NeuroCube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory

Duckhwan Kim¹, Jaeha Kung¹, Sek Chai², Sudhakar Yalamanchili¹, Saibal Mukhopadhyay¹

¹School of Electrical and Computer Engineering, Georgia Institute of Technology
²SRI International

This material is based on work supported in part by an ONR Young Investigator award, a National Science Foundation CAREER Award, and the National Science Foundation grant CCF 1337177
Outline

- Motivation
- Base Architecture of NeuroCube as PIM
- Programming NeuroCube
- Out-of-Order Packet Arrival
- Simulation
- Conclusion
Digital Accelerator Design for Neuromorphic Algorithm

- Low operation density (ops/byte)
- Massive date required

Digital neuro-inspired architecture with **programmability** to cover different types of neural networks, **scalability**, and **high energy efficiency**
NeuroCube: Process-in-Memory Architecture for Neural Computing

Programmable, scalable platform as processor in memory

Hybrid Memory Cube (HMC)
- Heterogeneous integration
- Flexible logic die design

Fabricated by Micron [J. Jeddeloh `12 TVLSI]

Q1. Neural computing layer should meet thermal and area constraint in 3D stacked DRAM

Q2. NeuroCube should be programmable to cover different types of neural network
Basic NeuroCube Architecture

Processor-in-memory + Parallelism

NeuroCube computing core
- Multiple MAC units and its temporal buffer
- Packet based NoC router
- **Programmable neurosequence generator**
- Buffer to handle out-of-order packet arrival
Operational Model of NeuroCube
Property of Neural Network: Deterministic Connections in Inference

General expression of artificial neural network (ANN)

\[ y_j = \varphi \left( \sum_{i \in I} w_{(j,i)} \cdot y_i \right) \]

\( I \): set of connected neurons

- Different NN layer can be mapped by changing set of connected neurons
  - Different data movements (memory address) can map different NN layers in NeuroCube
Property of Neural Network: Deterministic Connections in Training

- Backpropagation with gradient descent
  - $\delta_i = (W_{i,i+1}^T \times \delta_{i+1}) \times \varphi'(y_i)$ (hidden) or $= -(d - y_i) \times \varphi'(y_i)$ (output)
    - $\varphi'$: derivative of NL activation function
    - $\times$: element-wise multiplication
    - $\gamma$: learning rate
  - $\Delta W_{i-1} = \delta_i \times y_{i-1}^T$, $W_{i-1} = W_{i-1} + \gamma \Delta W_{i-1}$

- It is composed of
  - Matrix-vector multiplication, element-wise multiplication, and outer product
  - Can be mapped to FC layer with many zeros in matrix

- Training in neural network still has deterministic connections
Memory Centric Neural Computing

Programmable Neurosequence Generator (PNG)

- **Sequence of operands** is predetermined
- Based on the sequence, memory **can push the data** without request
- Data is delivered as packet through NoC

---

Conventional ISA operation

- Memory
  - Memory Controller
  - On-chip interface (bus)
  - Processing engine

Memory centric operation

- Memory
  - PNG
  - On-chip interface (bus)
  - Processing engine
  - Instruction decoder
  - Program counter
MCNC Data Flow

– Initial Memory Mapping

- Assume 2 vaults in NeuroCube, 3 MACs/PE
  - For synaptic weights
    - Divide synaptic weights matrix into 2 vaults
  - For previous layer
    - Divide input previous into 2 vaults or
    - Duplicate prev. layer into all vaults (reduce NoC traffic)

W_{1,2} = [5 by 4] matrix

- All neuron’s states and synaptic weights are 16bit fixed point

Memory range for this layer

Memory range for next layer
MCNC Data Flow: Address Generator

- Assume 2 vaults in NeuroCube, 3 MACs/PE
- For each vault, it will push data to NoC within packet form

```plaintext
for(j=0; j<1; j++) {//one MAC cover one neuron[j]
    for(i=0; i<4; i++) {//4 neurons[i] to one neuron[j]
        for(k=0; k<3; k++) {//state/weight/local grad
            for(m=0; m<3; m++) {//#MACs/PE = 3
                if k==0
                    addr = addr_gen(i, j, m, state)
                else if k==1
                    addr = addr_gen(i, j, m, weight)
                else
                    addr = addr_gen(i, j, j, local_grad)
            }
        }
    }
}
```

Three nested-counters
MCNC Data Flow (1)

Push packet (state[0]) to MAC [0]

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>State [0]</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
MCNC Data Flow (2)

Push packet (state[0]) to MAC [1]

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>State [0]</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
MCNC Data Flow (3)

Push packet (state[0]) to MAC [2]

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>State [0]</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

[Diagram showing data flow and connections between MAC nodes]
MCNC Data Flow (4)

Push packet (weight[0,0]) to MAC [0]

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>W [0, 0]</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Diagram showing data flow from SRC to MAC [0], MAC [1], and MAC [2].
MCNC Data Flow (5)

Push packet (weight$[0,1]$) to MAC [1]

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>W $[0, 1]$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
MCNC Data Flow (6)

Push packet (weight[0,2]) to MAC [2]

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>W [0, 2]</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
MCNC Data Flow (7)

Push packet (state[1]) to MAC [0]

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>State [1]</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

And so on …
Out-of-order Packet Arrival Problem in NeuroCube
Out-of-order Packet Arrival Issue: NoC Congestion

Although data access is sequential, data arrival can be out-of-order due to NoC congestion.

$W_{1,2} =$ [5 by 4] matrix
Out-of-Order data arrival (1)

Packet structure

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4bit</td>
<td>4bit</td>
<td>16bit</td>
<td>4bit</td>
<td>8bit</td>
</tr>
</tbody>
</table>

- **OP-ID == OP_CNT == 23**
  - This packet is for current operation
  - It moves to temporal buffer [15] directly
Out-of-Order data arrival (2)

- OP-ID \(!=\) OP_CNT
  - This packet is for next operation (OP_CNT == 24)
  - It moves to OOO buffer [8]
    - \(8 = \text{mod}(24,16)\)
    - OOO buffer [i] is FIFO with 64 depth
Out-of-Order data arrival (3)

- **Packet structure**
  - | SRC | DST | DATA | MAC-ID | OP-ID |
  - | 4bit | 4bit | 16bit | 4bit   | 8bit  |

- **OP-ID == OP_CNT == 23**
  - This packet is for current operation
  - It moves to temporal buffer [14] directly
  - Temp buffer (length 16) is full
    - It will trigger 16 MACs operation
Out-of-Order data arrival (4)

- Temp buffer trigger 16 MACs
- Increase OP_CNT
- Ready for 24th operation
Out-of-Order data arrival (5)

- Before capture new packet, check OOO buffer[8]
  - 8 = mod(24,16)
  - Full search OOO buffer [8] (64 depth)
  - Move [0;2;0x5A;0;24] to temp buffer [0]

Packet structure

<table>
<thead>
<tr>
<th>SRC</th>
<th>DST</th>
<th>DATA</th>
<th>MAC-ID</th>
<th>OP-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4bit</td>
<td>4bit</td>
<td>16bit</td>
<td>4bit</td>
<td>8bit</td>
</tr>
</tbody>
</table>
Layerwise Programming

Address generator can be designed using **three nested counters**

For each layer, host program the NeuroCube by **writing conf. registers**

For 2D-Conv: \(N: 26 \times 26, \ C: 3 \times 3\)

For FC: \(N: 10 \times 1, \ C: 676 \times 1\)

- Latency of writing configuration registers is negligible
- External interface is very rare
Layerwise Programming

Conv-NN for Scene labeling

- Handshaking to start programming or main operation

From host

Program

Configuration enable signal

Operation

NeuroCube

One layer done

ConvNN Operation

Programming Command Sent from Host

#Outputs [W, H] = [314, 234] [157, 117] [151, 111] [75, 55] [69, 49] [64, 1]

#Connections [r, r] = [7, 7] [2, 2] [7, 7] [2, 2] [7, 7] [64, 1]

2D-Conv Max-pooling 2D-Conv Max-pooling 2D-Conv Pixel-wise Fully connected

[S. Goud, `09 ICCV]
Simulation Results
Synthesis Result

28nm CMOS process

15nm FinFet process

- To utilize HMC internal throughput fully, $f_{PE} = 5$GHz
- Thermal analysis performed under 5GHz operating
- Max. allowable temp. of HMC: 378K

Footprint of HMC 1.0: 68mm$^2$
Hardware Power Breakdown

- Measured energy consumption [J. Jeddeloh, `12 VLSI]
  - 3.7 pj/bit for the DRAM layers
  - 6.78 pj/bit for the logic layer, (most power hungry = ext. interface SERDES)

- Single core power breakdown (15nm): 187mW

Actual logic die power consumption is much lower since external data movements are very rare

Actual MAC power consumption is much lower since its utilization ratio (activity) is very low: most of time MACs are idle until all operands are ready
Performance Simulation: Inference

To see system throughput, cycle-level simulation is performed

- **Inference**: scene labeling
- **Two operating modes**:
  1. Duplicating prev. layer to reduce NoC traffic
  2. No duplication to reduce memory overhead

```
15nm Finfet design
```

![System throughput](image)

- **Helpful in FC layer**
- **Memory overhead is not significant because weights are dominant**

```
System throughput (OPs/s)
```
```
Memory requirement (byte)
```

- **Duplicating prev. layer**
- **No duplicating**

```
2D Conv         FC             Total
```
```
2D Conv         FC             Total
```
Performance Simulation: Training

- **System throughput** (OPs/s)
  - Maintain constant system throughput

- **Memory requirement** (byte)
  - Memory requirement is high even for small image size (64 x 64)

15nm Finfet design
Performance Simulation: DDR3/Crossbar

Multiple channels in HMC improve system throughput

System throughput (OPs/s)

Duplicate
Non-duplicate

HMC
DDR3

FC layer

System throughput (OPs/s)

Mesh grid 2D NoC
Crossbar

2D Conv
FC Layer

Crossbar improves system throughput for FC layer

15nm Finfet design
Related Works

- Most of prev. work focused on specific NN
  - Shows high throughput based on optimized design (ASIC/FPGA)
  - Not programmable (not scalable)

- Programmable + Scalable design
  - General purposed architecture: mobile CPU or GPU
  - Integrated systems with external DRAM

<table>
<thead>
<tr>
<th></th>
<th>[L. Cabigelli `15 DAC]</th>
<th>NeuroCube</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform</strong></td>
<td>Tegra K1</td>
<td>GTX 780</td>
</tr>
<tr>
<td>Bit precision ctrl.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Throughput (GOPs/s)</td>
<td>76</td>
<td>1781</td>
</tr>
<tr>
<td>Computing Power (W)</td>
<td>11</td>
<td>206.8</td>
</tr>
<tr>
<td>Efficiency (GOPs/s/W)</td>
<td>6.91</td>
<td>8.61</td>
</tr>
<tr>
<td>Inference/training</td>
<td>inference</td>
<td>inference</td>
</tr>
</tbody>
</table>
Conclusion

- **NeuroCube: Neuro-inspired architecture as PIM in HMC**
  - Utilize high memory bandwidth
  - Integrated with high density memory
  - Meet thermal/area constraints

- **Programmable architecture to cover diff. NN types**
  - Programming memory access pattern
  - Simple memory address generator (PNG) is embedded in memory
  - Memory centric neural computing (MCNC) scheme

- **System performance is simulated**
  - Network-on-chip traffic is next bottleneck
  - Optimized NoC design, data re-usage should be studied
Thank you