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

Document number: ARM-ECM-0752397 Version: 1.0

Date of Issue: 13/03/2018

Abstract

This tutorial shows how to use Streamline on the AlexNet example application from the Compute Library which was demonstrated in the article, Running AlexNet on Raspberry Pi with Compute Library. The example is run on two different hardware platforms: HiKey 960 running Android AOSP and Raspberry Pi 3 running Ubuntu MATE. The tutorial shows the flexibility of the Compute Library, provides some tips on setting up each platform, and uses Streamline to show differences in how the Compute Library runs on different hardware.

Keywords

Machine Learning, Computer Vision, Streamline, Raspberry Pi, learning Arm, Compute Library.

Contents

1 Overview

Project Trillium, 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.

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

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

The Compute Library provides a way to address performance and portability challenges. It helps you to avoid re-writing 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.
Running on two different platforms demonstrates the flexibility of the Compute Library. This guide also provides tips on setting up each platform, and uses Streamline to show differences in how the 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.**

2 Setup your Pi

Prerequisites

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

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:

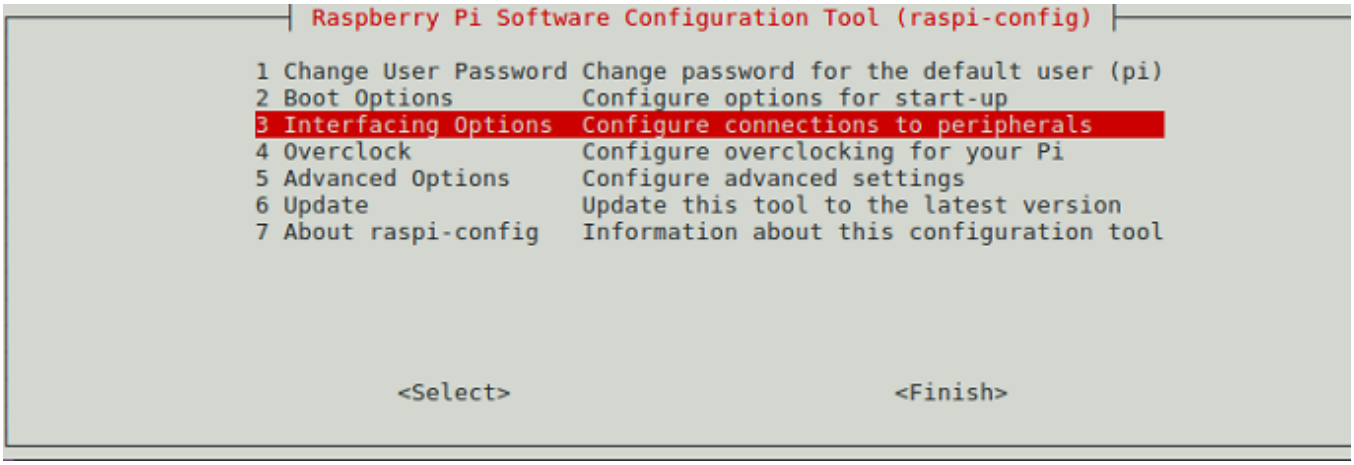
1. Download the Ubuntu MATE 16.04.02 image file.
2. Uncompress it using unxz.
3. Write the image to the MicroSD card, we use dd on Linux but there are many ways.
4. Insert the MicroSD card in a Raspberry Pi 3 and start it up.

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.

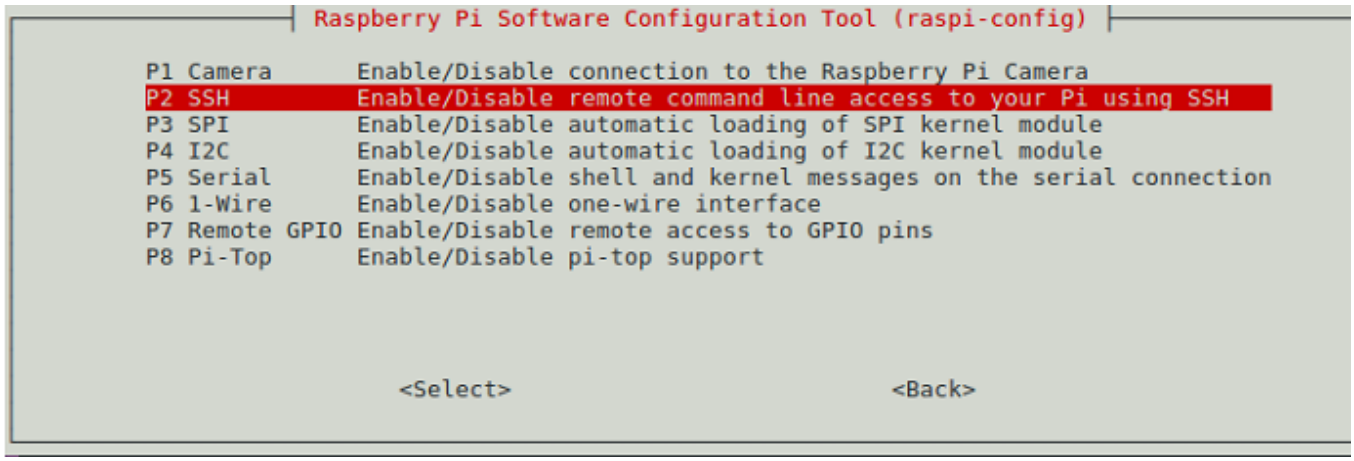
Next, enable SSH by entering this on the command line:

```
$ sudo raspi-config
```

Then select Interfacing Options:



And then select SSH:



With ssh enabled, use ifconfig to see the IP address of the Pi and check you can ssh to it from your host machine.

```
$ 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 setup SSH with no password. For example:

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

3 NFS 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 needs to be done every time the software changes.

Instead of `scp`, 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 setup 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 setup the directory to share. You can share the home directory of the Pi since it will contain 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/jasand01  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
```

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.

4 Build the Compute Library on Pi

The 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 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 break while the Compute Library compiles.

5 Run the graph_alexnet application on Pi

First, download the ZIP file 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 Compute Library.

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
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 will show more about what is happening.

6 Starting Streamline gatord on Pi

The Streamline gator daemon is a userspace 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 will not be 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 will provide profiling via 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.

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` o 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);
```

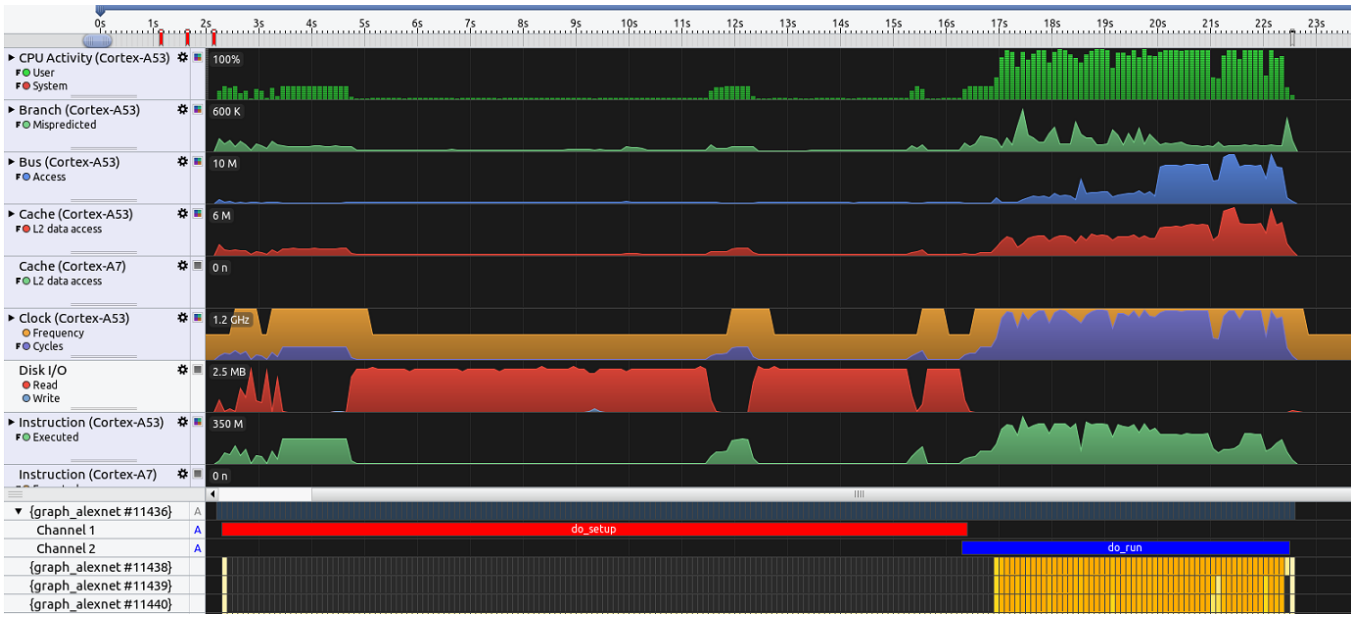
```
}
```

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 in the download can be used as a reference.
Note: **This download** 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:

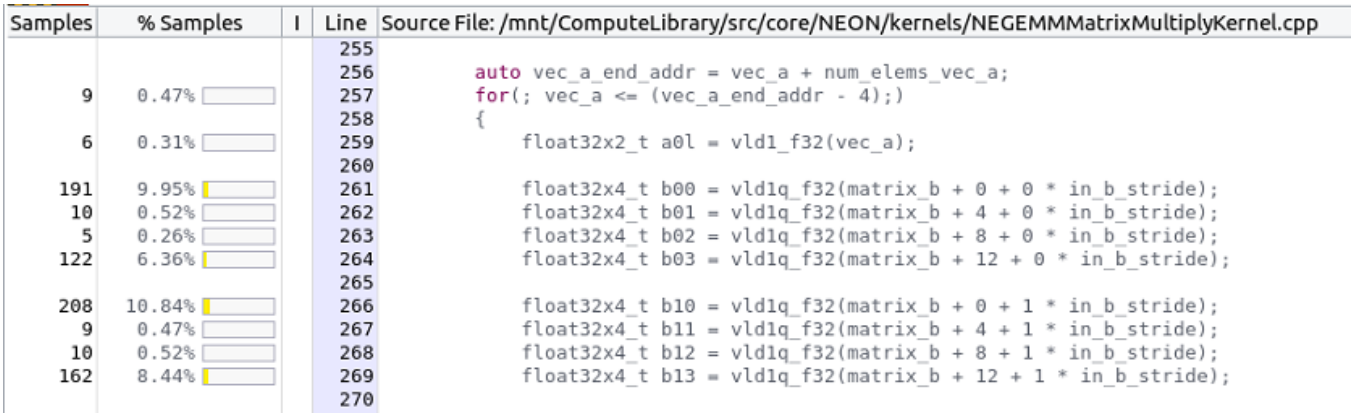
```
$ scons Werror=1 debug=1 asserts=0 neon=1 opencl=1 build=native -j2
```

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.**



Some immediate observations about 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 >200 Mb 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.

8 Build the Compute Library for 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 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 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 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 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
```

   Note: We tested with NDK version r16b which is newer than the r14 version that the 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 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 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 will be:

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

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:

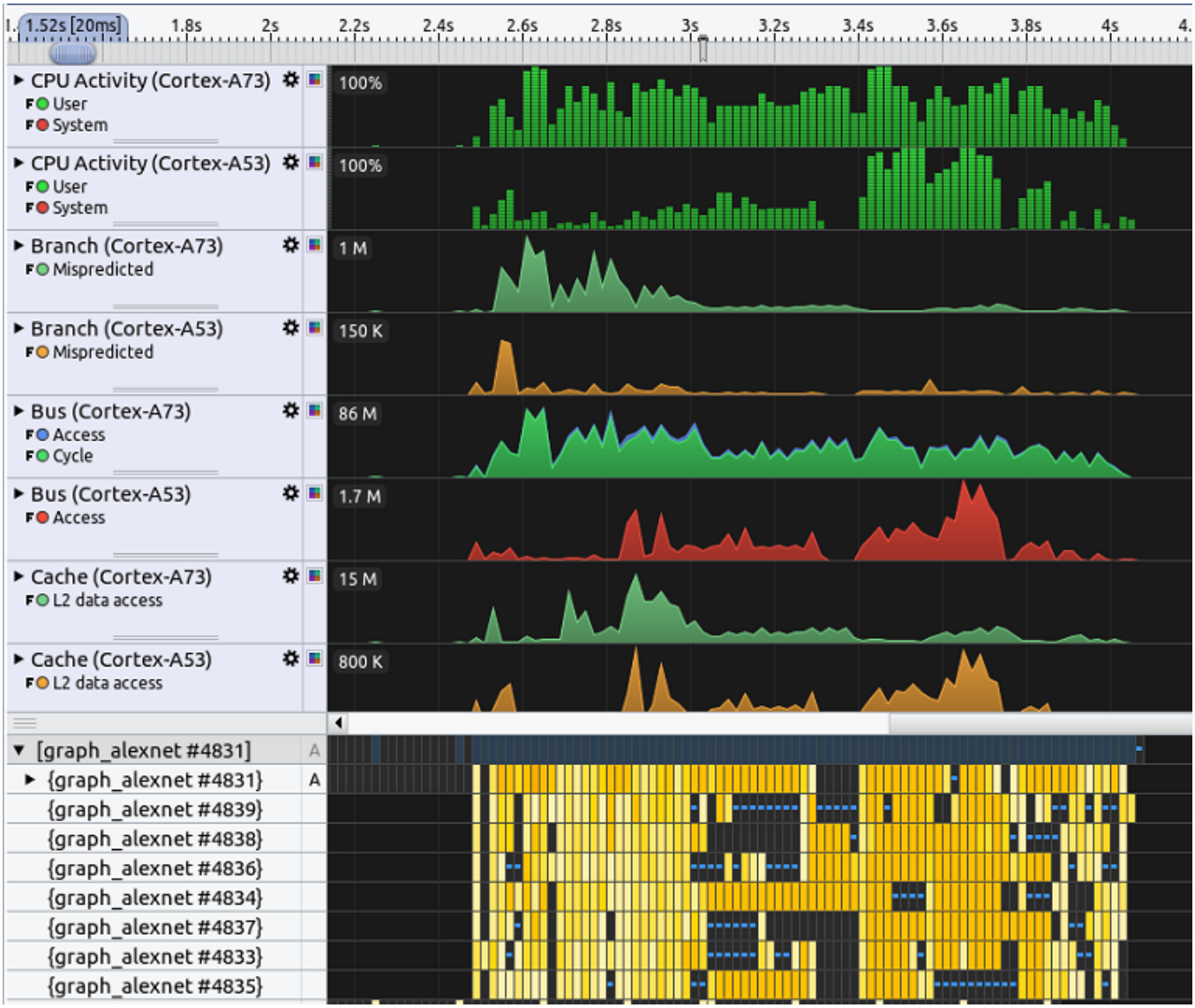
```
$ /bin/time ./graph_alexnet 0 ./assets/ ./assets/go_kart.ppm
./assets/labels.txt

---------- 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 multicore processing as shown by user time lasting longer than real time.



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 is the same as the Raspberry Pi.

10 Next steps

As we've seen, the 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 from the Compute Library for the AlexNet Convolutional Neural Network on two different hardware platforms with different operating systems.

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 Compute Library makes use of the available Neon hardware to perform efficient image inference.