Neural Acceleration for General-Purpose Approximate Programs

Hadi Esmaeilzadeh
Adrian Sampson
Luis Ceze
Doug Burger

University of Washington
Microsoft Research

University of Washington  MICRO 2012
Program

CPU
<table>
<thead>
<tr>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer vision</td>
</tr>
<tr>
<td>machine learning</td>
</tr>
<tr>
<td>sensory data</td>
</tr>
<tr>
<td>physical simulation</td>
</tr>
<tr>
<td>information retrieval</td>
</tr>
<tr>
<td>augmented reality</td>
</tr>
<tr>
<td>image rendering</td>
</tr>
</tbody>
</table>
computer vision
machine learning
sensory data
physical simulation
information retrieval
augmented reality
image rendering

Approximate computing

Probabilistic CMOS designs
[Rice, NTU, Georgia Tech…]

Stochastic processors
[Illinois]

Code perforation transformations
[MIT]

Relax software fault recovery
[de Kruijf et al., ISCA 2010]

Green runtime system
[Baek and Chilimbi, PLDI 2010]

Flikker approximate DRAM
[Liu et al., ASPLOS 2011]

EnerJ programming language
[PLDI 2011]

Truffle dual-voltage architecture
[ASPLOS 2012]
Accelerators

- Conservation
- Cores
- UCSD

- DySER
- Wisconsin

- CPU

- BERET
- Michigan

- GPU

- FPGA

- Vector Unit

- UCSD
Accelerators

- Conservation Cores
  - UCSD
- DySER
  - Wisconsin
- Vector Unit

Approximate computing

- computer vision
- machine learning
- sensory data
- physical simulation
- information retrieval
- augmented reality
- image rendering
An accelerator for approximate computations

✓ Mimics functions written in traditional languages!

✓ Runs more efficiently than a CPU or a precise accelerator!

✓ May introduce small errors!
Neural networks are function approximators

Trainable: implements many functions

Highly parallel

Very efficient hardware implementations

Fault tolerant

[Temam, ISCA 2012]
Neural acceleration

Program
Neural acceleration

Program

Annotate an approximate program component
Neural acceleration

- **Annotate** an approximate program component
- **Compile** the program and train a neural network

Program
Neural acceleration

- **Annotate** an approximate program component
- **Compile** the program and train a neural network
- **Execute** on a fast Neural Processing Unit (NPU)
Neural acceleration

1. **Annotate** an approximate program component
2. **Compile** the program and train a neural network
3. **Execute** on a fast Neural Processing Unit (NPU)
4. **Improve** performance 2.3x and energy 3.0x on average
Programming model

[[transform]]

float grad(float[3][3] p) {
    ...
}

void edgeDetection(Image &src, Image &dst) {
    for (int y = ...) {
        for (int x = ...) {
            dst[x][y] =
                grad(window(src, x, y));
        }
    }
}
Code region criteria

- ✔ Hot code
- ✔ Approximable
- ✔ Well-defined inputs and outputs

\[ \text{grad()} \]

run on every 3x3 pixel window

small errors do not corrupt output

takes 9 pixel values; returns a scalar
Empirically selecting target functions
Compiling and transforming

1. Code Observation

2. Training

3. Code Generation

- Annotated Source Code
- Training Inputs
- Trained Neural Network
- Augmented Binary
Code observation

```c
[[NPU]]
float grad(float[][3] p) {
    ...
}

void edgeDetection(Image &src, Image &dst) {
    for (int y = ...) {
        for (int x = ...) {
            dst[x][y] = grad(window(src, x, y));
        }
    }
}
```

Record (p);  record(result);

Test cases + Instrumented program = Sample arguments & outputs
Training

Backpropagation Training
Training

- Training Inputs
  - 70%
  - faster
  - less robust

- Training Inputs
  - 98%
  - slower
  - more accurate

- Training Inputs
  - 99%
void edgeDetection(Image &src, Image &dst) {
    for (int y = ...) {
        for (int x = ...) {
            p = window(src, x, y);
            NPU_SEND(p[0][0]);
            NPU_SEND(p[0][1]);
            NPU_SEND(p[0][2]);
            ...
            dst[x][y] = NPU_RECEIVE();
        }
    }
}
Neural Processing Unit (NPU)
Software interface: ISA extensions

Core

enq.d → input

← deq.d → output

enq.c → configuration

← deq.c →

NPU
Microarchitectural interface

Fetch
Decode
Issue
Execute
Memory
Commit

NPU

enq.d

S

NS

deq.d

S

NS

enq.c

configuration

deq.c
A digital NPU

Bus Scheduler

Processing Engines

input
output
A digital NPU

- Multiply-add unit
- Neuron weights
- Accumulator
- Sigmoid LUT

Processing Engines
Experiments

Several benchmarks; annotated **one hot function** each
FFT, inverse kinematics, triangle intersection, JPEG, K-means, Sobel

Simulated full programs on **MARSSx86**
Energy modeled with **McPAT** and **CACTI**
Microarchitecture like Intel Penryn: 4-wide, 6-issue
45 nm, 2080 MHz, 0.9 V
Two benchmarks

**edge detection**
- 88 static instructions
- 56% of dynamic instructions

18 neurons

**triangle intersection**
- 1,079 static x86-64 instructions
- 97% of dynamic instructions

60 neurons
- 2 hidden layers
Speedup with NPU acceleration

2.3x average speedup
Ranges from 0.8x to 11.1x
Energy savings with NPU acceleration

3.0x average energy reduction

All benchmarks benefit
Application quality loss

Quality loss below 10% in all cases
Based on application-specific quality metrics
Edge detection with gradient calculation on NPU
Also in the paper

Sensitivity to communication latency
Sensitivity to NN evaluation efficiency
Sensitivity to PE count
Benchmark statistics
All-software NN slowdown
Program
Program
Neural networks can efficiently approximate functions from programs written in conventional languages.
CPU

low power

parallel

flexible

regular

fault-tolerant

analog
Normalized dynamic instructions

FFT: 40%
Inverse k2j: 10%
Jmeint: 20%
JPEG: 30%
K-means: 70%
Sobel: 50%
Geometric mean: 45%

NPU queue instructions
Other instructions
Slowdown with software NN

20x average slowdown
Using off-the-shelf FANN library