arm

Linaro Virtual Connect 2020

Arm NN

New Features in 19.11 to 20.05 Release

Kevin May

23/09/2020

Agenda

- Introduction to Arm NN
  - What is Arm NN
  - Why use Arm NN
  - Real World Arm NN Use Case
- New Features added to Arm NN
  - PyArmNN
  - Backend Hint API
  - New API for passing parameters directly to backends
  - Arm NN data type support
  - Android NN Driver operators
- How To Contribute

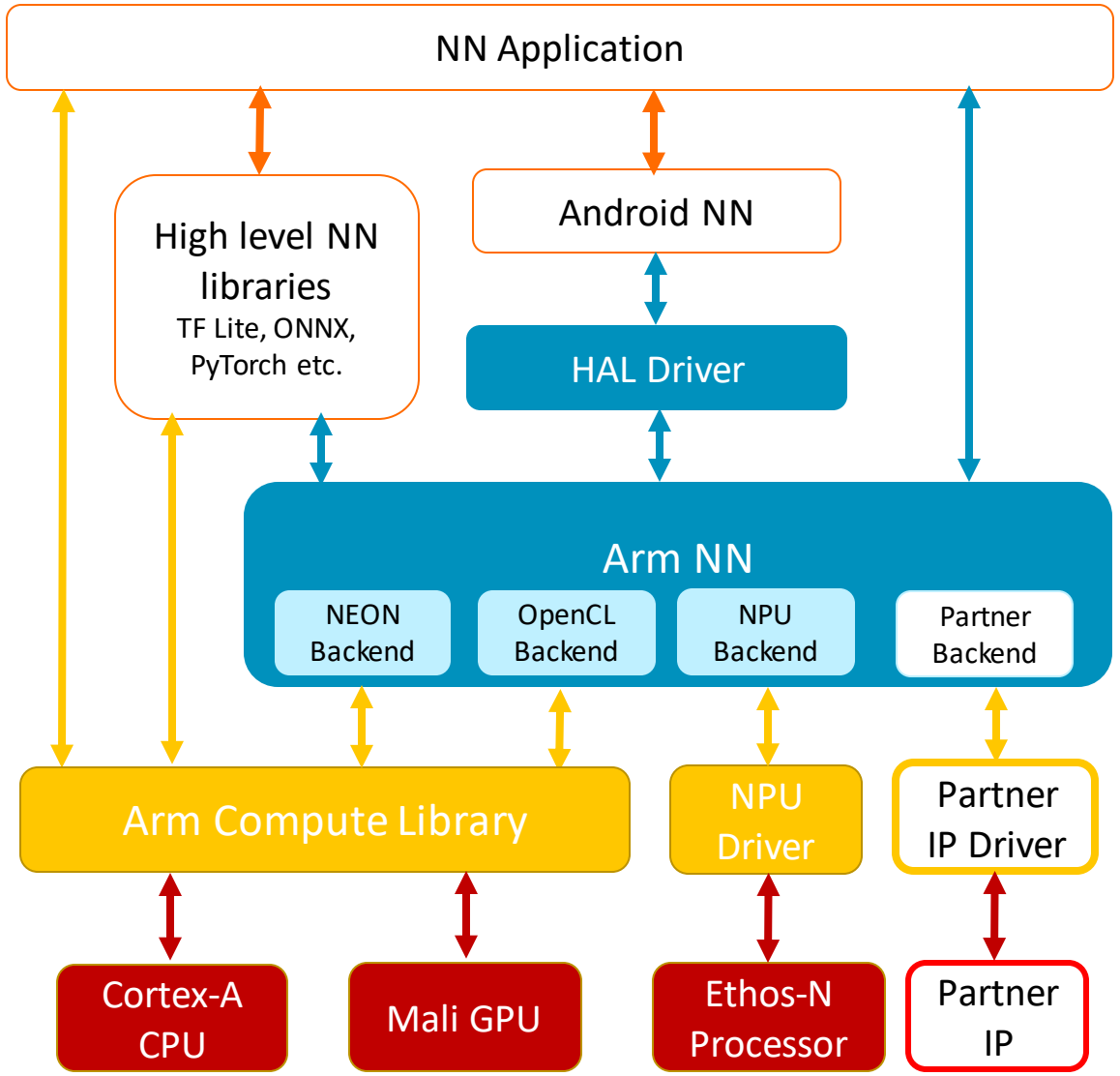
Introduction to Arm NN

What is Arm NN

- Arm NN is an inference engine for edge machine learning
- Arm NN bridges the gap between existing NN frameworks and underlying IP
  - Enables rapid application development through the support of commonly used frameworks such as TfLite, ONNX, PyTorch and Caffe
  - Provides efficient translation of these neural network frameworks allowing them to run efficiently across Arm Cortex-A CPUs, Mali GPUs, and Ethos-N NPUs
- Provides Android NNAPI Support
  - Arm NN interfaces with Googles Android NN using the HAL Driver to target Arm IP
- Provides Efficient targeting of Arm IP
  - Using Arm Compute Library and Ethos-N Driver Stack

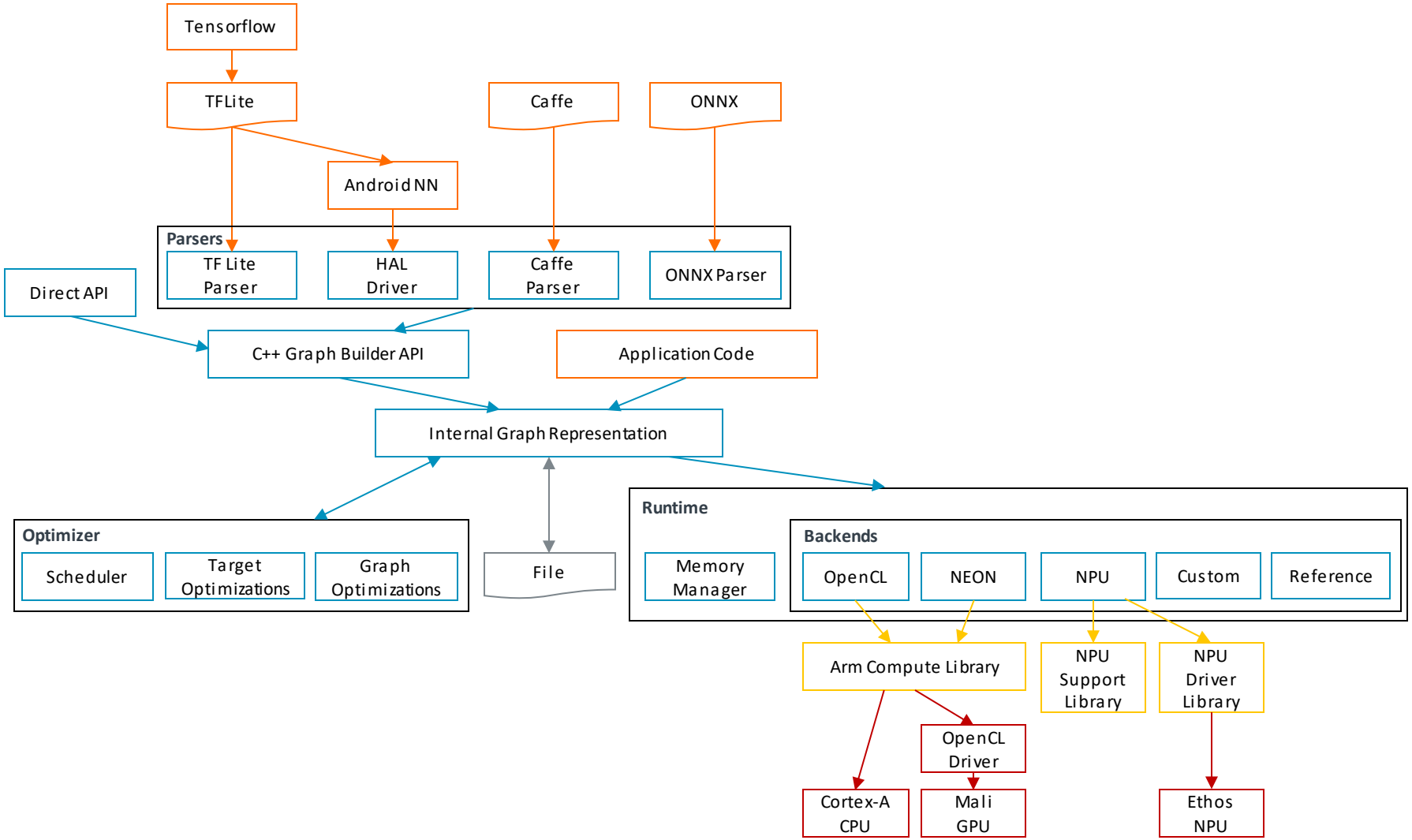
Arm NN Overview

- ML inference API for Linux written in **C++ 14**
- API to access many different NN accelerated devices
- Developed as **open source** and external contributions are always welcome
- Android NN **HAL driver** provides access to Arm NN for Android applications
- **Arm NN** provides the backends for the lower level libraries and hardware drivers
  - Third-party partners can add their own backends for Arm NN
  - Backends can be **dynamically loaded** to Arm NN during the runtime's start-up
- **Arm Compute Library (ACL)**
  - **Arm Cortex-A CPU** with NEON acceleration (ARMv7 and v8x)
  - **Arm Mali GPU** with OpenCL acceleration (Midgard and Bifrost architectures)



Arm NN Components

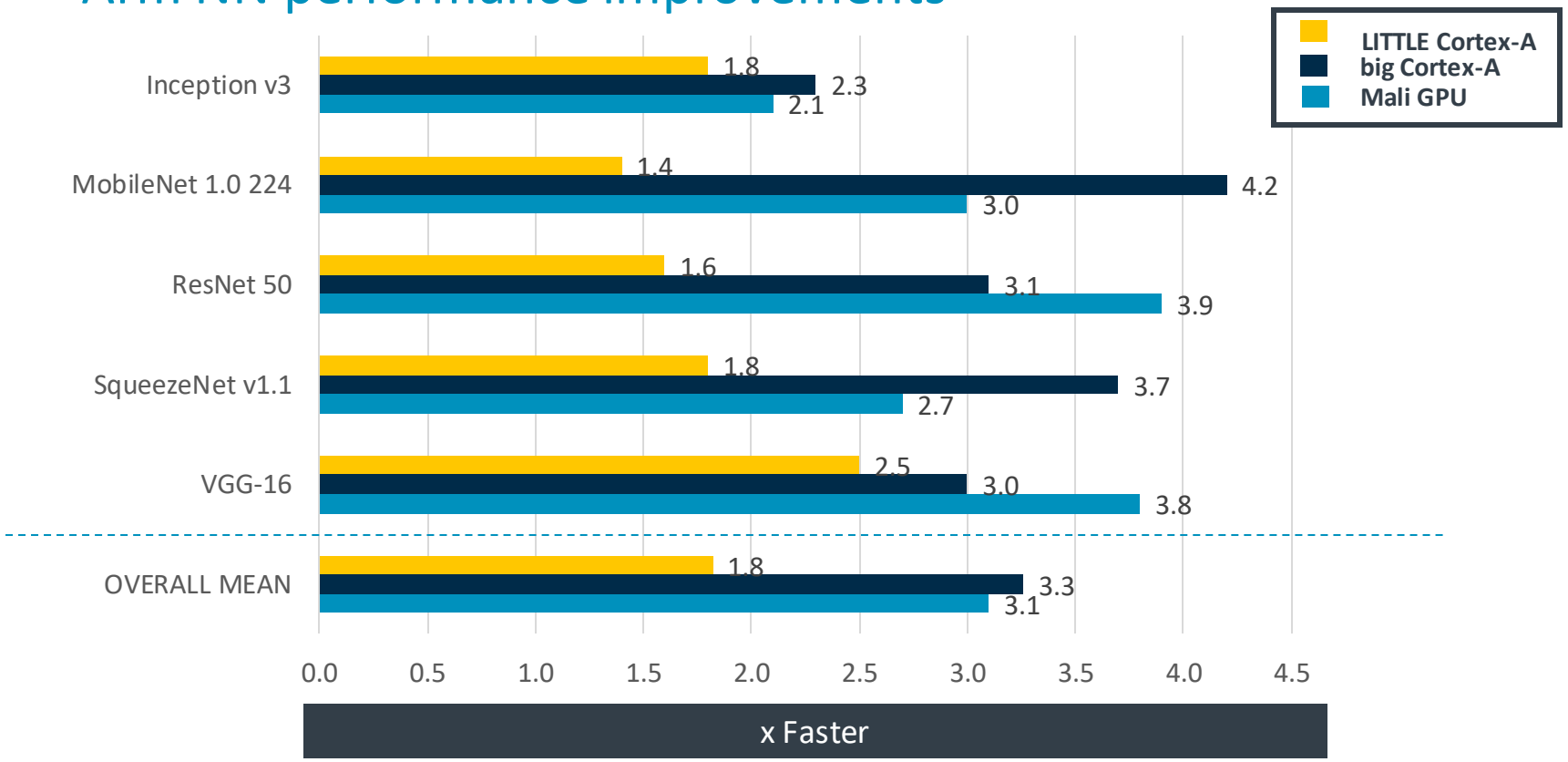
- Parsers
  - TensorFlow Lite
  - ONNX
  - PyTorch via ONNX
  - Caffe
- Android NN API
  - HAL Driver
- Core
  - Graph Builder API
  - Optimizer
  - Runtime



Why Arm NN?

Performance Improvements on Arm IP

Arm NN performance improvements*



- Arm NN shows what's possible for inference network optimizations on Arm IP
- Arm works with Partners to improve Arm hardware performance for other software libraries

*Mean performance improvements of Arm NN relative to up to six different industry software libraries

Real World Use Case – Arcturus Networks

- Arcturus Vision and AI Middleware transforms cameras from being passive observers into active detection systems
- Vision detection use cases for safety, security and surveillance devices
  - Motion / intrusion
  - Boundary crossing / zone incursion
  - Loitering / motionless behavior
  - Abandoned package
  - Face verification

Real World Use Case – Arcturus Networks

- Arcturus inference runtimes are optimized for edge processing by using Arm NN and TensorFlow Lite
- Models used include Yolo v3 and Inception v4
- Leveraging the Arm v8 NEON engine and utilizing the Arm NN inference framework to accelerate performance of a neural network allows Arcturus to run detection on Cortex-A cores, without the need for specialized ML hardware or a GPU
- For a public safety system like this one, edge processing improves response time and reliability by eliminating the need to ship pixel data from each camera across the network and up to the cloud

arm

PyArmNN

PyArmNN - Overview

- A Python extension for Arm NN SDK
- Provides a wrapper around the C++ interface
  - Python Api is very similar to C++
  - Does not implement any computational kernels, only a few helper functions
- Built around public headers of Arm NN
- All operations are delegated to the Arm NN library
- PyArmNN is integrated into Arm NN's CMake build system

Available at:

https://github.com/ARM-software/armnn/tree/branches/armnn_20_08/python/pyarmnn

PyArmNN – Code Example

```
# Import PyArmNN
import pyarmnn as ann

# Create a network from a model file and parser
parser = ann.IOnnxParser()
network = parser.CreateNetworkFromBinaryFile('mobilenetv2.onnx')

# Create a runtime initialized with creation options
options = ann.CreationOptions()
runtime = ann.IRuntime(options)
```

PyArmNN – Code Example

```
# Create a list of preferred backends
preferredBackends = [ ann.BackendId('GpuAcc'), ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]

# Optimize the network
optimized_network, _ = ann.Optimize(network,
                                    preferredBackends,
                                    runtime.GetDeviceSpec(),
                                    ann.OptimizerOptions())

# Load the optimized network into the runtime
net_id, _ = runtime.LoadNetwork(optimized_network)
```

PyArmNN – Code Example

```
# Get input names and binding info from model to create input tensors
input_names = parser.GetSubgraphInputTensorNames(0)
input_binding_info = parser.GetNetworkInputBindingInfo(0, input_names[0])
input_tensors = ann.make_input_tensors([input_binding_info], [image]) # Load image here

# Get output names and binding info from model to create output tensors
output_names = parser.GetSubgraphOutputTensorNames(0)
output_binding_info = parser.GetNetworkOutputBindingInfo(0, output_names[0])
output_tensors = ann.make_output_tensors([output_binding_info])

# Run inference
runtime.EnqueueWorkload(net_id, input_tensors, output_tensors)
```

PyArmNN

- More information on PyArmNN:

https://github.com/ARM-software/armnn/blob/branches/armnn_20_08/python/pyarmnn/README.md

- Examples can be found at:

https://github.com/ARM-software/armnn/tree/branches/armnn_20_08/python/pyarmnn/examples

Arm NN Backends

Backend Hint API

Backends - Overview

- A backend is an abstraction connecting layers of a graph to the underlying hardware through a driver or a compute engine
- Existing backends
  - OpenCL (Mali GPU)
  - NEON (Cortex-A CPU) using Arm Compute Library
  - reference (CPU-Ref testing)
  - NPU Backend (Ethos-N Processor)
- Arm NN allows adding custom backends through the **pluggable backend mechanism**
- Custom backends can also be loaded at runtime through the **dynamic backend interface**

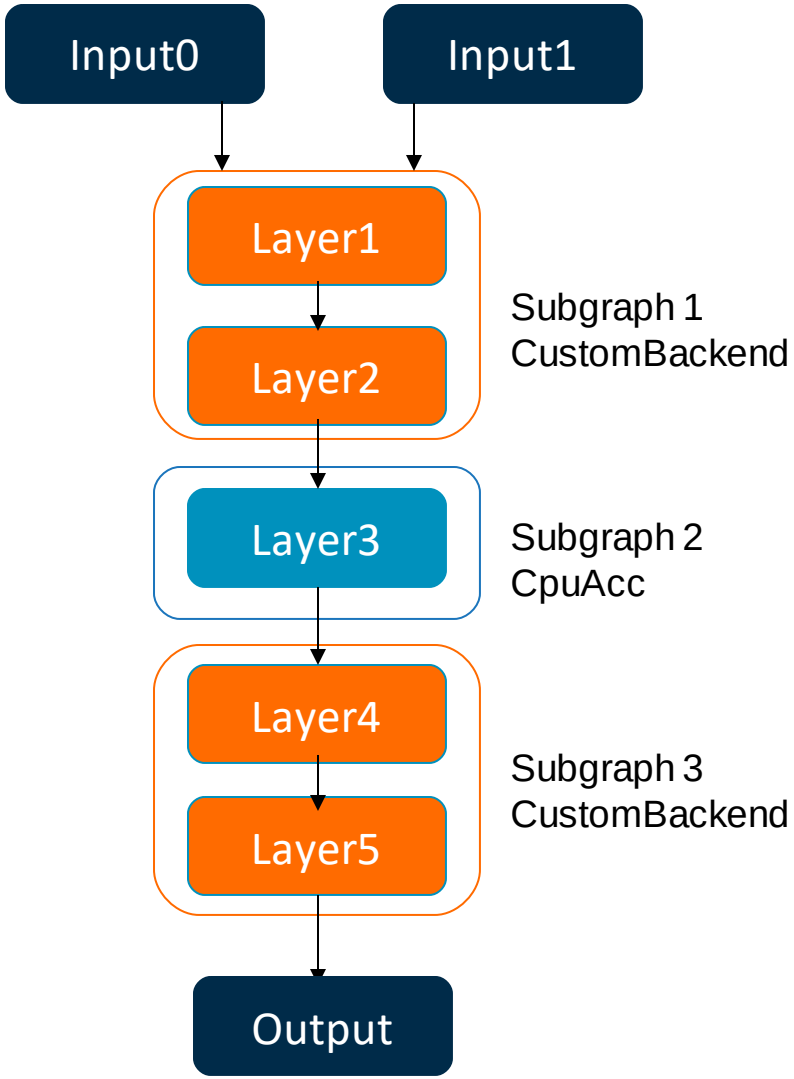
More information:

https://github.com/ARM-software/armnn/blob/branches/armnn_20_08/src/backends/README.md

https://github.com/ARM-software/armnn/blob/branches/armnn_20_08/src/dynamic/README.md

Backends – Specify Preference

- Arm NN optimizes the execution of a graph into continuous blocks (Subgraphs) which can be executed using multiple backends
- This happens in the Optimize function
- Preferred backend order is specified in code
  - `preferredBackends = [ ann.BackendId('CustomBackend') ann.BackendId('CpuAcc'), ann.BackendId('GpuAcc') ]`



Backends – Specify Preference Per Layer

- For power users of ArmNN we have added a new feature to specify a desired backend on a per-layer basis
- If set, this hint is treated as the first (most preferred) backend to select during the backend selection phase of the Optimizer
- If support for the layer cannot be found on the backend, we use the normal fallback mechanism to search for support on another backend
- Add the following method to the IConnectableLayer interface:
  - `void IConnectableLayer::BackendSelectionHint(BackendId id);`
- If the user wishes to load a model using the parsers, there are no handles to the layers while the InputNetwork is built up so they cannot explicitly set the backend hint
- The user must use the INetwork::Accept(ILayerVisitor* visitor) interface to provide a custom visitor class which identifies layers in the graph in order to set the backend hint based on custom rules.

Arm NN Backends

New API for passing parameters directly to backends

Backends – Pass Parameters Directly

- The 20.02 release introduced a new API (BackendOptions) for passing parameters directly to the backends
- BackendOptions are added during the initialisation of the Runtime
- When the runtime is initialized it creates an optional IBackendContext object and keeps this context alive for the Runtime's lifetime

Backends – Pass Parameters Directly

- We demonstrate the use of this new API using an existing feature called GpuAcc tuning which previously had its own GpuTuner API (now deprecated)
- The OpenCL tuner, a.k.a. CLTuner, is a module of Arm Compute Library that can improve the performance of the OpenCL kernels by tuning the Local-Workgroup-Size (LWS)
- The optimal LWS for each unique OpenCL kernel configuration is stored in a table
- This table can be either imported or exported from/to a file
- It supports three modes of tuning with different trade-offs between the time taken to tune and the kernel execution time achieved using the best LWS found

Backends – Pass Parameters Directly

```
// Create ArmNN runtime
IRuntime::CreationOptions options;
auto& backendOptions = options.m_BackendOptions;
backendOption.emplace_back(
    BackendOptions{
                        "GpuAcc",
                        {
                            {"TuningLevel", 2    },
                            {"TuningFile", filename}
                        }
                    }
    );
armnn::IRuntimePtr run(armnn::IRuntime::Create(options));



//  To execute with the tuning data start up with just the tuning file specified
```

Arm NN Data Type Support

Arm NN Data Type Support

Arm NN 19.08

```
enum class DataType
{
Float16 = 0,
Float32 = 1,
QuantisedAsymm8 = 2,
Signed32 = 3,
Boolean = 4,
QuantisedSymm16 = 5
};
```

Arm NN 20.05

```
enum class DataType
{
Float16 = 0,
Float32 = 1,
QAsymmU8 = 2, // RENAMED*
Signed32 = 3,
Boolean = 4,
QSymmS16 = 5, //RENAMED*
…
QSymmS8 = 7, // NEW
QAsymmS8 = 8, // NEW
BFloat16 = 9, // NEW
…
};
```

*Introduced new naming scheme for quantized types as follows:
Q[**Asymm**etric|**Symm**etric][**U**nsigned|**S**igned][**8**-bit|**16**-bit]

Date Type Support

QAsymmS8 and QSymmS8

- In the 20.02 release we updated the TfLite version to 1.15
- The new Tensorflow Lite 8-bit quantization specification uses int8 instead of uint8 to hold the payload
- We added support for two new data types:
  - QAsymmS8: Quantised Asymmetric Signed 8-bit integer type
  - QSymmS8: Quantised Symmetric Signed 8-bit integer type
- Activations are Asymmettric:
  - They can have their zero-point anywhere within the signed int8 range [-128, 127]
- Weights are Symmetric:
  - Weights are forced to have their zero-point equal to 0
- In addition we updated the Android NN Driver, Serializer and Quantizer to also support these new data types

Date Type Support

BFloat16

- In 20.05 we added Bfloat16 data type support
- bf16 provides an easy way to accelerate fp32 networks with a smaller loss in accuracy compared to fp16
- Benefits include reduction of memory footprint and computation speedup
- The existing fp16 turbo mode
  - this mode adds a conversion layer just after inputs and just before outputs so that the inputs and outputs are still fp32
- We added a second turbo mode which converts an fp32 network into Bfloat16
-
```
armnn::OptimizerOptions opOptions;
opOptions.m_ReduceFp32ToBf16 = true;
armnn::IOptimizedNetworkPtr optNet = Optimize(*network,
                                              preferredBackends,
                                              runtime->GetDeviceSpec(),
                                              opOptions);
```

Android NN Driver Operators

Android NN Driver Operators

Write in your subtitle here

HAL 1.0 & 1.1

- ADD
- AVERAGE POOLING 2D
- CONCATENATION
- CONV 2D
- DEPTH TO SPACE
- DEPTHWISE CONV 2D
- DEQUANTIZE
- FLOOR
- FULLY CONNECTED
- LOCAL RESPONSE NORMALIZATION
- LOGISTIC
- LSTM
- L2 NORMALIZATION
- L2 POOLING 2D
- MAX POOLING 2D
- MUL
- RELU, RELU1, RELU6
- SOFTMAX
- SPACE TO DEPTH
- TANH
- RESHAPE
- RESIZE BILINEAR

Android NN Driver Operators

*Operators added since 19.11 Release

HAL 1.2

- ABS
- ARGMAX
- ARGMIN
- DIV
- EQUAL
- EXPAND DIMS
- DEPTH TO SPACE
- DEQUANTIZE
- GATHER
- GREATER, GREATER EQUAL
- GROUPED CONV 2D
- INSTANCE NORMALIZATION
- LESS, LESS EQUAL, NOT EQUAL
- LOG SOFTMAX
- MAXIMUM
- MEAN
- NEG
- PAD, PAD V2
- PRELU
- RESIZE NEAREST NEIGHBOUR
- RSQRT, SQRT
- SQUEEZE
- STRIDED SLICE
- TRANSPOSE
- TRANSPOSE CONV 2D
- SPACE TO BATCH ND
- SUB

- HAL 1.3
  - ELU
  - FILL
  - HARD SWISH
  - QLSTM
  - RANK

arm

How to Contribute

Arm NN is Open Source

- Currently released under an **MIT license**

- **ArmNN** uses a **continuously integrated** development model
  - When accepted into **Linaro** as part the official Artificial Intelligence Initiative
  - Every commit to master is publicly available on review.mlplatform.org

- The master branch from review.mlplatform.org is automatically mirrored to GitHub
- Releases are made to **GitHub** every quarter

- Note: we are planning to change license from **MIT** to **Apache 2.0** to provide clarity over patents to both contributors and users

Contributing code to Arm NN

- All code reviews are performed on Linaro ML Platform Gerrit
- GitHub account credentials are required for creating an account on ML Platform
- Setup Arm NN git repo
  - git clone https://review.mlplatform.org/ml/armnn
  - cd armnn
  - git config user.name "FIRST_NAME SECOND_NAME"
  - git config user.email your@email.address
- Commit using sign-off and push patch for code review
  - git commit **-s**
  - git push origin HEAD:refs/for/master
- Patch will appear on ML Platform Gerrit here
- The Contributor Guide contains details of copyright notice and developer certificate of origin sign off

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה

Arm NN Tutorials

- Configuring Arm NN
  - TF Lite: Configuring the Arm NN SDK build environment for TensorFlow Lite
  - ONNX: Configuring the Arm NN SDK build environment for ONNX

- Deployment Examples
  - Quantized TF Lite: Deploying a quantized TensorFlow Lite MobileNet v1 model
  - Style Transfer on Android: Implementing a neural style transfer on Android
  - Text-to-Speech: Creating a Text-to-speech engine with Tesseract and Arm NN on Raspberry Pi
  - PyArmNN: Accelerating ML Inference on Raspberry Pi with PyArmNN

- Customization
  - Custom backends: Building Arm NN custom backend plugins

arm