Neural Acceleration for GPU Throughput Processors

Amir Yazdanbakhsh  Jongse Park  Hardik Sharma
Pejman Lotfi-Kamran*  Hadi Esmaeilzadeh

Alternative Computing Technologies (ACT) Lab
Georgia Institute of Technology

*The Institute for Research in Fundamental Sciences

NGPU
Neurally Accelerated GPU
Approximate computing
Embracing imprecision

Relax the abstraction of “near perfect” accuracy in

Accept imprecision to improve
performance
energy dissipation
resource utilization efficiency
Opportunity

Many GPU applications are amenable to approximation

Augmented Reality
Computer Vision
Robotics
Machine Learning
Sensor Processing
Multimedia
More than 55% of application runtime and energy is in neurally approximable regions
Neural Transformation for GPUs
Challenges

Many Core
Challenges

Many Core
Simple In Order
Challenges

- Many Core
- Simple In Order
- Data-level Parallelism
Augmenting the CPU based neural processing units to each SIMD lane imposes 31.2% area overhead.
NGPU

Neurally-Accelerated GPU Architecture

Fetch
Decode
Issue
Operand Collection
Active Mask
SIMT Stack

src_reg3 src_reg2 src_reg1 dst_reg

SIMD Lane
LSU Pipeline
Write Back
D-$

Memory Partition
L2 Cache
Off-chip DRAM

Interconnection Network
Streaming Multiprocessor (SM)
Neuronal Network Operations

\[ y_j = \text{sigmoid} \left( w_{j,0} \times x_{j,0} + \ldots + w_{j,i} \times x_{j,i} + \ldots + w_{j,n} \times x_{j,n} \right) \]
NGPU

Neurally-Accelerated GPU Architecture

1. Weight FIFO
2. Controller
3. Input FIFO
4. Output FIFO
5. Sig. Unit
6. Acc Reg

SIMD Lane
NGPU reuses the existing ALU in each SIMD lane
NGPU
Neurally-Accelerated GPU Architecture

Weight FIFO is shared among all the SIMD lanes
Overall NGPU has \( \leq 1\% \) area overhead
NGPU Execution Model

ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;

Neurally Accelerated GPU Application

Neural Network
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;

\[ \begin{align*}
& w_0 \times (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
& + w_2 \times (\text{in}_1, \text{in}_1, \ldots, \text{in}_1) \\
& \text{sigmoid} \left( \tilde{\text{in}}_0, \ldots, \tilde{\text{in}}_0 \right) \\
& w_1 \times (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
& + w_3 \times (\text{in}_1, \text{in}_1, \ldots, \text{in}_1) \\
& \text{sigmoid} \left( \tilde{\text{in}}_1, \ldots, \tilde{\text{in}}_1 \right) \\
& w_4 \times (\text{n}_0, \text{n}_0, \ldots, \text{n}_0) \\
& + w_5 \times (\text{n}_1, \text{n}_1, \ldots, \text{n}_1) \\
& \text{sigmoid} \left( \tilde{\text{out}}_0, \ldots, \tilde{\text{out}}_0 \right) \\
& \left( \text{out}_0, \text{out}_0, \ldots, \text{out}_0 \right)
\end{align*} \]
NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;
```

SIMD lanes are in normal mode and performs precise computation
NGPU Execution Model

ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;

\[
\begin{align*}
\text{w}_0 \times & (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
+ \text{w}_2 \times & (\text{in}_1, \text{in}_1, \ldots, \text{in}_1) \\
\text{sigmoid} & \left( \begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array} \right) \\
\text{w}_1 \times & (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
+ \text{w}_3 \times & (\text{in}_1, \text{in}_1, \ldots, \text{in}_1) \\
\text{sigmoid} & \left( \begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array} \right) \\
\text{w}_4 \times & (\text{n}_0, \text{n}_0, \ldots, \text{n}_0) \\
+ \text{w}_5 \times & (\text{n}_1, \text{n}_1, \ldots, \text{n}_1) \\
\text{sigmoid} & \left( \begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array} \right) \\
\left( \text{out}_0, \text{out}_0, \ldots, \text{out}_0 \right) & \left( \begin{array}{c} \vdots \\ \vdots \end{array} \right)
\end{align*}
\]

SIMD lanes enter neural mode
NGPU Execution Model

```assembly
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;
```

SIMD starts the calculation of the neural network
NGPU Execution Model

The neurally accelerated SIMD lanes autonomously calculate the neural outputs in lock-step.
NGPU Execution Model

ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;

\[
\begin{align*}
& w_0 \times (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
& + w_2 \times (\text{in}_1, \text{in}_1, \ldots, \text{in}_1) \\
& \text{sigmoid} \\
& w_1 \times (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
& + w_3 \times (\text{in}_1, \text{in}_1, \ldots, \text{in}_1) \\
& \text{sigmoid} \\
& w_4 \times (\text{n}_0, \text{n}_0, \ldots, \text{n}_0) \\
& + w_5 \times (\text{n}_1, \text{n}_1, \ldots, \text{n}_1) \\
& \text{sigmoid} \\
& (\text{out}_0, \text{out}_0, \ldots, \text{out}_0)
\end{align*}
\]

The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step.
NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
```

```
sta.global [addr2], %r2;
```

The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step
NGPU Execution Model

The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step.
NGPU Execution Model

ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;

st.global [addr2], %r2;

The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step
NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
```

```
\[
\begin{align*}
& \text{w}_0 \times (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
& + \text{w}_2 \times (\text{in}_1, \text{in}_1, \ldots, \text{in}_1)
\end{align*}
\]

sigmoid

```
\[
\begin{align*}
& \text{w}_1 \times (\text{in}_0, \text{in}_0, \ldots, \text{in}_0) \\
& + \text{w}_3 \times (\text{in}_1, \text{in}_1, \ldots, \text{in}_1)
\end{align*}
\]

sigmoid

```
\[
\begin{align*}
& \text{w}_4 \times (\text{n}_0, \text{n}_0, \ldots, \text{n}_0) \\
& + \text{w}_5 \times (\text{n}_1, \text{n}_1, \ldots, \text{n}_1)
\end{align*}
\]

sigmoid

```
\[
\begin{align*}
& (\text{out}_0, \text{out}_0, \ldots, \text{out}_0)
\end{align*}
\]

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
NGPU Execution Model

The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step.
NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;
```

The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step
NGPU Execution Model

The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step.
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;

SIMD lanes exit neural mode
NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;
```

SIMD lanes are in normal mode
Experimental Setup

Machine Learning, Finance, Vision
3D Gaming, Medical Imaging
Numerical Analysis, Image Processing

**GPU Simulator**

- GPGPUSim Cycle-Level Simulator
- Fermi-based GTX 480, Shader Core Frequency 1.4 GHz
- NVCC Compiler –O3

**Power Model**

- Technology Node 40 nm
- GPUWattch
- McPAT and CACTI, Verilog
Most applications see speedup with NGPU.
The speedup for **bandwidth-sensitive** applications is limited.
Bandwidth-sensitive applications see speedup with 2x bandwidth.
NGPU Energy Savings with Baseline Bandwidth

NGPU eliminates the von Neumann overhead which results in energy reduction
Energy Reduction with Baseline Bandwidth

NGPU Energy Savings with Baseline Bandwidth

Even bandwidth-sensitive applications see energy saving
Quality loss is below 10% in all cases
NGPU is a Fair Bargain

**Overhead**
- Area Overhead ≤ 1.0%
- Quality ≥ 97.5%
- Quality ≥ 90.0%

**Benefits**
- 1.9× Speedup
- 2.1× Energy Reduction

- 2.4× Speedup
- 2.8× Energy Reduction
Backup Slides
## Overheads

<table>
<thead>
<tr>
<th>GNPU Components</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight FIFO</td>
<td>2 KB / SM</td>
</tr>
<tr>
<td>Input FIFO</td>
<td>256 bytes / SIMD Lane</td>
</tr>
<tr>
<td>Output FIFO</td>
<td>256 bytes / SIMD Lane</td>
</tr>
<tr>
<td>Sigmoid LUT</td>
<td>2048 x 32-bit / SIMD Lane</td>
</tr>
</tbody>
</table>
### Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Application</th>
<th>PTX Instructions</th>
<th>Instruction Pipeline</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Processing</td>
<td>Binarization</td>
<td>27</td>
<td>3 → 4 → 2 → 1</td>
<td>8.23%</td>
</tr>
<tr>
<td>Finance</td>
<td>BlackScholes</td>
<td>96</td>
<td>6 → 8 → 1</td>
<td>4.35%</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>Convolution</td>
<td>886</td>
<td>17 → 2 → 1</td>
<td>5.25%</td>
</tr>
<tr>
<td>Robotics</td>
<td>InverseK2J</td>
<td>132</td>
<td>2 → 16 → 3</td>
<td>8.73%</td>
</tr>
<tr>
<td>3D Gaming</td>
<td>Jmeint</td>
<td>2,250</td>
<td>18 → 8 → 2</td>
<td>9.70%</td>
</tr>
<tr>
<td>Image Processing</td>
<td>Laplacian</td>
<td>51</td>
<td>9 → 2 → 1</td>
<td>6.01%</td>
</tr>
<tr>
<td>Machine Vision</td>
<td>MeanFilter</td>
<td>35</td>
<td>7 → 4 → 1</td>
<td>7.06%</td>
</tr>
<tr>
<td>Numerical Analysis</td>
<td>Newton-Raphson</td>
<td>44</td>
<td>5 → 2 → 1</td>
<td>3.08%</td>
</tr>
<tr>
<td>Image Processing</td>
<td>Sobel</td>
<td>86</td>
<td>9 → 4 → 1</td>
<td>5.45%</td>
</tr>
<tr>
<td>Medical Imaging</td>
<td>Srad</td>
<td>110</td>
<td>5 → 4 → 1</td>
<td>7.43%</td>
</tr>
</tbody>
</table>
Motivation

Runtime Breakdown

- binarization
- blackscholes
- convolution
- inversek2j
- jmeint
- laplacian
- meanfilter
- newton-raph
- sobel
- sram
- gmean

Approximable
Non-Approximable
Motivation

Energy Breakdown

<table>
<thead>
<tr>
<th>Function</th>
<th>Non-Approximable</th>
<th>Approximable</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarization</td>
<td>30%</td>
<td>70%</td>
</tr>
<tr>
<td>blackscholes</td>
<td>40%</td>
<td>60%</td>
</tr>
<tr>
<td>convolution</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>inversek2j</td>
<td>60%</td>
<td>40%</td>
</tr>
<tr>
<td>jmeint</td>
<td>70%</td>
<td>30%</td>
</tr>
<tr>
<td>laplacian</td>
<td>80%</td>
<td>20%</td>
</tr>
<tr>
<td>meanfilter</td>
<td>90%</td>
<td>10%</td>
</tr>
<tr>
<td>newton-raph</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>sobel</td>
<td>95%</td>
<td>5%</td>
</tr>
<tr>
<td>sram</td>
<td>99%</td>
<td>1%</td>
</tr>
<tr>
<td>gmean</td>
<td>98%</td>
<td>2%</td>
</tr>
</tbody>
</table>
Motivation

Software Implementation

<table>
<thead>
<tr>
<th>Function</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarization</td>
<td>10</td>
</tr>
<tr>
<td>blackscholes</td>
<td></td>
</tr>
<tr>
<td>convolution</td>
<td></td>
</tr>
<tr>
<td>inversek2j</td>
<td></td>
</tr>
<tr>
<td>jmeint</td>
<td></td>
</tr>
<tr>
<td>laplacian</td>
<td></td>
</tr>
<tr>
<td>meanfilter</td>
<td></td>
</tr>
<tr>
<td>newton-raph</td>
<td></td>
</tr>
<tr>
<td>sobel</td>
<td></td>
</tr>
<tr>
<td>sram</td>
<td></td>
</tr>
<tr>
<td>gmean</td>
<td></td>
</tr>
</tbody>
</table>
Adding a third dimension
Embracing Error
NGPU Execution Model

ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;

SIMD lanes are in normal mode

The SM performs precise computation

st.global [addr2], %r2;
NGPU Execution Model

The SM sends the inputs to each of the SIMD lanes

ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;

SIMD lanes enter neural mode
NGPU Execution Model

SIMD lanes autonomously calculate the neural outputs

The SM waits for the neurally enhanced SIMD lanes to finish calculating the neural outputs
NGPU Execution Model

SIMD lanes autonomously calculate the neural outputs

ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;

The SM waits for the neurally enhanced SIMD lanes to finish calculating the neural outputs
NGPU is a Fair Bargain

A scalable neurally accelerated architecture for GPUs with \( \leq 1.0\% \) area overhead

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Energy Reduction</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.9\times</td>
<td>2.1\times</td>
<td>\geq 97.5%</td>
</tr>
<tr>
<td>2.4\times</td>
<td>2.8\times</td>
<td>\geq 90.0%</td>
</tr>
</tbody>
</table>
Navigating a three dimensional space
Neural Accelerator Design Alternatives

- CPU
- GPU
- FPGA
- Digital ASIC
- FPAA
- Analog ASIC

Comparison:
- CPU: Speed: $1.8 \times \uparrow$, Energy: $1.7 \times \downarrow$, Quality: $10\% \downarrow$
- GPU: Speed: $2.3 \times \uparrow$, Energy: $3.0 \times \downarrow$, Quality: $10\% \downarrow$
- FPGA: Speed: $3.7 \times \uparrow$, Energy: $6.3 \times \downarrow$, Quality: $10\% \downarrow$
NGPU

Neurally-Accelerated GPU Architecture

Streaming Multiprocessor (SM)

- Fetch
- Decode
- Issue
- Operand Collection
- Active Mask
- SIMT Stack

Interconnection Network

- LSU Pipeline
- SIMD Lane
- src_reg3, src_reg2, src_reg1, dst_reg
- Write back

Memory Partition

- L2 Cache
- Off-chip DRAM

Off-chip DRAM
Neural Networks and Intermediate Representation

Source Codes

Common Intermediate Representation

Acceleration

Neural Accelerated GPU

Neural Representation

Code\(_1\) Code\(_2\) Code\(_3\) Code\(_4\) Code\(_5\) Code\(_6\) ...

Neurally Accelerated
Data Growth vs. Performance


NGPU

Neurally-Accelerated GPU Architecture

Controller handles the neural execution of all the SIMD lanes
Area Overhead \leq 1\%