SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration

Thierry Moreau
Mark Wyse
Jacob Nelson
Adrian Sampson

Hadi Esmaeilzadeh
Luis Ceze
Mark Oskin
Approximate Computing

Expose quality-performance trade-offs
Approximate Computing

Expose quality-performance trade-offs

✅ Accurate
❌ Expensive

❌ Approximate
✅ Cheap
Approximate Computing

Expose quality-performance trade-offs

✅ Accurate  ✗ Expensive  ✗ Approximate  ✅ Cheap

Domains include image processing, machine learning, search, physical simulation, multimedia etc.
float foo (float a, float b) {
    ...
    return val;
}
Neural Acceleration

float foo (float a, float b) {
    ... return val;
}

approximation  acceleration

Esmaeilzadeh et al.
[MICRO 2012]
SNAP

Programmable SoCs

3.8x speedup and 2.8x efficiency gains

offers an alternative to HLS tools for neural acceleration
Talk Outline

Introduction

Programming model

SNNAP design:

• Efficient neural network evaluation

• Low-latency communication

Evaluation & Comparison with HLS
Background:Compilation

1. Region detection
   code annotation
   region detection & program instrumentation

2. ANN Training
   [training.data] → back prop. & topology search

3. Code Generation
   binary generation
   SNNAP
   CPU
Programming Model

float sobel (float* p);
...

Image src;
Image dst;

while (true) {
    src = read_from_camera();
    for (y=0; y < h; ++y) {
        for (x=0; x < w; ++x) {
            dst.p[y][x] = sobel(& src.p[y][x]);
        }
    }
    display(dst);
}
Programming Model

```c
APPROX float sobel (APPROX float* p);
.
.
APPROX Image src;
APPROX Image dst;

while (true) {
    src = read_from_camera();
    for (y=0; y < h; ++y) {
        for (x=0; x < w; ++x) {
            dst.p[y][x] = sobel(& src.p[y][x]);
        }
    }
    display(dst);
}
```

✅ no side effects
✅ executes often

ACCEPT: compilation framework for approximate programs
Talk Outline

Introduction

Programming model

SNNAP design:

- Efficient neural network evaluation
- Low-latency communication

Evaluation & Comparison with HLS
Background: Multi-Layer Perceptrons

neural network

computing a single layer

\[
\begin{bmatrix}
  x_7 \\
  x_8 \\
  x_9 \\
\end{bmatrix} = f \left( \begin{bmatrix}
  W_{67} & W_{57} & W_{47} \\
  W_{68} & W_{58} & W_{48} \\
  W_{69} & W_{59} & W_{49} \\
\end{bmatrix} \begin{bmatrix}
  x_6 \\
  x_5 \\
  x_4 \\
\end{bmatrix} \right)
\]

activation function \( f \)
Background: Systolic Arrays

computing a single layer

\[
\begin{bmatrix}
  x_7 \\
  x_8 \\
  x_9
\end{bmatrix}
= f
\left(\begin{bmatrix}
  w_{67} & w_{57} & w_{47} \\
  w_{68} & w_{58} & w_{48} \\
  w_{69} & w_{59} & w_{49}
\end{bmatrix}\begin{bmatrix}
  x_6 \\
  x_5 \\
  x_4
\end{bmatrix}\right)
\]
Background: Systolic Arrays

systolic array

\[ \begin{array}{c}
  x_6 \\
  x_5 \\
  x_4 \\
  W_{49} \ W_{48} \ W_{47} \\
  W_{59} \ W_{58} \ W_{57} \\
  W_{69} \ W_{68} \ W_{67} \\
  f \\
  x_7 \\
  x_8 \\
  x_9 
\end{array} \]
PU Micro-Architecture

systolic array

processing unit

1 - processing elements in DSP logic

2 - local storage for synaptic weights

3 - sigmoid unit implements non-linear activation functions

4 - vertically micro-coded sequencer
Multi-Processing Units

AXI Master

scheduler

bus

PU
control
Storage
PE
PE
PE
PE
f

PU
control
Storage
PE
PE
PE
PE
f

PU
control
Storage
PE
PE
PE
PE
f

PU
control
Storage
PE
PE
PE
PE
f
Talk Outline

Introduction

Programming model

SNNAP design:

• Efficient neural network evaluation

  • Low-latency communication

Evaluation & Comparison with HLS
CPU-SNNAP Integration

Interface requirements:
- Low-latency data transfer
- Fast signaling
CPU-SNNAP Integration

Interface requirements:
- Low-latency data transfer
- Fast signaling
CPU-SNNAP Integration

Interface requirements:
- Low-latency data transfer
- Fast signaling

coherent reads & writes with accelerator coherency port

custom mastering interface

ACP

DMA master

scheduler

bus

PU
PU
PU
PU
PU

CPU

$L1$

$L2$
CPU-SNNAP Integration

Interface requirements:
- Low-latency data transfer
- Fast signaling

- coherent reads
- writes
    with accelerator
coherecy port

- custom
    mastering
interface

- low-latency
  event
  signaling,
  sleep &
wakeup

- SEV
- WFE

- ACP

- $L1

- $L2

- bus

DMA master

scheduler

CPU

PU
PU
PU
PU
PU
Talk Outline

Introduction

Programming model

SNNAP design:

- Efficient neural network evaluation
- Low-latency communication

Evaluation & Comparison with HLS
Evaluation

Neural acceleration on SNNAP (8x8 configuration, clocked at 1/4 of $f_{\text{CPU}}$) vs. precise CPU execution
Evaluation

Neural acceleration on SNNAP (8x8 configuration, clocked at 1/4 of $f_{CPU}$) vs. precise CPU execution

<table>
<thead>
<tr>
<th>application</th>
<th>domain</th>
<th>error metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>option pricing</td>
<td>MSE</td>
</tr>
<tr>
<td>fft</td>
<td>DSP</td>
<td>MSE</td>
</tr>
<tr>
<td>inversek2j</td>
<td>robotics</td>
<td>MSE</td>
</tr>
<tr>
<td>jmeint</td>
<td>3D-modeling</td>
<td>miss rate</td>
</tr>
<tr>
<td>jpeg</td>
<td>compression</td>
<td>image diff</td>
</tr>
<tr>
<td>kmeans</td>
<td>ML</td>
<td>image diff</td>
</tr>
<tr>
<td>sobel</td>
<td>vision</td>
<td>image diff</td>
</tr>
</tbody>
</table>
Speedup

Factors:
- Amdahl’s Speedup
- Cost of instructions on CPU vs. cost of NN on SNNAP
Speedup

Factors:
- Amdahl’s Speedup
- Cost of instructions on CPU vs. cost of NN on SNNAP

<table>
<thead>
<tr>
<th>Function</th>
<th>Speedup</th>
<th>Amdahl’s Speedup</th>
<th>CPU cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>inversek2j</td>
<td>10.8</td>
<td>&gt;100x</td>
<td>1660 cycles</td>
</tr>
<tr>
<td>bscholes</td>
<td>2.7</td>
<td>1.47x</td>
<td>29 cycles</td>
</tr>
<tr>
<td>jpeg</td>
<td>38.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jmeint</td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jpeg</td>
<td>2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kmeans</td>
<td>1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sobel</td>
<td>2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEOMEAN</td>
<td>3.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Energy Savings

Energy = Power on (DRAM + SoC) * Runtime +36%
HW Acceleration

Neural Acceleration with SNNAP

vs.

High Level Synthesis Compilers

which one should you use?
HLS Comparison Study

HLS

neural transform

NN executes compiled down

FPGA design

SNNAP

netlist compiled down

HLS

HLS compiled down netlist neural transform
HLS Comparison Study

Resource-normalized throughput:
- pipeline invocation interval
- maximum frequency
- resource utilization
HLS Comparison Study

Normalized Throughput Improvement over HLS

- bsoholes: 1.6
- fft: 0.4
- inversek2j: 1.3
- jmeint: 7.9
- jpeg: 43.7
- kmeans: 0.2
- sobel: 0.5
- GEOMEAN: 1.6

Neural Acceleration is better

HLS is better
HLS Comparison Study

Normalized Throughput Improvement over HLS

<table>
<thead>
<tr>
<th>Function</th>
<th>Neural Accel.</th>
<th>HLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>bscholes</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>fft</td>
<td></td>
<td>.4</td>
</tr>
<tr>
<td>inversek2i</td>
<td></td>
<td>1.3</td>
</tr>
<tr>
<td>jmeint</td>
<td></td>
<td>7.9</td>
</tr>
<tr>
<td>jpeg</td>
<td></td>
<td>43.7</td>
</tr>
<tr>
<td>kmeans</td>
<td></td>
<td>.2</td>
</tr>
<tr>
<td>sobel</td>
<td></td>
<td>.5</td>
</tr>
<tr>
<td>GEOMEAN</td>
<td></td>
<td>1.6</td>
</tr>
</tbody>
</table>

- Precision: ✓
- Virtualization: ✓
- Performance: 
- Programmability: 

Neural Accel. vs HLS Comparison study results.
HLS Comparison Study

Normalized Throughput Improvement over HLS

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Normalized Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>bscholes</td>
<td>1.6</td>
</tr>
<tr>
<td>fft</td>
<td>0.4</td>
</tr>
<tr>
<td>inversek2j</td>
<td>1.3</td>
</tr>
<tr>
<td>jmeint</td>
<td>7.9</td>
</tr>
<tr>
<td>jpeg</td>
<td>43.7</td>
</tr>
<tr>
<td>kmeans</td>
<td>0.2</td>
</tr>
<tr>
<td>sobel</td>
<td>0.5</td>
</tr>
<tr>
<td>GEOMEAN</td>
<td>1.6</td>
</tr>
</tbody>
</table>

- **Neural Accel.**
- **HLS**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Neural Accel.</th>
<th>HLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>✅</td>
<td>✓</td>
</tr>
<tr>
<td>Virtualization</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Performance</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Programmability</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Conclusion

SNNAP: apply approximate computing on programmable SoCs through neural acceleration

3.8x speedup & 2.8x energy savings

neural acceleration is a viable alternative to HLS
SNNAP:
Approximate Computing on Programmable SoCs via Neural Acceleration

Thierry Moreau: moreau@uw.edu

Mark Wyse
Jacob Nelson
Adrian Sampson

Hadi Esmaeilzadeh
Luis Ceze
Mark Oskin

http://sampa.cs.washington.edu/