ML Inference at the Edge

Felix Baum
Senior Director, Product Management
Qualcomm Technologies, Inc.

@qualcomm
To scale, the center of gravity of AI processing is moving to the edge.

We are leading the realization of the hybrid AI

Convergence of:
Wireless connectivity
Efficient computing
Distributed AI

Unlocking the data that will fuel our digital future and generative AI

Cost  
Energy  
Reliability, latency, & performance  
Privacy & security  
Personalization
Choose env., config., model and framework

Does the model meet performance metrics?

Train, Finetune model

Model Compilation/Runner

Does the model compile?

Accuracy Evaluation

Add custom layers and/or fix errors

Debug and fix errors

Profile model Perf

Did that work?

Use advanced optimization techniques

Is the model's accuracy acceptable?

Is the model's output & latency acceptable?

Integrate model into App or pipeline

Deploy App

Data Scientist

ML Training Engineer

ML Inference Engineer

App Developer

DevOps Engineer

Environment

Training

Compilation

Accuracy analysis

Optimizations

Integration

Deployment

Qualcomm® AI Engine direct Converter and Quantizer

Custom Ops

LLVM (C/C++)

TVM (Python)

Qualcomm® Neural Processing SDK

Qualcomm® AI Studio

Quantization Checker

Detect potential accuracy issues by flagging suboptimal quantization

Architecture Checker

Automated model topology and architecture

NAS

QAT

AI model Efficiency Toolkit (AIMET)

Accuracy Evaluator

Automated quantization using all quantization options for best accuracy

Accuracy Debugger

Per layer output analysis for backends.

Performance Analyzer

Hexagon Processor profiler that provides guidance on expected performance.

Hexagon VS Code

Improved IDE for Debugging, Profiling and Trace Analysis

Qualcomm® Hexagon™ Processor 8mm Simulator

Simulation environment for bit accurate validation

AIMET is a product of Qualcomm Innovation Center, Inc
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

Scalar
(Zero-order Tensor)

Vector
(First-order Tensor)

Matrix
(Second-order Tensor)

3D Array
(Third-order Tensor)
Optimizing Hardware for AI
Neural Networks: A mundane pile of linear algebra

Scalar
(Zero-order Tensor)

Vector
(First-order Tensor)

Matrix
(Second-order Tensor)

3D Array
(Third-order Tensor)

Setup

28 x 28 x 1
24 x 24 x 32
12 x 12 x 32
8 x 8 x 64
4 x 4 x 64
1024
10
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

Scalar (Zero-order Tensor)

Vector (First-order Tensor)

Matrix (Second-order Tensor)

3D Array (Third-order Tensor)

Hexagon Processor

Setup

28 x 28 x 1
24 x 24 x 32
12 x 12 x 32
8 x 8 x 64
4 x 4 x 64
1024
10
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

**Scalar**
(Zero-order Tensor)

**Vector**
(First-order Tensor)

**Matrix**
(Second-order Tensor)

**3D Array**
(Third-order Tensor)

**Setup**

**Hexagon Processor**

- Scalar Threads
- Scalar Processor
- Hexagon Processor

**Pooling**
2 x 2

**Conv**
5 x 5 x 64

**Pooling**
2 x 2

**FC**

<table>
<thead>
<tr>
<th>Layer Description</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>28 x 28 x 1</td>
<td></td>
</tr>
<tr>
<td>24 x 24 x 32</td>
<td></td>
</tr>
<tr>
<td>12 x 12 x 32</td>
<td></td>
</tr>
<tr>
<td>8 x 8 x 64</td>
<td>4 x 4 x 64</td>
</tr>
<tr>
<td>1024</td>
<td>10</td>
</tr>
</tbody>
</table>
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

**Scalar**
(Zero-order Tensor)

**Vector**
(First-order Tensor)

**Matrix**
(Second-order Tensor)

**3D Array**
(Third-order Tensor)

Hexagon Processor

<table>
<thead>
<tr>
<th>Scalar</th>
<th>Vector</th>
<th>Matrix</th>
<th>3D Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Processor</td>
<td>Processor</td>
<td>Processor</td>
</tr>
</tbody>
</table>

Setup

- 28 x 28 x 1
- 24 x 24 x 32
- 12 x 12 x 32
- 8 x 8 x 64
- 4 x 4 x 64
- 1024
- 10
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

Scalar (Zero-order Tensor)

Vector (First-order Tensor)

Matrix (Second-order Tensor)

3D Array (Third-order Tensor)

Setup

Scalar Processor

Vector Processor

Tensor Processor

Vector Processor

Hexagon Processor

Scalar Threads

HVX

Matrix

HMX

INT8

HMX

FP16
Optimizing Hardware for AI
Neural Networks: A mundane pile of linear algebra

Scalar (Zero-order Tensor)

Vector (First-order Tensor)

Matrix (Second-order Tensor)

3D Array (Third-order Tensor)

Setup

Hexagon Processor

Scalar Threads

Memory Processing

DMA Engine

L2 Cache

Vector

Matrix

HVX

TCM

INT8

L2 Cache

Vector Processor

Tensor Processor

Vector Processor

Scalar Processor
Optimizing Hardware for AI
Neural Networks: A mundane pile of linear algebra
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

Scalar (Zero-order Tensor)
Vector (First-order Tensor)
Matrix (Second-order Tensor)
3D Array (Third-order Tensor)

Setup

Optimization Goals
1. Maximize parallelism
2. Minimize data movement

Hexagon Processor
Scalar Threads
Memory Processing
DMA Engine
Scalar
L2 Cache
Memory Processing
Vector
HVX
Matrix
HMX INT8
HMX FP16

Memory Access
DRAM
Optimizing Hardware for AI
Neural Networks: A mundane pile of linear algebra

Scalar (Zero-order Tensor)
Vector (First-order Tensor)
Matrix (Second-order Tensor)
3D Array (Third-order Tensor)

Setup

Optimization Goals
1. Maximize parallelism
2. Minimize data

Hexagon Processor
Scalar
Memory Processing
DMA Engine
L2\$
TCM
HVX
Matrix
HVX
TCM
HMX INT8
HMX FP16
Scalar Threads
L2\$
DMA Engine
Optimizing Hardware for AI: Transformers

Neural Networks: A mundane pile of linear algebra

Scalar (Zero-order Tensor)  Vector (First-order Tensor)  Matrix (Second-order Tensor)  3D Array (Third-order Tensor)

Setup

Scalar Processor

Transformer Architecture

Optimization Goals
1. Maximize parallelism
2. Minimize data

Hexagon Processor

Memory Processing
Scalar Threads DMA Engine L2$ Vector Matrix

HVX (~70%) HMX INT8 HMX FP16
Optimizing Hardware for AI: Super Resolution

Neural Networks: A mundane pile of linear algebra

Optimization Goals
1. Maximize parallelism
2. Minimize data

Hexagon Processor

- Scalar Processor
- Memory Processing
  - DMA Engine
  - L2 Cache
- Vector
  - HVX (~30%)
- Matrix
  - HMX (~90%)

Super Resolution Architecture

Scalar (Zero-order Tensor)
Vector (First-order Tensor)
Matrix (Second-order Tensor)
3D Array (Third-order Tensor)

Setup

Neural Networks:
- ABPN, SESR, XLSR, Q-SRNet

Notations:
- $f$: number of channels
- $m$: number of intermediate conv layers
- $\cdots$: collapsible residual connection
- $\cdot\partial$: "partial" collapsible residual connection
- $\cdot\Pi$: "repeat-interleaving" collapsible residual connection
Hexagon Processor: Execution of ML use cases

Optimization Goals
1. Maximize parallelism
2. Minimize data movement

Hexagon Processor

Scalar Processor
Vector Processor
Tensor Processor
Vector Processor

Scalar (Zero-order Tensor)
Vector (First-order Tensor)
Matrix (Second-order Tensor)
3D Array (Third-order Tensor)

Setup

Scalar
Vector
Matrix

Memory Processing

DMA Engine
L2 Cache
TCM
HVX
HMX
HMX INT8
HMX FP16
Hexagon Processor: Execution of end-to-end use cases
Hexagon Processor: Concurrency Model

- Pre-Processing
- ML
- Post-Processing

Implementing concurrency

- ML
- CV

HVX

- HVX
- HVX, HMX
- HVX

Monochrome (Mono)

RGB

HVX, HMX

- HVX

Vector 1

Vector 2

Vector 3

Vector 4

Thread 1

Thread 2

Thread 3

Thread 4

Thread 5

Thread 6

DMA Engine

L2 Cache

Matrix Engine

VTCM
Hexagon Processor: Concurrency Model

- **HVX**
  - Pre-Processing
  - ML
  - Post-Processing

- **HVX, HMX**
  - ML
  - Post-Processing

- **HVX**
  - Post-Processing

- **HVX**

**Mono**
- Pre-Processing

**RGB**
- Pre-Processing

**Hexagon Processor**
- **Scalar**
  - Thread 1
  - Thread 2
  - Thread 3
  - Thread 4
  - Thread 5
  - Thread 6

- **Memory Processing**
  - DMA Engine
  - L2$	ext{H}$

- **HVX**
  - Vector 1
  - Vector 2
  - Vector 3
  - Vector 4

- **HMX**
  - VTCM
  - Matrix Engine

**VTCM Partitioning be ML en CV**
AI Model Compilation: Steps

- Framework optimizations
- Converter
- Quantizer
- Common Graph optimizations
AI Model Compilation: Steps

- Framework optimizations
- Converter
- Quantizer
- Common Graph optimizations

Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
AI Model Compilation: Steps

- Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
- Framework graph is translated to the IR Graph
AI Model Compilation: Steps

Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters
AI Model Compilation: Steps

- Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
- Framework graph is translated to the IR Graph
- If required, graph can be quantized according to various config. parameters
- Framework agnostic graph optimizations are applied such as batchnorm folding
AI Model Compilation: Steps

- Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
- Framework graph is translated to the IR Graph
- If required, graph can be quantized according to various config. parameters
- Framework agnostic graph optimizations are applied such as batchnorm folding

Naive sequencers executed Nets “Layer-by-Layer”, sequentially. Sometimes 1-3 layers can be aggregated (e.g., conv followed by RELU). “Layer by Layer” leaves performance and memory bandwidth on the table:
- If we exploit concurrences and simultaneously operate on data from multiple layers, execution finishes faster
- A layer’s output, once consumed by next layer, is discorable. This saves DDR bandwidth, but TCM must be large enough, or data unit small enough, to store intermediate output
AI Model Compilation: Steps

Conversion / Quantization

Framework opt

Converter

Quantizer

Common Graph optimizations

Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

Framework agnostic graph optimizations are applied such as batchnorm folding

Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

HW-aware ML Graph Compiler

HW Optimize d ML Kernel Library

Backend opt

Tiling

Sequencer

Scheduler

Sequence gen
**AI Model Compilation: Steps**

**Framework opt**
- Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

**Converter**
- Framework graph is translated to the IR Graph

**Quantizer**
- If required, graph can be quantized according to various config. parameters

**Common Graph optimizations**
- Framework agnostic graph optimizations are applied such as batchnorm folding

**HW-aware ML Graph Compiler**
- Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.
- Break neural network layers into smaller data piece / execution chunks (tiles)

**HW Optimize d ML Kernel Library**
- Backend opt
- Tiling
- Sequencer
- Scheduler
- Sequence gen
**AI Model Compilation: Steps**

1. **Framework opt**
   - Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

2. **Converter**
   - Framework graph is translated to the IR Graph

3. **Quantizer**
   - If required, graph can be quantized according to various config. parameters

4. **Common Graph optimizations**
   - Framework agnostic graph optimizations are applied such as batchnorm folding

5. **Backend opt**
   - Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

6. **Tiling**
   - Break neural network layers into smaller data piece / execution chunks (tiles)

**Scheduler** and **Sequencer** tools dictate order of tile execution to get best performance (completion time) & reduce DDR BW and power. To handle variety of network architectures, different types of sequencers are created. Each Sequencer is composed of cooperating algos & heuristics where some can be non-linear - small changes in networks give different results.
AI Model Compilation: Steps

**Conversion / Quantization**
- Framework opt
- Converter
- Quantizer
- Common Graph optimizations

**HW-aware ML Graph Compiler**
- Backend opt
- Tiling
- Sequencer
- Sequence gen

**Framework level (Pytorch, TF, etc) optimizations, op folding, etc.**

**Framework graph is translated to the IR Graph**

**If required, graph can be quantized according to various config. parameters**

**Framework agnostic graph optimizations are applied such as batchnorm folding**

**Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.**

**Break neural network layers into smaller data piece / execution chunks (tiles)**

To minimize DDR Bandwidth pressure, utilize locality between successive layers to reduce DDR BW. Consider layers:
- Output of layer n is the input of layer (n+1)
- Output of layer (n+1) is the input of layer (n+2)
- Output of Layer (n+2) is divided into four portions
  - Each portion results in a separate into computation 'cones'.
  - Intermediate results within a cone are stored in local TCM – do not consume DDR bandwidth.
AI Model Compilation: Steps

Framework opt

Converter

Quantizer

Common Graph optimizations

- Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
- Framework graph is translated to the IR Graph
- If required, graph can be quantized according to various config. parameters
- Framework agnostic graph optimizations are applied such as batchnorm folding
- Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.
- Break neural network layers into smaller data piece / execution chunks (tiles)

To minimize DDR Bandwidth pressure, utilize locality between successive layers to reduce DDR BW. Consider three sequential layers:

- Output of layer n is the input of layer (n+1)
- Output of layer (n+1) is the input of layer (n+2)
- Output of Layer (n+2) is divided into four portions

Each portion results in a separate into computation 'cones'. Intermediate results within a cone are stored in local TCM – do not consume DDR bandwidth.
**AI Model Compilation: Steps**

1. **Framework opt**
   - Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

2. **Converter**
   - Framework graph is translated to the IR Graph

3. **Quantizer**
   - If required, graph can be quantized according to various config. parameters

4. **Common Graph optimizations**
   - Framework agnostic graph optimizations are applied such as batchnorm folding

---

**HW-aware ML Graph Compiler**

- **Backend opt**
  - Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

- **Tiling**
  - Break neural network layers into smaller data piece / execution chunks (tiles)

- **Sequencer**
  - Define order of execution for each data piece and data movement

- **Scheduler**
  - **Sequence gen**
AI Model Compilation: Steps

**Framework opt**
- Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

**Converter**
- Framework graph is translated to the IR Graph

**Quantizer**
- If required, graph can be quantized according to various config. parameters

**Common Graph optimizations**
- Framework agnostic graph optimizations are applied such as batchnorm folding

**HW-aware ML Graph Compiler**
- Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

- Break neural network layers into smaller data piece / execution chunks (tiles)

- Define order of execution for each data piece and data movement

- Define parallel execution on engines for Performance/BW
### AI Model Compilation: Steps

<table>
<thead>
<tr>
<th>HW-aware ML Graph Compiler</th>
<th>Backend opt</th>
<th>Tiling</th>
<th>Sequencer</th>
<th>Scheduler</th>
<th>Sequence gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW Optimized ML Kernel Library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. **Framework opt**
   - Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

2. **Converter**
   - Framework graph is translated to the IR Graph

3. **Quantizer**
   - If required, graph can be quantized according to various config. parameters

4. **Common Graph optimizations**
   - Framework agnostic graph optimizations are applied such as batchnorm folding

5. **Sequencer**
   - Simplify graph by op fusion, const-prop, common sub expressions, etc.

6. **Scheduler**
   - Break neural network into smaller data pieces

7. **Backend opt**
   - Define order of execution for each data piece and data movement

8. **Sequence gen**
   - Define parallel execution on engines for Performance/BW

   - Generate the optimized list of functions to run on hardware
What order do I execute each operation?

All orders must follow a topological sort.

Very simple network for illustration with only 10 operations

1102 Valid topological sorts for this simple network of 10 operations!

Compiler algos trade-off DDR BW & Performance (latency) for each network.
AI Model Compilation: Optimal Execution Order

Threads, Run Orders, Timelines

Run Order:
A B C

Execution Cycles

Foreground Process
Background Process 1
Background Process 2
**AI Model Compilation: Optimal Execution Order**

Threads, Run Orders, Timelines

Run Order:

A  B  C

Foreground Process

```plaintext
0 1 2 3 4 5 6 7 8
```

Backgroud Process 1

Backgroud Process 2

**Execution Cycles**
AI Model Compilation: Optimal Execution Order

Tiling

Run Order: A B C
AI Model Compilation: Optimal Execution Order

Tiling

Run Order:
A B1 B2 C1 C2 C3 C4

Execution Cycles
AI Model Compilation: Optimal Execution Order

Tiling

Run Order:

A B1 B2 C1 C2 C3 C4

<table>
<thead>
<tr>
<th>Foreground Process</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B1</td>
<td>B2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Background Process</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**AI Model Compilation: Optimal Execution Order**

Scheduling

Run Order:
A  B1  B2  C1  C2  C3  C4

Execution Cycles
AI Model Compilation: Optimal Execution Order

**Run Order:**
A B1 C1 C2 B2 C3 C4

**Execution Cycles**

- **Foreground Process:** A, B1, B2
- **Background Process 1:** C1, C3
- **Background Process 2:** C2, C4

C1 and C2 only depend on B1, so they can be reordered with B2.
AI Model Compilation: Optimal Execution Order

Optimal ordering

Run Order:
A B1 C1 C2 B2 C3 C4

C1 and C2 only depend on B1, so they can be reordered with B2.

Foreground Process

Background Process

Background Process

Execution Cycles

0 1 2 3 4 5 6 7 8
AI Model Performance: inf/sec

**Super resolution (RDN)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

**Face recognition (FaceNet)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

**Bokeh (DeeplabV3+)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

**Natural language processing (MobileBERT)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

*Qualcomm Technologies internal test results*
AI Model Performance: inf/sec per watt

---

**Super resolution (RDN)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

**Face recognition (FaceNet)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

**Bokeh (DeeplabV3+)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

**Natural language processing (MobileBERT)**
- Snapdragon 8 Gen2
- Competitor A
- Competitor B

*Qualcomm Technologies internal test results*