DEEP LEARNING FRAMEWORKS WITH MPI

Jeff Adams
Intel HPC Platform Software Product Manager
Data Center Solutions Group
Session Agenda and Objective

Why?
- Bring HPC scale out capabilities to Deep Learning (DL)—HPC can take advantage of multiple nodes to conduct training
- Enable DL capabilities in your existing HPC system or application—provide options that do not require additional hardware

How can I do it?
- Key learnings, steps and findings—leverage your existing HPC infrastructure

What can I expect?
- Proof of concept initial results

Guidance for your next steps—get started!
Session Framework

- Proof of concept and demonstration details that will help you get underway
  - Assumes an understanding HPC middleware software
  - Explored DL frameworks with MPI for CPU only

- DL is a rapidly changing (and understandably hyped) environment
  - Frameworks are evolving rapidly
  - Only a few frameworks implement scalable parallelism through MPI today
  - Generally speaking, frameworks are not yet designed for optimal MPI scaling
Proof of Concept and Demo Qualifications

- Did not explore all angles to vet and optimize DL frameworks
  - Tuning hyperparameters
  - Environmental variables
  - Configuration modifications
  - Algorithmic modifications
  - Other troubleshooting

- Did not profile to determine processing bottlenecks (TAU, Intel® V-Tune™ Amplifier...)
  - Did not troubleshoot all slowdowns for certain increases in worker counts
Scalable DL Frameworks Proof of Concepts

- Investigated DL frameworks atop OpenHPC based system; May & June 2017
  - Used installed libraries, rather than distro or independent libraries
  - Learnings & findings expected to be applicable with the OpenHPC software stack

- Attempted to scale out jobs to multiple nodes using the cluster workload management component (SLURM) and message passing library (MPI)

- With successful scale out, benchmarked and documented any improvement found through use of multi-node processing

- Frameworks:
  - Intel® Optimization for Caffe*: [https://github.com/intel/caffe](https://github.com/intel/caffe)
  - Baidu Tensorflow-AllReduce*: [https://github.com/baidu-research/tensorflow-allreduce](https://github.com/baidu-research/tensorflow-allreduce)
  - Microsoft Cognitive Toolkit* (CNTK): [https://github.com/Microsoft/CNTK](https://github.com/Microsoft/CNTK)
Proof of Concept System Configuration

- Four compute nodes
  - Intel® Xeon® CPU E5-2699 v3 @ 2.30GHz
  - 2 processors
  - 18 physical cores/processor
  - Intel® TrueScale Infiniband

- OpenHPC based middleware
  - SLURM workload manager
  - OpenMP
  - gnu & Intel® toolchains
  - OpenMPI & Intel® MPI

- CNTK & Tensorflow AllReduce conducted May 2017
Proof of Concept Dataset

Language Model:
Employ a convolutional neural network (CNN) as inputs into long short term memory (LSTM)

The “Billion Words” dataset and vocabulary for predicting the next word in a sentence are at the links below (used in allreduce-test.py, etc.):

http://download.tensorflow.org/models/LM_LSTM_CNN/vocab-2016-09-10.txt

http://statmt.org/wmt11/training-monolingual.tgz
Intel® Optimization for Caffe* Workflow Summary – CentOS

- Install Dependencies
  - Intel® Math Kernel Library
  - Boost >= 1.55
  - Hdf5
  - ...

- Enable BLAS for better CPU performance
  - Set BLAS := mkl in Makefile.config

- Compile
  - For CPU only runtime, uncomment CPU_ONLY := 1 in Makefile.config
Findings & Recommendation Summary

- Microsoft CNTK* and Intel® Optimization for Caffe* offer “decent” & scalable out of the box performance
  - Caffe demonstrated linear scaling to eight nodes with 96% efficiency
  - CNTK 3rd most active framework at time of PoC, development continuing

- Intel® Optimization for Caffe* and CNTK* are relatively simple to install atop an OpenHPC based stack
  - CNTK* automatically incorporates OpenMP for processes that use the BLAS library

- Tensorflow-AllReduce* challenging for initial experimentation
  - Found to be relatively difficult to install and run due to build program, dependencies, unit test script modifications, and patches required—your mileage may vary
CNTK Benchmark #1 Case Study

- Fully Connected Neural Network: FCN-5.
  - 3 hidden layers, 55 million parameters
  - Threading helps single-node case
  - 8-process MPI case gives 6.1x speedup over single process
  - OMP_NUM_THREADS for 16 process MPI modified to reduces time from 2.5 min.
  - May be too small data set to see MPI benefit
CNTK Benchmark #2 Case Study

- Convolutional Neural Network (CNN) - AlexNet
  - 2 hidden layers, 61 million parameter
  - Threading helps single-node case
  - 8-process MPI case gives 20.7x speedup over single processor
  - 16-process case required mod to OMP_NUM_THREADS

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. Copyright © 2017, Intel Corporation
CNTK Benchmark #3 Case Study

- Recurrent Neural Network (RNN), Long-short Term Memory (LSTM-32)
  - 2 LSTM layers, 13 million parameters
  - Threading helps single-node for < 32 processes
  - 8-process MPI case gives 31.7x speedup over single processor
  - 16- and 32-process MPI cases require mods to OMP_NUM_THREADS

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. Copyright © 2017, Intel Corporation
TensorFlow-AllReduce Benchmark

Billion Words, 20 iterations

- Using MPI option
  - `--map-by node:pe=# process`

- MPI has default bindings:
  - For # processes <= 2, `--bind-by core`
  - For # processes > 2, `--bind-by socket`

- Setting `--bind-by none` opens up processing to additional cores
  - Time to complete = 50 sec for any # processes
Tensorflow-AllReduce Preliminary Results

- Attempts to scale out to multiple nodes essentially runs same processes on each node, slowing overall processing time.
  - e.g., setting n=8 (2 instances/node) takes 1.3 min, compared to 0.8 min with n=1

- Need to look into modifying configuration for MPI functionality

- Shelved TensorFlow-AllReduce to focus on other ML frameworks due to difficulties
ISC 2017 Demonstration

- The Intel® Optimization for Caffe* was demonstrated live at ISC 2017 in the Intel booth with an eight node Intel® Xeon Phi® cluster using a publicly available dataset.

- Even though the Intel® Optimization for Caffe* is not integrated into an OpenHPC based system today, it was simple to build, install, and run it within the OpenHPC environment.
ISC 2017 Demonstration System Configuration

**Software**
- Intel® HPC Orchestrator 17.01.005.candidate1 [or use OpenHPC]
- Intel® Optimization for Caffe* 1.0.0
  [https://github.com/intel/caffe/releases/tag/1.0.0](https://github.com/intel/caffe/releases/tag/1.0.0)
- Intel® Parallel Studio XE 2017 Cluster Edition Update 1
- CentOS 7.2
- BBBC021 dataset from the Broad Institute
  [https://data.broadinstitute.org/bbhc/BBBC021/](https://data.broadinstitute.org/bbhc/BBBC021/)

**Hardware**
- Eight node Intel® Xeon Phi® Processors
- Intel® Omni-Path Fabric
ISC Intel® Optimization for Caffe* Demonstration Results

- Up to 8x scaling performance [7.8-7.9x] from 1 node to 8 nodes
- Single test showed similar linear scaling to 16 nodes
ISC 2017 Demonstration – Intel® Optimization for Caffe* Installation

- Intel® Optimization for Caffe* was installed following the instructions from BVLC’s Caffe* project page: [http://caffe.berkeleyvision.org/install_yum.html](http://caffe.berkeleyvision.org/install_yum.html)
  - During the ‘compilation’ phase, select the CPU-only option by uncommenting CPU_ONLY := 1 in Makefile.config

- Build toolchain (e.g. compiler) was provided by Intel® HPC Orchestrator (based on OpenHPC)

- Based on ISC demo, library dependencies (e.g. Boost) for building & running Caffe* were upstreamed into OpenHPC
Launching DL Applications

_Familial: the standard conventions you know and love_

- To use SLURM, for example, use a shell script or the command line:
  
  ```
  Shell$> mpiexec.hydra -bootstrap slurm -hosts c1,c2,c3,c4 -n 8 -ppn 2 ...
  ```

- Intel® Optimization for Caffe* can run as an MPI application without a Workload Manager:
  
  ```
  Shell$> mpirun -n 4 -ppn 2 ./build/tools/caffe train --solver=models/bvlc_googlenet/solver_client.prototxt --engine=MKL2017 2>&1 | tee -i ./multinode_train.out
  ```
Conclusions

- You can enable DL capabilities in your existing HPC system or application *today*

- Take advantage of multiple nodes to conduct training
Additional Resources: Reference Design

Intel® Scalable System Framework (Intel® SSF) Reference Design Cluster installation (2017.03.31)
- Intel® Xeon® Processor E5-2699 v4
- Intel® Xeon Phi™ Processor 7230
- Intel® Omni-Path Fabric
- Intel® HPC Orchestrator
- Intel® Optimization for Caffe*

SSF General Reference Designs:
Additional Resources

Additional instructions for Microsoft CNTK or Tensorflow-AllReduce:

jeff.adams@intel.com
