Inferencing at the Edge and Fragmentation Challenges

Mark Charlebois
Director Engineering
Qualcomm Technologies, Inc.
Agenda

• Overview of Inferencing at the Edge
  ◦ Deployment Scenarios
  ◦ Application Development Options

• Runtimes

• Graph Compilers
  ◦ Four Levels of IR

• Leveraging Runtimes in Compiler Frameworks

• Conclusions
Inferencing at the Edge

• Models developed on servers, deployed to edge devices
• Devices can range from MCUs to edge servers
• Simple networks can run on MCU
• Huge diversity of HW at the edge for inferencing acceleration
• Growing number of runtimes and graph compilers
Deployment Scenarios

**Known Models, Unknown Devices**
- Developer
  - App Store
  - Phone

**Known Models, Known Device**
- OEM
  - Phone

**Known Models, Known Devices**
- OEM
  - Cloud
  - Managed Deployment
    - Device A
    - Device B
Application Development Options

App Using Vendor SDKs

- **Vendor runtime** → **Device** → **App**

App using OS / Framework API

- **Vendor runtime** → **Device** → **Platform/Framework API (e.g. NNAPI)** → **HAL A** → **Device** → **App**

App Using Compiled Model

- **Developer** → **Compiler Framework** → **Backend** → **App** → **Device** → **App** → **Device**
Runtimes
Android App Example (NNAPI)

- TFLite App running on Qualcomm® device using NNAPI

Trained TensorFlow Model

TFLite Converter

TFLite Model (.tflite)

Android App

TFLite

NNAPI

Neural Network Core

CPU ops

- Qualcomm® Math Libraries
- NEON

GPU ops

- Kernels
- OpenCL

DSP ops

- Qualcomm® Hexagon™ NN
- HVX
ARM NN Runtime Example

- Trained TensorFlow Model
- Trained ONNX, ...

App

ARM NN

Backends

- CPU ops
- Mali GPU
- ARM Compute Library (ACL)
- Other
CPU Runtime Fragmentation

- ARM NN
- Backends
- CPU ops
- ACL
- Qualcomm® Neural Processing SDK
- NN Core
- CPU ops
- Qualcomm® Math Libraries
- TFLite
- Deligate
- CPU ops
- Custom Kernels
- Android NN HAL
- Backend
- CPU ops
- Custom Kernels
Graph Compilers
Graph Compilers

nGraph  ONNX Runtime  ONNC  TVM  GLOW

CPU Binary  GPU Binary  DSP Binary

e.g. TensorFlow  Graph Compiler Framework
Framework Layer Cake - Example nGraph

TensorFlow -> Bridge -> nGraph -> CPU -> MKL-DNN
ONNX -> Bridge -> nGraph
mxnet -> Bridge -> nGraph
Caffe2 -> Bridge -> nGraph
PyTorch -> Bridge

PlaidML -> Intel GPU
nVidia GPU
Four Levels of IR

- **High-level IR**
  - "Graph IR"

- **Function-level IR**
  - "Operation IR"

- **Shader/AST-level IR**

- **Assembly-level IR**

---

**Developer APIs**

- Parse Graph, first pass optimization and partitioning
- Graph segment optimization
- Memory reuse (non-backend specific)
- Layout transformation

- HW specific subgraph optimizations
- HW optimizations:
  - Explicit memory layout, parallelization pattern

- Generate shaders, DSL code, bitcode
- Integrate hand optimized kernels

- Generate assembly and schedule for target ISA
Projects Mapped to Four Levels of IR

- **High Level IR**: NNVM/Relay, nGraph, XLA Frontend, GLOW Graph IR
- **Operator Level IR**: TVM, PlaidML, cuDNN, XLA Backend, GLOW Op IR
- **Shader/AST Level IR**: LLVM, OpenCL, CUDA
- **Assembly Level IR**: ARMv8 Assembly, Hexagon Assembly, PTX
Leveraging Runtimes in Compiler Frameworks
Frameworks, Platforms, HW, All Growing

<table>
<thead>
<tr>
<th>Framework</th>
<th>Runtime / Graph Compiler</th>
<th>OS</th>
<th>HW / Compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>Qualcomm Neural Processing SDK</td>
<td>Open Embedded</td>
<td>Hexagon SDK</td>
</tr>
<tr>
<td>Caffe</td>
<td>ARM NN</td>
<td>Android</td>
<td>OpenCL</td>
</tr>
<tr>
<td>Caffe2</td>
<td>TFLite</td>
<td>Ubuntu / Fedora / etc</td>
<td>CUDA</td>
</tr>
<tr>
<td>MxNet</td>
<td>Relay / TVM / LLVM</td>
<td>Windows</td>
<td>ARM M4 / A53 / A76 / etc</td>
</tr>
<tr>
<td>Pytorch</td>
<td>Glow / LLVM</td>
<td>RTOS</td>
<td></td>
</tr>
</tbody>
</table>

...
Frameworks, Platforms, HW, All Growing

<table>
<thead>
<tr>
<th>Framework</th>
<th>Runtime / Graph Comp</th>
<th>HW / Compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>Qualcomm Neural Processing SD</td>
<td>Hexagon SDK</td>
</tr>
<tr>
<td>Caffe</td>
<td>ARM NN</td>
<td>OpenCL</td>
</tr>
<tr>
<td>Caffe2</td>
<td>TFLite</td>
<td>CUDA</td>
</tr>
<tr>
<td>MxNet</td>
<td>Relay / TVM / LLVM</td>
<td>ARM M4 / A53 / A76 / etc</td>
</tr>
<tr>
<td>Pytorch</td>
<td>Glow / LLVM</td>
<td>etc</td>
</tr>
</tbody>
</table>

...
HW Diversity Is a Scalability Issue
Tradeoff Some Performance for Scale
Conclusions
Runtime Components Map to Op Level IR

Framework Model

Container

Qualcomm Neural Processing SDK
Hexagon NN

ARM NN
ARM NN Backend API

TEngine
TEngine executor

Graph Compiler
Op Level IR
Using Full Runtime as a HW Backend

mxnet  ➔ Relay  ➔ TVM  ➔ CPU  ➔ LLVM

mxnet  ➔ Relay  ➔ TVM  ➔ Runtime  ➔ CPU  ➔ GPU
Using Full Runtime as a HW Backend

mxnet → Relay → TVM → CPU → LLVM
mxnet → Relay → TVM → CPU → ACL
mxnet → Relay → TVM → Runtime → CPU → GPU
mxnet → ONNX → Runtime → CPU → GPU
ARM Inferencing Edge Server

- **TensorFlow** → Bridge
- **ONNX** → Bridge
- **mxnet** → Bridge
- **Caffe2** → Bridge
- **PyTorch** → Bridge
- **nGraph** → CPU
- **Mali CPU** → ACL
- **Accel HW** → compiler
- **ACL** → compiler
Addressing Graph Compiler IR Fragmentation

• Each compiler framework makes different tradeoffs, but eventually may be able to share components: e.g. code generation for OpenCL, LLVM, etc
• Too many frameworks, compilers and formats to address them all.
• Too much disparate HW for all frameworks to support.
• TensorFlow/TFLite and ONNX formats can provide the most scale for edge device inferencing runtimes.
• If Compiler Frameworks supported a common runtime backend API (like ARM NN Backend API) to bind to operator IR would enable graph compilers to support more edge devices with optimized backends, and would provide a common API for new backends (e.g. Hexagon NN) vs individual ports to each project.
Addressing CPU Runtime Fragmentation

• Make ACL the “best of breed” CPU Runtime
• Consolidate TFLite CPU runtime, Android NN CPU runtime, and ARM NN CPU backend
• Paves the way for others to follow: TEngine, MACE, Qualcomm® Neural Processing SDK, …