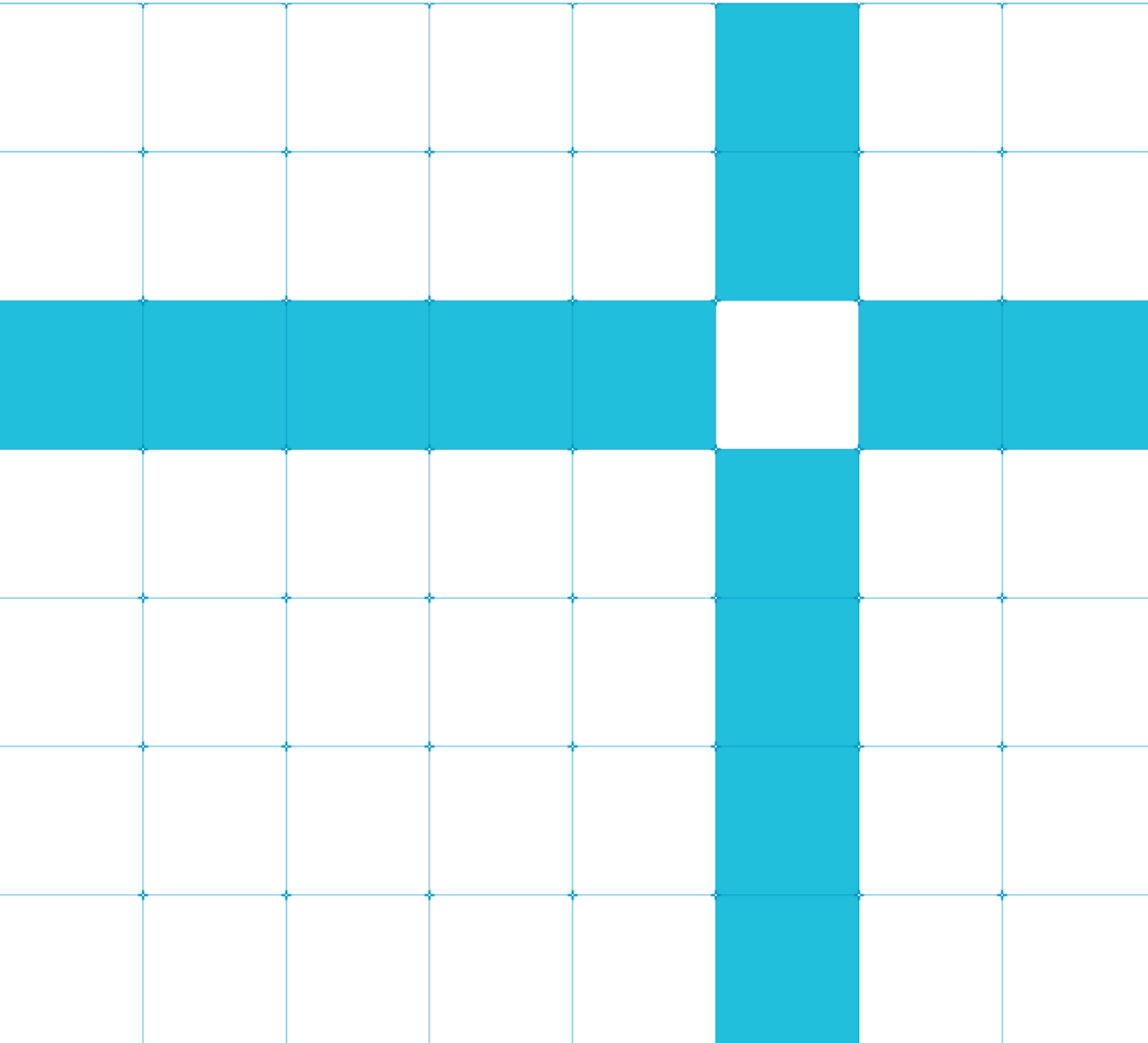
Configuring the Arm NN SDK Build Environment for ONNX

Version 1.2



Configuring the Arm NN SDK Build Environment for ONNX

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	20th November 2018	Non-Confidential	First release
1.1	25th January 2019	Non-Confidential	Minor enhancements
1.2	07th June 2019	Non-Confidential	Minor enhancements to the Before you begin section

Non-Confidential Proprietary Notice

convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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

www.arm.com

Contents

1 Overview

Arm NN is an inference engine for CPUs, GPUs, and NPUs. It bridges the gap between existing neural network frameworks and the underlying hardware IP. It enables efficient translation of existing neural network frameworks, such as TensorFlow and Caffe, allowing them to run efficiently and without modification across Arm Cortex CPUs and Arm Mali GPUs.

Arm NN now supports networks defined using the **Open Neural Network Exchange (ONNX)** format.

This guide shows you how to set up and configure your Arm NN build environment, so that you can use the ONNX format with Arm NN. This guide also shows how to 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.

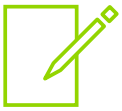
These instructions assume that you use Ubuntu 16.04 or Debian 9.0, but should work on most Linux distributions. Arm tests the Arm NN SDK on Ubuntu and Debian.

Duration: We estimate that you will need about 90-120 minutes to complete the instructions in this guide.

3 Setup and download libraries

First, you need to create a new directory on the platform or board that you will use to build your Arm NN distribution for ONNX. In this guide, we use a base directory called `armnn-onnx`. Use the following commands to create the directory:

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



Note: Some of the 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 this 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 v3.5.0 https://github.com/google/protobuf.git
```

4 Build the Compute Library

The **Compute Library** is a machine learning library that 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, go to the Compute Library directory, and follow the instructions below.

For example:

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

Compile the Compute Library using SCons. To do this, change your directory to the Compute Library git repository on your machine, and enter for **Arm7-A**:

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

Enter for **Armv8-A**:

```
$ 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 **NEON** on the CPU, and 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
```

If you want 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. Other versions may 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, are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. The ONNX files are generated using protobuf to serialize their ONNX model data. This means that Arm NN needs to use protobuf to load and interpret the ONNX files.

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 3.5.0. Other versions might work, too.

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 ONNX protobuf source files

The ONNX protobuf source files are required for the ArmNN ONNX parser. Generate these source files based on the ONNX message formats defined in the `onnx.proto` library. Use the instructions in the **ONNX GitHub**:

```
$ cd $BASEDIR
$ export ONNX_ML=1 #To clone ONNX with its ML extension
$ git clone --recursive https://github.com/onnx/onnx.git
$ unset ONNX_ML
$ cd onnx
$ export LD_LIBRARY_PATH=$BASEDIR/protobuf-host/lib:$LD_LIBRARY_PATH
$ $BASEDIR/protobuf-host/bin/protoc onnx/onnx.proto --proto_path=. --proto_path=$BASEDIR/protobuf-host/include --cpp_out $BASEDIR/onnx
```

The ONNX protobuf source files `onnx.pb.cc` and `onnx.pb.h`, are generated in the `$BASEDIR/onnx` directory ready for the Arm NN build.

8 Build Arm NN

Configure the Arm NN SDK build using CMake.

To do this, 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).
`-DPROTOBUF_ROOT`	The location of your protobuf install directory.
`-DBUILD_ONNX_PARSER`	This parameter is =1 to ensure ONNX parser is built.
`-DONNX_GENERATED_SOURCES`	The location of your ONNX generated sources.

For example:

```
$ cd $BASEDIR/armnn
$ mkdir build
$ cd build
$ cmake .. -DARMCOMPUTE_ROOT=$BASEDIR/ComputeLibrary -
DARMCOMPUTE_BUILD_DIR=$BASEDIR/ComputeLibrary/build -DBOOST_ROOT=$BASEDIR/boost -
DPROTOBUF_ROOT=$BASEDIR/protobuf-host -DBUILD_ONNX_PARSER=1 -DONNX_GENERATED_SOURCES=$BASEDIR/onnx
$ make
```

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
```

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

- `libarmnnOnnxParser.so`
- `libarmnn.so`
- `libarmnnUtils.a`

9 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 319 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, repeat the steps and check that all the commands have been entered correctly.

10 Next steps

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

Arm NN also provides a 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.