arm

Configure the Arm NN SDK build environment for TensorFlow Lite

Version 1.5

Configure the Arm NN SDK for TensorFlow Lite

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	20 November 2018	Non-Confidential	First release.
1.1	07 March 2019	Non-Confidential	Minor enhancements.
1.2	15 May 2019	Non-Confidential	Removed incorrect build parameter.
1.3	14 November 2019	Non-Confidential	Adds two extra commands for downloading the repositories and bundles.
1.4	28 February 2020	Non-Confidential	Adds instructions on including standalone dynamic backend tests. Also, updates the git checkout command in the Setup and download libraries section.
1.5	02 September 2020	Non-Confidential	Updates to reflect support for Google protobuf library 3.5.2.

Non-Confidential Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

http://www.arm.com

Contents

1 Overview

Arm NN is Arm's inference engine designed to run networks trained on popular frameworks, such as TensorFlow and Caffe, optimally on Arm IP. Arm NN now supports networks that are defined using TensorFlow Lite.

This guide shows you how to set up and configure your Arm NN build environment so you can use the TensorFlow Lite networks with Arm NN, it also shows how you can test that your build has completed successfully.

2 Before you begin

Your platform or board must have:

- An Armv7-A or Armv8-A CPU, and optionally an Arm Mali GPU using the OpenCL driver.
- At least 4GB of RAM.
- At least 1GB of free storage space.

Before you configure and build your environment, you must install the following tools on your platform or board:

- A Linux distribution.
- Git.
- SCons. Arm has tested SCons 2.4.1 (Ubuntu) and 2.5.1 (Debian). Other versions might work.
- CMake. Arm has tested CMake 3.5.1 (Ubuntu) and 3.7.2 (Debian). Other versions might work.

In this guide, we assume that you are using Ubuntu 16.04 or Debian 9.0, but these instructions should work on most Linux distributions. We test the Arm NN SDK on Ubuntu and Debian.

We estimate that you will need about 60-90 minutes to complete the instructions in this guide.

3 Setup and download libraries

First, you need to create a new directory on your platform or board that you will use for building Arm NN for TensorFlow Lite.

Open a new terminal session and enter these commands on the command line to create a new directory called `armnn-tflite`:

```
$ mkdir armnn-tflite && cd armnn-tflite
$ export BASEDIR=`pwd`
```

Some of the example commands that we use in this guide expect that the `$BASEDIR` environment variable is set correctly. So, if you use multiple terminal sessions then ensure that the variable is set correctly in each session.

Next, use these commands to download the required Git repositories and source bundles.

```
$ git clone https://github.com/Arm-software/ComputeLibrary.git
$ git clone https://github.com/Arm-software/armnn
$ wget https://dl.bintray.com/boostorg/release/1.64.0/source/boost_1_64_0.tar.bz2
$ tar xf boost_1_64_0.tar.bz2
$ git clone -b v3.5.2 https://github.com/google/protobuf.git
$ git clone https://github.com/tensorflow/tensorflow.git
cd tensorflow/
git checkout 590d6eef7e91a6a7392c8ffffb7b58f2e0c8bc6b
$ wget -O flatbuffers-1.12.0.tar.gz https://github.com/google/flatbuffers/archive/v1.12.0.tar.gz
$ tar xf flatbuffers-1.12.0.tar.gz
```

4 Build the Compute Library

The Compute Library is a machine learning library. It provides a set of functions that are optimized for both Arm CPUs and GPUs. The Compute Library is used directly by Arm NN to optimize the running of machine learning workloads on Arm CPUs and GPUs.

To build the Compute Library on your platform or board, open a terminal or bash screen and go to the Compute Library directory:

```
$ cd $BASEDIR/ComputeLibrary
```

Compile the Compute Library using SCons. To do this, change your directory to the Compute Library Git repository on your machine.

To do this for Armv7-A, enter this command:

```
$ scons extra_cxx_flags="-fPIC" benchmark_tests=0 validation_tests=0
```

Or for an Armv8-A system, enter this command:

```
$ scons arch=arm64-v8a extra_cxx_flags="-fPIC" benchmark_tests=0 validation_tests=0
```

Duration: About 15-20 minutes.

If you want to enable benchmark tests, set `benchmark_tests` to 1. If you want to enable validation tests, set `validation_tests` to 1.

You can enable support for OpenCL on an Arm Mali GPU if you have one.

If you want to support OpenCL for your Arm Mali GPU, add these arguments to the SCons command:

```
opencl=1 embed_kernels=1
```

You can enable support for NEON on the CPU. To support NEON, add this argument to your SCons command:

```
neon=1
```

5 Build the Boost library

Boost provides free peer-reviewed portable C++ source libraries that work well with the C++ Standard Library. Arm NN uses these libraries.

Now that you have downloaded Boost you need to build it. Arm has tested version 1.64 although other versions might work too. For instructions, see the Boost getting started guide.

When you build Boost, include the following flags:

```
link=static cxxflags=-fPIC --with-filesystem --with-test --with-log --with-program_options --
prefix=path/to/installation/prefix
```

For example, to build version 1.64 of the library, enter:

```
$ cd $BASEDIR/boost_1_64_0
$ ./bootstrap.sh
$ ./b2 --build-dir=$BASEDIR/boost_1_64_0/build toolset=gcc link=static cxxflags=-fPIC --with-
filesystem --with-test --with-log --with-program_options install --prefix=$BASEDIR/boost
```

Duration: About 15 minutes.

6 Build the Google protobuf library

Protocol Buffers, also known as protobuf, is Google's language-neutral, platform-neutral, extensible mechanism that is used to serialize structured data.

Build protobuf using the C++ installation instructions that you can find on the protobuf GitHub.

For example:

```
$ cd $BASEDIR/protobuf
$ git submodule update --init --recursive
$ ./autogen.sh
$ ./configure --prefix=$BASEDIR/protobuf-host
$ make
```

Arm has tested version 3.5.2. Some other versions might work.

Duration: About 15 minutes.

Next, copy the built program and its libraries and documentation to the correct locations using this command:

```
$ make install
```

7 Generate the TensorFlow protobuf library

Use the script provided by Arm NN to generate C++ sources and headers using the protobuf compiler, to provide the protobuf code that can interpret the TensorFlow data format.

To do this, change to the TensorFlow directory and enter:

```
<Arm NN directory>/armnn/scripts/generate_tensorflow_protobuf.sh <OUTPUT_DIR> <the protobuf install
directory>
```

For example:

```
$ cd $BASEDIR/tensorflow
$ ../armnn/scripts/generate_tensorflow_protobuf.sh ../tensorflow-protobuf ../protobuf-host
```

You can also use the code to interpret the TensorFlow Lite format, by using the example:

```
$ mkdir tflite
$ cd tflite
$ cp $BASEDIR/tensorflow/tensorflow/lite/schema/schema.fbs .
$ $BASEDIR/flatbuffers-1.12.0/build/flatc -c --gen-object-api --reflect-types --reflect-names
schema.fbs
```

8 Build the Google FlatBuffers library

FlatBuffers is another efficient cross-platform serialization library for C++ developed by Google for performance-critical applications. The TensorFlow Lite files are generated using FlatBuffers to serialize their TensorFlow Lite model data so Arm NN needs to use FlatBuffers to load and interpret the TensorFlow Lite files.

The FlatBuffers library is required for the Arm NN TensorFlow Lite parser. To build the FlatBuffers library, use the instructions found in the FlatBuffers Building Guide.

For example:

```
$ cd $BASEDIR/flatbuffers-1.12.0
$ CXXFLAGS="-fPIC" cmake ..
$ -DFLATBUFFERS_BUILD_FLATC=1
$ -DCMAKE_INSTALL_PREFIX:PATH=$BASEDIR/flatbuffers
```

Duration: About 5 minutes.

9 Build Arm NN

Configure the Arm NN SDK build using CMake. To do this, you will need to change your directory to the Arm NN directory and enter the following parameters to CMake:

`-DARMCOMPUTE_ROOT`	The location of your Compute Library source files directory.
`-DARMCOMPUTE_BUILD_DIR`	The location of your Compute Library build directory.
`-DBOOST_ROOT`	The directory used for Boost (see prefix flag used above).
`-DTF_GENERATED_SOURCES`	The location of your protobuf generated source files.
`-DPROTOBUF_ROOT`	The location of your protobuf install directory.
`-DBUILD_TF_LITE_PARSER`	Set this =1 to ensure the TensorFlow Lite parser is built.
`-DTF_LITE_GENERATED_PATH`	The location of the TensorFlow Lite schema directory.
`-DFLATBUFFERS_ROOT`	The root directory where FlatBuffers was installed.
`-DFLATC_DIR`	The path to the schema compiler.

For example:

```
$ cd $BASEDIR/armnn
$ mkdir build
$ cd build
$ cmake .. -DARMCOMPUTE_ROOT=$BASEDIR/ComputeLibrary \ -
DARMCOMPUTE_BUILD_DIR=$BASEDIR/ComputeLibrary/build \ -DBOOST_ROOT=$BASEDIR/boost \ -
DTF_GENERATED_SOURCES=$BASEDIR/tensorflow-protobuf \ -DPROTOBUF_ROOT=$BASEDIR/protobuf-host \ -
DBUILD_TF_LITE_PARSER=1 \ -DTF_LITE_GENERATED_PATH=$BASEDIR/tensorflow/tensorflow/lite/schema \ -
DFLATBUFFERS_ROOT=$BASEDIR/flatbuffers \ -DFLATC_DIR=$BASEDIR/flatbuffers-1.12.0/build
$ make
```

Note: Please remove the old `CMakeCache.txt` file before creating a new build. To remove the file, enter the following command:

```
rm -f CMakeCache.txt
```

Duration: About 12 minutes.

If you are supporting NEON, add this argument to the CMake command:

```
-DARMCOMPUTENEON=1
```

If you are supporting OpenCL, add this argument to the CMake command:

```
-DARMCOMPUTECL=1
```

If you want to include Arm NN reference support, add this argument to the CMake command:

```
-DARMNNREF=1
```

If you want to include standalone sample dynamic backend tests, add the following argument to enable the tests and the dynamic backend path to the CMake command:

```
-DSAMPLE_DYNAMIC_BACKEND=1 -DDYNAMIC_BACKEND_PATHS=<the location of the sample dynamic backend>
```

Also, after building Arm NN, build the standalone sample dynamic backend using the guide in the following path: `$BASEDIR/armnn/src/dynamic/README.md#standalone-dynamic-backend-build`.

The following Arm NN library files will be built in the `armnn/build` directory:

- `libarmnn.so`
- `libarmnnTfLiteParser.so`
- `libarmnnTfParser.so`
- `libarmnnUtils.a`
- `libgatordMockService.a`
- `libtimelineDecoderJson.so`
- `libtimelineDecoder.so`

10 Test your build

To check that your build of the Arm NN SDK is working correctly, you can run the unit tests. To do this, change to the Arm NN build directory and enter `./UnitTests`.

For example:

```
$ ./UnitTests
Running 424 test cases...
*** No errors detected
```

If the tests are successful, the output from the tests ends with `*** No errors detected`.

If some of the tests are unsuccessful, go back through the steps and check that all the commands have been entered correctly.

11 Next steps

Now that you have built your environment and your TensorFlow Lite parser for Arm NN, you are ready to begin programming with Arm NN, and to begin using Arm NN with TensorFlow Lite models.

Arm NN also provides a very basic example of how to use the Arm NN SDK API, here: `$BASEDIR/armnn/samples/SimpleSample.cpp`.

Other how-to guides are available for building Arm NN on other platforms and for using Arm NN with other model formats such as Caffe and TensorFlow. You can find more details in the `$BASEDIR/armnn/Readme.md` file.