WHITE PAPER

AI Products Group, Intel Corporation
Customer Solutions Technical Enabling

Best Practices for Scaling Deep Learning Training and Inference with TensorFlow* On Intel® Xeon® Processor-Based HPC Infrastructures

Table of Contents

Authors: Aishwarya Bhandare, Deepthi Karkada, Kushal Datta, Vamsi Sripathi, Sun Choi, Vikram Saletore, AI Products Group
Anupama Kurpad, Connectivity Group

1. Best Practices For Tensorflow* On Intel® Xeon® Processors

This document describes the setup, installation and procedure to run distributed Deep Learning training and inference using TensorFlow* with Uber Horovod* library on Intel® Xeon® processor-based infrastructure. The steps required to run the benchmark can vary depending on the user's environment. In case of a large cluster with the order of hundreds or thousands of nodes, we provide sample scripts that use the Slurm* scheduler. Alternatively, we also list out steps for smaller systems that may not have such a scheduler configured. Furthermore, we also provide scripts to build a singularity image for ease of deployment. You can download .txt files of all scripts **here**.

This document has been tested with TensorFlow 1.9.0, Horovod 0.13.11, Open MPI version 1.10.7, Python* 2.7 on CentOS* 7 and 7.5.

1.1 TensorFlow* Setup and Installation

TensorFlow can either be built from source or installed from a pre-built wheel file. The steps to build TensorFlow from source are clearly listed on the official TensorFlow documentation here: https://www.tensorflow.org/install/install_sources.

Also refer to the Best Practices documentation at the end of this document to use the recommended build flags to get the most optimized CPU build. We have consolidated these steps into a simple script: build_tensorflow_script, which can be run as follows:

```
./build_tf.sh <path/to/install/directory>
```

We recommend that the install directory be created on a shared filesystem so that the TensorFlow build is accessible by all nodes in a multi-node training environment.

At the prompt, choose "y" to build from source. The script creates a virtual environment and installs TensorFlow in this environment, as opposed to doing a system-wide install. The virtual environment is created in <path/to/install/directory>/virtenv/, and can be activated as follows:

```
source <path/to/install/directory>/virtenv/bin/activate
```

To deactivate the virtual environment, run:

```
deactivate
```

Alternatively, TensorFlow can be installed from a pre-built wheel available here:

https://software.intel.com/en-us/articles/intel-optimized-tensorflow-installation-guide

The script build_tf.sh can be used to do this by selecting 'No' when prompted about building TensorFlow from source.

Note: See section on Troubleshooting for some common problems encountered during the build process.

1.2 Install MPI, if not already installed

For Open MPI: Below are steps to install Open MPI. If sudo/root access is available:

```
sudo yum -y install openmpi openmpi-devel
```

```
module avail && module add mpi/openmpi-x86_64
```

Note: If you get an error 'module:command not found', use the following command:

```
source /etc/profile.d/modules.sh
```

This will install to standard paths, so there is no need to change environment variables.

This command installs Open MPI 1.10.7 as tested on CentOS 7.5. If neither sudo/root access is available, build Open MPI from source. Follow the steps on the Open MPI page to install and configure Open MPI. After installation, update PATH and LD_LIBRARY_PATH accordingly to point to this installation.

For MVAPICH2*: refer to the user guide in the official documentation here: http://mvapich.cse.ohio-state.edu/userguide/

For Intel-MPI*: refer to https://software.intel.com/en-us/intel-mpi-library. Add Intel-MPI related variables to PATH by running:

```
source <intel_mpi_install_dir>/bin/mpivars.sh intel64
```

1.3 Install Horovod* Library

Horovod is a distributed training framework developed by Uber Engineering for TensorFlow, Keras*, and PyTorch*, which makes distributed Deep Learning fast and easy to use: **https://github.com/uber/horovod**.

Horovod is available as a standalone Python package. The installer looks for an existing installation of MPI. **Note:** In an environment where there are potentially multiple MPI libraries, is important to ensure that only the required MPI libraries are enabled in Horovod environment. This can be done either with the mpi-selector utility, or by manually updating the PATH and LD_LIBRARY_PATH variables to point to the required library.

Horovod also requires an existing installation of TensorFlow. In step 1, we show how to install TensorFlow in a Python virtual environment. Before installing Horovod, ensure that the virtual environment is activated. Then install Horovod using the pip installer:

```
pip install --no-cache-dir horovod
```

Verify the installation. If the following steps work without errors, we are ready to run the benchmark.

```
python
Python 2.7.13 (default, Aug  1 2017, 16:43:54)
[GCC Intel C++ gcc 5.4 mode] on linux2
Type “help”, “copyright”, “credits” or “license” for more information.
>>> import tensorflow as tf
>>> import horovod.tensorflow as hvd
```

1.4 Installing tf_cnn_benchmarks

TensorFlow maintains a repository for performance benchmarking of few standard Convolutional Neural Network topologies at https://github.com/tensorflow/benchmarks.git. These benchmarks can be obtained by running:

```
git clone https://github.com/tensorflow/benchmarks.git
```

1.5 Preparing the ImageNet* 2012-1K Dataset

The dataset needed for performance evaluation is the ImageNet* 2012-1K dataset available here http:/www.image-net.org/challenges/LSVRC/2012/. The dataset can be obtained by first registering at the ImageNet website (http://image-net.org/download-images) and obtaining the username and access key. The raw images must be converted to the TensorFlow's TFRecord format which stores the raw image data along with metadata like class info, height, width, and so on.

SURFSara, one of our collaborators, identified a key requirement when preprocessing the dataset. In order to achieve State-Of-The-Art (SOTA) convergence after training, shuffling the input classes is very important when creating TFRecords from raw images. There is an existing script located in the tensorflow/tpu repository that can be used to do this: https://github.com/tensorflow/tpu/blob/master/tools/datasets/imagenet_to_gcs.py

The script contains several TPU references that must be removed when running on Intel Xeon processors. A version of the original script with the required modifications for CPU and Horovod and an accompanying Best Practices document is provided by SURFSara, which can be downloaded from: https://surfdrive.surf.nl/files/index.php/s/xrEFLPvo7IDRARs

This script can be used to download the dataset as well as create the TFRecords. Please note that there should be ~400GB disc space available before starting the process, and it might take several hours to complete depending on your network connectivity.

1.5.1 Steps to download and prepare the ImageNet Dataset

Follow the steps below for preparing the ImageNet-1K Dataset using scripts from SURFsara.

1. Untar the <tar_file>.tar.gz downloaded from SURFsara drive above with the command

```
tar –xzvf <tar_file>.tar.gz
cd tpu/tools/datasets
```

2. Download the dataset along with converting it to tf_records format

```
Python imagenet_to_gcs.py --local_scratch_dir <path-to-resulting tfrecords> \
--project="TEST_PROJECT" \
--imagenet_username=<username> \
--imagenet_access_key=<access_key>
```

Where "imagenet_username" and "imagenet_access_key" are to be obtained by registering on the ImageNet website as mentioned above.

The script does the following to download the data:

- Training images ILSVRC2012_img_train.tar are downloaded from http://www.image-net.org/challenges/LSVRC/2012/
- This tarball contains multiple tarballs in it. The script then un-tars each one of them and saves it in the form: Training images: train/n03062245/n03062245_4620.JPEG
  Note: in the training dataset, the images are grouped by the labels, meaning that all images in the folder train/n03062245 have the label n03062245.
- The synset_labels (validation labels) are downloaded from: https://raw.githubusercontent.com/tensorflow/models/master/research/inception/inception/data/imagenet_2012_validation_synset_labels.txt
- Then, the validation images ILSVRC2012_img_val.tar are downloaded from http://www.image-net.org/challenges/LSVRC/2012/

1.5.2 Already have the ImageNet-1K Dataset

If you have already downloaded the dataset, you can pass the path to the dataset in –raw_data_dir:

```
python imagenet_to_gcs.py --raw_data_dir <path-to-rawimage-dir> --local_scratch_dir <path-to-
resulting-tfrecords>
```

If raw data directory is provided, it should be in the format:

- Training images: train/n03062245/n03062245_4620.JPEG
- Validation Images: validation/ILSVRC2012_val_00000001.JPEG
- Validation Labels: synset_labels.txt

Note: If you have a pre-downloaded dataset, please make sure that the folder <path-to-rawimage-dir> has the directory structure specified above. Please make sure the validation labels and validation images are from the same sources as indicated earlier. A mismatch can give lower validation accuracies.

The TF_Records can either be then copied on to the local drives of all the nodes, or a high-speed shared file system like Lustre* can be used.

*1.5.3 Dataset Striped on Lustre**

While running multi-node experiments on large number of nodes, striping the dataset can give significance performance gains in terms of scaling efficiency. We have found that it can improve scaling efficiency for a 32 node run from ~40% to ~90%, for a stripe count of 64 and stripe size of 32M for ImageNet data. So, the conclusion is that Lustre striping makes a big difference by allowing us to have good scaling efficiency at 64-128 nodes, and possibly beyond.

1.6 Example: Running ResNet-50 with tf_cnn_benchmarks

In this section, we describe the command line parameters used for both single and multi-node Training and Inference with ResNet-50 using tf_cnn_benchmarks scripts.

1.6.1 tf_cnn_benchmarks: ResNet-50

Let us first look at the common arguments to tf_cnn_benchmarks for ResNet-50 model:

```
COMMON_ARGS="\
--batch_size=64 \
--model=resnet50 \
--num_inter_threads=2 \
--display_every=5 \
--data_format=NCHW \
--optimizer=momentum \
--device=cpu \
--mkl=TRUE \
--variable_update=horovod \
--horovod_device=cpu \
--local_parameter_device=cpu \
--kmp_blocktime=1"
```

The explanation of the values used are:

- --batch_size=64; Number of images/worker to train in parallel
- --model=resnet50. This is an ad-hoc pre-determined tag used by tf_cnn_benchmarks to define/use the ResNet-50 v1 model. The other preset model names are available in tf_cnn_benchmarks/model/model_config.py script.
- --num_inter_threads=2 or 3; this sets the number of ops that can be executed in parallel.
- --display_every=5 means after how many batches will the training loss will be printed in stdout. We have used 5, but any value between 5 & 20 gives useful information on the progress of the training job.
- --data_format=NCHW. NCHW format means the dimensions of the tensor containing the input images are in the order of BATCHES, CHANNELS, HEIGHT and WIDTH. In other words, "channels first" as it appears before height and width. Other possible value is NHWC where channels appear last.
- --optimizer=momentum is the gradient descent optimizer used in training. In place of Momentum optimizer, you can use adagrad, sgd (stochastic gradient descent), rmsprop, and others.
- --device=cpu signifies CPU device to execute the operations
- --mkl=True enables the use of Intel® Math Kernel Library (Intel® MKL) for optimized kernels for blocked matrix multiplication, vector-matric multiplication and other operations
- --variable_update=horovod; Use Horovod for gradient synchronization between the training workers.
- --horovod_device=cpu; Use CPU to execute the gradient synchronization
- --kmp_blocktime=0 or 1; optimal value may vary with topology. This is the time in milliseconds that a thread waits before sleeping after completion of execution.

For training the model to convergence, following additional arguments will need to be included in the `COMMON_ARGS` for saving check-point and summaries in the appropriate path to directory.

`--train_dir:` directory where model checkpoints and graph are saved

`--save_model_secs:` time in seconds, recurrently save model after every given duration.

`--init_learning_rate:` initial learning rate

`--num_epochs_per_decay:` learning rate is decayed after these many epochs

`--num_learning_rate_warmup_epochs:` Run warmup for these many epochs

`--learning_rate_decay_factor:` learning rate decayed by this factor

`--summary_verbosity=1:` verbosity of summary statistics

`--save_summaries_steps:` save summaries after every given steps

1.6.2 Training on Single-Node with Multiple Workers

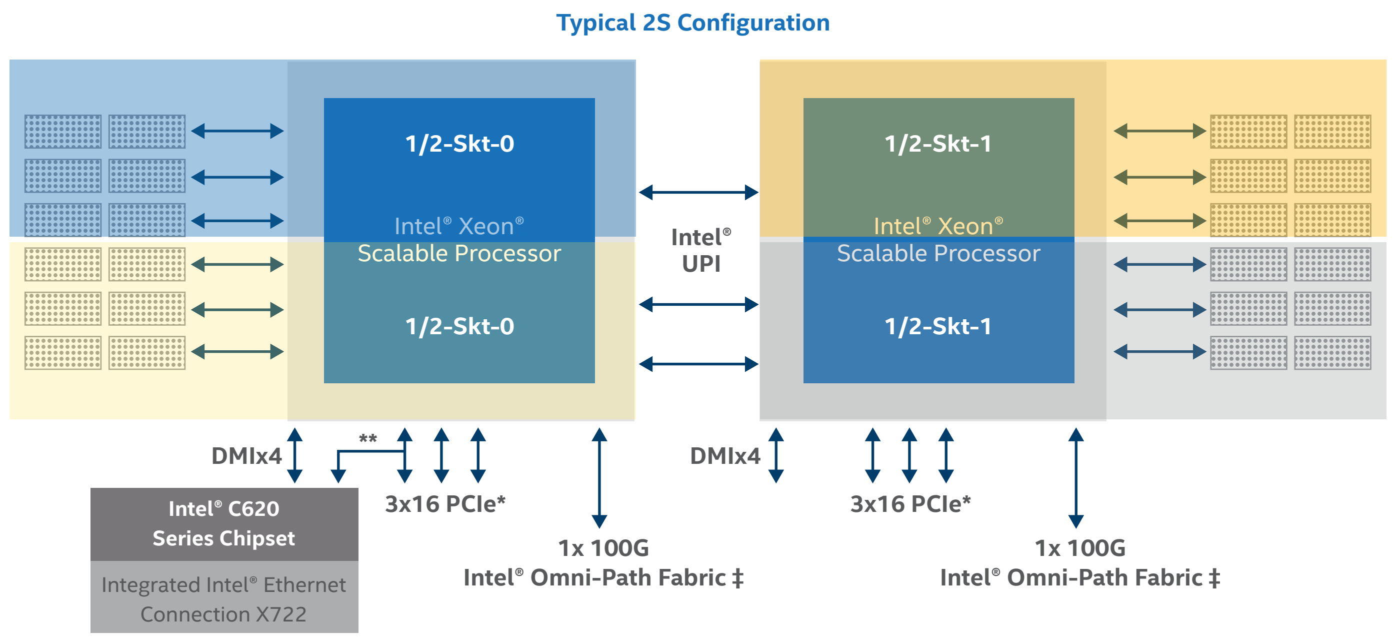
The key idea here is to use multiple training instances or workers even on a single CPU and divide the cores uniformly across them. In addition to core-pin, the workers also use NUMA-aware core affinity and data placement to exploit local memory channels of the sockets. The detailed methodology and analysis are presented at https://software.intel.com/en-us/articles/boosting-deep-learning-training-inference-performance-on-xeon-and-xeon-phi

One example is showcased below. In this scenario, we are running training ResNet-50 on 2S Intel® Xeon® CPU.

Platform Configuration: 2S Intel® Xeon® Gold 6148 processor @ 2.40 GHz, 20 cores/CPU, Intel® Hyper-Threading Technology (Intel® HT Technology) enabled, turbo disabled, scaling governor set to "performance" via intel_pstate driver, 192 GB DDR4-2666 ECC RAM. CentOS Linux* release 7.3.1611 (Core), Linux kernel 3.10.0-514.10.2.el7.x86_64. SSD: Intel® Solid State Drive Data Center (Intel® SSD DC) S3700 Series.

For a single-node using Horovod, a ResNet-50 training job with 4 TensorFlow worker instances per node can be launched as described below. **Note:** Several parameters including OMP_NUM_THREADS, affinity domains, etc., need to be set according to the number of physical cores available. To know which CPUs to bind the processes to, run "numactl –H" and observe how the CPUs are mapped to sockets and nodes.

The MPI tasks partitioning on the sockets of the node in this case is shown in figure 1.



Typical 2S Configuration
1/2-Skt-0
Intel® Xeon® Scalable Processor
1/2-Skt-0
1/2-Skt-1
Intel® Xeon® Scalable Processor
1/2-Skt-1
Intel® UPI
DMIx4
**
3x16 PCIe*
1x 100G Intel® Omni-Path Fabric ‡
DMIx4
3x16 PCIe*
1x 100G Intel® Omni-Path Fabric ‡
Intel® C620 Series Chipset
Integrated Intel® Ethernet Connection X722

Figure 1. MPI tasks Partitioning across Dual-Socket Intel® Xeon® platform

The MPI task Layout is shown in figure 2 below:

Compute Node 0																			
Socket 0																			
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 8	Core 9	Core 10	Core 11	Core 12	Core 13	Core 14	Core 15	Core 16	Core 17	Core 18	Core 19
Task 0	Task 0	Task 0	Task 0	Task 0	Task 0	Task 0	Task 0	Task 0	Task 0	Task 1	Task 1	Task 1	Task 1	Task 1	Task 1	Task 1	Task 1	Task 1	Task 1
Socket 1																			
Core 20	Core 21	Core 22	Core 23	Core 24	Core 25	Core 26	Core 27	Core 28	Core 29	Core 30	Core 31	Core 32	Core 33	Core 34	Core 35	Core 36	Core 37	Core 38	Core 39
Task 2	Task 2	Task 2	Task 2	Task 2	Task 2	Task 2	Task 2	Task 2	Task 2	Task 3	Task 3	Task 3	Task 3	Task 3	Task 3	Task 3	Task 3	Task 3	Task 3

Figure 2. MPI task layout on host0

1.6.3 Using Open MPI

```
HOROVOD_FUSION_THRESHOLD=134217728 \
mpirun -np 4 \
--map-by ppr:2:socket:pe=10 \
-H localhost \
--report-bindings \
--oversubscribe \
-x HOROVOD_FUSION_THRESHOLD \
-x OMP_NUM_THREADS=10 \
python /path/to/tf_cnn_benchmarks.py  $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS \
--data_dir=/path/to/imagenet-1k-TFRecords \
--data_name=<dataset_name>
```

In the above command line, use localhost to run the MPI job locally. Otherwise, use the target hostname on which you wish to run the workload. $COMMON_ARGS contains the arguments to the benchmark script. Rest of the arguments and their recommended values are:

- HOROVOD_FUSION_THRESHOLD=user defined size in bytes, default is 67108864(64Mbytes). This sets the size of the fusion buffer for Horovod's Tensor Fusion, which controls how tensors are batched to do allreduce. For more information please see: https://github.com/uber/horovod/blob/master/docs/tensor-fusion.md
- --np 4: runs 4 copies of the program on the given node(s).
- --map-by ppr:2:socket:pe=10: starts two processes per socket and binds each of them to 10 processing elements (pe), where "ppr" here stands for processes per resource, and "pe" should correspond to the number of physical cores per socket. The number for "pe" should be the same as num_intra_threads as described later.
- -H localhost: run the copies of the program on localhost
- --report-bindings: flags provides a verbose output of the mapping of cores to processes.
- --oversubscribe: is necessary to run multiple processes on a node.
- OMP_NUM_THREADS specifies the number of threads to be used by the OpenMP* (shared memory) library used by Eigen and Intel MKL. This should be set to number of physical cores (same as num_intra_threads).
- python /path/to/tf_cnn_benchmarks.py $COMMON_ARGS: Python program to run with CLI arguments
- --num_intra_threads=set to (number of physical cores per node)/(number of workers per node). The idea is to equally divide cores among the workers. This sets the number of threads an op can use to parallelize execution, used by Eigen and Intel Math Kernel Library.
- --data_dir: path to dataset TFRecords
- --data_name: dataset name

1.6.4 Using Intel® MPI

```
mpiexec \
--machinefile <hostfile> \
-genv I_MPI_DEBUG 4 \
-np 4 \
--ppn 4 \
--genv OMP_NUM_THREADS $OMP_NUM_THREADS \
--genv I_MPI_PIN_DOMAIN 20:compact \
--genv HOROVOD_FUSION_THRESHOLD 134217728 \
python /path/to/tf_cnn_benchmarks.py  $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS \
--data_dir=/path/to/imagenet-1k-TFRecords \
--data_name=<dataset_name>
```

Where hostfile is the file containing the hostname.

- I_MPI_DEBUG=4 prints out the debug information about process pinning. Please refer https://software.intel.com/en-us/mpi-developer-reference-linux-other-environment-variables for more details.
- I_MPI_PIN_DOMAIN "20:compact" maps each process to 20 logical processors located as close to each other in terms of common resources. This should be set according to the number of available physical cores on the system. Please refer https://software.intel.com/en-us/mpi-developer-reference-linux-interoperability-with-openmp-api for more details.

*1.6.5 Using MVAPICH2**

```
/usr/mpi/gcc/mvapich2-2.2-hfi/bin/mpirun_rsh \
--export -np 4 \
--hostfile <hostfile> \
MV2_SHOW_CPU_BINDING=1 \
OMP_NUM_THREADS=10 \
MV2_ENABLE_AFFINITY=1 \
MV2_CPU_MAPPING=0-9,40-49:10-19,50
-59:20-29,60-69:30-39,70-79 \
HOROVOD_FUSION_THRESHOLD 134217728 \
python /path/to/tf_cnn_benchmarks.py  $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS \
--data_dir=/path/to/imagenet-1k-TFRecords \
--data_name=<dataset_name>
```

Where hostfile is the file containing the hostname.

- MV2_SHOW_CPU_BINDING=1 displays the CPU bindings
- MV2_ENABLE_AFFINITY=1 and MV2_CPU_MAPPING are used to map each process to specific cores. This ensures that there is no thread migration to different sockets. To know which CPUs to bind the processes to, run "numactl –H" and observe how the CPUs are mapped to sockets and nodes.

1.6.6 Training on Multiple Nodes with Multiple Workers

To scale to multiple nodes, we spawn multiple workers per node and utilize Horovod over MPI to synchronize gradients (as shown in Section 3.2). The MPI command looks different now as it must include all the host names. Example command line parameters are shown to train ResNet-50 model on multiple dual socket Intel Xeon processors saving model periodically in a model check-point directory.

Cluster Configuration: The 2x dual socket Intel Xeon Gold 6148 CPU platforms are connected with Intel® Omni-Path Host Fabric Interface (Intel® OP HFI), Intel® Omni-Path Driver (Intel® OP Driver) version 10.4.2.0.7. SSD: Intel SSD DC S3700 Series.

For a training job on 2 nodes with 4 TensorFlow workers per node using Open MPI is launched as follows:

```
HOROVOD_FUSION_THRESHOLD=134217728 \
mpirun --np 8 \
--hostfile <hostfile> \
--map-by ppr:2:socket:pe=10 \
--report-bindings \
--oversubscribe \
-x LD_LIBRARY_PATH \
-x HOROVOD_FUSION_THRESHOLD
-x OMP_NUM_THREADS=10 \
python /path/to/tf_cnn_benchmarks.py  $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS \
--data_dir=/path/to/imagenet-1k-TFRecords \
--data_name=<dataset_name> \
--train_dir=/path/to/ model-checkpoint-dir \
--save_model_secs=<time-in-seconds-to-save-models-periodically>
```

It is important to ensure that mpirun binary, path to the training dataset and path to the tf_cnn_benchmarks are consistent in all the nodes and is in the PATH environment variables.

1.6.7 Evaluating the Accuracy of the Trained Model

Inference is the process to evaluate the classification or (prediction) score of the trained model on a validation dataset (images in case of ResNet-50). Due to artifacts of the model restore method in TensorFlow, the best practice is to repeat all the command line parameters used during training for inference with the evaluation flag set to True. Hence, we are going to use the Horovod variable update command line parameters similar to training although they are not relevant for inference. In our example for ResNet-50 model, evaluation is run with and rest of the training parameters as shown below.

Platform Configuration: 2S Intel Xeon Gold 6148 CPU @ 2.40GHz, 20 cores/CPU, Intel HT Technology enabled, turbo disabled, scaling governor set to "performance" via intel_pstate driver, 192GB DDR4-2666 ECC RAM. CentOS Linux release 7.3.1611 (Core), Linux kernel 3.10.0-514.10.2.el7.x86_64. SSD: Intel SSD DC S3700 Series.

```
mpirun --np 8 \
--hostfile <hostfile>  \
--map-by ppr:2:socket:pe=10 \
--report-bindings \
--oversubscribe \
-x LD_LIBRARY_PATH \
-x HOROVOD_FUSION_THRESHOLD
-x OMP_NUM_THREADS=10 \
python /path/to/tf_cnn_benchmarks.py  $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS \
--data_dir=/path/to/imagenet-1k-validation-TFRecords \
--train_dir=/path/to/model-checkpoint-dir \
--num_epochs=1 \
--data_name=<dataset_name> \
--eval=True
```

Be sure to use the appropriate executable according to the version of MPI you are using. Note that the path to the data directory must point to validation dataset files, not training files. Also, the path to the checkpoint directory is passed as the train_dir. The required files in the checkpoint directory are:

- checkpoint – contains the latest checkpoint number
- graph.pbtxt – the text version of the protocol buffer graph definition. This file contains the operation names and its inputs and outputs
- model.ckpt – Model checkpoint
- model.ckpt.index – Index of the weights and tensors
- model.ckpt.data – Weights of the model checkpoint

The output of running ImageNet2012-1K validation dataset (50K Images) for one epoch i.e. ONCE over the entire validation records for ResNet-50 trained model is shown below:

```
Accuracy @ 1 = 0.7513 Accuracy @ 5 = 0.9234
```

1.6.8 Multi-Stream Inference on the Trained Model

We create multiple independent deep learning inference framework instances, and set affinity for each instance to a partitioned set of cores and memory locality on single or multiple socket systems. Figure 3 shows an example of 8 framework instances, each concurrently processing a separate stream of input data on affinitized threads and memory locality. Depending on the inference batch size and system memory capacity, one could have even larger number of frameworks and streams, each mapped to different cores, even 1 stream per core.

In this scenario, we have K workers per node. The performance is measured by the total throughput in images/sec per node with K streams of input each at a given batch size BSize and processed by the K workers. The total number of batches per node on K workers for inference would then be equal to K*BSize. Please refer to the white paper: "Boosting Deep Learning Training & Inference Performance on Intel® Xeon® and Intel® Xeon Phi™ Processors", January 2018, for more information.

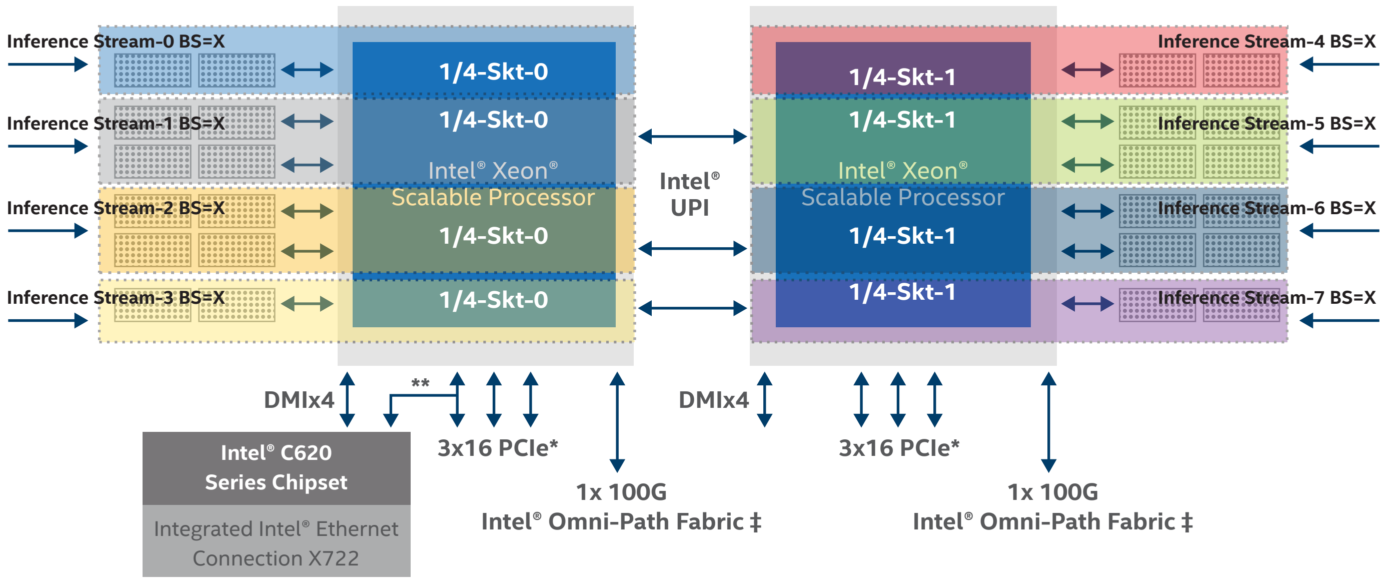


Figure 3. Sub-socket Partitioning across Dual-Socket Intel® Xeon® platforms for Multiple Inference Streams

1.6.8.1 Inference on the Trained Model

To run Inference on the trained model, we use the same command that was used for evaluating the accuracy with following modifications. *data_dir* would need to be pointed to the inference dataset and *train_dir* needs to point to the directory containing the checkpoints for inference model.

The commands below show how users can run eight inference streams on a dual-socket Intel Xeon Gold 6148 processor platform with 20 cores per CPU, with each stream affinitized to five cores. Note the use of `numactl` and `KMP_AFFINITY` for improved performance.

1.6.8.2 Inference Commands

For example, on a dual socket Intel Xeon Gold 6148 processor system with Intel HT Technology enabled, the core mapping using *`numactl –H`* will show as following, i.e. cores on CPU0 (socket-0) are mapped sequentially from 0-19 & 40-59 and cores on CPU1 (socket-1) are mapped sequentially from 20-39 & 60-79.

```
$ numactl –H

available: 2 nodes (0–1)

node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46 47 48 49 50 51 52
53 54 55 56 57 58 59

node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79
```

The commands below, one for each stream show how each TensorFlow instance is mapped to specific cores in the NUMA domain with mandatory memory locality.

```
#!/bin/bash
sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"
TF_CNN_BMARKS=/path/to/tf_cnn_benchmarks/
DATA_DIR=/path/to/imagenet-1k-validation-TFRecords/
TRAIN_DIR=/path/to/model-checkpoint-dir/

export OMP_NUM_THREADS=5 # 20C/skt * 2 Skt / Num_Streams
export KMP_BLOCKTIME=1
export KMP_SETTINGS=0

NUMA_DOMAIN0=0
NUMA_DOMAIN1=1

COMMON_ARGS="--batch_size=64 --model=resnet50 --num_inter_threads=2 --display_every=5 --data_format=NCHW
--optimizer=momentum --device=cpu --mkl=TRUE --variable_update=horovod --horovod_device=cpu --local_
parameter_device=cpu --kmp_blocktime=1"

#start Inference stream 0 on CPU0
KMP_AFFINITY='granularity=thread,proclist=[0-4,40-44],explicit,verbose' \
taskset -c 0-4,40-44 numactl -m $NUMA_DOMAIN0 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-0.log &

#start Inference stream 1 on CPU0
KMP_AFFINITY='granularity=thread,proclist=[5-9,45-49],explicit,verbose' \
taskset -c 5-9,45-49 numactl -m $NUMA_DOMAIN0 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-1.log &

#start Inference stream 2 on CPU0
KMP_AFFINITY='granularity=thread,proclist=[10-14,50-54],explicit,verbose' \
taskset -c 10-14,50-54 numactl -m $NUMA_DOMAIN0 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-2.log &

#start Inference stream 3 on CPU0
KMP_AFFINITY='granularity=thread,proclist=[15-19,55-59],explicit,verbose' \
taskset -c 15-19,55-59 numactl -m $NUMA_DOMAIN0 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-3.log &

#start Inference stream 4 on CPU1
KMP_AFFINITY='granularity=thread,proclist=[20-24,60-64],explicit,verbose' \
taskset -c 20-24,60-64 numactl -m $NUMA_DOMAIN1 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-4.log &

#start Inference stream 5 on CPU1
KMP_AFFINITY='granularity=thread,proclist=[25-29,65-69],explicit,verbose' \
taskset -c 25-29,65-69 numactl -m $NUMA_DOMAIN1 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-5.log &

#start Inference stream 6 on CPU1
KMP_AFFINITY='granularity=thread,proclist=[30-34,70-74],explicit,verbose' \
taskset -c 30-34,70-74 numactl -m $NUMA_DOMAIN1 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-6.log &

#start Inference stream 7 on CPU1
KMP_AFFINITY='granularity=thread,proclist=[35-39,75-79],explicit,verbose' \
taskset -c 35-39,75-79 numactl -m $NUMA_DOMAIN1 \
python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS --eval=True \
--train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-7.log &
```

Script 1: Sample Commands for 8 Concurrent Inference Streams

1.6.8.3 Inference Script

We also provide a generalized Inference script that can be used with different Intel Xeon processor SKUs. The Inference script is written with the assumption that core IDs are mapped sequentially to nodes as indicated earlier.

```
#!/bin/bash
sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"
export OMP_NUM_THREADS=5
export MKL_NUM_THREADS=5
export KMP_BLOCKTIME=1
export KMP_SETTINGS=0

NUM_CORES=40
NUM_CORES_PER_SOCK=20

num_workers=$(($NUM_CORES/$OMP_NUM_THREADS))
echo "Total Cores: $NUM_CORES"
echo "Num Workers: $num_workers"
echo "Physical Cores per worker: $OMP_NUM_THREADS"
ht=$(lscpu | grep "Thread(s) per core:" | awk '{print $NF}')

COMMON_ARGS="--batch_size=64 --model=resnet50 --num_inter_threads=2 --display_every=5 --data_format=NCHW
--optimizer=momentum --device=cpu --mkl=TRUE --variable_update=horovod --horovod_device=cpu --local_
parameter_device=cpu --kmp_blocktime=1"
TF_CNN_BMARKS=/path/to/tf_cnn_benchmarks/
DATA_DIR=/path/to/imagenet-1k-validation-TFRecords/
TRAIN_DIR=/path/to/model-checkpoint-dir/

for ((i=0;i<$num_workers;i++));
do
  phy_core_start=$(($i*$OMP_NUM_THREADS))
  log_core_start=$((($i*$OMP_NUM_THREADS)+$NUM_CORES))

  phy_core_list=()
  log_core_list=()
  for ((t=0;t<$OMP_NUM_THREADS;t++));
  do
     phy_core_list+=("$(($t+$phy_core_start))")
     log_core_list+=("$(($t+$log_core_start))")
  done;

  kmp_affinity="explicit,granularity=fine,proclist=[$(echo ${phy_core_list[*]} | sed 's/ /,/g')]"
  export KMP_AFFINITY=${kmp_affinity}

  if [ $phy_core_start -le $(($NUM_CORES_PER_SOCK-1)) ]; then
```

```
      numa_domain=0
  else
      numa_domain=1
  fi


  if [ $ht -gt 1 ]; then
    echo “taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’),$(echo ${log_core_list[*]} | sed ‘s/ 
/,/g’) numactl –m $numa_domain “
    taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’),$(echo ${log_core_list[*]} | sed ‘s/ /,/g’) 
numactl –m $numa_domain python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS --num_intra_threads=$OMP_
NUM_THREADS --eval=True --train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-$i.log &
  else
    echo “taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’) numactl –m $numa_domain “
    taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’) numactl –m $numa_domain python $TF_CNN_BMARKS/
tf_cnn_benchmarks.py $COMMON_ARGS --num_intra_threads=$OMP_NUM_THREADS --eval=True --train_dir=$TRAIN_DIR 
--data_dir=$DATA_DIR 2>&1 | tee worker-$i.log &
  fi
done;
```

Script 2: Sample Multi-Stream Inference Script

2. Using Singularity*

2.1 Installing Singularity*

If Singularity* is already installed on your cluster/system, then you can skip this step. If not, install Singularity as root/sudo user using the Script_4 below by replacing the --prefix path accordingly. Latest installation instructions are also available at https://singularity.lbl.gov/install-linux

Also refer to instructions at http://opensciencegrid.org/docs/worker-node/install-singularity/#enabling-unprivileged-mode-for-singularity for more instructions and options for installing singularity on your cluster.

2.2 Building Singularity Image

If you already have a Singularity image/container, then you can skip this step. If not, build a Singularity image comprising of CentOS, Intel® Optimization for TensorFlow*, Horovod, Open MPI and TensorFlow benchmarks as root/sudo user

```
sudo /path/to/singularity/install/bin/singularity build tf-horovod.simg tf-horovod.singularity
```

where tf-horovod.singularity is the Singularity recipe file that defines the configuration of the image. If you want the image to be writable, you can pass the –writable flag while building and invoking the container. For more details, refer to the documentation at https://singularity.lbl.gov/docs-build-container#--writable.

Below are two recipe files, either of which can be used based on your use case.

The singularity image can be built with the pre-built optimized TensorFlow wheel pulled from a URL or from conda*. A sample recipe file for this is provided in Script_5.

Alternatively, one can also build a singularity image using a TensorFlow wheel that was built from source on the local file system. The sample recipe file for this is Script_6.

2.3 Running TensorFlow With Singularity

Check the environment of the Singularity Image with Singularity run

```
 [user@linux ~]$ singularity run tf-horovod.simg
This is a Singularity image containing Intel optimized TensorFlow installation with Horovod (OpenMPI)
Operating System: CentOS Linux release 7.5.1804 (Core)
GCC: gcc (GCC) 6.2.0
```

```
TensorFlow: 1.9.0
Horovod: 0.13.11
OpenMPI: 1.10.7
TensorFlow Benchmarks: /opt/tensorflow-benchmarks
```

Run TensorFlow Benchmarks using Singularity shell:

For interactive development, shell into the image:

```
[user@linux ~]$ singularity shell tf-horovod.simg
Singularity: Invoking an interactive shell within container...
Singularity tf-horovod.simg:~ >
```

To run using *Singularity shell*, start a Singularity shell as described above, and launch the mpirun command (described in section: Running tf_cnn_benchmarks (Example ResNet-50) above). Make a note to use the right path for the TensorFlow benchmarks. This path should point to the location of tf_cnn_benchmarks within the image where they were cloned while creating the image

(eg.: `"/opt/tensorflow-benchmarks/"`**).**

Run TensorFlow Benchmarks using Singularity exec:

To run using *singularity exec*, the same version of Open MPI needs to exist within the image, and on the host system. For example, if Open MPI 1.10.7 is installed on the host system, make sure the image is built with the same Open MPI 1.10.7. Failure to match the MPI versions might cause unexpected failures. The Script_7 (run_singularity_multiworker.sh) included in this document uses the 'singularity exec' command to launch the multi-node tf_cnn_benchmark run with ResNet 50. This script should be launched from the host machine.

Eg:

```
[user@linux ~]$chmod 755 run_singularity_multiworker.sh
[user@linux ~]$./run_singularity_multiworker.sh
```

Note that the script uses --*bind* to bind the directory containing the data on the host to the image file system. This enables us to specify the data directory and data name while launching the benchmarks.

Use `--bind <path/to/some/host/directory>:<path/to/directory/in/image>` while launching singularity shell/exec to bind the host directory to the specified directory in the image. Some additional information about binding directories is available at http://singularity.lbl.gov/docs-mount. This can be used to bind the directory containing the ImageNet TF_Records to the image and running the benchmarks.

Some HPC clusters (e.g. TACC) do not allow user-specified binding. In case of TACC, some directories (e.g. /home1, /work, /scratch) are bound directly. A bind point needs to exist for these paths in the image. In that case, while building the image, these directories can be created in the image, and the data on the host can be stored in one of these to be accessed from within the image.

To run the benchmarks using dummy data, just exclude the arguments `data_dir` and `data_name` from the run command.

3. Using Nfs and Slurm*

While the previous section showed the job command line, the user may need to change how the jobs are launched depending on the environment. Large scale systems with thousands of nodes typically use schedulers such as Slurm to launch and manage jobs. We provide example scripts to run the benchmark in this environment. Additionally, we also cover alternatives in smaller scale systems that may not have such a scheduler system in place. The scripts include the run-time optimization Best Practices described above. The data set must be preprocessed as described in the previous section and copied onto the local SSDs of all nodes.

3.1 Using NFS Mounted File System

If your cluster has an NFS mounted drive, the TensorFlow installation and the benchmarks can reside on the NFS mounted drive. However, the dataset must preferably reside on local disc storage (preferably on SSDs) on all nodes as opposed to NFS drives, unless Lustre is being used.

- Ensure that you set up all nodes similarly (GCC (GNU* Compiler Collection) version, MPI version) as described in the steps above

- Install TensorFlow in a Python virtual environment as described in sections 1 and 2 by creating the virtualenv directory on the NFS mounted drive
- Install Horovod in the virtualenv following the steps described above
- Clone the tf_cnn_benchmarks in the NFS mounted directory

```
mkdir <NFS_dir>/tensorflow_benchmarks
cd <NFS_dir>/tensorflow_benchmarks
git clone https://github.com/tensorflow/benchmarks.git
```

- Run the benchmark by adapting the sample command line above to the cluster environment

3.2 Using Slurm* Scheduler

We provide example scripts showing how to launch jobs with Slurm. The scripts provided use the Best Practices from these optimizations listed in the blog above. In the scripts given in Section 8.1, edit the #SBATCH parameters accordingly to change the number of nodes and ranks per node.

4. Tensorflow* Build Instructions

4.1 Building TensorFlow

1. Clone TensorFlow from: https://github.com/tensorflow/tensorflow, and checkout r1.9.0.
2. Build TensorFlow 1.9 with instructions from AI Products Group/Intel TensorFlow optimization team described in the Appendix at the end of this document described below.
3. The process for building with Intel MKL ML or Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN) is very similar. No specific steps are needed during configuration to build TensorFlow using Intel MKL. It is automatically configured when you run configure. You need to only add --config=mkl to Bazel* build. In addition, make sure to set $TEST_TMPDIR to a local directory, instead of an NFS mount directory, or some random errors could happen.
4. When you get to the question about "copt" parameters, just accept the default. You'll specify the flags explicitly in the Bazel build command
5. Use the default for all other settings. Choose as following:

```
a. No cloud
b. No Hadoop
c. No opencl
d. No rdma/verbs
e. No cuda
f. No mpi
g. Yes xla, xla may give improved performance
```

6. Building for Intel MKL-DNN: Please use the following build command:
7. `bazel build --config=mkl --copt="-mfma" --copt="-mavx2" --copt="-march=broadwell" --copt="-O3" -s -c opt //tensorflow/tools/pip_package:build_pip_package`
8. When everything finally works, this is what you'll see: Success!
    a. Target //tensorflow/tools/pip_package:build_pip_package up-to-date: bazel-bin/tensorflow/tools/pip_package/build_pip_package
9. Building the Python Wheel
    a. Remove any existing wheels
    b. `rm ~/tmp/*.whl`
    c. From the root directory of your TensorFlow sources, create the whl file:

```
./bazel-bin/tensorflow/tools/pip_package/build_pip_package ~/tmp
```

10. Installing and running TensorFlow
    a. Remove any existing TensorFlow installations
    b. `pip uninstall tensorflow`
11. Install the wheel for your user account.
    a. `pip install ~/tmp/*.whl --user` #This will install any missing dependencies into your user profile
    b. `pip install ~/tmp/*.whl --user  --no-deps` #This will skip installing dependencies
12. Make sure everything is working. Try importing the TensorFlow lib. If you don't get any errors, all is well
    a. python
    b. >>> import tensorflow
    c. >>> quit()
13. Try a simple kernel test
    a. `python <tensorflow src directory>/tensorflow/python/kernel_tests/relu_op_test.py`

Build Notes

1. -mavx512f -march=broadwell and other AVX512 flags need GCC 4.9 or higher (Recommended to use GCC 5.4 or higher)
2. Currently, TensorFlow build with Intel® C Compilers and Intel® C++ Compilers has some issues.
3. To use an alternate compiler (e.g. GCC 6.3):
4. change PATH to point to alternate compiler:
5. export PATH=/opt/intel/gcc6.3/bin:$PATH
6. add -L flag to Bazel build command line above: --copt="-L/opt/intel/gcc6.3/lib64"
7. change LD_LIBRARY_PATH when running the TensorFlow programs to point to new GLIBC:

```
export LD_LIBRARY_PATH=/opt/intel/gcc6.3/lib64:$LD_LIBRARY_PATH
```

For debug builds (no optimization), use

```
bazel build --config=mkl --copt="-DEIGEN_USE_VML" --copt="-g" -s -c dbg //tensorflow/tools/pip_
package:build_pip_package
```

For debug symbols only add --copt="-g" to Bazel build command.

4.2 Install TensorFlow using script

The build_tensorflow_script can be used to install TensorFlow in a virtual environment. At the prompt, choose "y" to build from source. The script creates a virtual environment and installs TensorFlow in this environment, as opposed to doing a system-wide install. The virtual environment is created in <path/to/install/directory>/virtenv/ , and can be activated as follows:

```
source <path/to/install/directory>/virtenv/bin/activate
```

To deactivate the virtual environment, run:

```
deactivate
```

5. Sample Scripts

5.1 TensorFlow build script

In the script below, change the highlighted parts according to the TensorFlow version that you want to work with.

The script saves the TensorFlow wheel that is installed in the virtual environment (either built from source or downloaded) in the folder <path/to/install/directory>/tf_whl/.

```
#!/bin/bash

set -e
set -o pipefail

if [[ $# -lt 1 ]]
then
    echo "Usage $0 <dir>"
    echo "TensorFlow will be cloned/installed into <dir>"
    exit 1
fi

dir=$1
rm -rf $dir

export PYTHONPATH=`which python`
echo "Default python found at $PYTHONPATH"

while true; do
    read -p "Do you want to use this python? [y/n] " yn
    case $yn in
        [Yy]* ) py_ins=1; break;;
        [Nn]* ) py_ins=0; break;;
        * ) echo "Please answer y/n.";;
    esac
done

if [[ $py_ins -eq 0 ]]
then
       read -e -p "Enter location of python to be used: " new_path
       python_vers=`$new_path --version`
       echo "Python at $new_path, of version $python_vers"
       export PYTHONPATH=$new_path
fi

while true; do
    read -p "Do you want to build tensorflow from source? [y/n] " yn
    case $yn in
        [Yy]* ) tf_ins=1; break;;
        [Nn]* ) tf_ins=0; break;;
        * ) echo "Please answer y/n.";;
    esac
done
```

```
clonedir=`realpath $dir`
mkdir -p $clonedir
cd $clonedir

if [[ $tf_ins -eq 1 ]]
then
    echo “Building from source at: $clonedir”

    # Check GCC version
    minv=”5.2.0”
    gccv=`gcc --version | grep “(GCC)” | awk ‘{print $3}’`
    if [ “$(printf ‘%s\n’ $minv $gccv | sort -V | head -n 1)” != “$minv” ]; then
      echo “Need a newer GCC than $gccv”
      exit
    fi
    [[ $(type -P bazel) ]] ||
        { echo “Could not find bazel. Is it in PATH?” 1>&2; exit 1; }

    git clone https://github.com/tensorflow/tensorflow.git
    cd tensorflow
    git checkout r1.9

    testdir=/tmp/$USER
    whldir=”$testdir/tf_whl”
    mkdir -p $whldir
    export TEST_TMPDIR=$testdir

    # TF related env var
    export TF_NEED_GCP=0
    export TF_NEED_HDFS=0
    export TF_NEED_JEMALLOC=1
    export TF_NEED_OPENCL=0
    export TF_NEED_CUDA=0
    export TF_NEED_GDR=0
    export TF_CUDA_CLANG=0
    export TF_ENABLE_XLA=0
    export TF_ENABLE_GDR=0
    export TF_NEED_S3=0
    export TF_NEED_KAFKA=0
    export TF_NEED_OPENCL_SYCL=0
    export TF_NEED_VERBS=0
    export TF_NEED_MPI=0
    export TF_SET_ANDROID_WORKSPACE=0
    
    export CC_OPT_FLAGS="-march=broadwell --copt=\"-mfma\" --copt=\"-mavx2\" --copt=\"-O3\" "
    echo "Configuring tensorflow build..."
    ./configure

    bazel build --config mkl --copt="-mavx2" --copt="-O3" --copt="-mfma" --copt="-march=broadwell"  -s -c 
opt //tensorflow/tools/pip_package:build_pip_package
    
    cd $clonedir/tensorflow
    echo "Building wheel..."
    bazel-bin/tensorflow/tools/pip_package/build_pip_package $whldir

else
   echo "Downloading Optimized pre-built wheel..."
   
   whldir="$clonedir/tf_whl"
   mkdir -p $whldir
   cd $whldir
   wget https://storage.googleapis.com/intel-optimized-tensorflow/tensorflow-1.9.0-cp27-cp27mu-linux_x86_64.
whl
fi

virtdir="$clonedir/virtenv"
echo "Creating virtualenv directory in $virtdir..."
mkdir -p $virtdir
cd $virtdir

echo "Creating virtualenv..."
$PYTHONPATH -m virtualenv $virtdir

whl=`ls $whldir/*.whl`

echo "$whl"
cd $virtdir
source ./bin/activate
echo "Installing wheel in virtualenv..."
pip install --upgrade pip
pip install $whl
echo "TensorFlow is installed in $virtdir using wheel $whl"
```

Script 1: build_tf.sh: file to either build TensorFlow* from source, or download the pre-built Intel® optimized wheel and install in a Python* virtual environment

5.2 Slurm scripts

Here are two sample scripts, skx_2nodes.job (Script 2) and run.sh (Script 3). Script 2 controls the arguments to the multi-node job, and Script 3 controls the arguments to the Python script. The Script 2 runs the script 3.

```
#!/bin/bash

#SBATCH -J skx_resnet50
#SBATCH -o out_skx_resnet50_%j
#SBATCH -e err_skx_resnet50_%j
#SBATCH -N 2
#SBATCH -n 8
#SBATCH -p skx-normal
#SBATCH -t 00:45:00

cur_date=`date +%F-%H-%M-%S`
model="resnet50"

# edit result directory path
result_dir="path_to_res_dir"
mkdir -p ${result_dir}
mkdir -p ${result_dir}/train
curdir=`pwd`
script="${curdir}/run.sh"
echo "Running script $script"

unset OMP_NUM_THREADS
unset KMP_BLOCKTIME
unset KMP_AFFINITY

export OMP_NUM_THREADS=24
export KMP_BLOCKTIME=1
export I_MPI_FABRICS="shm:tmi"
export I_MPI_TMI_PROVIDER="psm2"
export I_MPI_PIN_DOMAIN="socket"
export I_MPI_FALLBACK=0

# optional for debug
export I_MPI_DEBUG="3"
export I_MPI_DEBUG_OUTPUT="${result_dir}/debug_%r.dbg"

thresh=$(( 16 * 1024 * 1024 ))
unset HOROVOD_FUSION_THRESHOLD
export HOROVOD_FUSION_THRESHOLD=$thresh
```

```
mpienv="-genv I_MPI_DEBUG $I_MPI_DEBUG -genv LD_LIBRARY_PATH $LD_LIBRARY_PATH -genv HOROVOD_FUSION_
THRESHOLD $HOROVOD_FUSION_THRESHOLD -genv OMP_NUM_THREADS $OMP_NUM_THREADS -genv KMP_BLOCKTIME $KMP_
BLOCKTIME -genv I_MPI_FABRICS $I_MPI_FABRICS -genv I_MPI_TMI_PROVIDER $I_MPI_TMI_PROVIDER -genv I_MPI_PIN_
DOMAIN $I_MPI_PIN_DOMAIN -genv I_MPI_FALLBACK $I_MPI_FALLBACK"


module load impi
which mpirun
mpirun --np 8 $mpienv -l ${script} ${thresh} ${result_dir} ${model}


mv out_skx_resnet* ${result_dir}
mv err_skx_resnet* ${result_dir}


# For TACC environment, ibrun is recommended
# export IBRUN_TASKS_PER_NODE=4
# ibrun --np 8 ${script} ${thresh} ${result_dir} ${model}
```

Script 2: Example skx_2nodes.job sample file launches the batch job by defining the SBATCH parameters for 2 nodes, and launches run.sh with mpirun. This is a sample script to run on TACC, and uses Intel® MPI.

```
#!/bin/bash


if [ $# -lt 1 ]
then
    echo "Usage: $0 fusion_threshold"
    exit 1
fi


killall -9 python


thresh=$1
resdir=$2
model=$3
train_dir=$resdir/train
total_procs=${SLURM_NPROCS}
num_nodes=$SLURM_NNODES
procs_per_node=`echo ${SLURM_NPROCS}/$SLURM_NNODES | bc`


if [ ${num_nodes} == "" ]
then
    echo "SLURM_NNODES returned $SLURM_NNODES"
    exit 1
fi
script="/path/to/tensorflow/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py"
```

```
intra_threads=24
inter_threads=2
num_batches=500
data_dir="/tmp/imgnet/"

batch_size=`echo 256/${procs_per_node} | bc`
host=$(hostname)

module load impi
env
lscpu

echo "Num nodes: ${num_nodes} Node: $host total procs: $total_procs batch size: $batch_size intra threads: 
$intra_threads"

args="--batch_size=$batch_size --model=resnet50 \
--num_intra_threads $OMP_NUM_THREADS \
--num_inter_threads 2 \
--display_every 5 \
--data_format NCHW \
--optimizer momentum \
--device cpu \
--enable_layout_optimizer=TRUE"

cmd="cd /path/to/virtualenv/virtenv; source ./bin/activate; python -u $script --model $model $args 
--mkl=TRUE \
--variable_update=horovod \
--horovod_device=cpu \
--local_parameter_device=cpu \
--kmp_blocktime=1 \
--enable_layout_optimizer=TRUE; deactivate "

echo $cmd
eval $cmd
```

Script 3: run.sh Launching Job with Slurm*. run.sh controls launching the Python* script and its arguments

5.3 Singularity scripts

5.3.1 Install script

Edit the highlighted portions below to change the version and the path where Singularity is to be installed

```
#!/bin/bash


VERSION=2.5.1
sudo yum –y install libarchive-devel
sudo yum –y install squashfs-tools


wget https://github.com/singularityware/singularity/releases/download/$VERSION/singularity-$VERSION.tar.gz
tar xvf singularity-$VERSION.tar.gz
cd singularity-$VERSION
./configure --prefix=/path/to/where/singularity/will/be/installed
make
sudo make install
```

Script 4: Installing Singularity*

5.3.2 Recipe file for TensorFlow wheel downloaded from a URL

```
Bootstrap: docker
From: centos:latest


%setup
  export SINGULARITY_SHELL=/bin/bash


%post -c /bin/bash
# If your environment requires proxy to reach internet, then export the http, https proxy variables
accordingly
  export http_proxy=http://proxy-server-name:port
  export https_proxy=http://proxy-server-name:port


  yum clean all
  rm -rf /var/cache/yum
  yum -y install epel-release


  yum -y groupinstall “Development Tools”
  yum -y install wget
  yum -y install git


  #installing gcc


  yum -y install gmp-devel mpfr-devel libmpc-devel glibc-devel.i686 libgcc.i686 gcc-c++
  wget https://ftp.gnu.org/gnu/gcc/gcc-6.2.0/gcc-6.2.0.tar.gz

  tar -xzf gcc-6.2.0.tar.gz
  mkdir build
  cd build
  ../gcc-6.2.0/configure --prefix=/opt/gcc-6.2.0
  make -j 20
  make install

 #adding GCC to $PATH
  export PATH=/opt/gcc-6.2.0/bin:$PATH
  export LD_LIBRARY_PATH=/opt/gcc-6.2.0/lib64:/opt/gcc-6.2.0/lib



  #using anaconda, conda's default tensorflow is Intel optimized
  wget https://repo.anaconda.com/archive/Anaconda2-5.2.0-Linux-x86_64.sh
  bash Anaconda2-5.2.0-Linux-x86_64.sh -b -p /opt/anaconda2
  export PATH=/opt/anaconda2/bin:$PATH

  rm -rf Anaconda2-5.2.0-Linux-x86_64.sh

  yum -y install openmpi openmpi-devel

  yum -y install environment-modules
  source /etc/profile.d/modules.sh
  module add mpi/openmpi-x86_64
  #In case module throws an error (eg. on TACC), use:
  # export PATH=/usr/lib64/openmpi/bin:$PATH
  # export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH

  conda install -y tensorflow
  pip install --no-cache-dir Horovod

  git clone https://github.com/tensorflow/benchmarks /opt/tensorflow-benchmarks

  #uncomment following for running on TACC
  #mkdir –p /home1
  #mkdir –p /scratch
  #mkdir –p /work

%environment
  export http_proxy=http://proxy-server-name:port
  export https_proxy=http://proxy-server-name:port
  export PATH=/opt/gcc-6.2.0/bin:$PATH
  export LD_LIBRARY_PATH=/opt/gcc-6.2.0/lib64:/opt/gcc-6.2.0/lib

  export PATH=/opt/anaconda2/bin:$PATH

  #openmpi
  source /etc/profile.d/modules.sh
  module add mpi/openmpi-x86_64
  #Again, in case module throws an error(eg. on TACC), use:
  # export PATH=/usr/lib64/openmpi/bin:$PATH
  # export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH

%runscript
  echo "This is a Singularity image containing Intel optimized TensorFlow installation with Horovod 
(OpenMPI)"
  echo "Operating System: $(cat /etc/redhat-release)"
  echo "GCC: $(gcc --version)"
  echo "TensorFlow: $(pip show tensorflow --disable-pip-version-check | grep "^Version:" | awk '{print 
$NF}')"
  echo "Horovod: $(pip show horovod --disable-pip-version-check | grep "^Version:" | awk '{print $NF}')"
  echo "OpenMPI: $(ompi_info | grep "Open MPI:" | awk '{print $NF}')"
  echo "TensorFlow Benchmarks: /opt/tensorflow-benchmarks"
```

Script 5: Script 5: tf-horovod.singularity file contents for the case where TensorFlow* is installed from conda*.

5.3.3 Recipe file for TensorFlow wheel on local file system

```
Bootstrap: docker
From: centos:latest

%setup
# Adjust the below source path and file name accordingly
  cp /path/on/local/filesystem/to/tensorflow.whl ${SINGULARITY_ROOTFS}/tensorflow.whl
  export SINGULARITY_SHELL=/bin/bash

%post -c /bin/bash
  # If your environment requires proxy to reach internet, then export the http, https proxy variables 
accordingly
  export http_proxy=http://proxy-server-name:port
  export https_proxy=http://proxy-server-name:port

  yum clean all
  rm -rf /var/cache/yum
  yum -y install epel-release
  yum -y groupinstall "Development Tools"
  yum -y install wget
  yum -y install git

  #installing gcc
  yum -y install gmp-devel mpfr-devel libmpc-devel glibc-devel.i686 libgcc.i686 gcc-c++

  wget https://ftp.gnu.org/gnu/gcc/gcc-6.2.0/gcc-6.2.0.tar.gz

  tar -xzf gcc-6.2.0.tar.gz
  mkdir build
  cd build
  ../gcc-6.2.0/configure --prefix=/opt/gcc-6.2.0
  make -j 20
  make install

  #adding GCC to $PATH
  export PATH=/opt/gcc-6.2.0/bin:$PATH
  export LD_LIBRARY_PATH=/opt/gcc-6.2.0/lib64:/opt/gcc-6.2.0/lib:$LD_LIBRARY_PATH

  yum -y install python-pip python-wheel python-devel

  yum -y install openmpi openmpi-devel


  yum -y install environment-modules
  source /etc/profile.d/modules.sh
  module add mpi/openmpi-x86_64
  #In case module throws an error (eg. on TACC), use:
  # export PATH=/usr/lib64/openmpi/bin:$PATH
  # export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH

  # Replace the tensorflow.whl file in the installation below to the actual file name used in the setup
section
  pip install /tensorflow.whl
  pip install --no-cache-dir Horovod

  # Download the TF benchmarks and place them in /opt inside the image
  git clone https://github.com/tensorflow/benchmarks /opt/tensorflow-benchmarks

  #For running on TACC: uncomment the following lines to bind these directories
  #  mkdir -p /home1
  #  mkdir -p /scratch
```

```
  #  mkdir –p /work

%environment
  export http_proxy=http://proxy-server-name:port
  export https_proxy=http://proxy-server-name:port

  #gcc 6.2.0
  export PATH=/opt/gcc-6.2.0/bin:$PATH
  export LD_LIBRARY_PATH=/opt/gcc-6.2.0/lib64:/opt/gcc-6.2.0/lib:$LD_LIBRARY_PATH

  #openmpi
  source /etc/profile.d/modules.sh
  module add mpi/openmpi-x86_64
  #Again, in case module throws an error(eg. on TACC), use:
  # export PATH=/usr/lib64/openmpi/bin:$PATH
  # export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH

%runscript
  echo “This is a Singularity image containing Intel optimized TensorFlow installation with Horovod
(OpenMPI)”
  echo “Operating System: $(cat /etc/redhat-release)”
  echo “GCC: $(gcc --version | grep “(GCC)” | awk ‘{print}’)”
  echo “TensorFlow: $(pip show tensorflow --disable-pip-version-check | grep “^Version:” | awk ‘{print
$NF}’)”
  echo “Horovod: $(pip show horovod --disable-pip-version-check | grep “^Version:” | awk ‘{print $NF}’)”
  echo “OpenMPI: $(ompi_info | grep “Open MPI:” | awk ‘{print $NF}’)”
  echo “TensorFlow Benchmarks: /opt/tensorflow-benchmarks”
```

Script 6: tf-horovod.singularity file contents for the case where TensorFlow* is installed from local file system. Edit highlighted text depending on your environment, paths and TensorFlow wheel.

5.3.4 Singularity run-script

```
#!/bin/bash

PATH_TO_SINGULARITY=”/path/to/singularity/installation/bin/singularity”
PATH_TO_SIMG=”/path/to/tf-horovod.simg”
SING_EXEC_CMD=”${PATH_TO_SINGULARITY} exec --bind /local/path/to/TF_Records: /image/path/to/TF_Records
${PATH_TO_SIMG}”
PATH_TO_SIMG_TF_BENCH=”/opt/tensorflow-benchmarks/”

OMP_NUM_THREADS=<positive_number>
HOSTNAMES=”hostname1,hostname2,and so on”

args=” \
--batch_size=64 \
--model=resnet50 \
--num_intra_threads=$OMP_NUM_THREADS \
--num_inter_threads=2 \
--display_every=5 \
--data_format=NCHW \
--optimizer=momentum \
--device=cpu "

HOROVOD_FUSION_THRESHOLD=134217728 mpirun --np 4 \
--map-by ppr:2:socket:pe=${OMP_NUM_THREADS} \
-H ${HOSTNAMES} \
--report-bindings \
--oversubscribe \
--allow-run-as-root \
-x LD_LIBRARY_PATH \
-x HOROVOD_FUSION_THRESHOLD \
-x OMP_NUM_THREADS=${OMP_NUM_THREADS} \
${SING_EXEC_CMD} \
python ${PATH_TO_SIMG_TF_BENCH}/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py $args \
--mkl=TRUE \
--variable_update=horovod --horovod_device=cpu \
--local_parameter_device=cpu --kmp_blocktime=1 \
--data_dir=/image/path/to/TF_Records --data_name=imagenet
```

Script 7: `Sample run_singularity_multiworker.sh` . Edit text highlighted in yellow depending on your environment and number of TF workers per node. Also, edit "args" settings for the TF benchmark topology.

5.4 Inference scripts

5.4.1

The Inference script is written with the assumption that core IDs are mapped sequentially to nodes. For example on 2 cpu Intel Xeon Gold 6148 processor 20 cores/cpu system with Intel HT Technology enabled, the core mapping using `numactl –H` will show as following, i.e. cores on CPU0 are mapped sequentially from 0-19 & 40-59 and cores on CPU1 are mapped sequentially from 20-39 & 60-79

```
$ numactl -H
```

available: 2 nodes (0-1)

```
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46 47 48 49 50 51 52
53 54 55 56 57 58 59

node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79
```

```
#!/bin/bash

export OMP_NUM_THREADS=5
export MKL_NUM_THREADS=5

NUM_CORES=40
NUM_CORES_PER_SOCK=20

num_workers=$(($NUM_CORES/$OMP_NUM_THREADS))

echo “Total Cores: $NUM_CORES”
echo “Num Workers: $num_workers”
echo “Physical Cores per worker: $OMP_NUM_THREADS”

ht=$(lscpu | grep “Thread(s) per core:” | awk ‘{print $NF}’)

COMMON_ARGS=”--batch_size=64 --model=resnet50 --num_inter_threads=2 --display_every=5 --data_format=NCHW 
--optimizer=momentum --device=cpu --mkl=TRUE --variable_update=horovod --horovod_device=cpu --local_
parameter_device=cpu --kmp_blocktime=1”
TF_CNN_BMARKS=/path/to/tf_cnn_benchmarks/
DATA_DIR=/path/to/imagenet-1k-validation-TFRecords/
TRAIN_DIR=/path/to/model-checkpoint-dir/

for ((i=0;i<$num_workers;i++));
do
  phy_core_start=$(($i*$OMP_NUM_THREADS))
  log_core_start=$((($i*$OMP_NUM_THREADS)+$NUM_CORES))

  phy_core_list=()
  log_core_list=()
  for ((t=0;t<$OMP_NUM_THREADS;t++));
  do
     phy_core_list+=(“$(($t+$phy_core_start))”)
     log_core_list+=(“$(($t+$log_core_start))”)
  done;

  kmp_affinity=”explicit,granularity=fine,proclist=[$(echo ${phy_core_list[*]} | sed ‘s/ /,/g’)]”
  export KMP_AFFINITY=${kmp_affinity}

  if [ $phy_core_start -le $(($NUM_CORES_PER_SOCK-1)) ]; then
     numa_domain=0
  else
     numa_domain=1
  fi
  if [ $ht -gt 1 ]; then
    echo “taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’),$(echo ${log_core_list[*]} | sed ‘s/ /,/g’) numactl —m $numa_domain “
    taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’),$(echo ${log_core_list[*]} | sed ‘s/ /,/g’) numactl —m $numa_domain python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS --num_intra_threads=$OMP_NUM_THREADS --eval=True --train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-$i.log &
  else
    echo “taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’) numactl —m $numa_domain “
    taskset -c $(echo ${phy_core_list[*]} | sed ‘s/ /,/g’) numactl —m $numa_domain python $TF_CNN_BMARKS/tf_cnn_benchmarks.py $COMMON_ARGS --num_intra_threads=$OMP_NUM_THREADS --eval=True --train_dir=$TRAIN_DIR --data_dir=$DATA_DIR 2>&1 | tee worker-$i.log &
  fi
done;
```

Script 8: Sample `inference.sh`

6. Troubleshooting

6.1 TensorFlow Import Issues

With the most recent pre-built 1.9 wheel, this error might occur on some systems while importing TensorFlow:

6.1.1 Importing TensorFlow

```
>>> import tensorflow
Traceback (most recent call last):
  File “<stdin>”, line 1, in <module>
 File “/path/to/virtualenv/virtenv/lib/python2.7/site-packages/tensorflow/__init__.py”, line 22, in <module>
  from tensorflow.python import pywrap_tensorflow  # pylint: disable=unused-import
 File “/path/to/virtualenv/virtenv/lib/python2.7/site-packages/tensorflow/python/__init__.py”, line 49, in <module>
  from tensorflow.python import pywrap_tensorflow
 File “/path/to/virtualenv/virtenv/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow.py”, line 74, in <module>
  raise ImportError(msg)
ImportError: Traceback (most recent call last):
 File “/path/to/virtualenv/virtenv/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow.py”, line 58, in <module>
 from tensorflow.python.pywrap_tensorflow_internal import *
 File “/path/to/virtualenv/virtenv/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow_internal.py”, line 28, in <module>
 _pywrap_tensorflow_internal = swig_import_helper()
 File “/path/to/virtualenv/virtenv/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow_internal.py”, line 24, in swig_import_helper
 _mod = imp.load_module(‘_pywrap_tensorflow_internal’, fp, pathname, description)
ImportError: /lib64/libm.so.6: version `GLIBC_2.23’ not found (required by /path/to/virtualenv/virtenv/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so)
Failed to load the native TensorFlow runtime.
```

This happens because the wheel requires GLIBC 2.23, but the system has an older version. You can use the following techniques to debug this issue.

6.1.2 Run ldd to find the dynamically linked libraries

```
(virt2.7) [user@linux singularity_testing]$ ldd /path/to/virtualenv/virt2.7/lib/python2.7/site-
packages/tensorflow/python/_pywrap_tensorflow_internal.so

    linux-vdso.so.1 =>  (0x00007fff4df5b000)

    libtensorflow_framework.so => /path/to/virtualenv/virt2.7/lib/python2.7/site-packages/tensorflow/
    python/../libtensorflow_framework.so (0x00007f426cefe000)

    libiomp5.so => /path/to/virtualenv/virt2.7/lib/python2.7/site-packages/tensorflow/python/../../_
    solib_k8/_U@mkl_Ulinux_S_S_Cmkl_Ulibs_Ulinux___Uexternal_Smkl_Ulinux_Slib/libiomp5.so
    (0x00007f426cb5a000)

    libmklml_intel.so => /path/to/virtualenv/virt2.7/lib/python2.7/site-packages/tensorflow/
    python/../../_solib_k8/_U@mkl_Ulinux_S_S_Cmkl_Ulibs_Ulinux___Uexternal_Smkl_Ulinux_Slib/libmklml_
    intel.so (0x00007f4263a89000)

    libdl.so.2 => /lib64/libdl.so.2 (0x00007f4263885000)

    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f4263669000)

    libm.so.6 => /lib64/libm.so.6 (0x00007f4263367000)

    libstdc++.so.6 => /path/to/gcc-7.2.0/lib64/libstdc++.so.6 (0x00007f4262fe6000)

    libgcc_s.so.1 => /path/to/gcc-7.2.0/lib64/libgcc_s.so.1 (0x00007f4262dcf000)

    libc.so.6 => /lib64/libc.so.6 (0x00007f4262a02000)

    /lib64/ld-linux-x86-64.so.2 (0x00007f4273a10000)
```

The libm.so.6 in /lib64 was not finding `GLIBC_2.23'.

6.1.3 Check by running:

```
(virt2.7) [user@linux singularity_testing]$ strings /lib64/libm.so.6 | grep "GLIBC_"

GLIBC_2.2.5

GLIBC_2.4

GLIBC_2.15

GLIBC_PRIVATE

GLIBC_2.15

__strtold_nan@@GLIBC_PRIVATE

__errno_location@@GLIBC_2.2.5

errno@@GLIBC_PRIVATE

qsort@@GLIBC_2.2.5

__strtod_nan@@GLIBC_PRIVATE

fputs@@GLIBC_2.2.5

memset@@GLIBC_2.2.5

feupdateenv@@GLIBC_2.2.5

GLIBC_2.4

_rtld_global_ro@@GLIBC_PRIVATE

__strtof_nan@@GLIBC_PRIVATE

memmove@@GLIBC_2.2.5

fwrite@@GLIBC_2.2.5
```

```
__cxa_finalize@@GLIBC_2.2.5
stderr@@GLIBC_2.2.5
GLIBC_2.2.5
```

Finally, one fix is to build the required version of GLIBC in a user-specified path, and then copying the libm.so.6 to a path that is imported in LD_LIBRARY_PATH.

```
mkdir ~/glibc
cd ~/glibc

wget http://ftp.gnu.org/gnu/glibc/glibc-2.23.tar.gz
tar zxvf glibc-2.23.tar.gz
cd glibc-2.23
mkdir build
cd build

../configure --prefix=/opt/glibc-2.23
make -j4
sudo make install

cp /opt/glibc-2.23/lib/libm.so.6 </some/path>/lib

export LD_LIBRARY_PATH=</some/path>/lib
```

Please make sure that you do not replace the system's GLIBC. Adding /opt/glibc-2.23/lib to LD_LIBRARY_PATH might break other things like ls.

6.1.4 Another Common Error when Importing TensorFlow

```
>>> import tensorflow
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/__init__.py", line 22, in
  <module>
    from tensorflow.python import pywrap_tensorflow  # pylint: disable=unused-import
  File "/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/__init__.py", line 49,
  in <module>
    from tensorflow.python import pywrap_tensorflow
  File "/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow.py",
  line 74, in <module>
    raise ImportError(msg)
ImportError: Traceback (most recent call last):
  File "/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow.py",
  line 58, in <module>
    from tensorflow.python.pywrap_tensorflow_internal import *
  File "/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow_
  internal.py", line 28, in <module>

   _pywrap_tensorflow_internal = swig_import_helper()

  File "/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/pywrap_tensorflow_
  internal.py", line 24, in swig_import_helper

   _mod = imp.load_module('_pywrap_tensorflow_internal', fp, pathname, description)

ImportError: /lib64/libstdc++.so.6: version `CXXABI_1.3.8' not found (required by /path/to/
virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so)

Failed to load the native TensorFlow runtime.
```

This happens if you are using an older version of GCC. To fix this, make sure the correct version of GCC (6.2 or higher) is in PATH and LD_LIBRARY_PATH.

6.1.5 Verify that TensorFlow is Using right the version of GCC

```
(test) [user@linux OOCpackage_testing]$ ldd /path/to/virtualenv/test/lib/python2.7/site-packages/
tensorflow/python/_pywrap_tensorflow_internal.so

/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so:
/lib64/libstdc++.so.6: version `CXXABI_1.3.8' not found (required by /path/to/virtualenv/test/lib/
python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so)

/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so:
/lib64/libstdc++.so.6: version `GLIBCXX_3.4.21' not found (required by /path/to/virtualenv/test/lib/
python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so)

/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so:
/lib64/libstdc++.so.6: version `GLIBCXX_3.4.20' not found (required by /path/to/virtualenv/test/lib/
python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so)

/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so:
/lib64/libm.so.6: version `GLIBC_2.23' not found (required by /path/to/virtualenv/test/lib/python2.7/
site-packages/tensorflow/python/_pywrap_tensorflow_internal.so)

/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so:
/lib64/libstdc++.so.6: version `GLIBCXX_3.4.20' not found (required by /path/to/virtualenv/test/lib/
python2.7/site-packages/tensorflow/python/../libtensorflow_framework.so)

/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so:
/lib64/libstdc++.so.6: version `GLIBCXX_3.4.21' not found (required by /path/to/virtualenv/test/lib/
python2.7/site-packages/tensorflow/python/../libtensorflow_framework.so)

/path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/_pywrap_tensorflow_internal.so:
/lib64/libstdc++.so.6: version `CXXABI_1.3.8' not found (required by /path/to/virtualenv/test/lib/
python2.7/site-packages/tensorflow/python/../libtensorflow_framework.so)

    linux-vdso.so.1 =>  (0x00007ffc2c326000)

    libtensorflow_framework.so => /path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/
    python/../libtensorflow_framework.so (0x00007fec5a4dd000)

    libiomp5.so => /path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/../../_
    solib_k8/_U@mkl_Ulinux_S_S_Cmkl_Ulibs_Ulinux___Uexternal_Smkl_Ulinux_Slib/libiomp5.so
    (0x00007fec5a139000)

    libmklml_intel.so => /path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/
    python/../../_solib_k8/_U@mkl_Ulinux_S_S_Cmkl_Ulibs_Ulinux___Uexternal_Smkl_Ulinux_Slib/libmklml_
    intel.so (0x00007fec51068000)

    libdl.so.2 => /lib64/libdl.so.2 (0x00007fec50e64000)

    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fec50c48000)

    libm.so.6 => /lib64/libm.so.6 (0x00007fec50946000)

    librt.so.1 => /lib64/librt.so.1 (0x00007fec5073e000)

    libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007fec50437000)

    libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007fec50221000)
    libc.so.6 => /lib64/libc.so.6 (0x00007fec4fe54000)

    /lib64/ld-linux-x86-64.so.2 (0x00007fec615b2000)
```

6.1.6 Run ldd again after adding the correct version of GCC

```
(test) [user@linux OOCpackage_testing]$ ldd /path/to/virtualenv/test/lib/python2.7/site-packages/
tensorflow/python/_pywrap_tensorflow_internal.so

    linux-vdso.so.1 =>  (0x00007ffe334f6000)

    libtensorflow_framework.so => /path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/
    python/../libtensorflow_framework.so (0x00007efff9f26000)

    libiomp5.so => /path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/python/../../_
    solib_k8/_U@mkl_Ulinux_S_S_Cmkl_Ulibs_Ulinux___Uexternal_Smkl_Ulinux_Slib/libiomp5.so
    (0x00007efff9b82000)

    libmklml_intel.so => /path/to/virtualenv/test/lib/python2.7/site-packages/tensorflow/
    python/../../_solib_k8/_U@mkl_Ulinux_S_S_Cmkl_Ulibs_Ulinux___Uexternal_Smkl_Ulinux_Slib/libmklml_
    intel.so (0x00007efff0ab1000)

    libdl.so.2 => /lib64/libdl.so.2 (0x00007efff08ad000)

    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007efff0691000)

    libm.so.6 => /lib64/libm.so.6 (0x00007efff038e000)

    librt.so.1 => /lib64/librt.so.1 (0x00007efff0186000)

    libstdc++.so.6 => /path/to/gcc-7.2.0/lib64/libstdc++.so.6 (0x00007effefe05000)

    libgcc_s.so.1 => /path/to/gcc-7.2.0/lib64/libgcc_s.so.1 (0x00007effefbee000)

    libc.so.6 => /lib64/libc.so.6 (0x00007effef821000)

    /lib64/ld-linux-x86-64.so.2 (0x00007f0000ffb000)
```

6.2 TensorFlow Build Issues

At the time of writing this document, TensorFlow fails to build with the latest Bazel version 0.12.0. The error is shown below:

```
[user@linux tensorflow]$  bazel build --config mkl --copt="-mavx2" --copt="-O3" --copt="-mfma"
--copt="-march=broadwell"  -s -c opt //tensorflow/tools/pip_package:build_pip_package

WARNING: /home/user/.cache/bazel/_bazel_user/8aaf3223e362cc8d0a1f01ee89fea818/external/protobuf_
archive/WORKSPACE:1: Workspace name in /home/user/.cache/bazel/_bazel_user/8aaf3223e362cc8d0a1f01ee8
9fea818/external/protobuf_archive/WORKSPACE (@com_google_protobuf) does not match the name given in
the repository's definition (@protobuf_archive); this will cause a build error in future versions

WARNING: /home/user/.cache/bazel/_bazel_user/8aaf3223e362cc8d0a1f01ee89fea818/external/grpc/
WORKSPACE:1: Workspace name in /home/user/.cache/bazel/_bazel_user/8aaf3223e362cc8d0a1f01ee89fea818/
external/grpc/WORKSPACE (@com_github_grpc_grpc) does not match the name given in the repository's
definition (@grpc); this will cause a build error in future versions

WARNING: /home/user/tf_test_new/tensorflow/third_party/mkl/BUILD:20:12: in srcs attribute of cc_
library rule //third_party/mkl:intel_binary_blob: please do not import '@mkl//:lib/libmklml_intel.so'
directly. You should either move the file to this package or depend on an appropriate rule there

WARNING: /home/user/tf_test_new/tensorflow/third_party/mkl/BUILD:20:12: in srcs attribute of cc_
library rule //third_party/mkl:intel_binary_blob: please do not import '@mkl//:lib/libiomp5.so'
directly. You should either move the file to this package or depend on an appropriate rule there

ERROR: /home/user/.cache/bazel/_bazel_user/8aaf3223e362cc8d0a1f01ee89fea818/external/jpeg/
BUILD:126:12: Illegal ambiguous match on configurable attribute "deps" in @jpeg//:jpeg:

@jpeg//:k8

@jpeg//:armeabi-v7a

Multiple matches are not allowed unless one is unambiguously more specialized.

ERROR: Analysis of target '//tensorflow/tools/pip_package:build_pip_package' failed; build aborted:
```

```
/home/user/.cache/bazel/_bazel_user/8aaf3223e362cc8d0a1f01ee89fea818/external/jpeg/BUILD:126:12:
Illegal ambiguous match on configurable attribute "deps" in @jpeg//:jpeg:
@jpeg//:k8
@jpeg//:armeabi-v7a
Multiple matches are not allowed unless one is unambiguously more specialized.
INFO: Elapsed time: 7.813s
FAILED: Build did NOT complete successfully (135 packages loaded)
  currently loading: tensorflow/core/kernels
```

The workaround is to downgrade to the previous Bazel version 0.11.1

6.3 Horovod Install Issues

On trying to import Horovod, if there are errors such as this:

```
>>> import horovod.tensorflow
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "/home/user/tf_test_new/virtenv/lib/python2.7/site-packages/horovod/tensorflow/__init__.py",
 line 41, in <module>
  from horovod.tensorflow.mpi_ops import allgather
 File "/home/user/tf_test_new/virtenv/lib/python2.7/site-packages/horovod/tensorflow/mpi_ops.py", line
 56, in <module>
  ['HorovodAllgather', 'HorovodAllreduce'])
 File "/home/user/tf_test_new/virtenv/lib/python2.7/site-packages/horovod/tensorflow/mpi_ops.py", line
 43, in _load_library
  library = load_library.load_op_library(filename)
 File "/home/user/tf_test_new/virtenv/lib/python2.7/site-packages/tensorflow/python/framework/load_
 library.py", line 58, in load_op_library
  lib_handle = py_tf.TF_LoadLibrary(library_filename, status)
 File "/home/user/tf_test_new/virtenv/lib/python2.7/site-packages/tensorflow/python/framework/errors_
 impl.py", line 516, in __exit__
  c_api.TF_GetCode(self.status.status))
tensorflow.python.framework.errors_impl.NotFoundError: /home/user/tf_test_new/virtenv/lib/python2.7/
site-packages/horovod/tensorflow/mpi_lib.so: undefined symbol: _ZNK10tensorflow8OpKernel4nameEv
```

Ensure that the same GCC version that was using to build TensorFlow is in PATH when installing Horovod. If different versions were used, especially GCC versions older than 5.1, then object files get compiled with for different versions of the ABI (_GLIBCXX_USE_CXX11_ABI), resulting in this error at link time. To fix this, uninstall Horovod, fix PATH and LD_LIBRARY_PATH to point to the correct GCC, and then reinstall Horovod.

6.4 Verify Intel® Omni-Path Architecture (Intel® OPA)

The official documentation contains in-depth information on setup and configuration of Intel® OPA: https://www.intel.com/content/www/us/en/support/articles/000016242/network-and-i-o/fabric-products.html

6.4.1 Verify that Intel® OPA is Up and Running

```
opaconfig -V
10.3.1.0.22
```

6.4.2 Verify Install (Example of a good install)

```
ibstat
CA 'hfi1_0'
```

```
    CA type:
    Number of ports: 1
    Firmware version:
    Hardware version: 11
    Node GUID: 0x00117501017a04c9
    System image GUID: 0x00117501017a04c9
    Port 1:
        State: Active
        Physical state: LinkUp
        Rate: 100
        Base lid: 19
        LMC: 0
        SM lid: 1
        Capability mask: 0x00410020
        Port GUID: 0x00117501017a04c9
        Link layer: InfiniBand
opainfo
hfi1_0:1                          PortGID:0xfe80000000000000:001175010174447d
   PortState:     Active
   LinkSpeed      Act: 25Gb         En: 25Gb
   LinkWidth      Act: 4            En: 4
   LinkWidthDnGrd ActTx: 4  Rx: 4   En: 3,4
   LCRC           Act: 14-bit       En: 14-bit,16-bit,48-bit   Mgmt: True
   LID: 0x00000031-0x00000031       SM LID: 0x00000001 SL: 0
   QSFP: PassiveCu, 3m   Hitachi Metals    P/N IQSFP26C-30       Rev 03
   Xmit Data:           10922365 MB Pkts:           1929517624
   Recv Data:           10893103 MB Pkts:           1809969601
   Link Quality: 5 (Excellent)
```

6.4.3 Verify Intel® Omni-Path Fabric Performance

We recommend sanity checking the Intel OPA network connectivity by using a test benchmark. If the Intel® Omni-Path Fabric (Intel® OP Fabric) Suite FastFabric Toolset is installed, a number of such test suites are automatically included in the installation. See section 8 of the Intel OP Fabric FastFabric Toolset documentation here:

https://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/Intel_OP_FabricSuite_FastFabric_UG_H76469_v8_0.pdf

If not, the link can be checked manually by running benchmarks such as the OSU benchmark, which are easily obtainable on the official page here:

http://mvapich.cse.ohio-state.edu/benchmarks/

6.4.4 Intel OPA Install Issues

Some things to check if the hfi1 module does not load automatically on reboot: You may have to load the driver manually:

```
sudo service rdma start
sudo modprobe hfi1
service irqbalance restart
lsmod | grep hfi1
```

```
hfi1                     697628  1
rdmavt                     63294  1 hfi1
ib_core                   210381  13 hfi1,rdma_cm,ib_cm,iw_cm,rpcrdma,ib_ucm,rdmavt,ib_iser,ib_umad,ib_
uverbs,rdma_ucm,ib_ipoib,ib_isert
i2c_algo_bit               13413  3 igb,hfi1,mgag200
i2c_core                   40756  7 drm,igb,hfi1,i2c_i801,drm_kms_helper,mgag200,i2c_algo_bit
```

Check if device is present and recognized in the PCIe* tree:

```
[user@linux nompi]$ ls –l /sys/class/infiniband
lrwxrwxrwx 1 root root 0 Oct 10 12:14 hfi1_0 -> ../../devices/pci0000:00/0000:00:02.0/0000:02:00.0/
infiniband/hfi1_0
[user@linux nompi]$ ls -l /dev/hfi1*
crw-rw-rw- 1 root root 245,   0 Apr 12 14:33 /dev/hfi1_0
crw------- 1 root root 245, 128 Apr 12 14:33 /dev/hfi1_diagpkt
crw------- 1 root root 245, 200 Apr 12 14:33 /dev/hfi1_diagpkt0
[bduser@node27 ~]$ lspci | grep -i hfi
03:00.0 Fabric controller: Intel Corporation Omni-Path HFI Silicon 100 Series [discrete] (rev 11)
```

Look at dmesg. Following shows driver loaded correctly:

```
[   22.987380] hfi1 0000:03:00.0: hfi1_0: set_link_state: current INIT, new ARMED
[   22.987398] hfi1 0000:03:00.0: hfi1_0: logical state changed to PORT_ARMED (0x3)
[   22.987409] hfi1 0000:03:00.0: hfi1_0: send_idle_message: sending idle message 0x103
[   22.988637] hfi1 0000:03:00.0: hfi1_0: set_link_state: current ARMED, new ACTIVE
[   22.988651] hfi1 0000:03:00.0: hfi1_0: logical state changed to PORT_ACTIVE (0x4)
[   22.988715] hfi1 0000:03:00.0: hfi1_0: send_idle_message: sending idle message 0x203
```

Additional driver parameters are listed in the **performance tuning guide:**

https://www.intel.com/content/www/us/en/support/articles/000016242/network-and-i-o/fabric-products.html

Current version (subject to update):

https://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/Intel_OP_Performance_Tuning_UG_H93143_v11_0.pdf

To test basic functionality, run a Verbs or IPoFabric benchmark. See section 7 of performance tuning guide for complete details. Below is a snippet showing how to run ib_write_bw from the perftest benchmark to test for Verbs performance.

```
sudo modprobe hfi1
sudo modprobe ib_ipoib
sudo ifup ib0
ib_write_bw -F -R -s 1048576                          // on server node
ib_write_bw -F -R -s 1048576 <server's IPoIB address> // on client node
```

To unload the driver manually:

```
sudo rmmod hfi1
sudo rmmod rdmavt
```