DISTRIBUTED TENSORFLOW* TRAINING WITH INTEL® NGRAPH LIBRARY ON INTEL® XEON® SCALABLE PROCESSORS

Jianying Lang, Deep Learning Engineer

Artificial Intelligence Products Group

Data Center Group

Intel Corporation

July 11th, 2019

Outline

- Why and what is nGraph?
- Distributed training architecture on nGraph
- Scaling and convergence results
- Conclusion and next steps

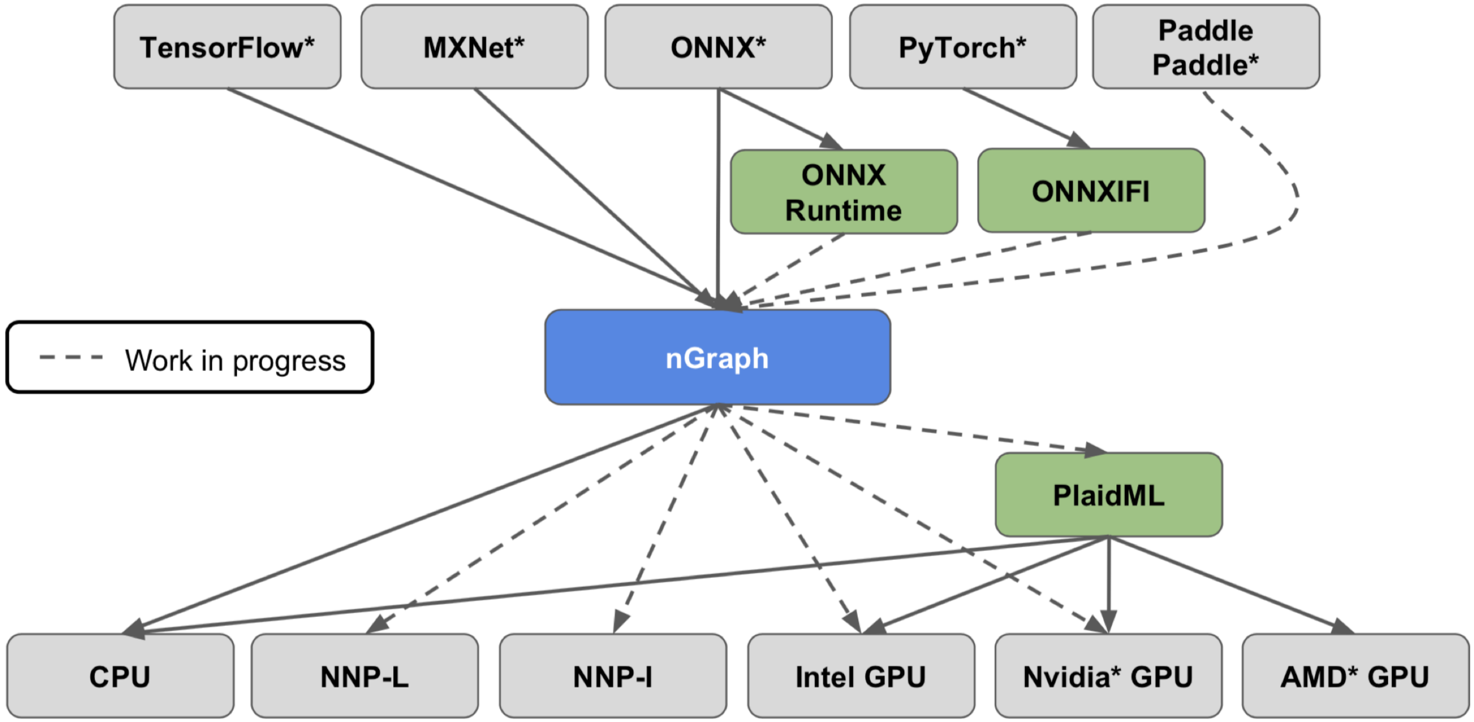
Why nGraph

https://vimeo.com/347401000

Courtesy to AIPG marketing team

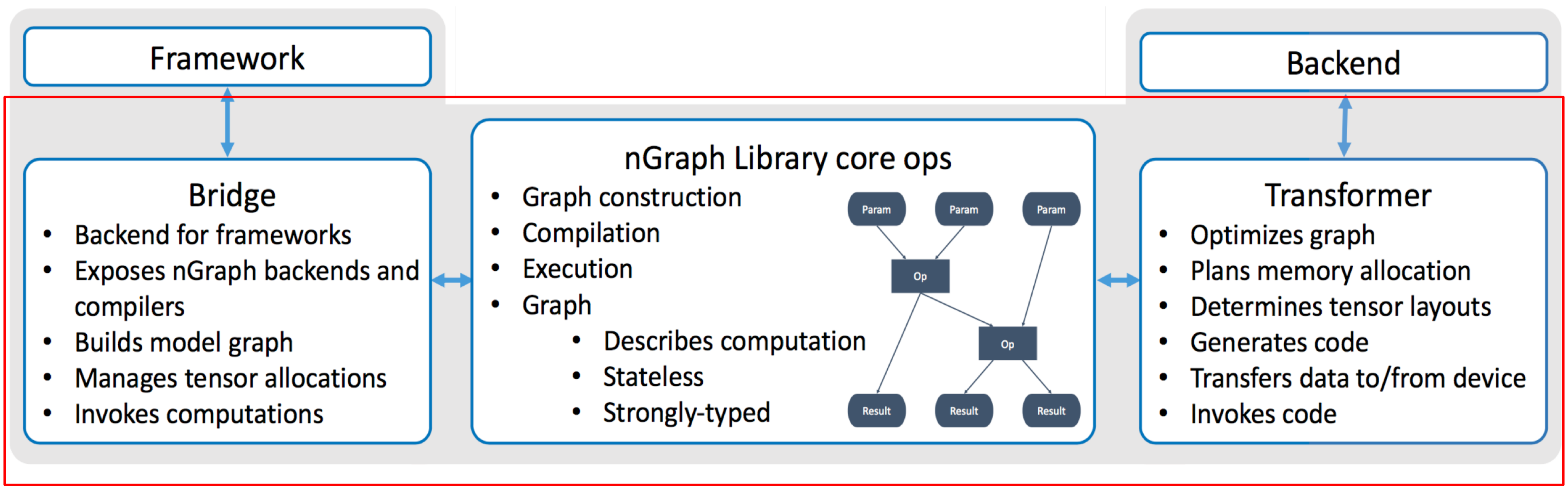
Reduces the optimization complexity from O(m*n) to O(m+n)

nGraph - Open source C++ library, compiler & runtime for DL



Intel nGraph Compiler stack contains trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others; see the branding notice and documentation for further details: https://ngraph.nervanasys.com/docs/latest

nGraph Stack

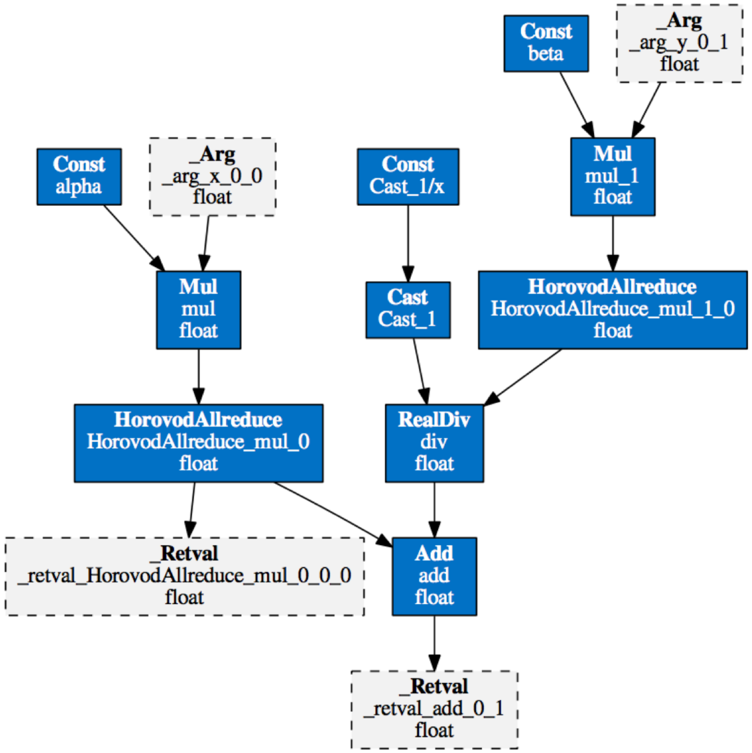


Easy to use

- Easy to install and use
- Support both training and inference
- Performance is on par or better than the direct optimization

- Minimize the changes from the user side
  - -Take Tensorflow as an example
  - -Users only need to add the following in their script
    import ngraph_bridge

TensorFlow Graph: Graph Marked for Clustering



Computing architecture for machine learning tasks

- The ultimate goal is to design a heterogeneous computing system with multiple kinds of processors to gain performance and energy efficiency in handling large-scale machine learning tasks.
- As a first step, we are focusing on distributed training using one architecture during the runtime.
- One goal is to develop generic distributed feature in nGraph which could be easily applied to all the framework platforms without much change from user side.

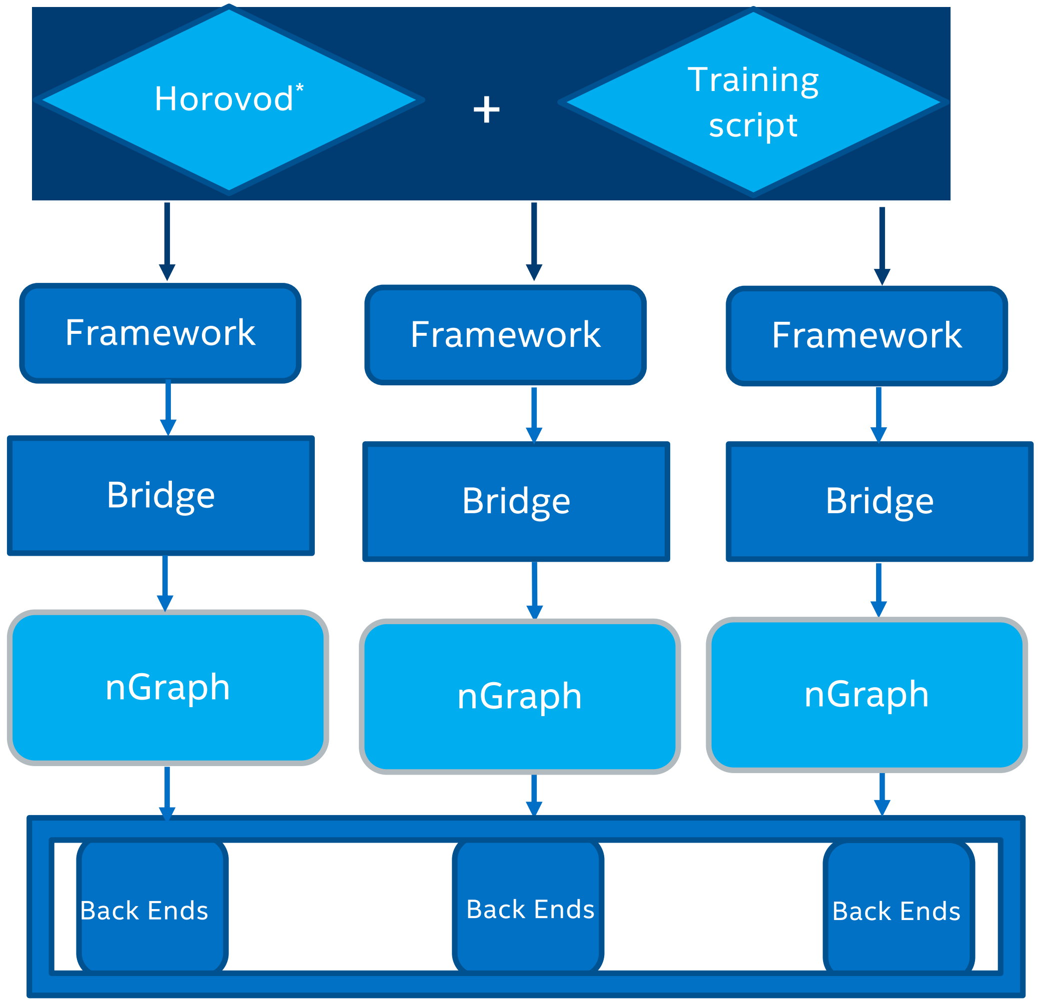
Data Parallel on nGraph

Each device will have a replica of the training script (graph)

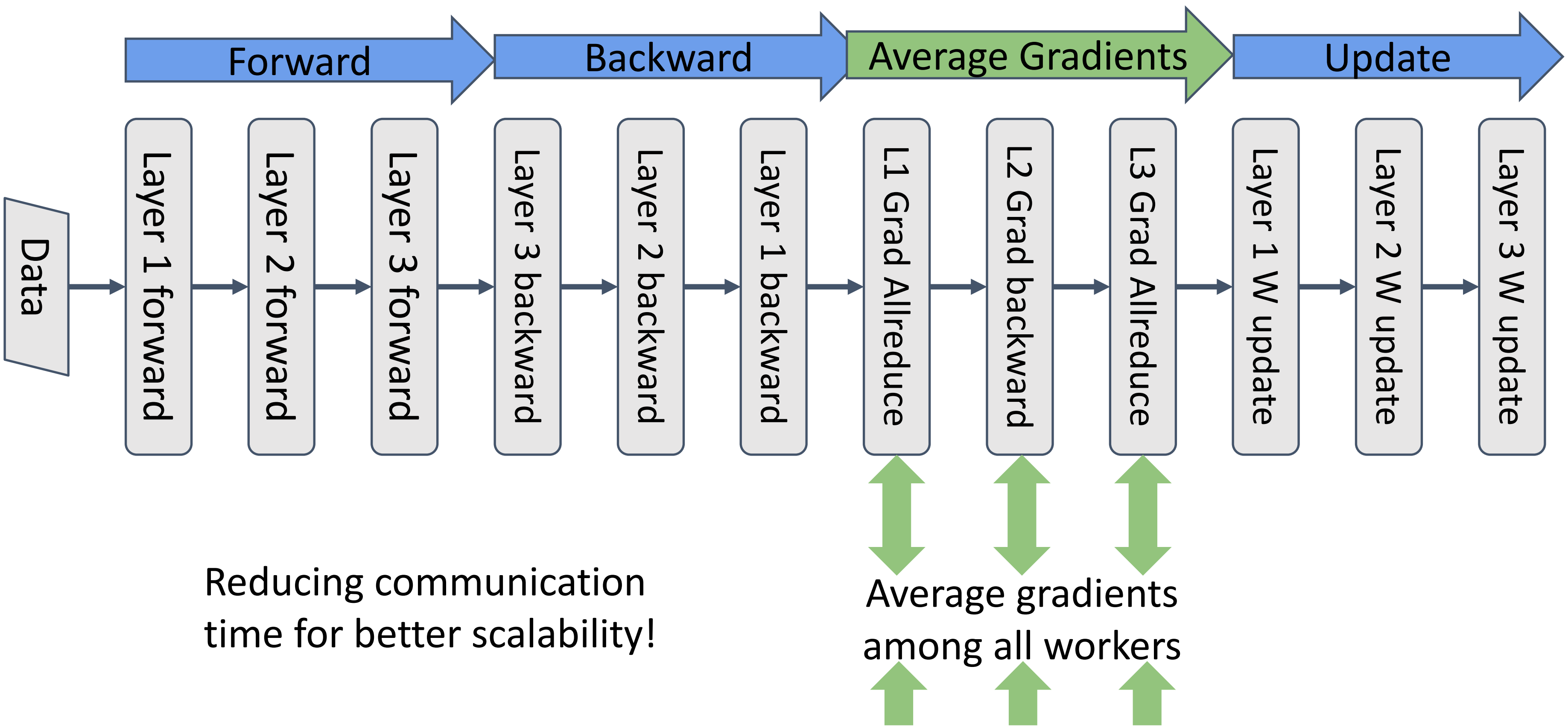
The training data is distributed on each device

- Each device runs its own data through the model and computes the local gradient
- Gradients on each device will be averaged though a gather-scattering process (allreduce)

The model will be updated with the averaged gradients

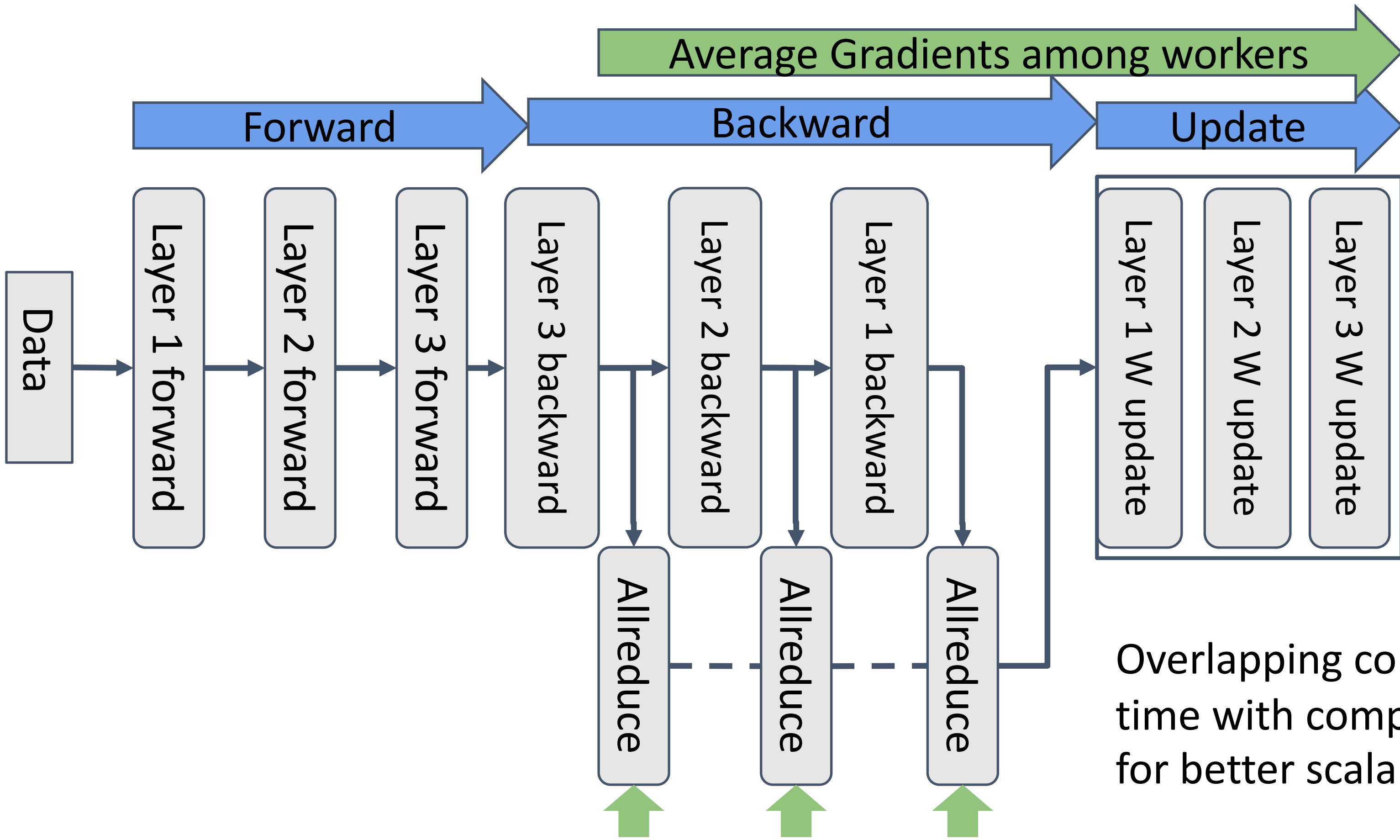


Synchronous SGD



Reducing communication time for better scalability!

Asynchronous SGD



Overlapping communication time with computation time for better scalability!

Horovod* -- MPI

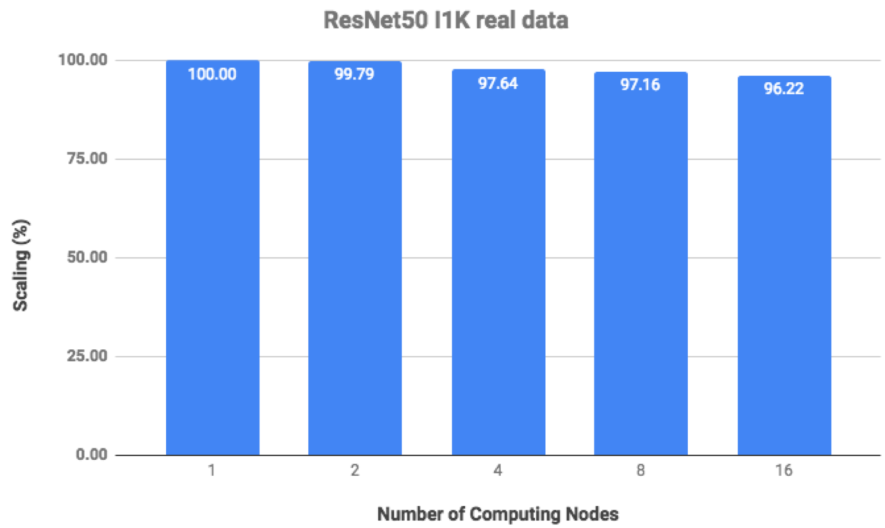
- Horovod* is a distributed training framework for TensorFlow*, Keras*, PyTorch*, and MxNet*.
- It utilize the MPI technique for the communication
- The installation requires MPI library and then pip install Horovod*
- This framework is user friendly and it requires less code change from user side than parameter server does
- Better scaling performance
  - MPI transparently sets up the distributed infrastructure necessary for workers to communicate with each other.
  - The MPI communication coordination implementation reduces the wait time for each device

Distributed training results

- The distributed model needs to achieve the scaling efficiency from training time to standard top-1/top-5 accuracy and scale up to a large number of nodes.
- The current distributed development in nGraph focuses on main framework: TensorFlow* , PaddlPaddle* , and MxNet*.
- This presentation will only showTensorFlow* results.
- Data parallel is used for distributed training. The batch size per device keeps the same.

Performance benchmark with Horovod*

We have achieved good multi-node scaling in nGraph using Horovod* by placing the communication op (AllReduce) on Intel® Xeon®



ResNet-50 with Imagenet 1K dataset using Horovod and nGraph TensorFlow

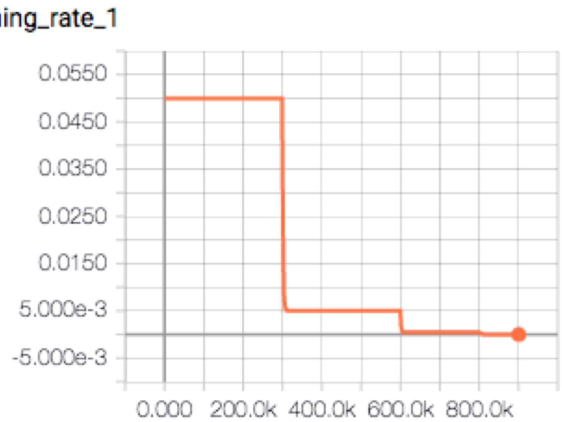
- Single node performs better with 2 processes mapped by socket using mpirun
- >96% scaling up to 16 nodes with Ethernet interconnect
- 16 2S Intel® Xeon® Platinum 8180 CPU nodes RN50 training achieves 15.4x performance gain compared to the single 2S node

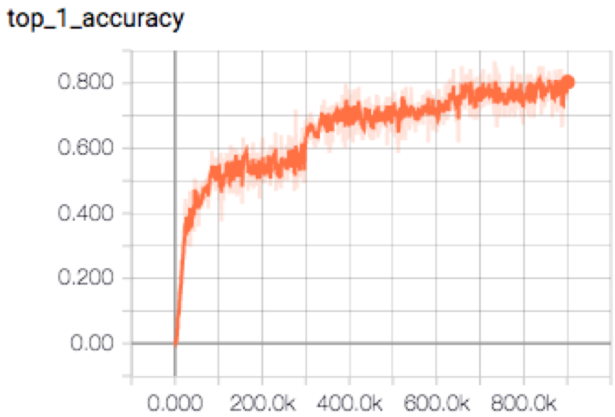
Convergence test (single node)

Hyper parameter tuning is the key to get convergence to the targeted accuracy for multi-node training

Larger batch size requires larger learning rate (lr ∝ Total Batch Size)

Take tf_cnn_benchmark model script as an example, the validation accuracy @top-1 is 75.5% and accuracy@top-5 is 92.54% at 90 epochs





Performance results are based on testing as of 10/20/2018, and may not reflect all publicly available security updates. See configuration disclosure for details on slide 27. No product can be absolutely secure. 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. 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

Multi-node convergence for ResNet50 I1K

By 90 epochs on 8 2S Intel® Xeon® Platinum 8180 CPU nodes

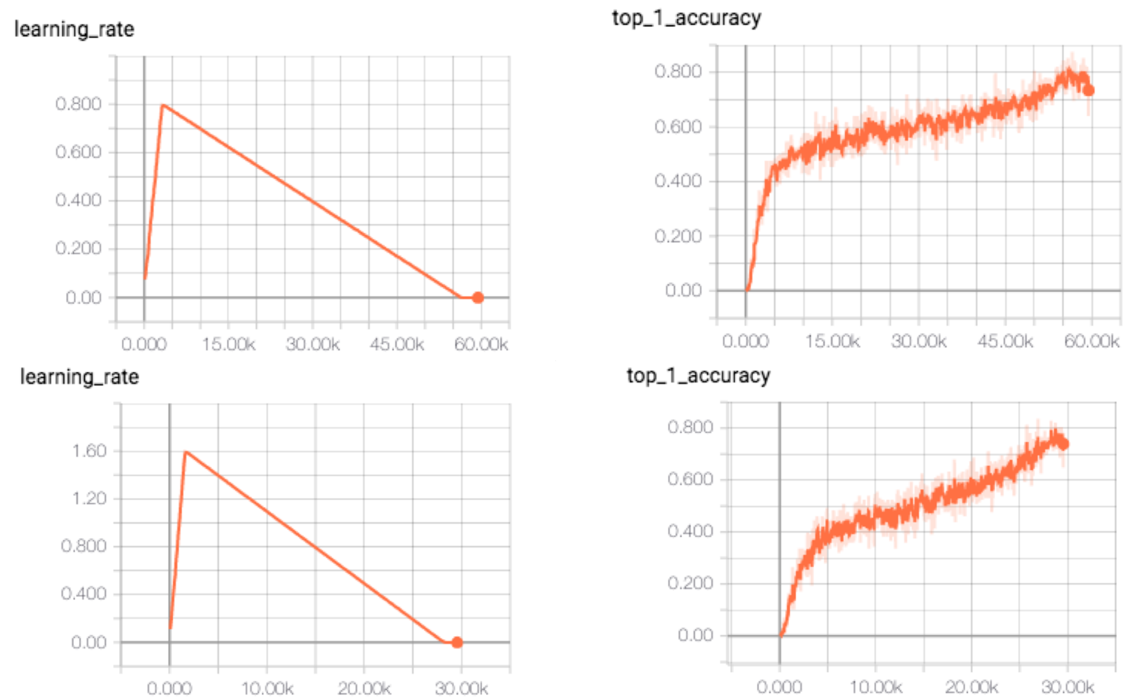
•75.5% accuracy@top-1

•92.5% accucary@top-5

By 90 epochs on 16 2S Intel® Xeon® Platinum 8180 CPU nodes

•75.0% accuracy@top-1

•92.3% accuracy@top-5



Conclusion

- Intel® developed open source code nGraph (C++) support distributed training and we have successfully run distributed ResNet50 I1K through nGraph on Intel® Xeon®.

- On multi-node Intel® Xeon® nodes , we have achieved the start-of-art ResNet50 accuracy@top-1 and accuracy@top-5.

- The performance reaches >96% of scaling efficiency up to 16 2s Intel® Xeon® Platinum 8180 CPU nodes.

- Work in progress on scaling nGraph to a large number of cluster nodes and developing model parallel support.

Configuration Details

Platforms: Intel® Xeon® Platinum 8180 CPU @ 2.50GHz, 2 sockets, 28 cores per socket, Link encap:Ethernet
model repos: https://github.com/tensorflow/models.git
Benchmark scripts: https://github.com/tensorflow/benchmarks.git
Tensorflow framework: https://github.com/tensorflow/tensorflow.git
Ngraph-TF: https://github.com/NervanaSystems/ngraph-tf.git
Horovod: https://github.com/uber/horovod.git

Run command:
export OMP_NUM_THREADS=28 export PYTHONPATH=$PYTHONPATH:<path-to-models>/models

mpirun --mca btl_tcp_if_include eno1 -np 16 -x LD_LIBRARY_PATH -H skx123,skx124,skx125,skx126, skx127,nervana-skx128,skx129,skx130 -x OMP_NUM_THREADS -x PYTHONPATH -cpus-per-proc 28 -map-by socket --oversubscribe --report-bindings python tf_cnn_benchmarks.py --variable_update horovod --model=official_resnet50 --batch_size=128 --data_format NCHW --num_intra_threads 28 --num_inter_threads 2 --horovod_device cpu --nodistortions --num_epochs=90 --num_warmup_batches=3125 --save_model_secs=625 --mkl=true --data_name=imagenet --data_dir /dataset/TF_Image Net_latest/ --train_dir 8node_resnet50 --datasets_use_prefetch=False --kmp_blocktime=1 --kmp_affinity=granularity=fine,compact,1,0 --eval_interval_secs=625 --print_training_accuracy=True --summary_verbosity=2 --benchmark_log_dir=benchmark_logs --optimizer=sgd --save_summaries_steps=20

mpirun --mca btl_tcp_if_include eno1 -np 32 -x LD_LIBRARY_PATH -H skx117, skx118, skx119,skx120, skx121, skx122, skx123, skx124,skx125, skx126, skx127, skx128, skx129, skx130, skx131, skx132 -x OMP_NUM_THREADS -x PYTHONPATH -cpus-per-proc 28 -map-by socket --oversubscribe --report-bindings python tf_cnn_benchmarks.py --variable_update horovod --model=official_resnet50 --batch_size=128 --data_format NCHW --num_intra_threads 28 --num_in ter_threads 2 --horovod_device cpu --nodistortions --num_epochs=90 --num_warmup_batches=1563 --save_model_secs=313 --mkl=true --data_name=imagenet --data_dir /dataset/TF_ImageNet_latest/ --train_dir /mnt/data/langjian/16node_resnet50_accu racy_sgd_90epoch --datasets_use_prefetch=False --kmp_blocktime=1 --kmp_affinity=granularity=fine,compact,1,0 --eval_interval_secs=313 --print_training_accuracy=True --summary_verbosity=2 --benchmark_log_dir=benchmark_logs --optimizer=sgd - -save_summaries_steps=20

NOTICES AND DISCLAIMERS