TensorFlow™ with LIBXSMM

Getting Started

Previously, this document covered building TensorFlow with LIBXSMM's API for Deep Learning (direct convolutions and Winograd). LIBXSMM's Deep Learning domain (DL) is under active research and quickly evolving, and hence reintegration with TensorFlow may be needed. This document focuses on building TensorFlow from source with Intel MKL and MKL-DNN plus LIBXSMM's code for sparse Matrix Dense-Matrix multiplication (SpMDM). LIBXSMM SpMDM is rather stable and integrated with TensorFlow since TF 1.1 (`--define tensorflow_xsmm=1`). To start building TensorFlow, one may clone the source from the official Git-repository:

```
git clone https://github.com/tensorflow/tensorflow.git
```

MKL, MKL-DNN, and LIBXSMM do not impose to build for a specific code path or target flags and attempt to exploit the most recent instruction set extensions. For most other code it is 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), TensorFlow emits a warning at runtime which is reasonable given that libraries such as Eigen may contribute performance critical code paths. With any recent Bazel version, a non-default GNU Compiler Collection 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 the de-facto prerequisite). It is recommended to use the default Python version available on the system (Linux distribution's default). For the configuration, all questions may be (interactively) answered with the suggested defaults. In earlier revisions of TensorFlow some frameworks could be disabled at configure-time in a non-interactive fashion using environment variables (`TF_NEED_GCP=0`, `TF_NEED_HDFS=0`, `TF_NEED_S3=0`, `TF_NEED_KAFKA=0`). However, the current mechanism to disable certain frameworks is per Bazel's build-line (`--config=noaws`, `--config=nogcp`, `--config=nohdfs`, `--config=noignite`, `--config=nokafka`, `--config=nonccl`).

```
cd /path/to/tensorflow
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` are considered by the Python package installer (pip) but they should carry `https://` and `http://` respectively (in the past `pip --proxy` was necessary despite of the environment variables being present, e.g., `pip --proxy proxy.domain.com:912`).

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

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). To start over completely, one may wipe directory caching the downloaded dependencies which is located by default in user's home and called ".cache" (`rm -rf $HOME/.cache`). 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 sources. In case of production code, it is recommended to rely on a moderate optimization level (`-c opt --copt=-O2`), and to better focus on a reasonable set of target-flags (`-mfma -mavx2`).

MKL, MKL-DNN, and LIBXSMM make use of CPUID-dispatch, and it is not too critical to pick for instance AVX-512 (even if AVX-512 is available on the intended production target). However, if the desired workload is bottlenecked by Eigen code paths that are not covered by the aforementioned libraries, one may be sufficiently served with Intel AVX2 instructions (`-mfma -mavx2`).

```
bazel build --config=mkl -c opt --copt=-O2 \
  --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0 --copt=-fopenmp-simd \
  --define tensorflow_xsmm=1 --copt=-mfma --copt=-mavx2 \
  //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: In the past, TensorFlow or specifically Eigen's packed math abstraction asserted an unmet condition in case of AVX-512. Therefore, one should either (1) limit the code to Intel AVX2 instructions, or (2) supply `-c opt` which 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).

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 code because of exceeding 2 GB for the size of the wheel).

```
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` (this flag does not require a user name argument but implicitly specifies the current user) to install locally for the current user rather than installing it in a system-wide fashion):

```
pip install -I /tmp/tensorflow_pkg/<package-name-build-above.whl>
```

The `-I` flag may be sufficient to reinstall the wheel even when the name of the wheel suggests that the same version is already installed. To make sure that no other bits are left, it is perhaps even better to remove all TensorFlow wheels (system-wide and user-local). In rare cases it can help to start over and to remove all locally installed Python packages (`rm -rf ~/.local`).

```
pip uninstall tensorflow
pip install /tmp/tensorflow_pkg/<package-name-build-above.whl>
```

Note: Unless a workload is symlinked and built underneath of the TensorFlow directory (for quicker development turnaround time; out of scope in this document), a wheel must be installed before it can be used to run any TensorFlow Python-code (the desired workload).

Performance Tuning

To use MKL and MKL-DNN effectively, the environment shall be setup 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. Please note that LIBXSMM uses the native TensorFlow (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 web-based client/server infrastructure TensorFlow Serving may be used to serve inference-requests.

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 can source the Intel VTune Amplifier and run:

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

Validation and Benchmarks

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 a suitable memory layout for the tensors. In general, the "channel-last" format may perform with best support (NHWC-format). If NHWC is not the default, the model (benchmark) should be adjusted.

```
git clone https://github.com/tensorflow/models.git tensorflow-models
cd /path/to/tensorflow
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 may be outdated due to helps to the Convnet Benchmarks being superseded (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
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:

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

Running Inception-v3

This section may be outdated, or data source may have moved to a different location! To run Inception-v3 inference on the ImageNet dataset, please follow the instructions 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 as well as the pretrained weights for Inception-v3. 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]
```

Development and Tests

This section focuses on LIBXSMM's integration with TensorFlow, which has two aspects: (1) sparse CNN using SpMDM routines, and (2) CNN using direct convolutions. To build and run the regression tests for the sparse routines (SpMDM):

```
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_
```

As suggested in the overview, it is still possible to exercise TensorFlow with LIBXSMM as a compute engine for a very limited set of operators (2d forward/backward direct convolutions), which may be desired for testing and development purpose. To enable LIBXSMM's convolutions, the flags `--define tensorflow_xsmm_convolutions=1` and/or `--define tensorflow_xsmm_backward_convolutions=1` are supplied in addition to `--define tensorflow_xsmm=1`. It might be even possible to `--define eigen_xsmm=1` if not implied by the afore mentioned flags. Configuring MKL-DNN (`--config=mkl`) may take precedence over LIBXSMM, hence it is omitted.

```
bazel build --config=v2 -c opt --copt=-O2 \
  --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0 --copt=-fopenmp-simd \
  --define tensorflow_xsmm_convolutions=1 \
  --define tensorflow_xsmm_backward_convolutions=1 \
  --define tensorflow_xsmm=1 --copt=-mfma --copt=-mavx2 \
  //tensorflow/tools/pip_package:build_pip_package \
  //tensorflow/core/kernels:sparse_matmul_op_test \
  //tensorflow/core/kernels:conv_ops_test
```

To build and test the CNN routines:

```
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
```

For development and experiments, one may clone a fork of the original TensorFlow repository:

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

To get nicely named JIT-kernels when profiling a workload, LIBXSMM's support for JIT-profiling can be leveraged. In case of TensorFlow, the following flags can be added 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
```