INTEL AND MACHINE LEARNING
AN OPEN SOURCE ENDEAVOUR!

Tutorial: build and run TensorFlow™ optimized for Intel Architecture.

Intel Technical Computing Enabling
Hans Pabst, Zurich/Switzerland

July 17th 2017

Optimized TensorFlow on Intel Architecture is available from the public GitHub repository

TensorFlow source code

$ git clone https://github.com/tensorflow/tensorflow.git [master]
$ wget https://github.com/tensorflow/tensorflow/archive/v1.2.1.tar.gz [release]

Benchmarks and models

$ git clone https://github.com/soumith/convnet-benchmarks.git
$ git clone https://github.com/tensorflow/models.git

Prerequisites

- GNU GCC 5.x, 6.x, or 7.x, and current Binutils (AVX2, etc.)
- Bazel (e.g., 0.5.2), Python 2.7 (do not use Python 3.x)
Build TensorFlow for Intel Architecture
Configure TensorFlow

demo@genova:~/.tensorflow$ ./configure

[...]
Please specify the location of python. [Default is /usr/bin/python]:
Found possible Python library paths:
/usr/local/lib/python2.7/dist-packages
/usr/lib/python2.7/dist-packages
Please input the desired Python library path to use. Default is
/usr/local/lib/python2.7/dist-packages]

Make sure Python 2.7 is available
(do not use Python 3.x).

Build TensorFlow for Intel Architecture
Configure TensorFlow

demo@genova:~/.tensorflow$ ./configure

[...]
Do you wish to build TensorFlow with MKL support? [y/N] y
MKL support will be enabled for TensorFlow
Do you wish to download MKL LIB from the web? [Y/n] y
Please specify optimization flags to use during compilation when bazel option "--
config=opt" is specified [Default is -march=native]: --copt=-O3 --linkopt=-pthread -
copt=-mfma --copt=-mavx2
[...]
Configuration finished

This setting is apparently not working, hence all flags need to be (later)
passed to Bazel whenever an optimized build is needed.
Build TensorFlow for Intel Architecture

Install Benchmarks

Background

- TensorFlow functionality is exposed to Python (TensorFlow frontend)
- To use TensorFlow, one (usually) runs a Python script
  - If a TF/Python script runs outside of the TF development directory, the scripts relies on TensorFlow being installed. ("import tensorflow as tf").
  - To install TF, one may use "pip" (see later). However, if the script is underneath of TF’s main directory it just uses the this version of TF.

Optional: install benchmarks underneath of TensorFlow

$ cd /path/to/tensorflow; mkdir -p tensorflow/models

$ ln -s /path/to/convnet-benchmarks/tensorflow /
tensorflow/models/convnetbenchmarks

Build TensorFlow for Intel Architecture

Compiler/target flags and Bazel

GCC 5.x, 6.x, or 7.x are recommended

- Older Linux distros my build GCC+Binutils from source (prepend to PATH)
- Current issue(s): use only **AVX2 target flags**, or simply no target flags
  → MKL/MKL-DNN dispatches critical code paths based on CPUID

TensorFlow (baseline and with MKL/MKL-DNN)

- **Baseline**: --copt=-O3 --linkopt=-pthread --c=f --c=-mavx2
  (no MKL)
- **Optimized**: --c=-O3 --linkopt=-pthread --c=f --c=-mavx2 
  (with MKL) --config=mkl --c=-DEIGEN_USE_VML
- Intel AVX-512 target flags (Eigen and TF are not yet ready!)
  --c=-mavx --c=-mavx512f --c=-mavx512cd 
  --c=-mavx512bw --c=-mavx512vl --c=-mavx512dq
Build TensorFlow for Intel Architecture

Compile TensorFlow

Compile TensorFlow and benchmarks (symlinked underneath of TF directory)

```bash
$ bazel build -c opt --config=mkl --copt=-DEIGEN_USE_VML --copt=-O3
   --linkopt=-pthread -copt=-mfma -copt=-mavx2
   //tensorflow/models/convnetbenchmarks:benchmark_alexnet
   //tensorflow/models/convnetbenchmarks:benchmark_overfeat
   //tensorflow/models/convnetbenchmarks:benchmark_vgg
   //tensorflow/models/convnetbenchmarks:benchmark_googlenet
   //tensorflow/tools/pip_package:build_pip_package

( //tensorflow:all-targets would build all possible targets )
```

Start over

```bash
$ bazel clean --async    [Bazel's way to retire its most recent cache entry, etc.]
$ rm -rf ~/.cache        [The hard way :-)]
```

Build TensorFlow for Intel Architecture

Package and install the TensorFlow Wheel

Package the TensorFlow Wheel file

```bash
$ bazel-bin/tensorflow/tools/pip_package/build_pip_package
   /tmp/tensorflow_pkg
```

Optional (save Wheel file for future installation):

```bash
$ cp /tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl
   /path/to/mysafeplace
```

Install the TensorFlow Wheel

```bash
[user] $ install --user --upgrade -I
   /tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl
[root] $ sudo -H pip install --upgrade -I
   /tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl
```
Build TensorFlow for Intel Architecture
Package and install the TensorFlow Wheel

Package the TensorFlow Wheel file

$ bazel-bin/tensorflow/tools/pip_package/build_pip_package \ 
   /tmp/tensorflow_pkg

Optional (save Wheel file for future installation):

$ cp /tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl \ 
   /path/to/mysafeplace

Install the TensorFlow Wheel

[user] $ install --user --upgrade -I \ 
   /tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl

[root] $ sudo -H pip install --upgrade -I \ 
   /tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl

Optimized Performance (Intel MKL/MKL-DNN)
OpenMP Thread Affinity and Placement*

Interleaving pages might be beneficial for training:

$ numactl -i all command

To run only 1 thread per core can be beneficial. For example:

Xeon 8180: X=28, Y=56  Xeon 8168: X=24, Y=48  Xeon 6148: X=20, Y=40

- KMP_AFFINITY=compact,1,granularity=fine
- KMP_HW_SUBSET=Xc,1t  OMP_NUM_THREADS=Y

To run 2 threads per core, simply use:

- KMP_AFFINITY=compact,1,granularity=fine

ALWAYS: enable quicker transition between OpenMP and TF-threads

- KMP_BLOCKTIME=1

A low number [ms] releases OpenMP threads asap rather than “hot”-spinning for new work.

* Focus on Intel's OpenMP implementation (OpenMP 4.0 standardizes thread affinization and placement using OMP_* variables).
Baseline Performance (without MKL/MKL-DNN)
Likwid-pin* can be beneficial

The latest Likwid apparently supports SKX:

$ git clone https://github.com/RRZE-HPC/likwid.git
$ make; sudo make install  (installs under /usr/local/)

Baseline performance appears to be best with 1 thread per core:

Xeon Platinum 8180

$ /usr/local/bin/likwid-pin -c E:N:56:1:2 command

Xeon Platinum 8168

$ /usr/local/bin/likwid-pin -c E:N:48:1:2 command

Xeon Gold 6148

$ /usr/local/bin/likwid-pin -c E:N:40:1:2 command

* Used to improve the baseline performance. The MKL-enabled TF should rely on OMP_* and KMP_* environment variables.

Adjust the number of threads in TensorFlow*
Optional / Advanced

Replace code such as:

```python
sess = tf.Session()  
```

... with the following:

```python
sess = tf.Session(
    config=tf.ConfigProto(
        inter_op_parallelism_threads=X,  
        intra_op_parallelism_threads=Y
    )
)
```

Default behavior (0 means all threads):

- X=0
- Y=0

Same as if the session configuration is not present.

Recommended for baseline:

- X = <number-of-cores - 1> or <1>
- Y = <number-of-cores - 1>

* There are other ways depending on the scripts modeling the topology e.g., manipulating tf.app.flags.FLAGS
Convnet Benchmarks

For example, to run the GoogleNetv1 benchmark (inference only):

$ KMP_AFFINITY=compact,1,granularity=fine \nKMP_BLOCKTIME=1 \nnumactl -i all \nbazel-bin/tensorflow/models/convnetbenchmarks/benchmark_googlenet \n   --forward_only=true

Optimized Performance with Intel MKL/MKL-DNN

- Above, the GoogleNetv1 benchmark may be a proxy for any other topology
  or a topology used for production.

More Information

Steps that may be beneficial prior to running benchmarks, etc.

- Kill orphaned "java" process (see "top"); common issue with JDK
  $ killall -9 java -u user
- Equalize and compact memory across sockets
  $ echo 3 > /proc/sys/vm/drop_caches
  $ echo 1 > /proc/sys/vm/compact_memory

References

### Performance on Broadwell (2x22 Cores)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Metric</th>
<th>Batch Size</th>
<th>Optimized Performance Training</th>
<th>Optimized Performance Inference</th>
<th>Speedup Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet-Alexnet</td>
<td>Images/s</td>
<td>128</td>
<td>524</td>
<td>1696</td>
<td>15.6x</td>
</tr>
<tr>
<td>ConvNet-GoogleNet v1</td>
<td>Images/s</td>
<td>128</td>
<td>112.3</td>
<td>439.7</td>
<td>6.7x</td>
</tr>
<tr>
<td>ConvNet-VGG</td>
<td>Images/s</td>
<td>64</td>
<td>47.1</td>
<td>151.1</td>
<td>5.7x</td>
</tr>
</tbody>
</table>

Baseline using TensorFlow 1.0 release with standard compiler knobs

Optimized performance using TensorFlow with Intel optimizations and built with

- `bazel build --config=mkl --copt=\"-DEIGEN_USE_VML\"`

---

### Performance on Knights Landing (68 Cores)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Metric</th>
<th>Batch Size</th>
<th>Optimized Performance Training</th>
<th>Optimized Performance Inference</th>
<th>Speedup Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet-Alexnet</td>
<td>Images/s</td>
<td>128</td>
<td>549</td>
<td>2698.3</td>
<td>15.6x</td>
</tr>
<tr>
<td>ConvNet-GoogleNet v1</td>
<td>Images/s</td>
<td>128</td>
<td>106</td>
<td>576.6</td>
<td>6.7x</td>
</tr>
<tr>
<td>ConvNet-VGG</td>
<td>Images/s</td>
<td>64</td>
<td>69.4</td>
<td>251</td>
<td>5.7x</td>
</tr>
</tbody>
</table>

Baseline using TensorFlow 1.0 release with standard compiler knobs

Optimized performance using TensorFlow with Intel optimizations and built with

- `bazel build --config=mkl --copt=\"-DEIGEN_USE_VML\"`
Additional Performance Gains (Parameter Tuning)

Best Setting for Xeon (Broadwell – 2 Socket – 44 Cores)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Data Format</th>
<th>Inter_op</th>
<th>Intra_op</th>
<th>KMP_BLOCKTIME</th>
<th>Batch size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet-AlexnetNet</td>
<td>NCHW</td>
<td>1</td>
<td>44</td>
<td>39</td>
<td>2048</td>
</tr>
<tr>
<td>ConvNet-Googlenet V1</td>
<td>NCHW</td>
<td>2</td>
<td>44</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>ConvNet-VGG</td>
<td>NCHW</td>
<td>1</td>
<td>44</td>
<td>1</td>
<td>128</td>
</tr>
</tbody>
</table>

Best Setting for Xeon Phi (Knights Landing – 68 Cores)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Data Format</th>
<th>Inter_op</th>
<th>Intra_op</th>
<th>KMP_BLOCKTIME</th>
<th>OMP_NUM_THREADS</th>
<th>Batch size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet-AlexnetNet</td>
<td>NCHW</td>
<td>1</td>
<td>136</td>
<td>39</td>
<td>136</td>
<td>2048</td>
</tr>
<tr>
<td>ConvNet-Googlenet V1</td>
<td>NCHW</td>
<td>2 training</td>
<td>68</td>
<td>Infinite</td>
<td>68</td>
<td>256</td>
</tr>
<tr>
<td>ConvNet-VGG</td>
<td>NCHW</td>
<td>1</td>
<td>136</td>
<td>1</td>
<td>136</td>
<td>128</td>
</tr>
</tbody>
</table>

Parameter tuning

- Intra_op, inter_op and OMP_NUM_THREADS: set for best core utilization
- Higher batch size: more parallelism, but both practical/memory constraints
- Data format: CPU prefers NCHW data format