TensorFlow™ with LIBXSMM

Getting Started

This document is an early recipe for building and running TensorFlow with LIBXSMM. The amount of covered code paths as well as the performance of these code paths will be improved as the integration progresses. This means that executing TensorFlow for any real workload (or benchmark) beside of the cases shown below **may not use LIBXSMM at all** (same is/remains true for any system without Intel AVX2 instruction set extension). Relying on the master revision of TensorFlow is perfectly fine since the integration work is merged on a frequent basis. However, to capture the latest revision of the integration one may rely on:

```
git clone https://github.com/benoitsteiner/tensorflow-xsmm.git
wget https://github.com/hfp/libxsmm/archive/master.zip
sha256sum libxsmm-master.zip
```

To try LIBXSMM's master revision, the file `tensorflow/workspace.bzl` can be adjusted by using the SHA256 sum from above:

```
  native.new_http_archive(
      name = "libxsmm_archive",
      urls = [
          "https://github.com/hfp/libxsmm/archive/master.zip",
      ],
      sha256 = "<sha256sum␣libxsmm-master.zip>",
      strip_prefix = "libxsmm-master",
      build_file = str(Label("//third_party:libxsmm.BUILD")),
  )
```

Beside of the regular prerequisites, nothing else is needed to use TensorFlow with LIBXSMM. Prior to Bazel 0.4.5, it was helpful to change TensorFlow's `configure` script by replacing `bazel clean --expunge` with `bazel clean --expunge_async` (at least when NFS-hosted).

```
cd /path/to/tensorflow-xsmm
./configure
```

When behind a HTTP-proxy, the environment variables should carry `https://` or `http://`:

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

Please note that certain content under `tensorflow/models` may cause an error during the configuration. In such a case, simply configure with "models" temporarily not present. In general, if the build step of any of the Bazel commands goes wrong, `-s --verbose_failures` can be added to the command line (`-s` shows the full command of each of the build steps). The flags `--define tensorflow_xsmm=1`, `--define eigen_xsmm=1`, and `--define tensorflow_xsmm_backward=1` are not actually needed for all the cases, but are supplied for consistency.

More important, one line of below target flags should be added to Bazel's build line:

- AVX2/HSW/BDW: `--copt=-mfma --copt=-mavx2`
- AVX-512/SKX: `--copt=-mfma --copt=-mavx512f --copt=-mavx512cd --copt=-mavx512bw --copt=-mavx512vl` (and `--copt=-mavx512dq` depending on certain fixes being already present in TensorFlow)
- AVX-512/KNL: `--copt=-mfma --copt=-mavx512f --copt=-mavx512cd --copt=-mavx512pf --copt=-mavx512er`

NOTE: TensorFlow may run into issues, and one may temporarily apply `--copt=-mfma --copt=-mavx2` (even if Intel AVX-512 extensions are available). There are at least two issues: (1) there can be NaNs e.g., printed when training a network regardless of the GCC-version, or (2) TensorFlow *without* LIBXSMM may crash in TF's implementation of GEMM.

For AVX-512 in general, GCC 5.x (or higher) should be used (see section Non-default Compiler). LIBXSMM supports Intel AVX2 as the baseline code path for all JIT-generated DNN-code (SMM domain also supports AVX). For Intel AVX-512 (on top of AVX2), the foundational instructions are sufficient in many cases, but for the sparse domain the Core-flavor is a prerequisite ("Skylake server" or SKX), and VNNI/QFMA instructions are honored on Intel Xeon Phi code-named "Knights Mill" (KNM).

Generally, please follow the guide to build TensorFlow from the sources. Bazel command lines (like below) can be extended to build an optimized target (default: `-c opt`), or they can be extended to include debug symbols (per `-c dbg`, which is usually combined with `--copt=-O0`). Please invoke the following commands to build the pip-package (Python wheel):

```
bazel build --copt=-O3 --copt=-fopenmp-simd --copt=-DLIBXSMM_OPENMP_SIMD --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  <line-of-target-flags-from-above> \
  //tensorflow/tools/pip_package:build_pip_package

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 in a system-wide fashion):

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

Benchmarks

This document is an early recipe for building and running TensorFlow with LIBXSMM. Please do not expect any performance advantage (at this point) when comparing to TensorFlow without LIBXSMM!

Convnet Benchmarks

The section helps to quickly setup benchmarks for Alexnet, Overfeat, VGG, and Googlenet v1. To setup a more complete set of models, please have a look at the next subsection.

```
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 --copt=-O3 --copt=-fopenmp-simd --copt=-DLIBXSMM_OPENMP_SIMD --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  <line-of-target-flags-from-above> \
  //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 to build `//tensorflow/tools/pip_package:build_pip_package` as well. When completed (wheel installed!) e.g., 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
```

In case of an `ImportError: No module named builtins` one can resolve the problem with `sudo -H pip install future --upgrade`.

Non-default Compiler

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. However, LIBXSMM may also use non-JIT code paths which are CPUID-dispatched when the static code path has lower capabilities. This only works when using GCC 4.9 (or later) or the Intel Compiler. If TensorFlow does not match the highest possible CPU target (march=native), a performance penalty is possible.

It is recommended to rely on a pre-built compiler by using for instance the "devtools" package (RedHat) or similar (depends on the Linux distribution). To use a custom-built compiler with TensorFlow may not only ask to source this compiler:

```
export LD_LIBRARY_PATH=/software/gnu/gcc-6.3.0/lib64:/software/gnu/gcc-6.3.0/lib:${LD_LIBRARY_PATH}
export PATH=/software/gnu/gcc-6.3.0/bin:${PATH}
```

but to further advertise the different compiler-runtime to the linker (`ld`).

```
echo "/software/gnu/gcc-6.3.0/lib64" > software-gnu-gcc630.conf
sudo mv software-gnu-gcc630.conf /etc/ld.so.conf.d/
sudo ldconfig
```

If there are still problems when using the custom compiler (mismatched GLIBC version), the symbolic link `libstdc++.so.6` (under `/usr/lib64`) may be adjusted to point to the latest update of the same major GLIBC version.

Performance Profiling

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 -- \
  bazel-bin/tensorflow/models/convnetbenchmarks/benchmark_alexnet \
    --data_format=NHWC --forward_only=true --batch_size=64 --num_batches=50
```

To get named JIT-kernels, one may add the following flags to Bazel's build line:

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

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 --copt=-O3 --copt=-fopenmp-simd --copt=-DLIBXSMM_OPENMP_SIMD --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  <line-of-target-flags-from-above> \
  //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 --copt=-O3 --copt=-fopenmp-simd --copt=-DLIBXSMM_OPENMP_SIMD --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  <line-of-target-flags-from-above> \
  //tensorflow/python/kernel_tests:sparse_matmul_op_test
```

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

```
bazel build --copt=-O3 --copt=-fopenmp-simd --copt=-DLIBXSMM_OPENMP_SIMD --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  <line-of-target-flags-from-above> \
  //tensorflow/core/kernels:conv_ops_test

bazel-bin/tensorflow/core/kernels/conv_ops_test

bazel run --copt=-O3 --copt=-fopenmp-simd --copt=-DLIBXSMM_OPENMP_SIMD --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  <line-of-target-flags-from-above> \
  //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]
```