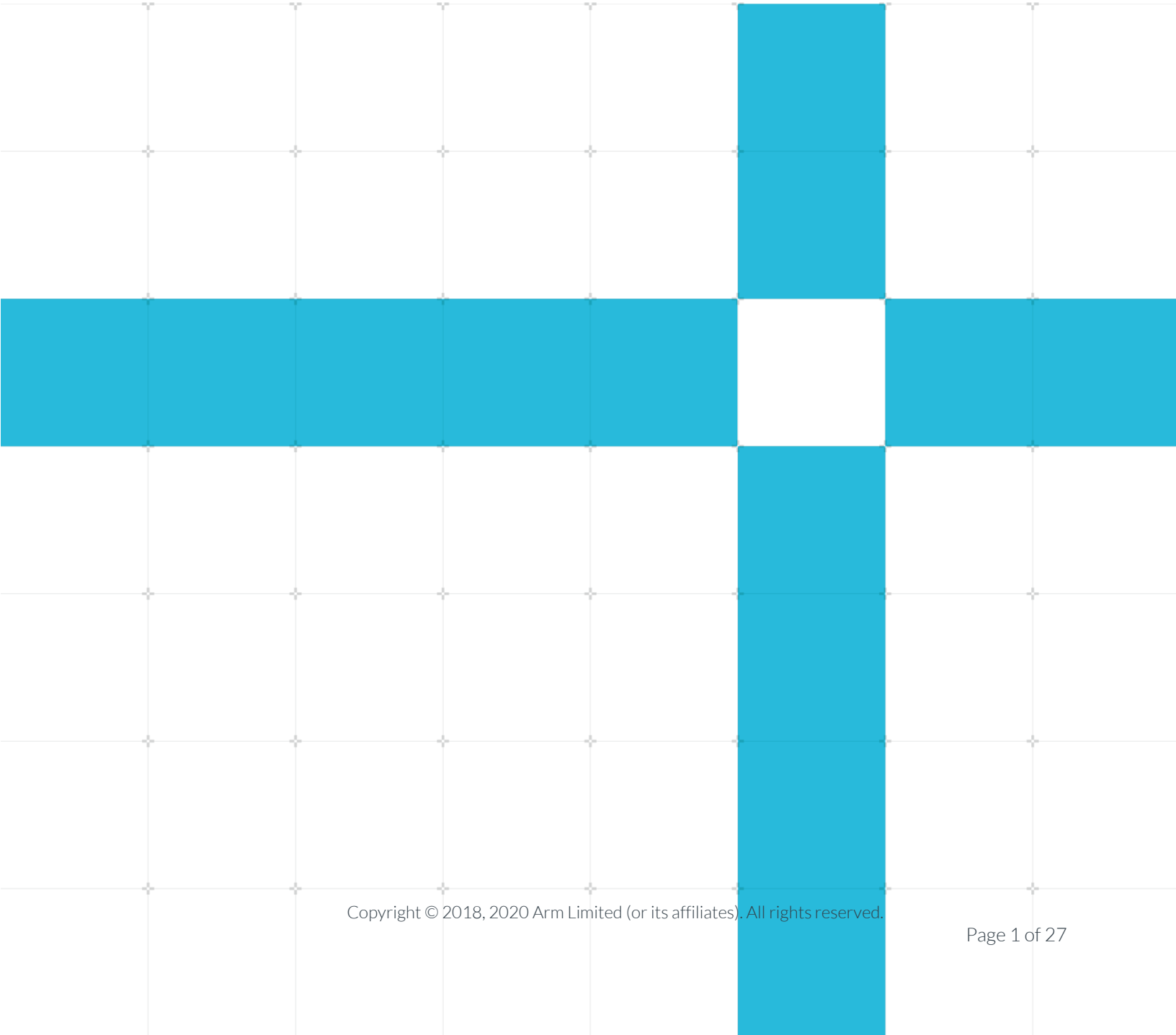
arm

Profiling AlexNet on Raspberry Pi and HiKey 960 with the Compute Library

Non-Confidential

Copyright © 2018, 2020 Arm Limited (or its affiliates).
All rights reserved.

Issue 1.2
ARM-ECM-0752397



Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Profiling AlexNet on Raspberry Pi and HiKey 960 with the Compute Library

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
1.1	13/03/2018	Non-Confidential	First release
1.2	26/06/2020	Non-Confidential	General editorial updates

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for 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.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at **http://www.arm.com/company/policies/trademarks**.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

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.

Web Address

www.arm.com

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Contents

1 Introduction 5
1.1 Product revision status 5
1.2 Intended audience 5
1.3 Conventions 5
1.3.1 Glossary 5
1.3.2 Typographical conventions 6
1.4 Feedback 7
1.4.1 Feedback on this product 7
1.4.2 Feedback on content 7

1 Overview 8

2 Before you begin 9
2.1 Set up Ubuntu MATE 9

3 NFS on Pi 11

4 Install and build the Arm Compute Library on Pi 13

5 Run the graph_alexnet application on Pi 14

6 Start Streamline gatord on Pi 16

7 Add Streamline annotations and rebuild on Pi 17

8 Build the Arm Compute Library on HiKey 960 19

9 Profile with Streamline on HiKey 960 21

10 Related information 23

11 Next steps 24

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

1 Introduction

1.1 Product revision status

The r*mpn* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rm
: Identifies the major revision of the product, for example, r1.

pn
: Identifies the minor revision or modification status of the product, for example, p2.

1.2 Intended audience

This guide is for software developers who want to know how to use Streamline on the AlexNet example application from the Compute Library.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: **https://developer.arm.com/glossary**.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

1.3.2 Typographical conventions

Convention	Use
italic	Introduces citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace` **`bold`**	Denotes language keywords when used outside example code.
`monospace underline`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
	This represents a recommendation which, if not followed, might lead to system failure or damage.
	This represents a requirement for the system that, if not followed, might result in system failure or damage.
	This represents a requirement for the system that, if not followed, will result in system failure or damage.
	This represents an important piece of information that needs your attention.
	This represents a useful tip that might make it easier, better or faster to perform a task.
	This is a reminder of something important that relates to the information you are reading.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

1.4 Feedback

Arm welcomes feedback on this product and its documentation.

1.4.1 Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

1.4.2 Feedback on content

If you have comments on content, send an email to errata@arm.com and give:

- The title [Product Name] [Document Title].
- The number [Document ID].
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader and cannot guarantee the quality of the represented document when used with any other PDF reader.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

1 Overview

This guide explains how to use Streamline to profile the AlexNet example application from the Arm Compute Library for both the Raspberry Pi and the HiKey 960 board.

Arm's Machine Learning (ML) platform, enables new applications and capabilities. It is important that you monitor your software's performance to ensure that algorithms and applications deliver excellent user experiences.

You can do this using Arm's Streamline, a performance analyzer that makes it easy to profile and optimize your software for running on Arm-based processors.

AlexNet is a convolutional neural network (CNN) that performs image feature classification from a training set of 1000 images.

The Arm Compute Library provides a way to address performance and portability challenges. It helps you to avoid rewriting applications for different target hardware and gives you confidence that lower-level functionality is optimized.

In this guide, we explain how to run the AlexNet example on two different hardware platforms. The first section covers Raspberry Pi 3 running Ubuntu MATE and the second section covers the HiKey 960 development platform running Android AOSP. The Raspberry Pi 3 contains four Arm Cortex-A53 cores, and the Hikey 960 is based on an Arm big.LITTLE processor with four Arm Cortex-A73 and four Cortex-A53 cores. You can also use Raspberry Pi 4 to do this.

Running on two different platforms demonstrates the flexibility of the Arm Compute Library. This guide also provides tips on setting up each platform, and uses Streamline to show differences in how the Arm Compute Library runs on different hardware.

Continue on through this guide to starting learning how you can profile on Raspberry Pi. Or, if you want to jump straight to how to profile for the Hikey 960, go to the section titled Install and build Compute Library on HiKey 960.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

2 Before you begin

Read this section and familiarize yourself with what you need before you start the guide.

- Raspberry Pi 2, 3 or 4 with internet access.
- A blank Micro SD card. We recommend an 8GB (minimum 6GB) Class 6 or Class 10 microSDHC card for installing the Raspberry Pi OS and storing the CNN model.
- **Arm DS-5** installed on a host PC running Windows or Linux.
- Router and ethernet cable. This is to connect to the Raspberry Pi from a host PC using SSH.

2.1 Set up Ubuntu MATE

Set up your Pi with Ubuntu MATE and prepare it for SSH access from a separate host machine. The general steps for this are:

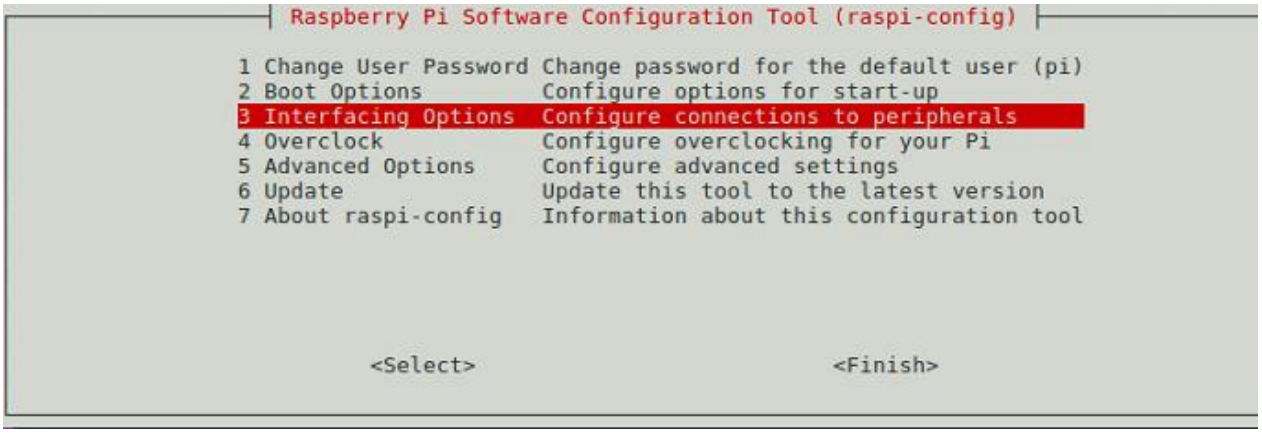
- Download the Ubuntu MATE 16.04.02 image file.
- Uncompress it using unxz.
- Write the image to the MicroSD card, we use dd on Linux but there are many ways.
- Insert the MicroSD card in a Raspberry Pi 3 and start it up.

Once the new system starts up, go through the configuration steps and connect to the network using a wired or wireless connection from your host machine.

Next, enable SSH by entering this on the command line:

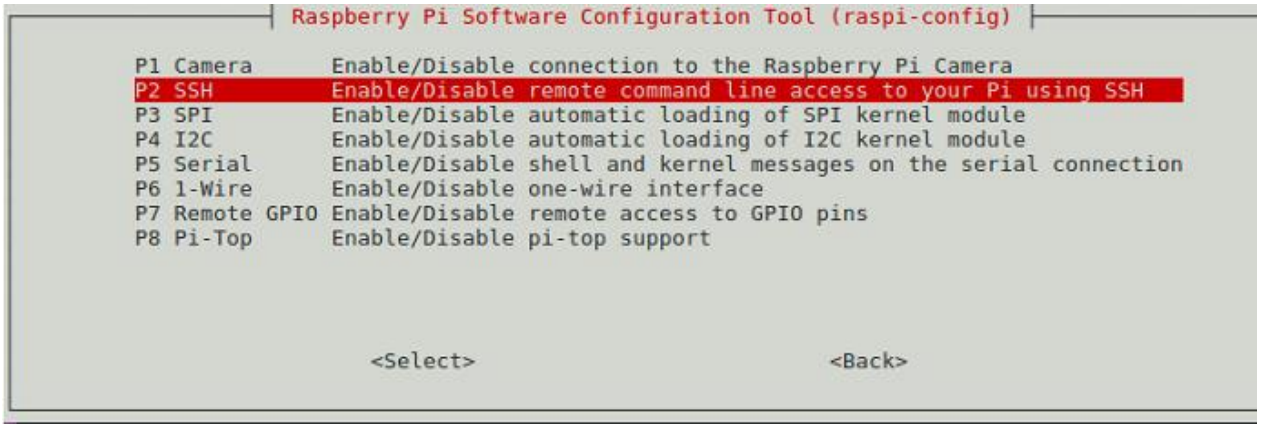
```
$ sudo raspi-config
```

Then select Interfacing Options:



And then select SSH:

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.



With SSH enabled, use the `ifconfig command` to see the IP address of the Pi:

```
$ ifconfig wlan0
 wlan0     Link encap:Ethernet  HWaddr b8:27:eb:22:ab:26
           inet addr:192.168.0.121  Bcast:192.168.0.255  Mask:255.255.255.0
```

And then check that you can SSH to it from your host machine.

To save time, you can set up SSH with no password. For example:

```
$ ssh-copy-id 192.168.0.121
```

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

3 Network File System on Pi

The easiest way to run the example on a Raspberry Pi 3 is to compile it natively on the Pi. This method is slow but avoids the possibility of missing or mismatched libraries on the target system. The downside of native compiling is that the host machine does not have the binaries for Streamline to analyze. Although the scp command can be used to copy them back to the host, this method is tedious because it must be done every time the software changes.

Instead of scp, the Network File System (NFS) can be used to share a directory on the Pi which can be mounted from the host machine. This way, the files are always in sync and Streamline is much easier to use. The steps to set up NFS on the Raspberry Pi are:

1. Install the NFS client and server files:

```
$ sudo apt-get install nfs-common nfs-server
```

2. Edit the `/etc/exports` configuration file, using `sudo`, to set up the directory to share. You can share the home directory of the Pi since it contains everything for the example. Because of dynamic IP addresses, use a range of addresses so any machine on the local network can mount the exported directory.

```
# /etc/exports: the access control list for filesystems which may be exported
#                    to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
# /home/<user-name> 192.168.0.0/255.255.255.0(rw,sync)
```

3. Update the exports using the `-ra` option the confirm the file system has been exported:

```
$ sudo exportfs -ra $ sudo exportfs /home/<user-name> 192.168.0.0/255.255.255.0
```

4. On the host machine, make sure the exported directory is visible and then mount it:

```
$ showmount -e 192.168.0.121
Export list for 192.168.0.121:
/home/<user-name> 192.168.0.0/255.255.255.0 $ sudo mount -t nfs
192.168.0.121:/home/<user-name> /mnt
```

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

The home directory of the Pi is now accessible under `/mnt` on the host machine. You can access files on the Pi here or copy new files to the Pi just by copying them to `/mnt`.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

4 Install and build the Arm Compute Library on Pi

The Arm Compute Library can be compiled natively on the Pi. It takes some time, but this is the easiest way. When building, `opencl=1` must be specified for the example application to be built, even though OpenCL is not going to be used on the Pi. Building the debug version, by specifying `debug=1`, provides useful information during the performance analysis.

To install and build the Arm Compute Library, enter the following commands:

```
$ sudo apt-get install git scons -y
$ git clone https://github.com/Arm-software/ComputeLibrary.git
$ cd ComputeLibrary
$ scons Werror=1 debug=1 asserts=0 neon=1 opencl=1 build=native -j2
```

This is a good time to take a short break while the Arm Compute Library compiles.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

5 Run the graph_alexnet application on Pi

1. First, download the zip file of AlexNet with the AlexNet model, input images, and labels onto the Pi. Create a new directory and unzip the file, as shown here:

```
$ cd ; mkdir assets_alexnet
$ unzip compute_library_alexnet.zip -d assets_alexnet
```

The AlexNet example performs a simple image classification task.

On Linux, the libraries are compiled as `.so` files, so the `LD_LIBRARY_PATH` environment variable is used to point to the .so files from the Arm Compute Library.

2. Next, run the application to ensure that the test passes. To do this, enter the following commands:

```
$ export LD_LIBRARY_PATH=$HOME/ComputeLibrary/build/
$ export PATH_ASSETS=$HOME/assets_alexnet
$ time ./build/examples/graph_alexnet 0 $PATH_ASSETS $PATH_ASSETS/go_kart.ppm
$PATH_ASSETS/labels.txt


Can't load libOpenCL.so: libOpenCL.so: cannot open shared object file: No such file or
directory
Can't load libGLES_mali.so: libGLES_mali.so: cannot open shared object file: No such
file or directory
Can't load libmali.so: libmali.so: cannot open shared object file: No such file or
directory
Couldn't find any OpenCL library


. ./build/examples/graph_alexnet [GRAPH][08-02-2018 06:39:39][INFO] Instantiating
NEConvolutionLayer


[GRAPH][08-02-2018 06:39:40][INFO] Data Type: F32 Input Shape: 227x227x3 Weights shape:
11x11x3x96 Biases Shape: 96 Output Shape: 55x55x96 PadStrideInfo: 4,4;0,0,0,0 Groups: 1
WeightsInfo: 0;0;0,0
[GRAPH][08-02-2018 06:39:53][INFO] Instantiating NESoftmaxLayer Data Type: F32 Input
shape: 1000 Output shape: 1000


---------- Top 5 predictions ----------
0.9736 - [id = 573], n03444034 go-kart
0.0118 - [id = 518], n03127747 crash helmet
0.0108 - [id = 751], n04037443 racer, race car, racing car
0.0022 - [id = 817], n04285008 sports car, sport car
```

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

```
0.0006 - [id = 670], n03791053 motor scooter, scooter


Test passed


real 0m20.017s
user 0m21.930s
sys 0m1.460s
```

The application prints some messages about missing libraries for OpenCL and OpenGL, debug messages that only occur for debug builds, and finally the predictions at the end. The user time is more than the real time which means not a lot of parallel computation is happening. Streamline shows more about what is happening.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

6 Start Streamline gatord on Pi

The Streamline gator daemon is a user space application that can be built using the Android NDK and the gator driver is a Linux kernel driver.

To profile the example, you need to run `gatord` using `sudo`. It can be run without `sudo`, but the sample-based profiling information is not available and this is one of the interesting things to see for this application. The `gator` kernel module is not needed for this tutorial as `gatord` with `sudo` provides profiling with the Linux perf API. Use NFS to copy the 32-bit `gatord` from DS-5 directly to the Pi using:

```
$ cp $DS5_HOME/sw/streamline/bin/arm/gatord /mnt
```

Then, `ssh` to the Pi and start the `gator` daemon:

```
$ sudo ./gatord
```

Now that everything is set up, the next step is to look at the performance details of the graph_alexnet application.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

7 Add Streamline annotations and rebuild on Pi

First, add a few annotations to the application so that we can see the phases where it spends time. To do this, edit the file `examples/graph_alexnet.cpp` or you can use the version of the file in this download.

Add the include file `streamline_annotate.h` and call the `ANNOTATE_SETUP` macro at the start of the program.

```
int main(int argc, char **argv)
{
     int st;
     ANNOTATE_SETUP;
     st= arm_compute::utils::run_example(argc, argv);
     ANNOTATE_MARKER_STR("main complete");
     return (st);
}
```

Next, annotate the `do_setup()` and the `do_run()` with the `ANNOTATE_CHANNEL_COLOR()` so the time spent in these is clearly visible. The modified `graph_alexnet.cpp` is attached as a reference.

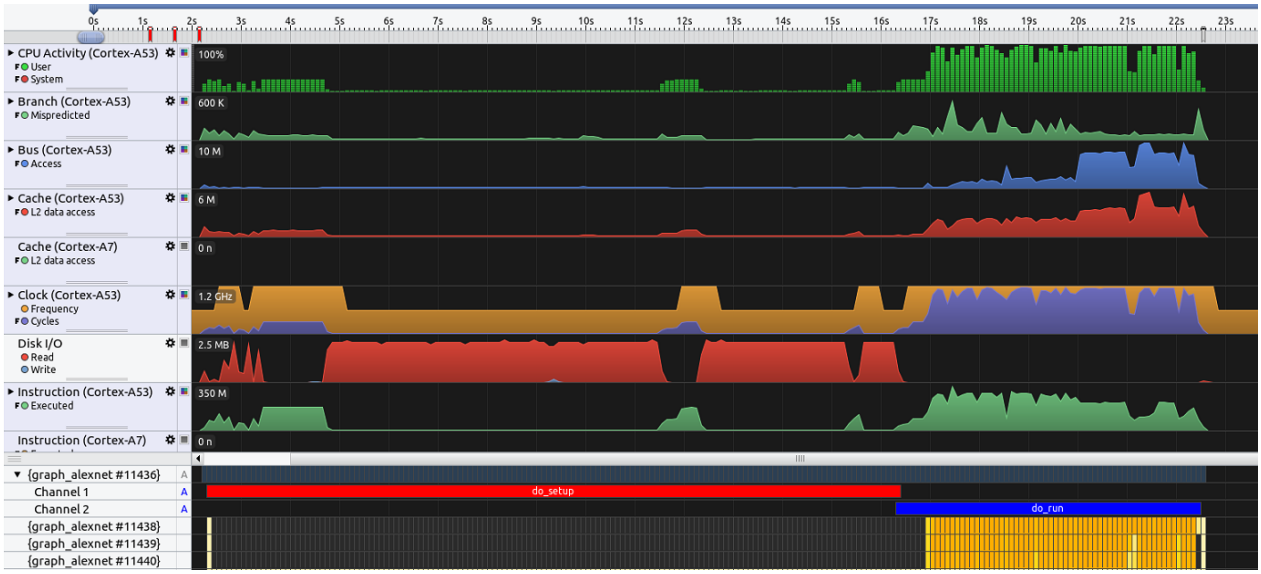
Note: The download also contains an updated `SConscript` file that shows you how to include Streamline annotations in the application.

Here, you are going to rebuild using the `scons` command. Only the example needs to be recompiled:

```
$ scons Werror=1 debug=1 asserts=0 neon=1 opencl=1 build=native -j2
```

Next, connect Streamline, start a capture, run the example, and stop the capture. For information and instructions on the connection process, read this article from the sentence starting with "Click the eye-ball, browse for a target" within the section Gator driver and daemon.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.



Some immediate observations on the capture:

- Setup (do_setup) takes up most of the application's length and lasts around 14 seconds. Most of the time it is spent performing >200Mb of Disk I/O.
- Once the run (do_run) starts there are 4 threads to fully utilize the Cortex-A53x4.
- As expected, much of the time for each thread is spent doing matrix multiply operations.

Samples	% Samples	I	Line	Source File: /mnt/ComputeLibrary/src/core/NEON/kernels/NEGEMMMatrixMultiplyKernel.cpp
			255	
			256	`auto vec_a_end_addr = vec_a + num_elems_vec_a;`
9	0.47%		257	`for(; vec_a <= (vec_a_end_addr - 4);)`
			258	`{`
6	0.31%		259	`float32x2_t a0l = vld1_f32(vec_a);`
			260	
191	9.95%		261	`float32x4_t b00 = vld1q_f32(matrix_b + 0 + 0 * in_b_stride);`
10	0.52%		262	`float32x4_t b01 = vld1q_f32(matrix_b + 4 + 0 * in_b_stride);`
5	0.26%		263	`float32x4_t b02 = vld1q_f32(matrix_b + 8 + 0 * in_b_stride);`
122	6.36%		264	`float32x4_t b03 = vld1q_f32(matrix_b + 12 + 0 * in_b_stride);`
			265	
208	10.84%		266	`float32x4_t b10 = vld1q_f32(matrix_b + 0 + 1 * in_b_stride);`
9	0.47%		267	`float32x4_t b11 = vld1q_f32(matrix_b + 4 + 1 * in_b_stride);`
10	0.52%		268	`float32x4_t b12 = vld1q_f32(matrix_b + 8 + 1 * in_b_stride);`
162	8.44%		269	`float32x4_t b13 = vld1q_f32(matrix_b + 12 + 1 * in_b_stride);`
			270	

Having now profiled an application on the Raspberry Pi, the remainder of this guide does the same on the HiKey 960 platform running Android.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

8 Build the Arm Compute Library on HiKey 960

Running the AlexNet example on the HiKey 960 with Android provides a comparison to the Raspberry Pi.

This article on **Profiling Android with the HiKey 960** provides information on how to get Android running on the HiKey 960. The only change is the latest **AOSP release on Linaro** is newer than it was at the time the article was written.

1. **Install Streamline** and gatord. As with the Pi, using gatord with the perf API is recommended. Gator can be compiled **from GitHub** using NDK or copied from the DS-5 directory as with the Pi, as shown here. To do this, copy Gatord from the DS-5 directory. Note that the path is different from the Pi, for 64-bit.

```
$ sudo adb remount<
$ sudo adb push $DS5_HOME/sw/streamline/bin/arm64/gatord /system
$ sudo adb root
$ sudo adb shell
# cd /system
# chmod +x gatord
# ./gatord &
```

2. Install the Android NDK. This can be added to Android SDK or used standalone. For Android, the Arm Compute Library is cross compiled using the Android NDK. Once compiled, the examples are copied to the HiKey 960 using the `adb push` command.

3. The Arm Compute Library should be compiled with Clang as gcc is no longer supported. Setup Clang by generating a standalone toolchain from the NDK with these commands:

```
$ export NDK=/home/<user-name>/Android/Sdk/ndk-bundle
$ $NDK/build/tools/make_standalone_toolchain.py --arch arm64 --api 23 --stl gnustl --
install-dir /ml/cl/toolchains/aarch64
```

This creates a standalone toolchain in the specified installation directory.

4. Next, add the `bin/` directory of the toolchain to the `PATH` environment variable and compile the Arm Compute Library for Android:

```
$ export PATH=/ml/cl/toolchains/aarch64/bin:$PATH
$ git clone https://github.com/Arm-software/ComputeLibrary.git
$ cd ComputeLibrary
$ CXX=clang++ CC=clang scons Werror=0 debug=1 asserts=0 neon=1 opencl=1 os=android
arch=arm64-v8a -j8
```

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Note: We tested with NDK version r16b which is newer than the r14 version that the Arm Compute Library was tested with. We had only one warning which blocked the compilation with `Werror=1`, but we submitted a patch which has already been incorporated into the Arm Compute Library.

5. Once the build is complete, copy the examples to the HiKey 960 using `adb`:

```
$ sudo adb push build/examples/graph_alexnet /data/local/tmp
```

6. Copy the data needed by the example. Adjust the path to the downloaded .zip file as needed, unzip it on the host machine, and copy the data to the HiKey 960 using `adb`:

```
$ unzip compute_library_alexnet.zip -d assets_alexnet
$ sudo ./adb push ./assets_alexnet /data/local/tmp
```

7. The Android version of the Arm Compute Library and the examples are statically linked for all libraries except `libOpenCL.so`. This means that unlike the Raspberry Pi, the `.so` files in the `build/` directory such as `libarm_compute.so` and `libarm_compute_core.so` are not needed on the target system, but the OpenCL library is required. To run the example, create a `libOpenCL.so` as shown here:

```
$ cp /system/lib64/egl/libGLES_mali.so /data/local/tmp/libOpenCl.so
$ export LD_LIBRARY_PATH=/data/local/tmp
```

If this is not done correctly, the resulting error is:

```
CANNOT LINK EXECUTABLE "./graph_alexnet": library "libOpenCL.so" not found
```

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

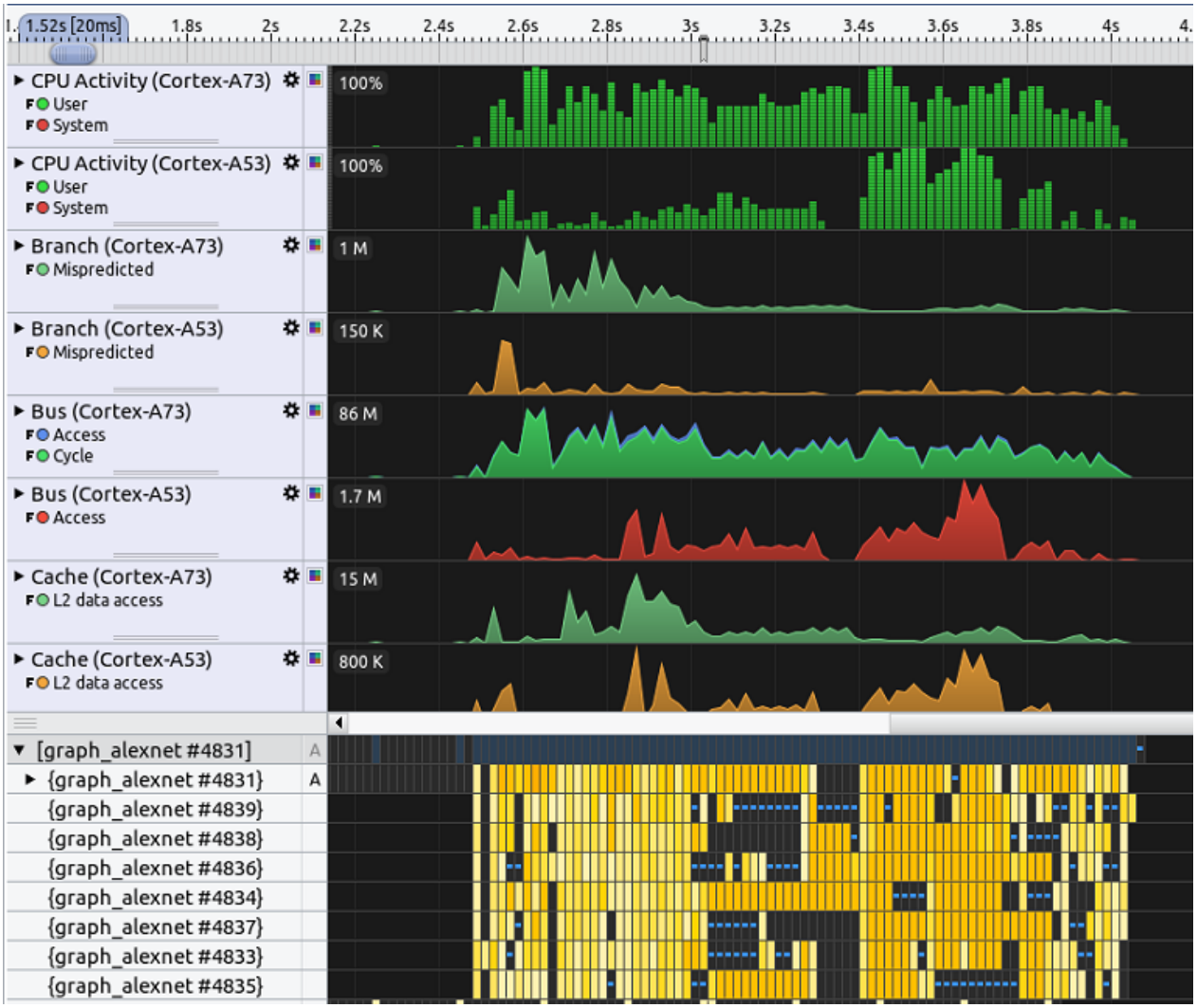
9 Profile with Streamline on HiKey 960

Before checking with Streamline, time the application and see how long it takes compared to the Pi and verify the results as the same as on the Pi:

```
$ /bin/time ./graph_alexnet 0 ./assets/ ./assets/go_kart.ppm ./assets/labels.txt

---------- Top 5 predictions ----------

0.9736 - [id = 573], n03444034 go-kart
0.0118 - [id = 518], n03127747 crash helmet
0.0108 - [id = 751], n04037443 racer, race car, racing car
0.0022 - [id = 817], n04285008 sports car, sport car
0.0006 - [id = 670], n03791053 motor scooter, scooter

Test passed
real 1.315090
user 4.900000
sys 0.188000
```

As expected, the application runs much faster on the HiKey 960 board. It is over a second faster than the Pi to complete the entire run. This is because the application is taking advantage of the multiprocessor as shown by user time lasting longer than real time.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.



Some observations when compared with the Raspberry Pi are:

- Setup is very fast, reading the model files takes very little time, less than ¼ of a second.
- Once the run starts there are 8 threads used which fully utilize the dual-cluster Cortex-A53x4, Cortex-A73x4 design.
- A large amount of time for each thread is spent doing matrix multiply operations, which are the same as the Raspberry Pi.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

10 Related information

Here are some resources related to the material in this guide:

- **Arm Compute Library**
- Machine Learning on Arm
- **Run AlexNet on Raspberry Pi with Arm Compute Library**
- **Streamline**

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

11 Next steps

As we have seen, the Arm Compute Library can be profiled with Arm Streamline to study the performance of Machine Learning and Computer Vision applications.

This guide demonstrates how you can use Streamline to profile an example application. This is from the Arm Compute Library for the AlexNet Convolutional Neural Network on two different hardware platforms with different operating systems. The next guide you can read is Install and build Compute Library on HiKey 960.

Going forward, you can apply the methods shown here to use Streamline to profile your own machine learning applications to help you optimize their performance for running on Arm-based systems. You can use Streamline to report further information such as memory used, disk I/O, threads created, and sample-based function profiling. Furthermore, the Arm Compute Library makes use of the available Neon hardware to perform efficient image inference.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.