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
```

In order 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. However, at least in some cases (NFS-hosted directory, etc.) it may help to adjust the `configure` script (change `bazel clean --expunge` into `bazel clean --expunge_async`).

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

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

```
bazel build -c opt --copt=-mavx2 --copt=-mfma \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/core/kernels:sparse_matmul_op_test

bazel run -c opt --copt=-mavx2 --copt=-mfma \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/python/kernel_tests:sparse_matmul_op_test

bazel-bin/tensorflow/core/kernels/sparse_matmul_op_test --benchmarks=all
bazel-bin/tensorflow/core/kernels/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 -c opt --copt=-mavx2 --copt=-mfma \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/core/kernels:conv_ops_test

bazel run -c opt --copt=-mavx2 --copt=-mfma \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/python/kernel_tests:conv_ops_test

bazel-bin/tensorflow/core/kernels/conv_ops_test
```

Generally, please follow the guide to build TensorFlow from the sources. Please invoke the following commands to build and install a pip-package, which represents your build of TensorFlow:

```
bazel build -c opt --copt=-mavx2 --copt=-mfma \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/tools/pip_package:build_pip_package

bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorflow_pkg
sudo pip install -I /tmp/tensorflow_pkg/<package-name-build-above.whl>
```

In general, if the build step of any of the above 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 above cases, but are supplied for consistency. More important, `--copt=-mavx2 --copt=-mfma` are only suitable for Intel AVX2 capable systems.

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).

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.

Changing the compiler when building TensorFlow appears to be brittle, if the compiler is only sourced in the usual way. For example:

```
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}
```

It is further necessary to 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
```