Analysing Machine Learning Inference with Arm Performance Tools

Stephen Barton

Product Manager

Arm

#ArmTechCon

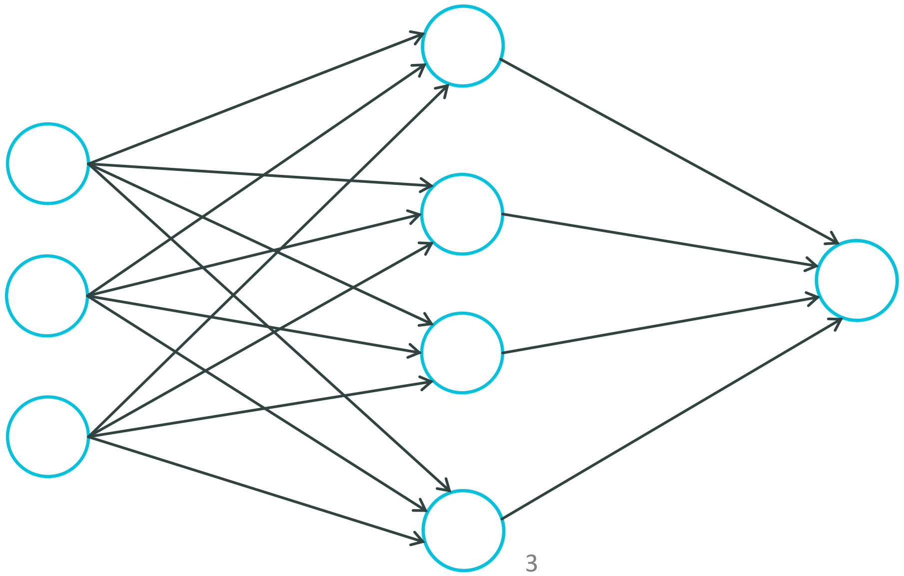
arm TechCon

Agenda

- Brief look at what we mean when we talk about machine learning and how it is related to mobile platforms
- A look at common ML related performance problems
- Current solutions that exist
- A look at what is coming in the future to tackle this problem

What do We Mean by Machine Learning?

- Training a computer to predict an output based on an input
- This is done by giving the computer a data set of sample inputs and a set of corresponding sample outputs
- From this it can predict an output given a completely new input - the larger the data the computer was trained with, the more accurate the output
- This is a very computationally expensive operation



Use Cases

Quvium

Elli Q

Kissfly Drones

Amazon Alexa

Huawei Mate 9

Buddyguard Flare

Training vs Inference

Training

- Training is done with a large data set and is usually still done offline
- The more valid data you are using, the more well-trained and well-tuned your network can become
- For each piece of data, all the model parameters are adjusted until the model is accurate enough for use

Inference

- Inferencing is about sending new data through the model to retrieve an output
- Mobile technology is at a state where inferencing can be done on a mobile device
- This process is only performed once per new piece of data and involves large numbers of multiply-add operations

Current Performance Problems with Machine Learning

Machine learning (ML) is very compute intensive

ML also is very bandwidth intensive

ML workloads can run more efficiently on different compute resources – CPU, GPU or NPU

It is important to select the appropriate network for the chosen use case

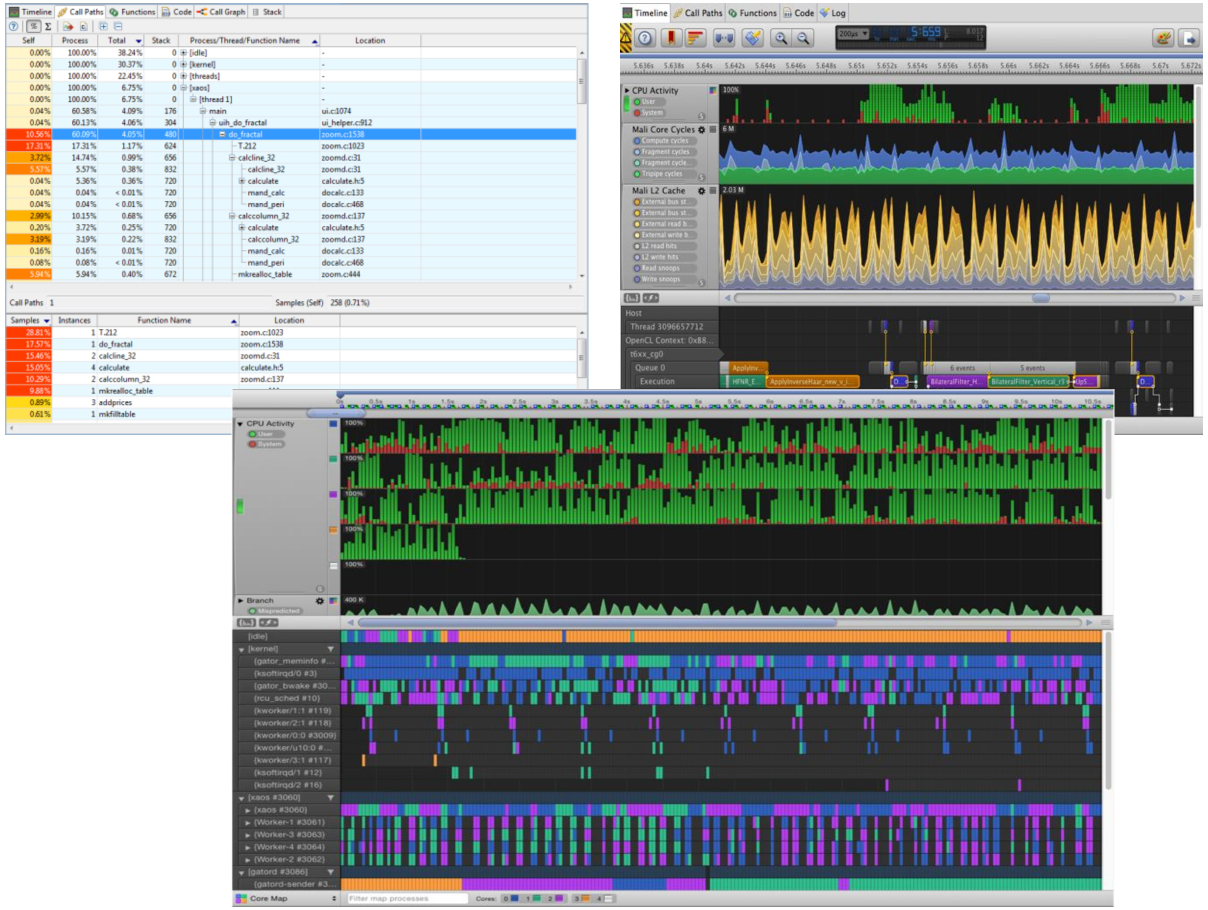
Arm DS Streamline Performance Analyzer

Speed up your code

- Monitor CPU and GPU cache usage
- Break performance down by function
- View it alongside the disassembly

OpenCL™ Visualizer

- Visualization of OpenCL dependencies, helping you to balance resources between GPU and CPU better than ever



Drill down to the Source Code

- Break performance down by function
- View it alongside the disassembly

#ArmTechCon

arm TechCon

Timeline: Heat Map

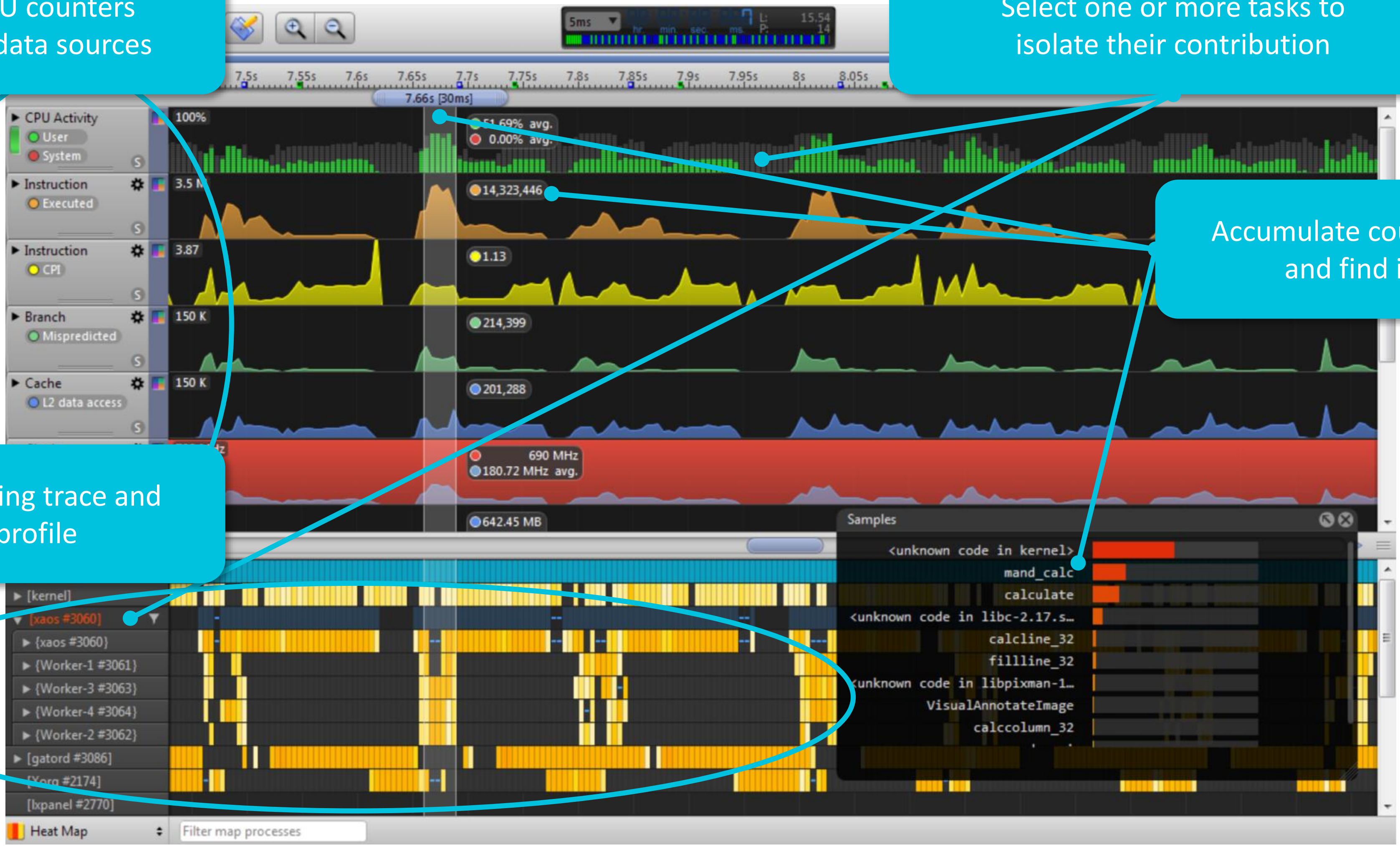
Identify hotspots and system bottlenecks at a glance

Select from CPU/GPU counters
OS level and custom data sources

Select one or more tasks to
isolate their contribution

Accumulate counters, measure time
and find instant hotspots

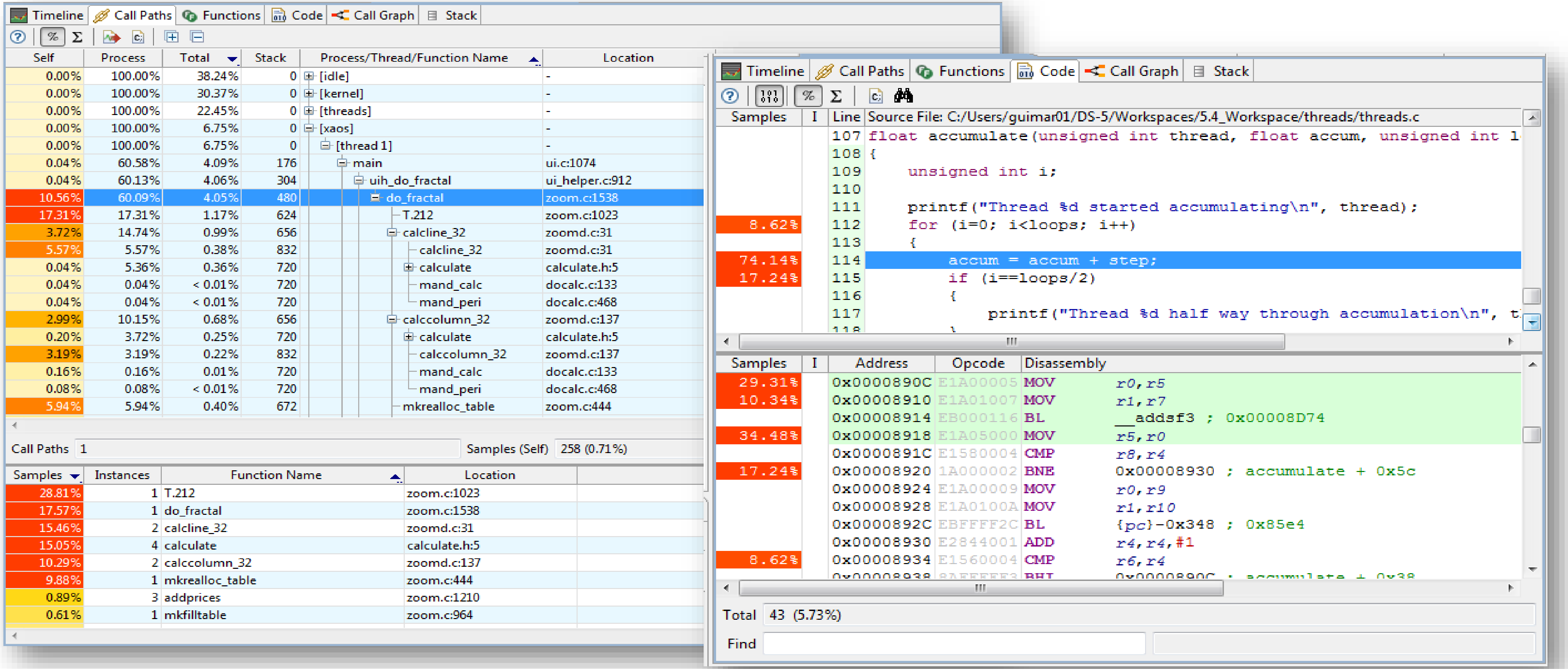
Combined task switching trace and
sample-based profile



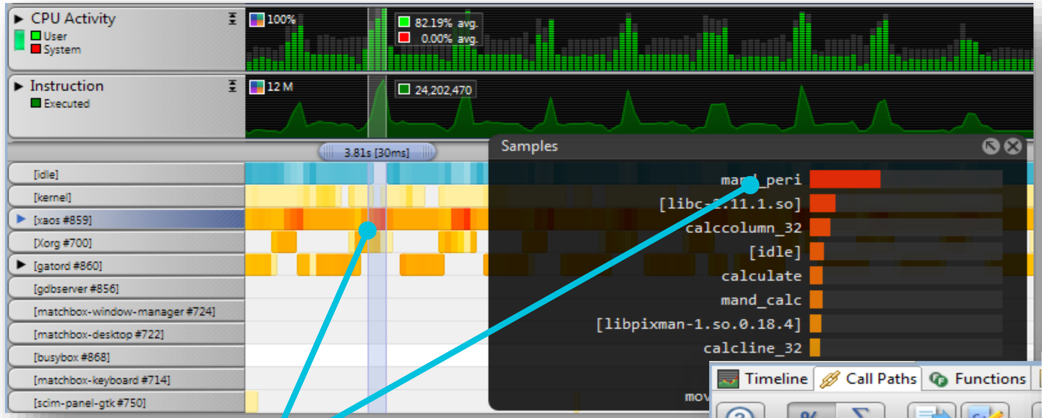
#ArmTechCon

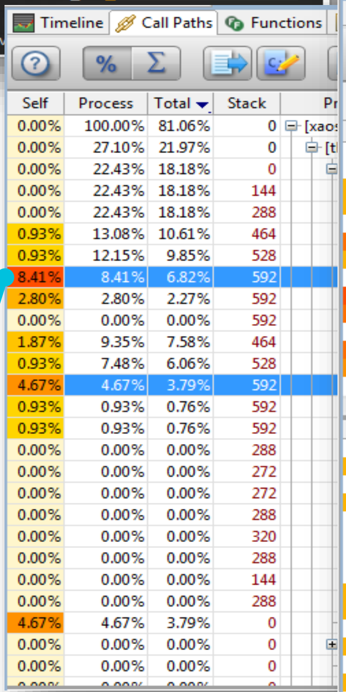
Profiling Reports

Analysis of call paths, source code, and generated assembly

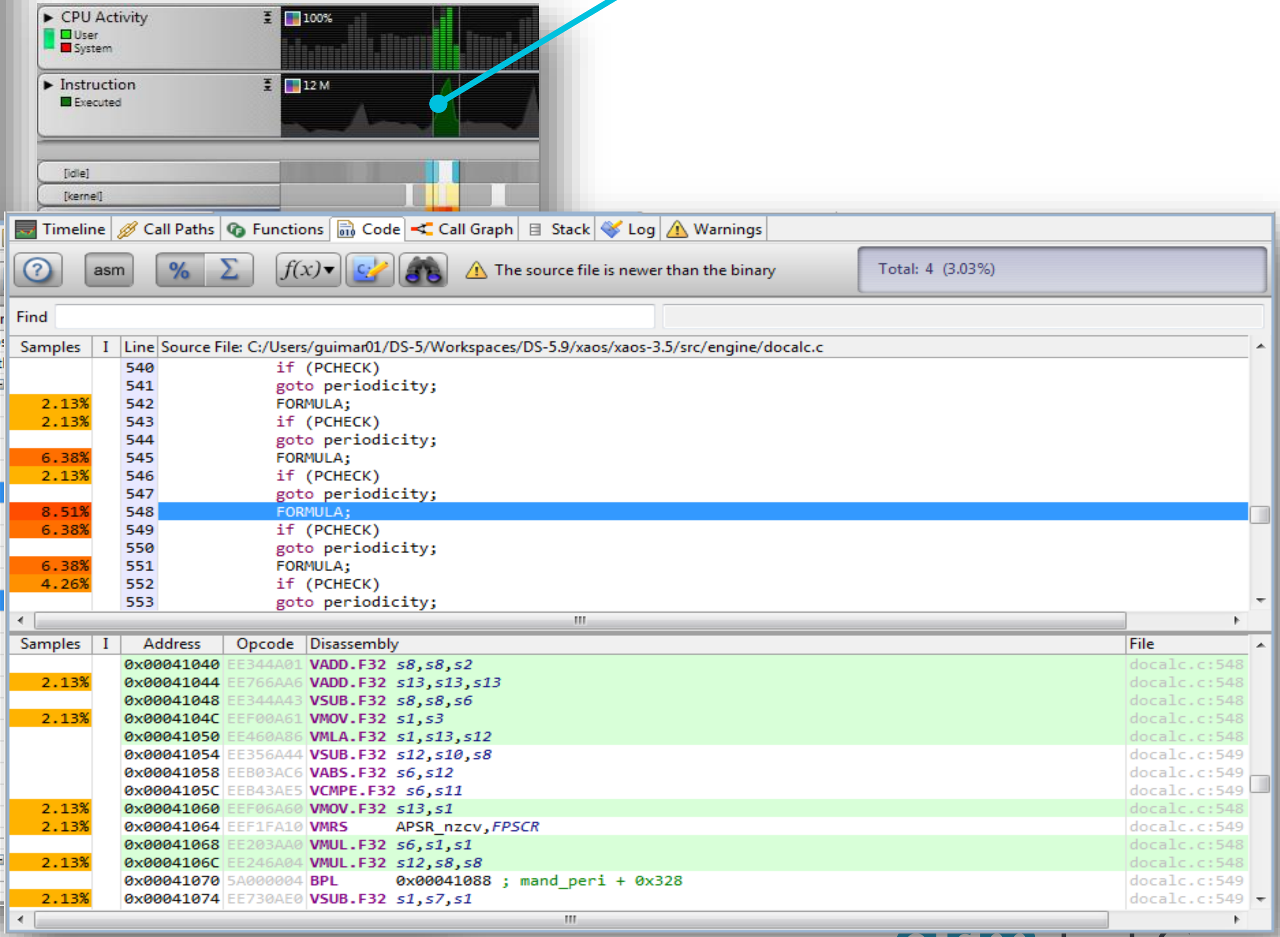


Top-Down Software Profiling





Filter timeline data to generate focused software profile reports



Quickly identify instant hotspots

Click on the function name
to go to source code level profile

#ArmTechCon

Traditional Streamline Use Cases

- Highlight peaks of CPU activity
- Investigate further through process level all the way down to function call level
- Line by line account of time spent on each line of source code in the CPU

A CPU bound case

- Determine whether you are Fragment bound or Vertex Bound
- Analyse further to see which pipeline or functional unit you are bound by

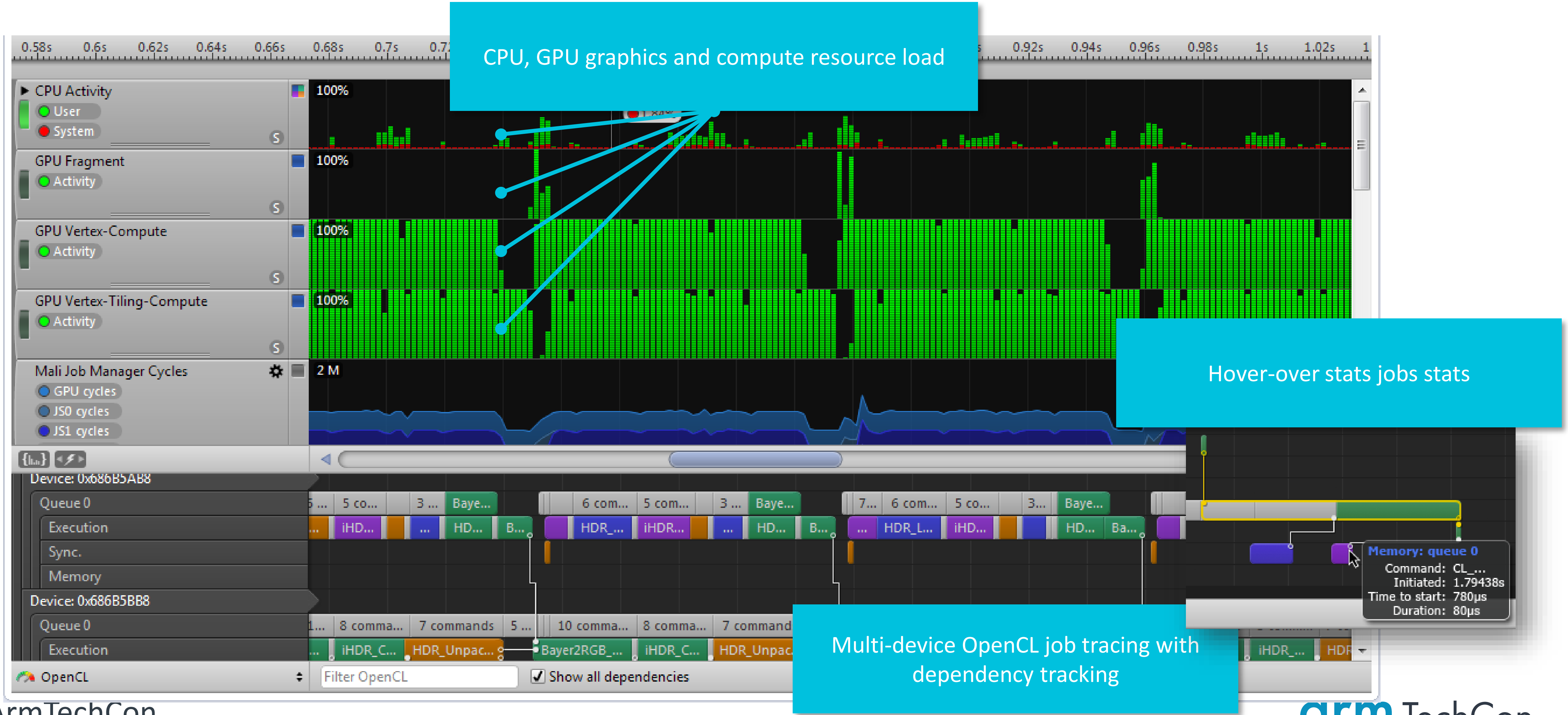
A GPU bound case

- See how much bandwidth is being used by the system
- Which IP block is responsible for the majority of the bandwidth usage

A bandwidth bound case

But what about the ML bound case?

OpenCL Support



Arm ML processor

Network control unit

- Overall programmability and high-level control flow

Onboard Memory

- Central storage for weights and feature maps

DMA

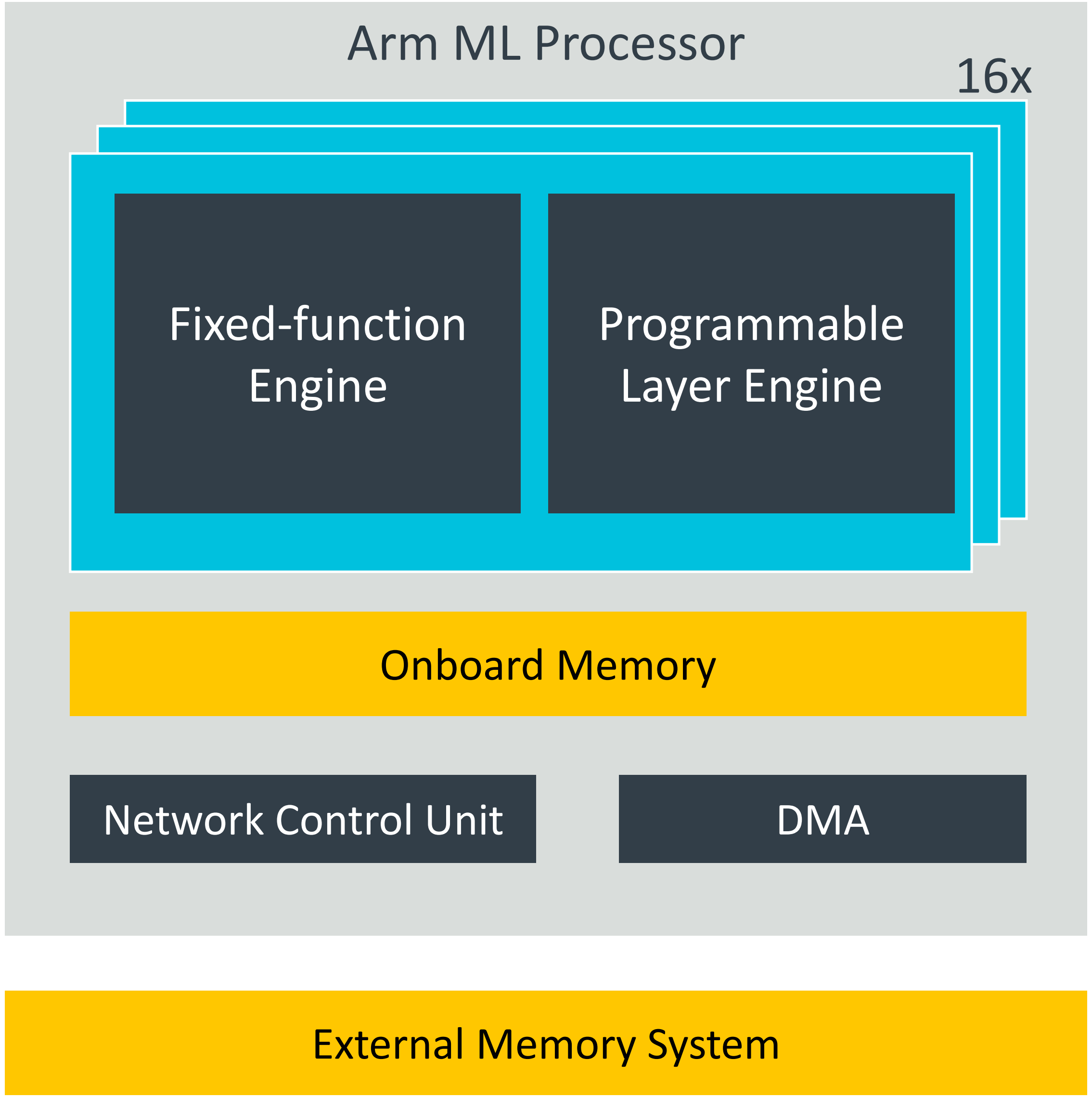
- Move data in and out of main memory

Fixed-function engines

- Main fixed-function compute engines

Programmable layer engines

- Enable post-deployment changes for future proofing



Arm ML processor information in Streamline

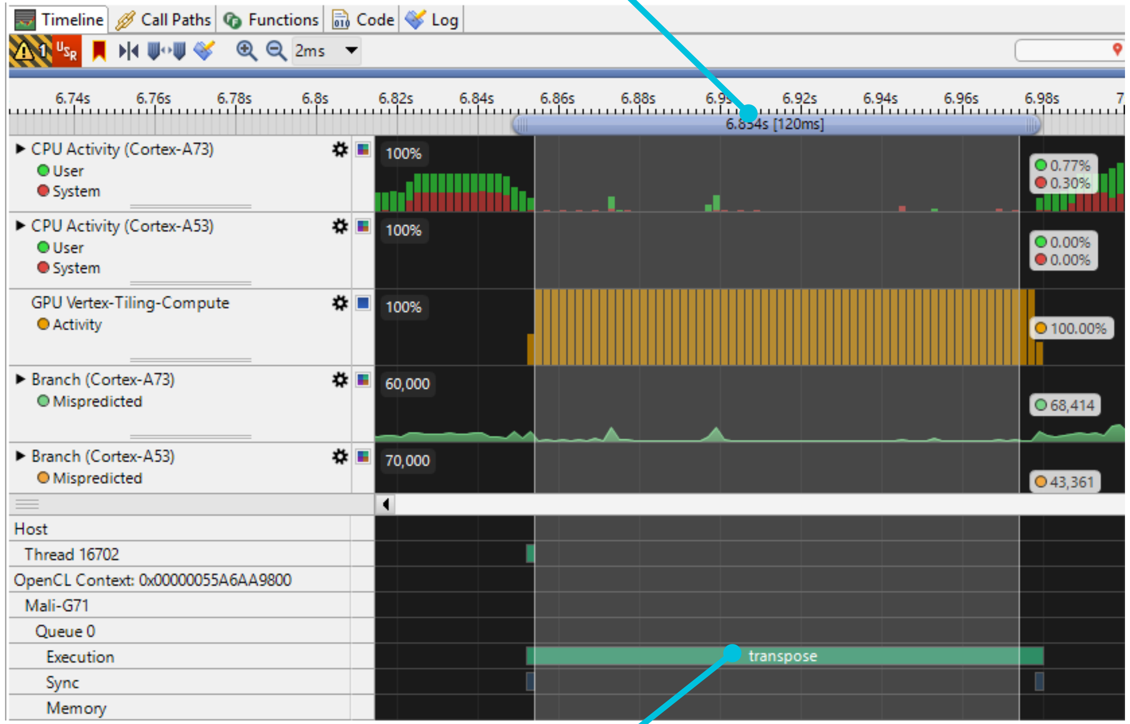
Streamline great for first pass analysis

Adding support for Arm's ML processor in Streamline

Allows the user to see:

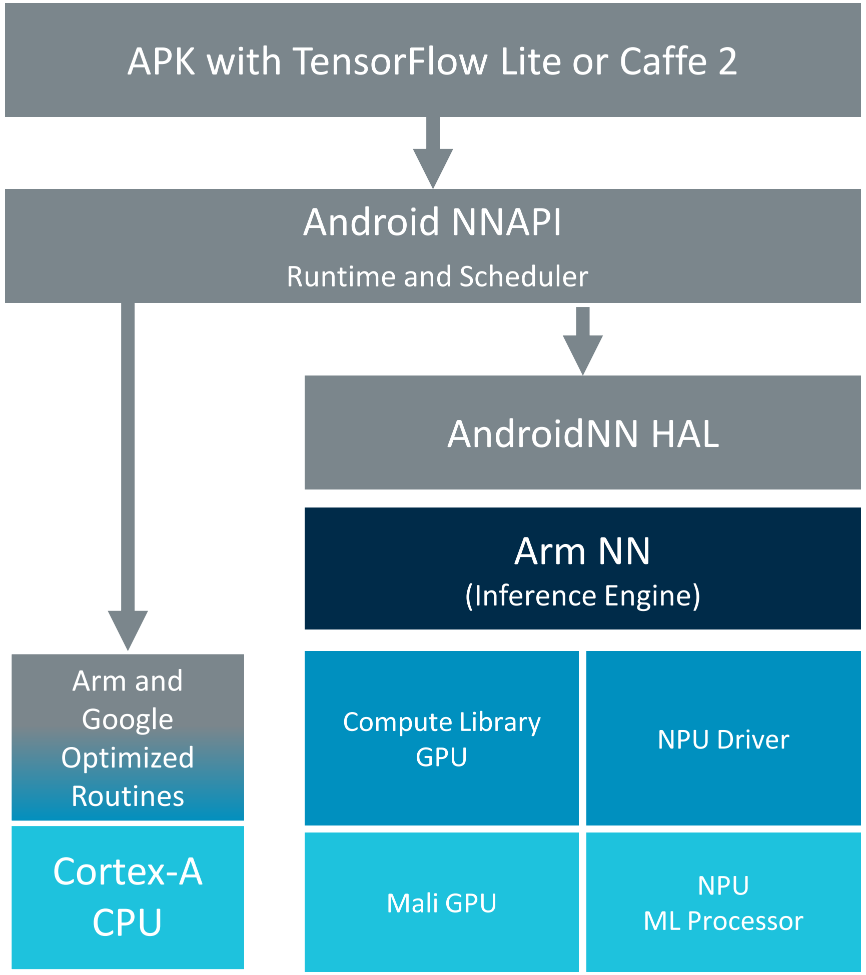
- Whether they are ML bound
- What IP block their ML content is running on:
  - CPU with Neon
  - GPU with OpenCL
  - Arm ML processor (NPU)

Duration of particular job



Transpose is taking a long time

Arm NN for Android & Linux: Overview

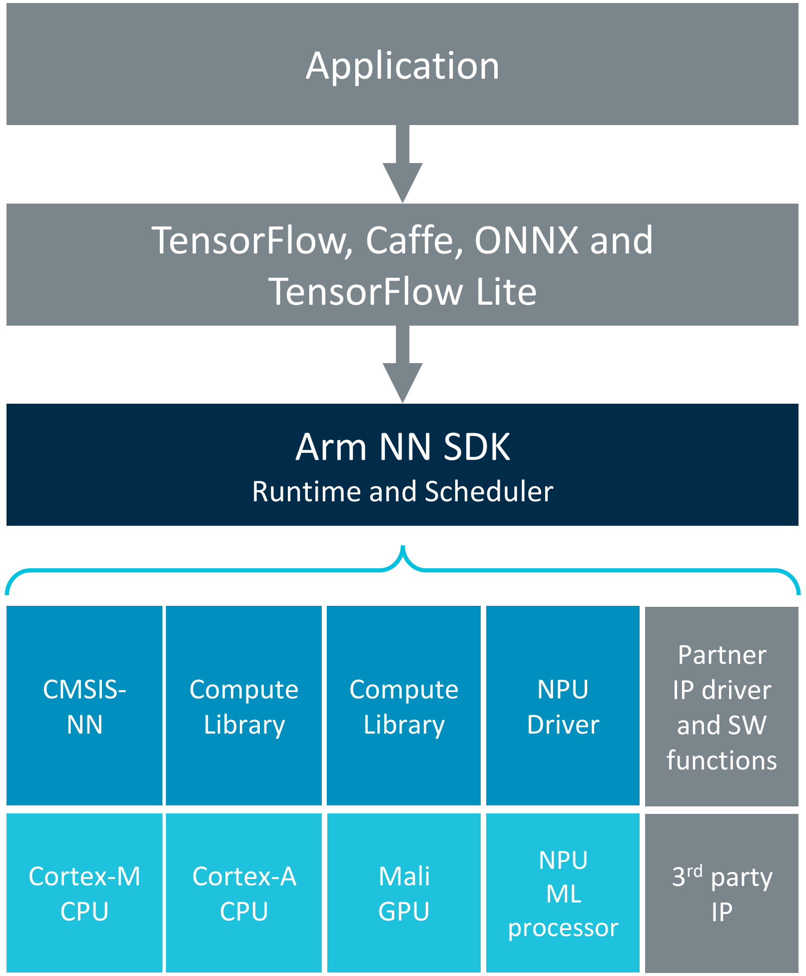


Arm NN providing support for Cortex-A CPUs and Mali GPUs under embedded Linux

Support for Cortex-M in development

Support for ML Processor available on release

Arm NN providing support for Mali GPUs under Android NNAPI



Compute Library

Optimized low-level functions for CPU and GPU

- Most popular CV and ML functions
- Supports common ML frameworks
- Over 80 functions in all
- Quarterly releases
- CMSIS-NN separately targets Cortex-M

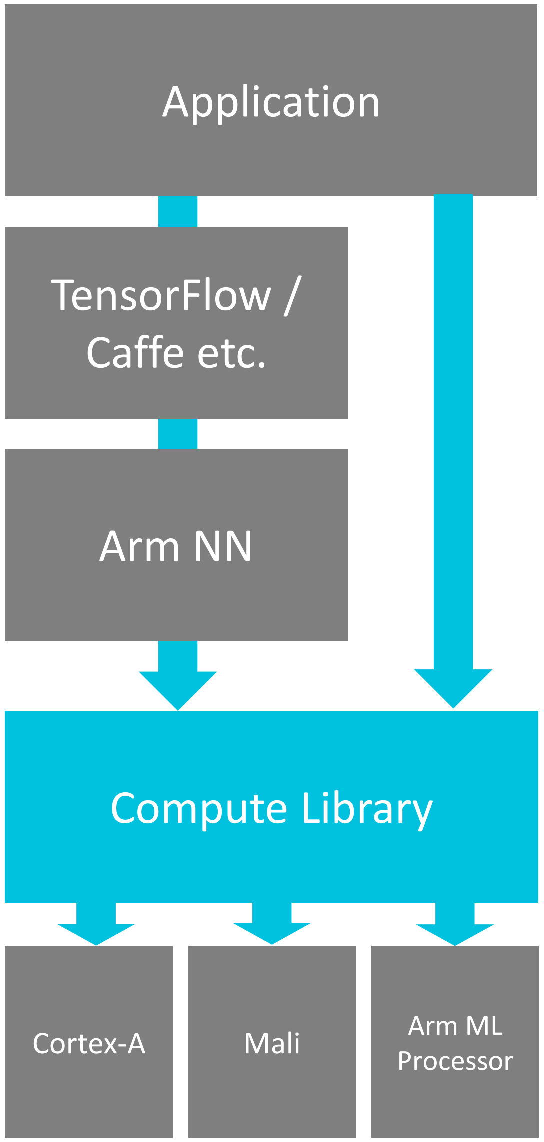
Enable faster deployment of CV and ML

- Targeting CPU (NEON) and GPU (OpenCL)
- Significant performance uplift compared to OSS alternatives (up to 15x)

Publicly available now (no fee, MIT license)

https://developer.arm.com/technologies/compute-library

Key Function Categories
Neural network
Convolutions
Colour manipulation
Feature detection
Basic arithmetic
GEMM
Pyramids
Filters
Image reshaping
Mathematical functions



Streamline annotations

- Custom Counters
- Text Based
- Visual
- Markers
- Custom Activity Map (CAM)
- Groups and Channels

- Annotations allow you to instrument your code to provide Streamline more information about your application
- There are a variety of different types that allow you to:
  - Highlight points of interest in your code
  - Show dependencies between different parts of your code
  - Show duration of execution time for various pieces of code
- Examples for all times can be found in the Streamline package

Annotations and the Compute Library

This allows the user to see:

- Which functions are called in the Compute Library
- How long each function took to execute
- The OpenCL kernels that were created and ran as a result

Functions that your ML network call in the Compute Library

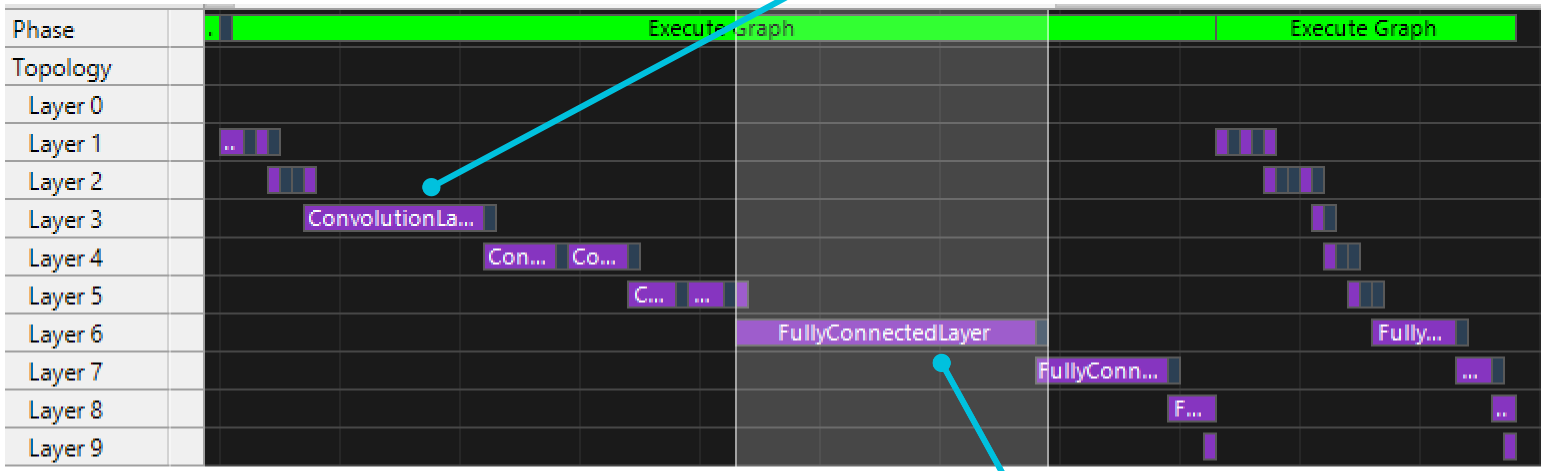
Work out duration every step of the way

The OpenCL kernels that get called

Annotations and Arm NN

- Visibility of how long each layer takes to execute
- See what layers are involved in the system and how they interact with each other
- Identify the phase of the ML graph the system is currently executing

Look at what layers are involved in your network



How long did each layer take to execute

New ML View in Streamline

- Brand new view that is dedicated to just information relating to ML
- Shows you the length of execution of the each of your Linux Kernels

List can be sorted on execution time

Timeline | Call Paths | Functions | Code | Log | ML Profile

All phases | Row Filter

Name	Execution Time [ns]
ML-ComputeLibrary-2982	9404221606
Kernels	0
OpenCL Kernels	222285885
softmax_layer_norm	11459
softmax_layer_max_shift_exp_sum_parallel	13020
im2col_reduced_dchw	13021
pooling_layer_optimized_3	42188
gemm_accumulate_biases	59375
col2im	328125
winograd_input_transform_4x4_3x3_stepz1_nchw	362500
fill_image_borders_replicate	412500
reshape_to_columns	594792
activation_layer	917187
im2col11x11_padx0_pady0_dchw	1018750
normalization_layer_cross_map	1048957
winograd_output_transform_4x4_3x3_nchw	1155728
fill_image_borders_constant	2130728
gemm_mm_floating_point_f32_bifrost_1000	2788020
gemm_mm_floating_point_f32_bifrost	9369269
direct_convolution5x5_f32_bifrost	10403644
gemm_mm_floating_point	20750516
winograd_filter_transform_4x4_3x3_nchw	64656235
transpose	106209871

Total of all the OpenCL Kernels that run

Time take for each individual kernel

New ML View in Streamline

- Shows you the time spent in each layer of your network
- Also shows you each time spent in each phase of the execution
- Gives you a complete overview of your ML system

Layers of your network

All phases | Row Filter

Name	Execution Time [ns]
ML-ComputeLibrary-2982	9404221606
Kernels	0
OpenCL Kernels	222285885
Functions	303547311
Devices	312678573
OpenCL Kernel Builds	1663480836
Topology	3065254441
Layer 0	29167
Layer 8	115417158
Layer 7	118447888
Layer 5	205372342
Layer 6	240836398
Layer 9	246002543
Layer 4	345410849
Layer 2	380911885
Layer 1	586833708
Layer 3	825992503
Phase	3836974560
Forcing graph to target	29687
Configuring Tensors	814062
Optimising Graph	1128645
Configuring Transition Manager	1385937
Validating Nodes	2382812
Execute Graph	54280715
Preparing CL Tasks	358125952
Allocating Tensors	387194695
Configuring Nodes	3031632055

Collapse and expand to reveal more detail about your system

Time spent with each phase of the network

Streamline support for Python

- A lot of machine learning networks start out in Python for ease of use
- Streamline supports the profiling of Python. Allowing you to optimize your algorithms right from the beginning
- You get access to the same visualizations as C or native code. Including:
  - Hardware counter information that happened as you were running your Python code
  - CPU sampling information showing where you spent the time in your code
- Simply use the gator.py Python module that gets shipped with Streamline with it you can:
  - Profile the entry and exit of every function
  - Trace every line of code

Python –m gator moduleToBeProfiled

Resources

Video Resources

- **MGD Introduction Part 1:** Shows how to create a trace in MGD and what options should be included.
- **MGD Introduction Part 2:** Shows how to look for problem areas in your scene using MGD.
- **MGD User Scripting:** Shows how to use the scripting feature to automate analysis and add new features.
- **MGD Automated Trace:** Shows how automate the trace and capture of applications
- **MGD Vulkan:** Shows the features in MGD that allow Vulkan traces and capture to work
- **Streamline Non-root**: How non-root mode works and what data you can get out of it

https://developer.arm.com/products/software-development-tools/ds-5-development-studio/resources/ds-5-media-articles

Blogs

- **Streamline for Developers**
- **Looking at performance analysis on your target**
- **Using Streamline with Fast Models**
- **Using tools with Unreal Engine and Unity**
- **Deploying Convolutional Neural Network on Cortex-M**
- **Using Arm to bring on-device machine learning inference to embedded world**

http://community.arm.com

Problems with Performance Analysis

Difficult to understand

Takes time to learn each different part of IP

Confusing with over 100 different counters to choose for each IP block

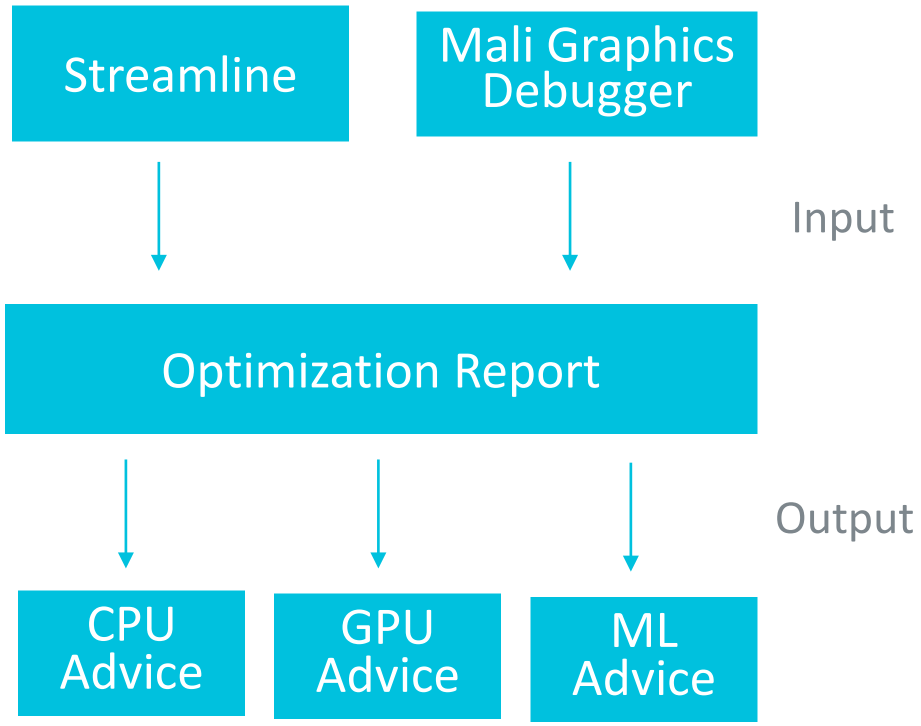
Many different systems out there. Can't learn them all

How this Relates to ML

The Machine Learning solution that we have detailed in the last few slides can suffer from the same problem:

- What do the ML counters actually mean?
- Which ones should a user select to get the most meaningful output to them?
- How do they learn which ones they should select?
- How can they take the timing information for each layer and turn that into something actionable?

Optimization Report



- Optimization report based on Arm Streamline and Mali Graphics Debugger data
    - Takes in MGD and Streamline data as inputs
    - Produces HTML-based reports based on the inputs
    - Offers advice on CPU, GPU, and ML-based workloads
    - Can integrate Perfdoc tool to give advice on optimizing for Vulkan API
- Streamline and MGD can generate directly which provides an improved user experience

Optimization Report for Arm Streamline and MGD

- A first pass analysis tool for ML-based content
  - Identifies which processor is running machine learning code
  - Provides timing information for each layer in the network
  - Visibility of the bandwidth and memory consumption of the neural network
  - Actionable advice based on the analysed data for targeted optimizations

Plan and Progress for ML

Currently Available

- **Streamline with the following features**
  - CPU Sampling
  - Hardware counter support
  - OpenCL Support

Coming Soon

- **Streamline with ML features**
  - Annotations in Compute Library
  - Annotations in Arm NN
  - New ML view showing time of layers and functions

Future

- **Optimization Reports**
  - First pass analysis of CPU, GPU & ML based workloads
  - Optimization advice given to the three above usecases

How to get access to Streamline

- Streamline is part of the DS-5 family. More information can be found at: https://developer.arm.com/products/software-development-tools/ds-5-development-studio/streamline
- There are several different editions that provide you different features. Evaluation copies of all the editions are available

Feature	DS Community Edition	DS Professional Edition	DS Ultimate Edition
GPU Support	Yes	Yes	Yes
OpenCL Support	Yes	Yes	Yes
Basic CPU Support	Yes	Yes	Yes
CPU Sampling	No	Yes	Yes
Full ArmV7 Support	No	Yes	Yes
Full ArmV8 Support	No	No	Yes

Summary

We have looked at:

- What data can you currently get out of your Machine Learning network
- How we have worked with the Arm NN team and the Compute Library team to provide ML information now
- Look at the future on what Arm is planning to do with ML tooling
- First Look at plan to generate automated optimization reports and how it makes performance analysis more accessible