Boosting Deep Learning Training & Inference Performance on Intel® Xeon® and Intel® Xeon Phi™ Processors

Vikram Saletore, Ph.D., Deepthi Karkada, Vamsi Sripathi, Kushal Datta, Ph.D., & Ananth Sankaranarayanan

Contents
1. Overview ..................................................................................................................................................... 2
2. Improving Deep Learning Performance ...................................................................................................... 3
3. Performance Metrics for Image Recognition ............................................................................................... 4
   A. Training Performance Metric .................................................................................................................. 4
   B. Baseline Performance for Single and Multi-Node Training .............................................................. 4
   C. Baseline Inference Performance: ........................................................................................................ 4
   A. Optimized Performance with Multiple Workers on Single- and Multi-Node Training ................. 5
   A. Optimized Inference Performance: ....................................................................................................... 6
6. TensorFlow Training Performance: ........................................................................................................... 7
7. Caffe Training Performance: ...................................................................................................................... 8
8. TensorFlow Inference Performance: ........................................................................................................... 9
9. Caffe* Inference Performance: .................................................................................................................. 10
    A. TensorFlow Training Performance on Intel Xeon Phi Processor Based Platforms: ................... 12
11. Platform Configurations ............................................................................................................................. 13
12. Deep Learning Framework Configurations ............................................................................................. 13
In this work we present how, without a single line of code change in the framework, we can further boost the performance for deep learning training by up to 2X and inference by up to 2.7X on top of the current software optimizations available from open source TensorFlow* and Caffe* on Intel® Xeon® and Intel® Xeon Phi™ processors. Our system level optimizations result in a higher throughput and a reduction in time-to-train for a given batch size per worker compared to the current baseline for image recognition Convolution Neural Networks (CNN) workloads.

1. **OVERVIEW**

Intel® Xeon® and Intel® Xeon Phi™ processors are extensively used in deep learning and high performance computing applications. Popular deep learning frameworks such as TensorFlow*, Caffe* and MxNet* have been optimized by Intel software teams to deliver optimal performance on Intel platforms for both deep learning training and inference workflows. With Intel and Google's continuing collaboration, the performance of TensorFlow has significantly improved with Intel® Math Kernel Library (Intel® MKL) and Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN). Similarly, the Intel® Distribution of Caffe* also delivers significant performance gains on Intel Xeon and Intel Xeon Phi processors.
Training deep Convolution Neural Networks (CNN) such as ResNet-50, GoogLeNet-v1, Inception-3, and others involves executing hundreds of compute-intensive functions such as two-dimensional convolutions, matrix multiplication, RELU activation, max-pool and softmax to name a few, for hundreds of thousands of iterations. These function kernels are mapped to libraries such as Intel MKL or Intel MKL-DNN which are highly optimized implementations of these kernels on Intel platforms. In our performance characterization of CNN applications, we have observed that even though Intel optimized deep learning frameworks are multi-threaded, the CPU cores are under-utilized during the execution of CNNs.

Although user controllable configuration parameters are provided in the frameworks, those are not sufficient to achieve optimal performance. TensorFlow, for example, utilizes *intra-op* and *inter-op* parallelism. *Intra-op* controls the size of the thread pool to parallelize kernels in a given operation and *inter-op* controls the size of thread pool to run operations in parallel. However, these user-level knobs do not provide users with sufficient micro-architectural information on the underlying NUMA configuration in multi-socket Intel Xeon processor-based platforms.

In addition, without the knowledge of CPU socket and NUMA configuration, simple thread affinitization (as in the case of thread pool) does not lead to optimal performance. In fact, it can sometimes prohibitively decrease throughput, as a core from socket 0 might have to continually access cache lines from the memory bank of socket 1 creating increased bandwidth pressure on the Intel® Ultra-Path Interconnect (Intel® UPI). This situation exacerbates with larger number of sockets found in 4, 8, and 16 socket systems. We believe that users need to be aware of system level optimizations in addition to framework specific configuration parameters to achieve the best performance for CNN workloads on CPU platforms.

2. IMPROVING DEEP LEARNING PERFORMANCE

In this section we present the methodology (or Best Known Methods – BKMs) on how to optimally run deep learning workloads on multi-socket Intel Xeon platforms. The BKMs achieve the following:

- Single-node multi-socket with Parameter Server (PS) (if required) deep learning training
- Multi-node multi-socket with PS (if required) distributed deep learning training
- Single-node multi-stream deep learning inference

In a later section, we will show that these BKMs are also applicable for Intel Xeon Phi processor-based platforms.
3. PERFORMANCE METRICS FOR IMAGE RECOGNITION

A. Training Performance Metric

The performance metric used to reach convergence at a given batch size per worker with a specific number of iterations for developing a trained model for a neural network on an image dataset is the Time-To-Train (TTT). With a given batch-size \( BSize/worker \), image throughput in images/sec, and assuming tuned hyper-parameters and convergence with a given number of Epochs:

For 1 worker, the TTT is given by:

\[
TTT_1 = \frac{Images_{Total}}{ImageThroughput_1} \times Epochs
\]

For \( W \) workers, the TTT is given by:

\[
TTT_W = \frac{Images_{Total}}{ImageThroughput_W} \times Epochs
\]

B. Baseline Performance for Single and Multi-Node Training

The current methodology is to train with a single worker per node with a batch size \( BSize \).

Single-node baseline performance is measured by TTT with 1 Worker/Node. Multi-Node baseline performance on \( N \) nodes is measured by TTT with \( N \) Workers, 1 worker on each node.

C. Baseline Inference Performance:

The current methodology is to run inference with a single stream of input with a single worker per node. Baseline Inference performance is measured by throughput in Images/sec achieved by a single node at a given batch size \( BSize \).

4. DEEP LEARNING TRAINING: PARTITIONING MULTI-SOCKET INTEL® XEON® PROCESSOR BASED SYSTEMS

To improve core utilization and ultimately performance for CNN workloads on multi-socket Intel Xeon platforms, we partition the sockets and the cores on the platform as separate computing devices and run multiple deep learning training instances. The term ‘instances’ refers to deep learning framework worker processes that are working in tandem, each on a local batch size of input data in a synchronous manner on a multi-socket or even a single-socket system. Each worker is process bound to a subset of the total number of cores and threads in the system using core and thread affinity settings.
We use libnumactl to control memory allocations to target NUMA domains and the KMP_AFFINITY environment variable provided by the OpenMP* runtime library to affinitize OpenMP threads to target CPU cores.

If a parameter server (PS) is required, it may be used to aggregate gradients, whether locally spawned as a separate thread in the host server or remotely spawned over the network on another server works without any change.

A. Optimized Performance with Multiple Workers on Single- and Multi-Node Training
In this scenario, the single-node optimized performance is measured by TTT with $K$ Workers/Node each with a batch size $BSize$ per worker. The batch size per node would then be equal to $K*BSize$. Multi-node optimized performance on $N$ nodes is measured by TTT with $K*N$ Workers, $K$ worker on each node. It is assumed that hyper-parameters for the neural network model are tuned for multiple workers for single and multiple nodes.
Figure 2. Sub-socket Partitioning across Dual-Socket Intel® Xeon® Platforms for Multiple Inference Streams

Similar methodology can be applied for deep learning inference. 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 2 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.

A. Optimized Inference Performance:
In this scenario, we have $K$ workers per node. The optimized 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 $K*BSIZE$. 
6. **TENSORFLOW TRAINING PERFORMANCE:**

Figure 3 shows deep learning training performance (Images/Sec) relative to the current optimization using TensorFlow 1.4.0 release version across 6 deep learning benchmark topologies. The 3 bars in the chart show the performance improvement on 1, 2, & 4 nodes of dual-socket Intel Xeon Platinum 8168 processor cluster over 10Gbit Ethernet fabric. The figure shows that we can improve the performance up to 2.1X even on a single node with 4 workers/node using core/thread affinity and memory locality optimizations.

**Intel® Xeon® Platinum 8168 Processor: TensorFlow® Multi-Node & Multiple Workers/Socket**
Training: TensorFlow 1.4, BS=64, Image Dataset, grpc/10GB Ethernet, Parameter Server on each node

![Figure 3. TensorFlow 1.4 Training Performance (Projected Time-To-Train (TTT)) Improvement with optimized affinity for cores and memory locality using 4 Workers/Node compared to current baseline with 1 Worker/Node. Platform Configuration: 2S Intel Xeon Platinum 8168 Processor @ 2.70GHz (24 cores), HT 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. Multiple nodes connected with 10Gbit Ethernet. Tensorflow 1.4.0, GCC 6.2.0, Intel MKL-DNN. Training measured with image data on SSD. Source: Intel internal testing, December 2017.**
7. CAFFE TRAINING PERFORMANCE:

Figure 4 shows that using our optimized BKMs for Intel® Distribution of Caffe, we are able to boost the performance of GoogLeNet-v1 by up to 1.2X on top of current optimizations for 1, 2, and 4 node clusters of dual-socket Intel Xeon Platinum 8170 processor-based systems. As the current Caffe available from GitHub is highly optimized for Intel CPUs and able to use cores more efficiently, the improvement is smaller compared to TensorFlow.

![Figure 4. Intel® Distribution of Caffe® Training Performance (Projected TTT) Improvement with optimized affinity for cores and memory locality using 2 Caffe Instances/Node compared to current optimized baseline with 1 Instance/Node. Source: Intel internal testing, December 2017.](image-url)
8. TENSORFLOW INFERENCE PERFORMANCE:

Figure 5. TensorFlow Inference Performance (Images/Sec) Improvement with optimized affinity for cores and memory locality using concurrent multiple 2, 4, & 8 Streams compared to current baseline with equivalent batch-size using 1 Stream. Platform Configuration: 2S Intel Xeon Platinum 8168 processor @ 2.70GHz (24 cores), HT 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. Tensorflow 1.4.0, GCC 6.2.0, Intel MKL-DNN. Inference measured with –forward_only. Image data on SSD. Source: Intel internal testing, December 2017.

Figure 5 shows deep learning inference performance (Images/Sec) relative to the current optimization using TensorFlow 1.4. The 3 bars in the chart show the performance improvements for global batch sizes of 512 (2 streams, each of batch-size of 256), 1024, and 2048 on a single-node, dual-socket Intel Xeon Platinum 8168 processor-based platform. For the optimized test, we have 2, 4, & 8 workers affinitized to cores and mapped to appropriate memory locality. Multiple streams of input data, each stream per worker is concurrently processed by the workers. E.g., for a global batch size of 2048, we use 8 streams each processing a batch size of 256. Performance data measured shows that we are able to boost inference performance up to 2.7X with our system level optimizations.
9. **CAFFE* INFEERENCE PERFORMANCE:**

Figure 6 shows deep learning Inference performance (Images/Sec) relative to the current optimization using Intel Distribution of Caffe. The 4 bars in the chart show the performance improvements for global batch sizes of 256, 512, 1024, and 2048 on a single-node, dual-socket Intel Xeon Platinum 8170 processor-based platform. We observe that although Caffe is well optimized, we are still able to improve the inference performance up to 1.8X for large batch sizes.

![Figure 6](image.png)

*Figure 6. Intel® Distribution of Caffe* Inference Performance (Images/Sec) Improvement with optimized affinity for cores and memory locality using concurrent multiple 2, 4, & 8 Streams/Node compared to current baseline with equivalent batch-size using 1 Stream/Node. Source: Intel internal testing, December 2017.*
10. DEEP LEARNING TRAINING: PARTITIONING SINGLE-SOCKET INTEL® XEON PHI™ PROCESSOR BASED SYSTEMS

We used optimization learnings from the Intel Xeon processor and applied them to single-socket Intel Xeon Phi processor-based platforms. The Intel Xeon Phi processor 7250 has 68 cores with 4 threads/core resulting in 272 threads. Figure 7 shows a symbolic view of how one could partition the socket for 4 instances of a framework, each instance affinitized to specific cores. The 4 instances run on 64 cores (16 cores/instance) in a distributed training manner with 4 cores allocated to I/O, Parameter Server (if required).

Figure 7. Symbolic Sub-socket Partitioning for Single Socket Intel® Xeon Phi™ Processor 7250
A. TensorFlow Training Performance on Intel Xeon Phi Processor Based Platforms:
To support multiple workers on a single Intel Xeon Phi processor-based system, we configure
the processor MCDRAM in Cache-Mode at system boot time. Figure 8 shows that we are
successfully able to apply the optimizations on the single-socket Intel Xeon Phi 7250
processor, boosting its performance up to 1.4X with 4 workers/node using TensorFlow 1.3 for
ResNet-50 neural network benchmark. The optimizations also hold for multiple worker and
multi-node (1, 2, and 4) distributed training using Intel® Omni-Path Architecture (Intel® OPA).

Figure 8. TensorFlow Training Performance (Projected TTT) Improvement with optimized affinity for cores and
memory locality using 4 Workers/Node compared to current optimized baseline with 1 Worker/Node. Source: Intel
internal testing, December 2017.
11. PLATFORM CONFIGURATIONS

Intel Xeon Platinum 8168 Processor
Dual-socket Intel Xeon Platinum 8168 processor @ 2.70GHz (24 cores), HT 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. Multiple nodes connected with 10Gbit Ethernet.

Intel Xeon Gold 6148 Processor
Dual-socket Intel Xeon Gold 6148 processor @ 2.40GHz (20 cores), HT 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. Multiple nodes connected with Intel Omni-Path Architecture Host Fabric, Intel OPA Interface Driver version 10.4.2.0.7. SSD: Intel SSD DC S3700 Series.

Intel Xeon Platinum 8170 Processor
Dual-socket Intel Xeon Platinum 8170 processor @ 2.10GHz (26 cores), HT enabled, turbo disabled, scaling governor set to “performance” via intel_pstate driver, 384GB DDR4-2666 ECC RAM. CentOS Linux release 7.3.1611 (Core), Linux kernel 3.10.0-514.10.2.el7.x86_64. Multiple nodes connected with Intel OPA Host Fabric, Intel OPA Interface Driver version 10.4.2.0.7. SSD: Intel SSD 800GB DC S3700 Series.

Intel Xeon Phi Processor 7250
Single-socket Intel Xeon Phi processor 7250, 68 Cores, 4 HW Threads per core, 1.4 GHz, 16GB high-speed MCDRAM set in Cache-Quadrant mode, 32KB L1 data cache per core, 1MB L2 per two-core tile, 96GB DDR4. Multiple nodes connected with Intel OPA Host Fabric, Intel OPA Interface Driver version 10.4.2.0.7, Intel SSD 480GB DC S3500 Series, Software: CentOS Linux release 7.3.1611, Linux kernel 3.10.0-514.10.2.el7.x86_64, Intel® MPI Library 2017 Update 4.

12. DEEP LEARNING FRAMEWORK CONFIGURATIONS

TensorFlow:
TensorFlow 1.4: (https://github.com/tensorflow/tensorflow), Tensorflow 1.4.0, GCC 6.2.0, Intel MKL-DNN. TensorFlow training measured with image data stored on the SSD storage, Inference measured with -forward_only option.

TensorFlow1.3: (https://github.com/tensorflow/tensorflow), Tensorflow 1.3.0, GCC 6.2.0, Intel MKL 2017. TensorFlow training measured with image data stored on the SSD storage, Inference measured with --forward_only option.

Intel Distribution of Caffe:
13. **BKMS FOR INTEL® XEON® PROCESSOR PERFORMANCE OPTIMIZATIONS + OPTIMIZED DEEP LEARNING FRAMEWORKS**

In this section we outline our Best Known Methods (BKMs) using TensorFlow and Caffe as examples. We have used Intel Xeon and Intel Xeon Phi processor-based platforms in our examples.

14. **BEST KNOWN METHODS FOR TENSORFLOW:**

   A. **Build Methodology for TensorFlow:**

   For Intel® optimized TensorFlow build please follow the BKMs specified for direct optimizations team or refer to this article: [https://software.intel.com/en-us/articles/intel-optimized-tensorflow-wheel-now-available](https://software.intel.com/en-us/articles/intel-optimized-tensorflow-wheel-now-available)

   B. **Optimized Run Time BKM for TensorFlow**

   We use the tf_cnn_benchmarks at TensorFlow GitHub to test and measure performance improvement using our runtime optimizations:

   TensorFlow tf_cnn_benchmarks:
   
   • tf_cnn_benchmarks code available here: [https://github.com/tensorflow/benchmarks](https://github.com/tensorflow/benchmarks)
   
   • Uses the latest APIs for the input pipeline, gradient updates hence designed to be fast

   • Can be easily integrated with custom CNN topologies

   C. **BKM for Single-Node Multi-Socket Distributed Training**

   **Example 1:** For 2S Intel Xeon Gold 6148 processor-based systems, multi-socket (sub-socket) with 20 Cores/Socket single-mode distributed training with 4 TensorFlow worker instances per node and 1 Parameter Server (PS) can be specified and launched as follows:

   ```
   PS_HOST: "hostname1"
   ps_list: "hostname1:2218"
   WK_HOST= hostname2"
   workers_list : "hostname2:2223,hostname2:2224,hostname2:2225,hostname2:2226"
   worker_env:"export OMP_NUM_THREADS=9; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"
   common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir '/path-to/TF_Records' --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ ps_list --worker_hosts $workers_list --display_every 10 
   ps_args: "$common_args --num_intra_threads 4 --num_inter_threads 2"
   worker_args: "$common_args --num_intra_threads 9 --num_inter_threads 4"
   
   To start the Parameter Server:
   
   ssh $PS_HOST; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &
   
   To start the Workers:
   
   ssh $PS_HOST; numactl -l python tf_cnn_benchmarks.py $worker_args --job_name worker --task_index 0 --worker_hosts $workers_list &
   ```
ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[0-9,40-49],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[10-19,50-59],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[20-29,60-69],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[30-39,70-79],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &

Where $ps_list and $workers_list are the comma separated list of hostname:port pairs of the parameter servers and worker hosts respectively. $ps_args are the arguments to the parameter server such as --num_inter_threads and --num_intra_threads. $worker_args are the arguments to the worker such as the model name, batch_size, data_format, data_dir, server_protocol, num_inter_threads and num_intra_threads values etc.

D. BKM for Multi-Node, Multi-Socket Training

Example 2: For 2S Intel Xeon Gold 6148 processor-based systems, multi-socket (sub-socket) with 20 Cores/Socket 2-node distributed training with 4 TensorFlow worker instances per node and 1 Parameter Server (PS) can be specified and launched as follows

```
PS_HOST_0: "hostname1"
ps_list: "hostname1:2218"
WK_HOST_0=hostname2, WK_HOST_1=hostname3
workers_list: 
worker_env:"export OMP_NUM_THREADS=9; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"
common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir /path-to/TF_Records' --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ ps_list --worker_hosts $workers_list --display_every 10 "
ps_args: "$common_args --num_intra_threads 4 --num_inter_threads 2"
worker_args: "$common_args --num_intra_threads 9 --num_inter_threads 4"
```

To start the parameter server:

```
ssh $PS_HOST_0; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &
```

To start the workers on node 0:

```
```
ssh $WK_HOST_0; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[0-9,40-49],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --
  worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[10-19,50-59],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --
  worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[20-29,60-69],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --
  worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[30-39,70-79],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --
  worker_hosts $workers_list &

To start the workers on node 1:

ssh $WK_HOST_1; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[0-9,40-49],explicit,verbose" --job_name worker --task_index 4 --ps_hosts $ps_list --
  worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[10-19,50-59],explicit,verbose" --job_name worker --task_index 5 --ps_hosts $ps_list --
  worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[20-29,60-69],explicit,verbose" --job_name worker --task_index 6 --ps_hosts $ps_list --
  worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --
  kmp_affinity="granularity=thread,proclist=[30-39,70-79],explicit,verbose" --job_name worker --task_index 7 --ps_hosts $ps_list --
  worker_hosts $workers_list &

Where $ps_list and $workers_list are the comma separated list of hostname:port pairs of the
parameter servers and worker hosts respectively. $ps_args are the arguments to the
parameter server such as --num_inter_threads and --num_intra_threads. $worker_args are the
arguments to the worker such as the model name, batch_size, data_format, data_dir,
server_protocol, num_inter_threads and num_intra_threads values etc.
E. Multi-Socket Deep Learning Inference on Intel® Xeon® Processor-Based Systems

Example 3: For 2S Intel Xeon Platinum 8170 processor-based systems, multi-socket (sub-socket) with 26 Cores/Socket with 8 TensorFlow instances per node running inference can be launched as follows:

```plaintext
common_args: "--model resnet50 --batch_size 256 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --num_warmup_batches 10 --device cpu --data_dir ~/tensorflow/TF_Records --data_name imagenet --display_every 10 "
WK_HOST= hostname"

worker_env:"export OMP_NUM_THREADS=6; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"
inf_args: "$common_args --num_intra_threads 6 --num_inter_threads 4"

To start 4 Inference streams on Socket-0:

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[0-5,52-57],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[6-12,58-64],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[13-18,65-70],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[19-25,71-77],explicit,verbose" &

To start 4 Inference streams on Socket-1:

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[26-31,78-83],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[32-38,84-90],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[39-44,91-96],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[45-51,96-102],explicit,verbose" &

Where $inf_args are the arguments to the TF instance running inference such as the model name, batch_size, data_format, data_dir, num_inter_threads and num_intra_threads values etc.
F. Optimized Run Time BKM for TensorFlow for Training on Intel Xeon Phi processor 7250

Example 4: For 1S Intel Xeon Phi processor 7250 based systems, multi-socket (sub-socket) with 68 Cores/Socket 1-node distributed training with 4 TensorFlow worker instances per node and 1 Parameter Server (PS) can be specified and launched as follows. We use 64 Cores for compute and the remaining 4 cores for I/O. We assume that the MCDRAM in the Intel Xeon Phi processor-based system is booted in Cache-Mode.

```
PS_HOST: "hostname1"
p_list: "hostname1:2218"
WK_HOST= hostname2"
workers_list : "hostname2:2223,hostname2:2224,hostname2:2225,hostname2:2226"
worker_env:"export OMP_NUM_THREADS=15; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"
common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir /path-to/TF_Records/ --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ps_list --worker_hosts $workers_list --display_every 10 "

ps_args: "$common_args --num_intra_threads 4 --num_inter_threads 2"
worker_args: "$common_args --num_intra_threads 15 --num_inter_threads 4"

To start the Parameter Server:

ssh $PS_HOST; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the Workers:

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[0-15,68-115],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[16-31,116-163],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[32-47,164-211],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[48-63,212-259],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &

Where $ps_list and $workers_list are the comma separated list of hostname:port pairs of the parameter servers and worker hosts respectively. $ps_args are the arguments to the parameter server such as --num_inter_threads and --num_intra_threads. $worker_args are the arguments to the worker such as the model name, batch_size, data_format, data_dir, server_protocol, num_inter_threads and num_intra_threads values etc.
Example 5: For 2S Intel Xeon Phi processor 7250 based systems, multi-socket (sub-socket) with 68 Cores/Socket 2-node distributed training with 4 TensorFlow worker instances per node and 1 Parameter Server (PS) can be specified and launched as follows:

PS_HOST_0: "hostname1"
ps_list: "hostname1:2218"
WK_HOST_0=hostname2, WK_HOST_1=hostname3

workers_list:
worker_env:"export OMP_NUM_THREADS=15; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"

common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir '/path-to/TF_Records' --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ps_list --worker_hosts $workers_list --display_every 10 "

ps_args:"$common_args --num_intra_threads 4 --num_inter_threads 2"
worker_args:"$common_args --num_intra_threads 15 --num_inter_threads 4"

To start the Parameter Server:

ssh $PS_HOST; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 0:

ssh $WK_HOST_0; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[0-15,68-115],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 1:

ssh $WK_HOST_1; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[16-31,116-163],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 2:

ssh $WK_HOST_0; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[16-31,116-163],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 3:

ssh $WK_HOST_1; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[32-47,164-211],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 4:

ssh $WK_HOST_0; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[32-47,164-211],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 5:

ssh $WK_HOST_1; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[48-63,212-259],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 6:

ssh $WK_HOST_0; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[48-63,212-259],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &

To start the workers on node 7:

ssh $WK_HOST_1; $worker_env;n unh numactl -m 0 python tf_cnn_benchmarks.py $worker_args -- kmp_affinity="granularity=thread,proclist=[64-79,260-307],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &
ssh $WK_HOST_1; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --
kmp_affinity="granularity=thread,proclist=[48-63,212-259],explicit,verbose" --job_name worker --task_index 7 --ps_hosts $ps_list --worker_hosts $workers_list &

Where $ps_list and $workers_list are the comma separated list of hostname:port pairs of the parameter servers and worker hosts respectively. $ps_args are the arguments to the parameter server such as --num_inter_threads and --num_intra_threads. $worker_args are the arguments to the worker such as the model name, batch_size, data_format, data_dir, server_protocol, num_inter_threads and num_intra_threads values etc.
**BEST KNOWN METHODS FOR OPTIMIZED INTEL DISTRIBUTION OF CAFFE**

**Build Methodology for Caffe:** For Intel Distribution of Caffe please follow the BKMs specified by Intel optimized Caffe: [https://github.com/intel/caffe](https://github.com/intel/caffe)

A. **BKM for Single & Multiple Node Multi-Socket Distributed Training Examples:**

Example 6: For 2S Intel Xeon Platinum 8170 processor-based systems, multi-socket (sub-socket) with 26 Cores/Socket Dist. Training with 4 Caffe Worker instances per node can be specified and launched as follows:

WK_HOST="hostname"
CORES_PER_NODE=52
P=2 #Processes per node
N=2 #Total number of processes calculated as num_nodes/P
CORES_PER_MPI_PROCESS=$((CORES_PER_NODE / P))
OMPTHREADS=$((CORES_PER_MPI_PROCESS - 2))

export I_MPI_DEBUG=5; mpiexec.hydra -v -l -ppn $P –n $N -f $WK_HOST -genv OMP_NUM_THREADS $OMPTHREADS -genv KMP_AFFINITY 'granularity=fine,compact,1,0' path-to-intelcaffe/build/tools/caffe train -solver $MODELDIR/solver.prototxt -engine MKL2017

Where OMP_NUM_THREADS is the Number of OpenMP threads used per process, CAFFEDIR is the path to the Caffe installation, MODELDIR is the path to the directory containing the model prototxt files (for ex. googlenet)

B. **Multi-Socket Inference Example**

Example 7: 2S Intel Xeon Platinum 8170 processor-based systems, multi-socket (sub-socket) multi stream inference with 26 Cores/Socket with 8 Caffe instances per node running inference can be launched as follows:

OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[0-5,52-57],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[6-12,58-64],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[13-18,65-70],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[19-25,71-77],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[26-31,78-83],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[32-38,84-90],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[39-44,91-96],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[45-51,96-102],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
PLATFORM CONFIGURATIONS

Intel Xeon Platinum 8168 Processor
2S Intel Xeon Platinum 8168 CPU @ 2.70GHz (24 cores), HT 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. Multiple nodes connected with 10Gbit Ethernet.

Intel Xeon Gold 6148 Processor
2S Intel Xeon Gold 6148 CPU @ 2.40GHz (20 cores), HT 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. Multiple nodes connected with Intel Omni-Path Architecture Host Fabric, Intel OPA Interface Driver version 10.4.2.0.7. SSD: Intel SSD DC S3700 Series.

Intel Xeon Platinum 8170 Processor
2S Intel Xeon Platinum 8170 CPU @ 2.10GHz (26 cores), HT enabled, turbo disabled, scaling governor set to “performance” via intel_pstate driver, 384GB DDR4-2666 ECC RAM. CentOS Linux release 7.3.1611 (Core), Linux kernel 3.10.0-514.16.1.el7.x86_64. Multiple nodes connected with Intel OPA Host Fabric, Intel OPA Interface Driver version 10.4.2.0.7. SSD: Intel SSD 800GB DC S3700 Series.

Intel Xeon Phi Processor 7250
1S Intel Xeon Phi processor 7250, 68 Cores, 4 HW Threads per core, 1.4 GHz, 16GB high-speed MCDRAM set in Cache-Quadrant mode, 32KB L1 data cache per core, 1MB L2 per two-core tile, 96GB DDR4, Multiple nodes connected with Intel OPA Host Fabric, Intel OPA Interface Driver version 10.4.2.0.7, Intel SSD 480GB DC S3500 Series, Software: CentOS Linux release 7.3.1611, Linux kernel 3.10.0-514.10.2.el7.x86_64, Intel MPI Library 2017 Update 4.

16. REFERENCES:
[2]: https://github.com/intel/caffe/
[3]: https://software.intel.com/en-us/articles/optimizing-applications-for-numa
[7]: https://www.open-mpi.org/doc/v2.0/man1/miexec.1.php
17. AUTHORS:

Vikram Saletore is a Principal Engineer and a Machine Learning and Deep Learning Performance Architect and leads the Performance Enabling team in the Customer Technical Solutions team in the Artificial Intelligence Products Group at Intel Corporation for Intel® Xeon® and Intel® Nervana™ products. He has delivered optimized parallel database software to ISVs (Oracle, Informix) and ML Analytics optimizations on Apache/Spark to Cloudera, led joint research with HP Labs and more recently Co-PI for research with SURFsara on deep learning. Prior to Intel, Vikram was a faculty member in Computer Science at OSU, Corvallis, OR and led NSF sponsored ($300K) research in parallel programming and distributed computing supervising 8 students (PhD, MS). He also worked for AMD and DEC on network and CPU architectures. Vikram received his PhD in Electrical Engineering from the University of Illinois at Urbana-Champaign and MSEE from Berkeley and holds six patents with two pending and has more than 40 research publications.

Deepthi Karkada is a Machine Learning Engineer in the Performance Enabling Team in the Customer Solutions in the Artificial Intelligence Products Group at Intel Corporation. She works on deep learning framework and platform optimizations and benchmarking targeted for Intel Xeon Architectures and Intel Nervana products. Earlier she worked on seamless integration of Intel® Math Kernel Library with Apache Spark for Machine Learning and data analytics for Cloudera* Distribution of Hadoop*.

Vamsi Sripathi is a Software Engineer at Intel since 2010. He has a Masters' degree in Computer Science from North Carolina State University, USA. During his tenure at Intel, he worked on the performance optimization of Basic Linear Algebra Subroutines (BLAS) in Intel Math Kernel Library spanning multiple generations of Intel Xeon and Intel Xeon Phi architectures. Recently, he has been working on the optimization of deep learning algorithms and frameworks for Intel architectures and Intel Nervana products.

Kushal Datta is a Research Scientist in the Performance Enabling team in the Customer Solutions in the Artificial Intelligence Products Group at Intel Corporation. His interests are in Machine Learning, Deep Learning, systems performance optimizations and CPU micro-architecture. He is one of the lead authors of TileDB – a performant storage library for multi-dimensional arrays and GenomicsDB – a genomics data storage system used in GATK 4.0. Prior to Intel, Kushal graduated from University of North Carolina at Charlotte where he won a $40,000 research grant for developing a cycle-accurate CPU simulator for SPARCv9 instruction set with Sun Microsystems*. He holds four patents and several research publications.

Ananth Sankaranarayanan is the Director of Engineering leading AI Solutions and Applied Machine Learning teams in the AI Products Group at Intel Corporation. He is responsible for enabling and scaling the Intel Xeon and Intel Nervana AI product portfolio worldwide across Cloud Service Providers, Enterprise, Government and Communication Service Providers. Ananth has been with Intel since 2001 in various engineering leadership roles and has received Intel Achievement Award for delivering Intel’s first production High Performance Computing capability and more than 30 Divisional Recognition Awards. Ananth earned B.E. in Computer Science and Engineering, MBA in Information Systems. He holds two patents and has authored several technical publications.
18. NOTICES AND DISCLAIMERS:

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit: http://www.intel.com/performance.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, Xeon, Xeon Phi and Nervana are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

© 2018 Intel Corporation. All rights reserved.