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 tensorflow-xsmm
./configure
```

For people working behind a HTTP/S proxy, the environment variables must be setup with prefixed URLs (`https://` and `http://`):

```
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 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, in addition one set of below target flags should be added:

- AVX2/HSW/BDW: `--copt=-mavx2 --copt=-mfma`
- AVX-512F (Foundational): `--copt=-mfma --copt=-mavx512f --copt=-mavx512cd`
- AVX-512/SKX: `--copt=-mavx512dq --copt=-mavx512bw --copt=-mavx512vl` (plus AVX-512F)
- AVX-512/KNL: `--copt=-mavx512pf --copt=-mavx512er` (plus AVX-512F)

In order for Intel AVX-512 support, GCC 5.x should be used (see more 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).

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 -c opt --copt=-O3 --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //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 -c opt --copt=-O3 --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //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 -c opt --copt=-O3 --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/core/kernels:conv_ops_test

bazel-bin/tensorflow/core/kernels/conv_ops_test

bazel run -c opt --copt=-O3 --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/python/kernel_tests: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=-O3 --linkopt=-pthread \
  --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>
```

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! Please note that the symbolic link ("models") shown below should may not be present when configuring TensorFlow.

```
git clone https://github.com/soumith/convnet-benchmarks.git
cd tensorflow-xsmm; mkdir -p tensorflow/models
ln -s /path/to/convnet-benchmarks/tensorflow-models tensorflow/models/convnetbenchmarks
bazel build -c opt --copt=-O3 --linkopt=-pthread \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/models/convnetbenchmarks:benchmark_alexnet \
  //tensorflow/models/convnetbenchmarks:benchmark_overfeat \
  //tensorflow/models/convnetbenchmarks:benchmark_vgg \
  //tensorflow/models/convnetbenchmarks:benchmark_googlenet
```

For example, to run the "Alexnet" benchmark one may enter the following command:

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

In case of an `ImportError: No module named builtins`, it resolves with `sudo 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 recommended to rely on a pre-built compiler by using for instance the "devtools" package (RedHat) or similar (depends on the Linux distribution). It apparently helps to build `bazel` using the compiler intended for building TensorFlow. 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
```