects:

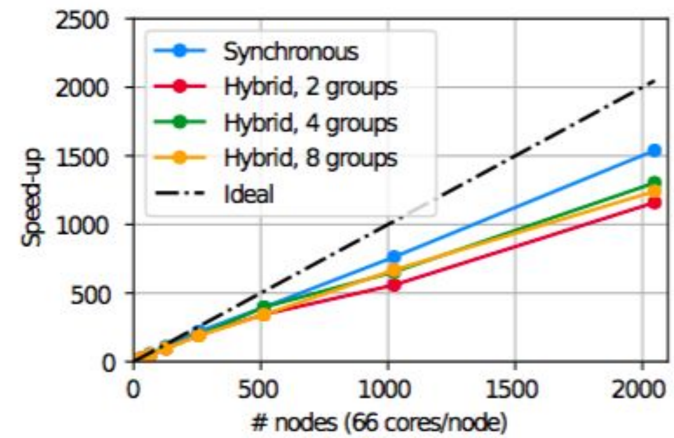
- **If strong scaling (small node count): decrease per-node batch size with increasing synchronous node count**
- **Experiment with increasing learning rate sub-linearly to linearly with number of workers:**
  - Warmup period starting at initial learning rate
  - Reduce learning rate if learning plateaus
- **Advanced Layer-wise adaptive (LARS/LARC) strategies**
- **Be careful with batch normalization for multi-node performance, consider Ghost Batch Normalization**
- **With 'global shuffle' of training examples, use of Burst Buffer can help with I/O at NERSC**

Performant Multi-Node Libraries

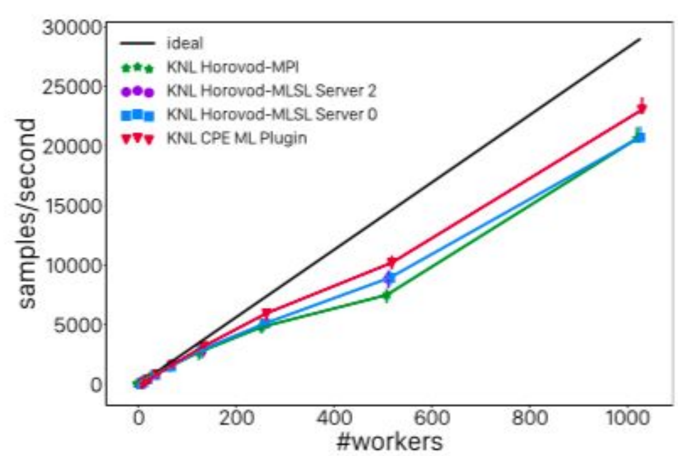
- **Initial scaling on NERSC involved a lot of work**
  - e.g. with Intel-Caffe and Intel-MLSL
- **Default TensorFlow uses gRPC for communication - non-ideal for Cori high-speed network**
  - See e.g. Mathuriya et. al (arXiv:1712.09388)
- **Fortunately now libraries based on MPI with Horovod and Cray PE ML Plugin**

For LHC-CNN:

Kurth et al. SC17 **arXiv:1708.05256**



Kurth et al.Concurrency Computat Pract Exper. 2018;e4989



Horovod Keras Simple Snippets

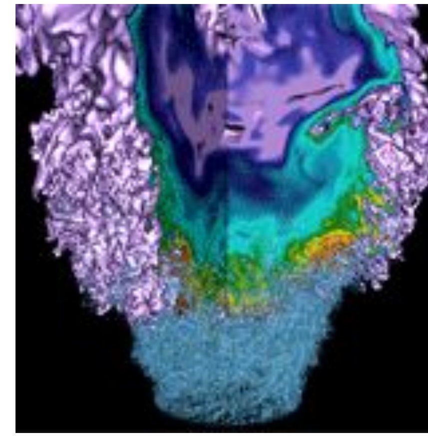
- **When building model:**

```
from keras import models
import horovod.keras as hvd
model = models.Model(inputs, outputs)
hvd.init()
model.compile(optimizer=hvd.DistributedOptimizer(optimizers.Adam),…
```

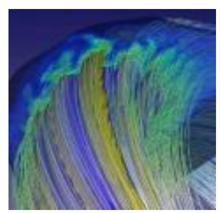
- **When training model:**

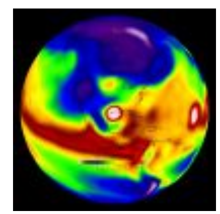
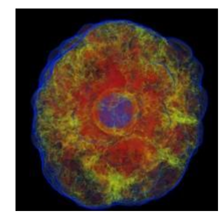
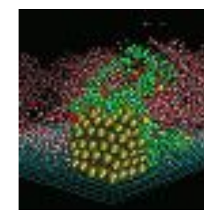
```
model.fit(callbacks=[hvd.callbacks.BroadcastGlobalVariablesCallback(0),...
```

Support



NERSC



Contact us

General help with deep learning modules;

and running DL at NERSC via:

consult@nersc.gov

Collaborations:

ML-Engineers@NERSC

Mustafa Mustafa: mmustafa@lbl.gov

Steve Farrell: SFarrell@lbl.gov

Questions on this talk: wbhimji@lbl.gov