General-Purpose Code Acceleration with Limited-Precision Analog Computation

Renée St. Amant  Amir Yazdanbakhsh  Jongse Park  Bradley Thwaites
Hadi Esmaeilzadeh  Arjang Hassibi  Luis Ceze  Doug Burger

Georgia Institute of Technology
Alternative Computing Technologies (ACT) Lab

Georgia Institute of Technology  The University of Texas at Austin
University of Washington  Microsoft Research

ISCA 2014
Input and Output
Display
Communication
Sensing

Analog Domain

Processing
Storage

Digital Domain

Analog Accelerator

DAC
ADC

DAC
ADC

DAC
ADC

ADC
DAC

GPU
CPU
DSP Unit
Memory

DAC
ADC

CPU
GPU
DSP Unit
Memory
How to use analog circuits for accelerating programs written in conventional languages?

1) Neural transformation
   [Esmaeilzadeh et. al., MICRO 2012]

2) Analog neurons

3) Compiler-circuit co-design
Challenges

- Analog circuits are mainly single function
- Instruction control cannot be analog
- Storing intermediate results in analog domain is not effective
- Analog circuits have limited operational range

1) Neural transformation

2) Analog neurons

3) Compiler-circuit co-design
Challenges

- Analog circuits are mainly single function
- Instruction control cannot be analog
- Storing intermediate results in analog is not effective
- Analog circuits have limited operational range

1) Neural transformation

2) **Analog neurons**

3) Compiler-circuit co-design
Challenges

- Analog circuits are mainly single function
- Instruction control cannot be analog
- Storing intermediate results in analog domain is not effective
- Analog circuits have limited operational range

1) Neural transformation

2) Analog neurons

3) Compiler-circuit co-design
1st Design Principle

Neural Transformation
Neural Transformation

A-NPU acceleration

Source Codes → Common Intermediate Representation → Acceleration

Code_1 → Code_2 → Code_3 → Code_4 → Code_5 → Code_6 → ... → Neural Representation

CPU → A-NPU
2\textsuperscript{nd} Design Principle

Analog Neurons
Analog Neurons for Accelerated Computation

\[ y = \text{sigmoid} \left( \sum (x_i w_i) \right) \]

\[ y \approx \text{sigmoid} \left( \sum (I(x_i) R(w_i)) \right) \]
Mixed-signal A-NPU

Row Selector

Weight Buffer

8-Wide Analog Neuron

Weight Buffer

8-Wide Analog Neuron

Weight Buffer

8-Wide Analog Neuron

Weight Buffer

8-Wide Analog Neuron

Column Selector

Input FIFO

Config FIFO

Output FIFO
\[ I(|x_0|) \]
\[ R(|w_0|) \]
\[ V(|w_0x_0|) \]
\[ I^+(w_0x_0) \]
\[ I^-(w_0x_0) \]
\[ V^+ \left( \sum w_ix_i \right) \]
\[ V^- \left( \sum w_ix_i \right) \]
\[ I^+(w_nx_n) \]
\[ I^-(w_nx_n) \]
\[ y \approx \text{sigmoid} \left( V \left( \sum w_ix_i \right) \right) \]
Limitations of Analog Neuron

Limited range of operation (e.g. 600mV)

Margins for noise resiliency (2-3 mV)

Limited Bit-width
Topology Restriction
Circuit Non-idealities (e.g., Sigmoid)
3rd Design Principle

Compiler-Circuit Co-design
Digital Compilation Workflow

- Source Code
- Programmer
- Source Code + Annotations
- Compiler + Training Algorithm
- Accelerator Config
- Instrumented Binary
- D-NPU
- CORE

Programming (Profiling, Training, Code Generation) → Compilation → Execution
Analog Compilation Workflow

Source Code

Compiler + Customized Training Algorithm

Accelerator Config
Instrumented Binary

Limited Bit-Width Topology Restriction Circuit Non-idealities

A-NPU
CORE

Programming (Profiling, Training, Code Generation)

Compilation

Execution
(1) Training with Limited Bit-width

Limited-Precision Network

Fully-Precise Network

Train a fully-precise neural network

Input the training data to the discretized neural network

Calculate the output error from the limited-precision neural network

Back propagate the error through the fully-precise neural network

Continuous-Discrete Learning Method (CDLM), E. Fiesler, 1990
(2) Training with topology restrictions and non-idealities

1) **Robust** to the topology restrictions

2) Tolerate a more **shallow sigmoid** activation steepness over all applications

Resilient Back Propagation (RPROP), M. Riedmiller, 1993
Measurements
Signal Processing, Robotics, 3D Gaming, Financial Analysis, Compression, Machine Learning, Image Processing

Analog A-NPU with 8 Analog Neurons
- Transistor-Level HSPICE Simulation
- Predictive Technology Models (PTM), 45nm
- Vdd: 1.2 V, f: 1.1 GHz

Digital Components
- Power Models: McPAT, CACTI, and Verilog

Processor Simulator
- Marssx86 Cycle-Accurate Simulation
- Intel Nehalem-like 4-wide/5-issue OoO processor
- Technology: 45 nm, Vdd: 0.9 V, f: 3.4 GHz
Ranges from $0.8\times$ to $24.5\times$ with Analog NPU

$1.2\times$ increase in application speedup with Analog over Digital NPU
Energy Savings

<table>
<thead>
<tr>
<th>Application</th>
<th>Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>51.2</td>
</tr>
<tr>
<td>fft</td>
<td>30.0</td>
</tr>
<tr>
<td>inversek2j</td>
<td>17.8</td>
</tr>
<tr>
<td>jmeint</td>
<td>42.5</td>
</tr>
<tr>
<td>jpeg</td>
<td>25.8</td>
</tr>
<tr>
<td>kmeans</td>
<td>17.8</td>
</tr>
<tr>
<td>sobel</td>
<td>30.0</td>
</tr>
<tr>
<td>geomean</td>
<td>6.3</td>
</tr>
</tbody>
</table>

Energy saving with Analog NPU is very close to ideal case (6.5x)
Application quality loss

Quality loss is below 10% in all cases but one
Based on application-specific quality metric
What is left?

3% Energy Reduction

46% Speedup

We can not reduce the energy of the computation much more.
3.7x × 6.3x

Speedup

Kirchhoff's Law

\[ I_{out} = I_0 + I_1 + I_2 \]

Energy Reduction

\[ I(x_n) + V_o - R(w_n) \]

\[ V_o = I(x_n) \cdot R(w_n) \]

\approx 23x

Energy-Delay Product

Quality Degradation: Avg. 8.2%, Max. 19.7%

Saturation Property of Transistors
It is still the beginning...

1) **Broad applicability** of the analog computation

2) Prototyping and integrating A-NPU within **noisy** high performance processors

3) Reasoning about the **acceptable level of error** at the programming level
Backup Slides
# Area Breakdown

<table>
<thead>
<tr>
<th>Sub-circuit</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A-NPU</strong></td>
<td></td>
</tr>
<tr>
<td>8x8-bit DAC</td>
<td>3,096 T</td>
</tr>
<tr>
<td>8xResistor Ladder (8-bit weights)</td>
<td>4,096 T + 1 KΩ  (≈ 450 T)</td>
</tr>
<tr>
<td>8xDifferential Pair</td>
<td>48 T</td>
</tr>
<tr>
<td>I-to-V Resistors</td>
<td>20 KΩ  (≈ 30 T)</td>
</tr>
<tr>
<td>Differential Amplifier</td>
<td>244 T</td>
</tr>
<tr>
<td>8-bit ADC</td>
<td>2,550 T + 1KΩ   (≈ 450)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>≈10,964 T</td>
</tr>
<tr>
<td><strong>D-NPU</strong></td>
<td></td>
</tr>
<tr>
<td>8x8-bit multiply-adds</td>
<td>≈56,000 T</td>
</tr>
<tr>
<td>8-bit Sigmoid lookup table</td>
<td>16,456 T</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>≈72,456</td>
</tr>
</tbody>
</table>

6.6x fewer transistors in the analog neuron implementation
## Power Breakdown

<table>
<thead>
<tr>
<th>Sub-circuit</th>
<th>Percentage of total power</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A-NPU</strong></td>
<td></td>
</tr>
<tr>
<td>SRAM-accesses</td>
<td>13%</td>
</tr>
<tr>
<td>DAC-Resistor Ladder-Diff Pair-Sum</td>
<td>54%</td>
</tr>
<tr>
<td>Sigmoid-ADC</td>
<td>33%</td>
</tr>
</tbody>
</table>

*Power numbers vary with applications*
<table>
<thead>
<tr>
<th>Application</th>
<th>Instructions</th>
<th>Dynamic Instructions</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signal Processing</strong></td>
<td>34 x86</td>
<td>67.4%</td>
<td>1 → 4 → 4 → 2</td>
</tr>
<tr>
<td><strong>Compression</strong></td>
<td>1,257 x86</td>
<td>56.3%</td>
<td>64 → 16 → 8 → 64</td>
</tr>
<tr>
<td><strong>Robotics</strong></td>
<td>100 x86</td>
<td>95.9%</td>
<td>2 → 8 → 2</td>
</tr>
<tr>
<td><strong>Machine Learning</strong></td>
<td>26 x86</td>
<td>29.7%</td>
<td>6 → 8 → 4 → 1</td>
</tr>
<tr>
<td><strong>3D Gaming</strong></td>
<td>1,079 x86</td>
<td>95.1%</td>
<td>18 → 32 → 8 → 2</td>
</tr>
<tr>
<td><strong>Image Processing</strong></td>
<td>88 x86</td>
<td>57.1%</td>
<td>9 → 8 → 1</td>
</tr>
<tr>
<td><strong>Financial</strong></td>
<td>309 x86</td>
<td>97.2%</td>
<td>6 → 8 → 8 → 1</td>
</tr>
<tr>
<td><strong>Robo2cs inversek2j</strong></td>
<td></td>
<td></td>
<td>2 → 8 → 2</td>
</tr>
<tr>
<td><strong>Machine Learning</strong></td>
<td>26 x86</td>
<td>29.7%</td>
<td>6 → 8 → 4 → 1</td>
</tr>
<tr>
<td><strong>Image Processing</strong></td>
<td>88 x86</td>
<td>57.1%</td>
<td>9 → 8 → 1</td>
</tr>
</tbody>
</table>
3.3× geometric mean speedup
Ranges from 1.8× to 15.2×
Energy savings with A-NPU over 8-bit D-NPU

12.1× geometric mean speedup
Ranges from 3.7× to 82.2×
Dynamic Instruction Reduction

Percentage of Instructions Subsumed

blackscholes, fft, inversek2j, jmeint, jpeg, kmeans, sobel, geomean

66.4%
Speedup with A-NPU acceleration

3.7× geometric mean speedup
Ranges from 0.8× to 24.5×
Energy savings with A-NPU acceleration

6.3× geometric mean energy reduction
All benchmarks benefit