TensorFlow™ with LIBXSMM

Getting Started

The TensorFlow repository (as cloned below) is tracking the master revision of the original TensorFlow and it is modified to use a recent revision of LIBXSMM (likely the master revision) as well as a recent revision of the Eigen library (which may be an improved version). It is also possible to build this fork of TensorFlow without LIBXSMM (see Special Builds section).

```
git clone https://github.com/hfp/tensorflow-xsmm.git
```

LIBXSMM does not impose to build for a specific code path, and always exploits the most suitable instruction set extension for JIT-enabled code paths. LIBXSMM may also use non-JIT code paths which are CPUID-dispatched (if the static code path has lower capabilities). This however only works with GCC 5.1 (v4.9 is fine if the intended target does not support AVX-512), Clang (not tested), or the Intel Compiler. It is hence recommended to use a recent GNU Compiler Collection to build TensorFlow. If the static code path does not match the highest possible CPU target (march=native), a very minor performance penalty (if at all) is expected (indirect calls due to CPUID-dispatch). With any recent Bazel version, a non-default compiler can be source'd i.e., it can be added to the environment just normally as shown below (the second block of exports may be safely omitted).

```
export PATH=/path/to/gcc/bin:${PATH}
export LD_LIBRARY_PATH=/path/to/gcc/lib64:/path/to/gcc/lib:${LD_LIBRARY_PATH}
export LIBRARY_PATH=/path/to/gcc/lib64:${LIBRARY_PATH}

export MANPATH=/path/to/gcc/share/man:${MANPATH}
export CXX=/path/to/gcc/bin/g++
export CC=/path/to/gcc/bin/gcc
export FC=/path/to/gcc/bin/gfortran
```

TensorFlow may be configured for the first time. In the past, Python 3 was problematic since it was not the primary development vehicle and Python 2.7 was de-facto prerequisite. It is recommended to use the default Python version available on the system (Linux distribution). For the configuration, all question may be (interactively) answered with their defaults. If not needed, some of the frameworks may be disabled (`TF_NEED_GCP=0`, `TF_NEED_HDFS=0`, `TF_NEED_S3=0`, and `TF_NEED_KAFKA=0`).

```
cd /path/to/tensorflow-xsmm
git pull

TF_NEED_GCP=0 TF_NEED_HDFS=0 TF_NEED_S3=0 TF_NEED_KAFKA=0 \
./configure
```

Bazel is downloading dependencies by default during the initial build stage and hence Internet access on the build system is highly desirable. When behind an HTTP-proxy, the environment variables `https_proxy` and `http_proxy` can be used but should carry `https://` and `http://` respectively.

```
export https_proxy=https://proxy.domain.com:912
export http_proxy=http://proxy.domain.com:911
```

In general, if the build step of any of the Bazel commands goes wrong, `-s --verbose_failures` can be used (`-s` shows the full command of each of the build steps). For non-production code such as for debug purpose, TensorFlow can be built with `-c dbg` (or at least `--copt=-O0`). For further reference, please consult the official guide to build TensorFlow from the sources. In case of production code, it is recommended the use a moderate optimization level (`-c opt --copt=-O2`), and to better focus on a reasonable set of target-flags (`--copt=-mfma --copt=-mavx2`). LIBXSMM (and MKL-DNN) both make use of CPUID-dispatch, and it is not too critical to pick for instance AVX-512 (even when available on the intended production target). However, if the desired workload is bottlenecked by Eigen code paths that are not covered by LIBXSMM (or MKL-DNN), one may be sufficiently served with Intel AVX2 instructions (`--copt=-mfma --copt=-mavx2`).

```
bazel build -c opt --copt=-O2 --linkopt=-pthread --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0 \
  --copt=-fopenmp-simd --copt=-DLIBXSMM_OPENMP_SIMD \
  --define tensorflow_xsmm=1 --define tensorflow_xsmm_convolutions=1 \
  --define tensorflow_xsmm_backward_convolutions=0 \
  //tensorflow/tools/pip_package:build_pip_package
```

If specific target flags are desired, one may select depending on the system capabilities:

- AVX2/HSW/BDW: `--copt=-mfma --copt=-mavx2` (as shown above, and typically sufficient)
- AVX-512/CORE/SKX: `--copt=-mfma --copt=-mavx512f --copt=-mavx512cd --copt=-mavx512bw --copt=-mavx512vl --copt=-m`

- AVX-512/MIC/KNL/KNM: `--copt=-mfma --copt=-mavx512f --copt=-mavx512cd --copt=-mavx512pf --copt=-mavx512er`

NOTE: TensorFlow or specifically Eigen's packed math abstraction may assert an unmet condition in case of AVX-512, hence one should either (1) limit the code to Intel AVX2 instructions, or (2) supply `-c opt` which in turn implies `--copt=-DNDEBUG` and thereby **disables** the assertions (at own risk). As a side-note (this is often missed in AVX2 vs. AVX-512 comparisons), AVX2 code can utilize twice as many registers (32) on an AVX-512 capable system (if instructions are EVEX encoded, which is practically always the case).

To finally build the TensorFlow (pip-)package ("wheel"), please invoke the following command (in the past the zip-stage ran into problems with Python wheels containing debug and thereby exceeding 2 GB for the archive).

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

The new Python TensorFlow wheel can be installed by the following command (use `sudo -H` in front to elevate your permissions, or add `--user` to install locally for the current user rather than installing it in a system-wide fashion):

```
pip install \
  --proxy proxy.domain.com:912 \
  -I /tmp/tensorflow_pkg/<package-name-build-above.whl>
```

The `-I` flag shall enforce to reinstall the wheel even when the name of the wheel suggests the same version. To really ensure that no other bits are left, please remove any previously installed TensorFlow wheel:

```
pip uninstall tensorflow
pip install \
  --proxy proxy.domain.com:912 \
  /tmp/tensorflow_pkg/<package-name-build-above.whl>
```

Performance Tuning and Profiling

As suggested in the overview, it is possible to build the mentioned fork of TensorFlow without LIBXSMM ("vanilla build"), or to try-out MKL-DNN as a compute engine. This may be desired to aim for performance, or to draw a performance baseline respectively. To omit LIBXSMM, just omit the flags `tensorflow_xsmm`, `eigen_xsmm`, and `tensorflow_xsmm_backward` (sometimes it may be desired to `--define tensorflow_xsmm_backward=0` but to keep `--define tensorflow_xsmm=1` and `--define eigen_xsmm=1`). To try MKL-DNN, simply use `--config=mkl` (instead of `tensorflow_xsmm`, `eigen_xsmm`, and `tensorflow_xsmm_backward`). To use MKL-DNN effectively, one may setup the environment with at least `KMP_BLOCKTIME=1` (perhaps more environment settings such as `KMP_AFFINITY=compact,1,granularity=fine`, `KMP_HW_SUBSET=1T`, and `OMP_NUM_THREADS=<number-of-physical-cores-not-threads>` are beneficial). The `KMP_BLOCKTIME` shall be set to a "low number of Milliseconds" (if not zero) to allow OpenMP workers to quickly transition between MKL's and TF's (Eigen) thread-pool.

It can be very beneficial to scale TensorFlow even on a per-socket basis (in case of multi-socket systems). Generally, this may involve (1) real MPI-based communication, or (2) just trivially running multiple instances of TensorFlow separately (without tight communication). For example, Horovod can be used to perform an almost "trivial" instancing of TensorFlow, and to add an intermittent averaging scheme for exchanging weights between independently learning instances (Horovod is out of scope for this document). Similarly, for inference all incoming requests may be dispatched (in batches) to independent instances of TensorFlow. For the latter, the TensorFlow Serving framework may be used to serve for inference-requests with an easy to use web-based client/server infrastructure.

However, to quickly check the benefits of scaling TensorFlow, one may simply use `numactl` to run on a single socket only; multiplying the achieved performance according to the number of sockets yields a quick estimate of scaling performance. Here is an example for a single dual-socket Skylake server system with HT enabled and sub-NUMA clustering disabled (2x24 cores, 96 threads in two memory-domains/sockets).

```
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 20 21 22 23 48 49 50 51 52 53 54 55 5
  node 0 size: 96972 MB
  node 0 free: 91935 MB
  node 1 cpus: 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 72 73 74 75 76
  node 1 size: 98304 MB
  node 1 free: 95136 MB
  node distances:
  node   0   1
    0:  10  21
    1:  21  10
```

To run a workload on a single socket (of the afore mentioned system), one may execute the following command:

```
$ numactl -C 0-23,48-71 ./my_tf_workload.py
```

It can be assumed that running on two sockets independently is twice as fast as the performance measured in the previous step. For any benchmarks, a freshly booted system shall be used (alternatively, a root/sudo user can drop filesystem caches and defragment memory pages):

```
echo 3 > /proc/sys/vm/drop_caches
echo 1 > /proc/sys/vm/compact_memory
```

To gain insight into performance bottlenecks, one might source the Intel VTune Amplifier and run:

```
amplxe-cl -r result -data-limit 0 \
  -collect advanced-hotspots -knob collection-detail=stack-sampling -- \
  ./my_tf_workload.py
```

To get have nicely named JIT-kernels, one may add the following flags to Bazel's build line (Intel VTune Amplifier 2018):

```
--copt=-DLIBXSMM_VTUNE=2 --linkopt=${VTUNE_AMPLIFIER_2018_DIR}/lib64/libjitprofiling.a
```

For Intel VTune Amplifier 2017 this looks like:

```
--copt=-DLIBXSMM_VTUNE=2 --linkopt=${VTUNE_AMPLIFIER_XE_2017_DIR}/lib64/libjitprofiling.a
```

Benchmarks

This document is a recipe for building and running TensorFlow with LIBXSMM. The amount of covered code paths as well as the performance of these code paths are under development. Please do not expect any performance advantage (at this point) when comparing to TensorFlow without LIBXSMM!

TensorFlow Model Repository

This section may help to quickly setup models from the TensorFlow repository. Care must be taken to ensure that the model in question uses the NHWC-format, which is assumed by LIBXSMM. In most (if not all) cases this is not the default, and the model must be adjusted.

```
git clone https://github.com/tensorflow/models.git tensorflow-models
cd /path/to/tensorflow-xsmm
ln -s /path/to/tensorflow-models tensorflow/models

bazel build <all-build-flags-used-to-build-the-wheel> //tensorflow/models/tutorials/image/alexnet:alex
```

The above command may be combined with `//tensorflow/tools/pip_package:build_pip_package` to build TF as well. Please remember, the TF wheel needs to be only installed if the model runs outside of TF's source tree. To run the "Alexnet" benchmark:

```
LIBXSMM_VERBOSE=2 \
bazel-bin/tensorflow/models/tutorials/image/alexnet/alexnet_benchmark \
  --batch_size=256 2>&1 \
| tee output_alexnet.log
```

Convnet Benchmarks

The section helps to quickly setup benchmarks for Alexnet, Overfeat, VGG, and Googlenet v1. Recently, the original Convnet benchmark **stopped working with current TensorFlow**: please rely on TensorFlow model repository (previous section).

```
git clone https://github.com/soumith/convnet-benchmarks.git
cd /path/to/tensorflow-xsmm
mkdir -p tensorflow/models
ln -s /path/to/convnet-benchmarks/tensorflow tensorflow/models/convnetbenchmarks

bazel build <all-build-flags-used-to-build-the-wheel> \
  //tensorflow/models/convnetbenchmarks:benchmark_alexnet \
  //tensorflow/models/convnetbenchmarks:benchmark_overfeat \
  //tensorflow/models/convnetbenchmarks:benchmark_vgg \
  //tensorflow/models/convnetbenchmarks:benchmark_googlenet
```

The above command may be combined with `//tensorflow/tools/pip_package:build_pip_package` to build TF as well. Please note, the wheel needs to be only installed if the model runs outside of TF's source tree. To run the "Alexnet" benchmark:

```
LIBXSMM_VERBOSE=2 \
bazel-bin/tensorflow/models/convnetbenchmarks/benchmark_alexnet \
  --data_format=NHWC --forward_only=true --batch_size=256 2>&1 \
| tee output_alexnet.log
```

Regression Tests

There are two aspects of LIBXSMM enabled within TensorFlow: (1) sparse CNN, and (2) CNN. To build and test the sparse routines:

```
bazel build <all-build-flags-used-to-build-the-wheel> //tensorflow/core/kernels:sparse_matmul_op_test

bazel-bin/tensorflow/core/kernels/sparse_matmul_op_test --benchmarks=all
bazel-bin/tensorflow/core/kernels/sparse_matmul_op_test

bazel run <all-build-flags-used-to-build-the-wheel> //tensorflow/python/kernel_tests:sparse_matmul_op_
```

To build and test the regular CNN routines (note that below `bazel run`... may be deadlocking during the test):

```
bazel build <all-build-flags-used-to-build-the-wheel> //tensorflow/core/kernels:conv_ops_test
bazel-bin/tensorflow/core/kernels/conv_ops_test

bazel run <all-build-flags-used-to-build-the-wheel> //tensorflow/python/kernel_tests:conv_ops_test
```

Running Inception-v3 inference on the ImageNet dataset

Please follow the instructions at the following link to download and preprocess the Inception-v3 dataset: The relevant part of the instructions are duplicated below for convenience.

```
# location of where to place the ImageNet data
DATA_DIR=$HOME/imagenet-data

# build the preprocessing script.
cd tensorflow-models/inception
bazel build //inception:download_and_preprocess_imagenet

# run it
bazel-bin/inception/download_and_preprocess_imagenet "${DATA_DIR}"
```

The final line of the output script should read something like this, note the number of images:

```
2016-02-17 14:30:17.287989: Finished writing all 1281167 images in data set.
```

Please download models/slim from this link. Please download the pretrained weights for Inception-v3 from here. Please setup the environment variables as follows:

```
export CHECKPOINT_FILE= location of downloaded inception-v3 pretrained weights
export DATASET_DIR=$DATA_DIR
```

Please modify the file eval_image_classifier.py in models/slim so that inter_op_parallelism_threads is set to 1 since TensorFlow/libxsmm does not support concurrent evaluations of subgraphs currently.

```
slim.evaluation.evaluate_once(
        master=FLAGS.master,
        checkpoint_path=checkpoint_path,
        logdir=FLAGS.eval_dir,
        num_evals=num_batches,
        eval_op=list(names_to_updates.values()),
        variables_to_restore=variables_to_restore,
        session_config= tf.ConfigProto(inter_op_parallelism_threads=1))
```

Run inference on ImageNet as follows:

```
python eval_image_classifier.py \
    --alsologtostderr \
    --checkpoint_path=${CHECKPOINT_FILE} \
    --dataset_dir=${DATASET_DIR} \
    --dataset_name=imagenet \
    --dataset_split_name=validation \
    --model_name=inception_v3
```

Please verify recall and accuracy as follows:

```
2017-07-13 21:21:27.438050: I tensorflow/core/kernels/logging_ops.cc:79] eval/Recall_5[0.93945813]
2017-07-13 21:21:27.438104: I tensorflow/core/kernels/logging_ops.cc:79] eval/Accuracy[0.77981138]
```