Neural Cache: Bit-Serial In-Memory Acceleration of Deep Neural Networks

Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, Reetuparna Das

ISCA 2018

Presented by Timo Loher

Computer Architecture Seminar
ETH Zürich
Overview

- Introduction
- Background
- Neural Cache Architecture
- Bit-Serial Computation
- Mapping CNN to Cache
- Evaluation
- Conclusion
Motivation

- Convolutional Neural Networks are popular and powerful, but complex
- Inference is relatively slow on common CPU/GPUs
- Much data movement -> memory gap
- Parallelism is not fully exploited yet
Neural Cache Overview

Goal:
- Reduce data movement
- Exploit parallelism of convolution

Approach:
- SRAM array design to support addition, multiplication and reduction
- Neural Cache architecture to use these operations to accelerate CNN inference
Overview

• Introduction
• **Background**
• Neural Cache Architecture
• Bit-Serial Computation
• Mapping CNN to Memory
• Evaluation
• Conclusion
Convolution: Image Filtering

Gaussian Filter

Laplace Filter

noise reduction

edge detection
# Convolution Operation

![Convolution Operation Diagram](https://cs231n.github.io/convolutional-networks/)

**Input Volume (+pad 1) (7x7x3)**

<table>
<thead>
<tr>
<th>x[i, j, k]</th>
<th>w0[i, j, k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>-1 0 1</td>
</tr>
<tr>
<td>0 0 1 0 2 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 1 0 2 0 1 0</td>
<td>1 -1 1</td>
</tr>
<tr>
<td>0 2 0 2 2 0 0</td>
<td>-1 0 1</td>
</tr>
<tr>
<td>0 2 1 2 2 0 0</td>
<td>1 -1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

**Output Volume (3x3x2)**

<table>
<thead>
<tr>
<th>o[i, j, k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3 3</td>
</tr>
<tr>
<td>3 7 3</td>
</tr>
<tr>
<td>8 10 -3</td>
</tr>
<tr>
<td>-8 -8 -3</td>
</tr>
<tr>
<td>-3 1 0</td>
</tr>
<tr>
<td>-3 -8 -5</td>
</tr>
</tbody>
</table>

**Filter W0 (3x3x3)**

<table>
<thead>
<tr>
<th>w0[i, j, k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 0 1</td>
</tr>
<tr>
<td>0 0 1</td>
</tr>
<tr>
<td>1 -1 1</td>
</tr>
</tbody>
</table>

**Filter W1 (3x3x3)**

<table>
<thead>
<tr>
<th>w1[i, j, k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 0 0</td>
</tr>
<tr>
<td>1 -1 0</td>
</tr>
<tr>
<td>1 -1 0</td>
</tr>
</tbody>
</table>

**Bias b0 (1x1x1)**

<table>
<thead>
<tr>
<th>b0[i, j, k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**Bias b1 (1x1x1)**

<table>
<thead>
<tr>
<th>b1[i, j, k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
Convolution Operation

Source: https://cs231n.github.io/convolutional-networks/
Background: Image Recognition

• Goal: Find image features to distinguish image classes

• Difficulty: Nearly impossible to find by a human

• Solution: Convolutional Neural Networks

Source: https://www.topbots.com/chihuahua-muffin-searching-best-computer-vision-api/
Convolutional Neural Network (CNN)

- Idea: Learn filter weights using machine learning
- Inference: Perform the necessary convolutions to then classify images

Source: https://cloud.google.com/tpu/docs/inception-v3-advanced
Real Life Example: Self Driving Cars

- Performance is critical!
- Human reaction time: ~1 sec
- Limited space and power
Challenges:

- **Memory intensity**
  - Need to load filters and input for every layer
  - “Memory Gap”: Memory latency bottleneck

- **Computational intensity**
  - Many parallel computations (> 1M)
  - Modern GPU has ~5’000 cores (Titan RTX)

Source: https://www.nvidia.com/de-ch/deep-learning-ai/products/titan-rtx/
Challenges:

- **Memory intensity**
  - Need to load filters and input for every layer
  - “Memory Gap”: Memory latency bottleneck
  - **Solution:** Minimize data movement using Processing-in-Memory (PIM)

- **Computational intensity**
  - Many parallel computations (> 1M)
  - Modern GPU has ~5’000 cores (Titan RTX)
  - **Solution:** Turn L3 cache into highly parallel computational unit

Source: https://www.nvidia.com/de-ch/deep-learning-ai/products/titan-rtx/
Overview

- Introduction
- Background
- **Neural Cache Architecture**
  - Bit-Serial Computation
  - Mapping CNN to Memory
- Evaluation
- Conclusion
Neural Cache Architecture

18-core Xeon processor
45 MB LLC

18 LLC slices
Neural Cache Architecture

18-core Xeon processor
45 MB LLC

2.5MB LLC slice

18 LLC slices

360 ways
Neural Cache Architecture

18-core Xeon processor
45 MB LLC

18 LLC slices

2.5MB LLC slice

32kB data bank

8kB array

360 ways

8kB SRAM array

5760 arrays
Neural Cache Architecture

18-core Xeon processor
45 MB LLC

2.5MB LLC slice

8kB SRAM array

18 LLC slices

360 ways

32kB data bank

8kB array

1,474,560 ALUs

5760 arrays

WL

Row decoders

A + B = A + B

S = A ^ B ^ C

Cout

Cin

Vref

C_EN

~A & ~B

A & B

255

0

BL/BLB

Logic

Array A

Array B

Array C

Array D

Bitline ALU

18 LLC slices

360 ways

5760 arrays

1,474,560 ALUs
Overview

• Introduction
• Background
• Neural Cache Architecture
• **Bit-Serial Computation**
• Mapping CNN to Memory
• Evaluation
• Conclusion
Recap: SRAM

- Bit-line drain
- Bit-line amplification
Logical Operations in SRAM

Changes

- Additional row decoder
- Reconfigurable sense amplifiers

Additional row decoder

Reconfigurable sense amplifiers

Row Decoder

Row Decoder

Bitlines

Wordlines

Single-ended Sense Amplifiers

SA

Differential Sense Amplifiers

Vref

SA

SA

SA

BL0

BL0

BLn

BLn
Logical Operations in SRAM

- Bitwise **AND**
- Bitwise **NOR**
Arithmetic Operations

- Neural Cache supports addition, multiplication and reduction operations
- Data needs to be transposed (Bit-Serial)
- Operations are executed for the whole SRAM array in parallel
Adding one Bit

Standard Full adder

Bit-Serial adder
Addition: Data Layout

A[0]: 11
B[0]: 10
Carry[0]: 0

---------

Sum[0]: 000
Addition: First Bit

A[0]: 11
B[0]: 10
Carry[0]: 0

Sum[0]: 000
Addition: Second Bit

A[0]: 11  
B[0]: 10  
Carry[0]: 10  

Sum[0]: 001
Addition: Carry Write

\[
\begin{align*}
\text{A}[0] & : 11 \\
\text{B}[0] & : 10 \\
\text{Carry}[0] & : 1 \\
\text{Sum}[0] & : 001 \\
\end{align*}
\]
Multiplication: Full Circuit

- Full Adder
- Bitline Driver
- Predication to enable Tag as a mask for bitline driver
- DIN and DOUT for data movement
Multiplication

A[0]: 10  
B[0]: 01  
Carry[0]: 0  
--------------------------  
Product [0]: 0000
Multiplication

A[0]: 10
B[0]: 01
Carry[0]: 0

Product [0]: 0000
Multiplication

A[0]: 10
B[0]: 01
Carry[0]: 0

Product [0]: 0000
Multiplication

- **A[0]:** 10
- **B[0]:** 01
- **Carry[0]:** 0

--------------------------
**Product [0]:** 0001
Multiplication

A[0]: 10
B[0]: 01
Carry[0]: 0

Product [0]: 0010
Multiplication

A[0]: 10
B[0]: 01
Carry[0]: 0

Product [0]: 0010
Multiplication

A[0]: 10
B[0]: 01
Carry[0]: 0

Product [0]: 0010
Multiplication

A[0]: 10
B[0]: 01
Carry[0]: 0
--------------------------
Product [0]: 0010
Reduction

• Adding up multiple numbers

• By exploiting parallelism we can reduce the number of operations
Transposing

Transposing Memory Unit (TMU)

Row Decoder

Regular read/write

8-T transpose bit-cell

Transpose read/write

WL
BLBv
BLv
VDD
VSS

Way 1
Way 2
Way 19
Way 20
Transposing

<table>
<thead>
<tr>
<th>Row Decoder</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A₂</td>
<td>A₁</td>
<td>A₀</td>
<td></td>
</tr>
<tr>
<td>B₂</td>
<td>B₁</td>
<td>B₀</td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td>C₁</td>
<td>C₀</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Row Decoder

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
<tr>
<td>DR</td>
<td>VS</td>
<td>DR</td>
</tr>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DR</td>
<td>VS</td>
<td>DR</td>
</tr>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
<tr>
<td>DR</td>
<td>VS</td>
<td>DR</td>
</tr>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Decoder</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C₂</td>
<td>B₂</td>
<td>A₂</td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td>B₁</td>
<td>A₁</td>
<td></td>
</tr>
<tr>
<td>C₀</td>
<td>B₀</td>
<td>A₀</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Row Decoder

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
<tr>
<td>DR</td>
<td>VS</td>
<td>DR</td>
</tr>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DR</td>
<td>VS</td>
<td>DR</td>
</tr>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
<tr>
<td>DR</td>
<td>VS</td>
<td>DR</td>
</tr>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
<tr>
<td>DR</td>
<td>VS</td>
<td>DR</td>
</tr>
<tr>
<td>VS</td>
<td>DR</td>
<td>VS</td>
</tr>
</tbody>
</table>

Legend:
- A: Row Decoder
- B: Col Decoder
- C: Control
Overview

- Introduction
- Background
- Neural Cache Architecture
- Bit-Serial Computation
- Mapping CNN to Memory
- Evaluation
- Conclusion
A Convolutional Layer

3D Filters (M)
each filter: C channels
each channel: RxS weights

Input Activations (C channels)
Output Activations (M channels)
Mapping CNN to Neural Cache

Unroll

Filter Weights

Input Activations

Output Activations

Unroll

Partial Sum

1 Output Activation

MAC

Reduction

8 kB SRAM Array

256 Bitlines

256 Wordlines

Input Activations

Output Activations

Weights

Partial Sum

4x8

4x8

4x8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8

RxSx8
Mapping CNN to Neural Cache
Mapping of Convolution to Array
Overview

- Introduction
- Background
- Neural Cache Architecture
- Bit-Serial Computation
- Mapping CNN to Memory

**Evaluation**

- Conclusion
Evaluation

- Inception V3

<table>
<thead>
<tr>
<th></th>
<th>CPU (2 sockets)</th>
<th>GPU (1 card)</th>
<th>Neural Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>Intel Xeon E5-2597 v3, 2.6GHz, 28 cores, 56 threads</td>
<td>Nvidia Titan Xp, 1.6GHz, 3840 cuda cores</td>
<td>2.5GHz Compute SRAM, 1032192 Bit-serial ALUs</td>
</tr>
<tr>
<td><strong>On-chip memory</strong></td>
<td>78.96 MB</td>
<td>9.14 MB</td>
<td>70 MB (Dual Socket)</td>
</tr>
<tr>
<td><strong>Off-chip memory</strong></td>
<td>64 GB DRAM</td>
<td>12 GB DRAM</td>
<td>64 GB DRAM</td>
</tr>
<tr>
<td><strong>Profiler / Simulator (Performance)</strong></td>
<td>TensorFlow tfprof</td>
<td>TensorFlow tfprof</td>
<td>Cycle accurate simulator + C Microbench</td>
</tr>
<tr>
<td><strong>Profiler / Simulator (Energy)</strong></td>
<td>Intel RAPL Interface</td>
<td>NVIDIA System Management Interface</td>
<td>SPICE simulation + Intel RAPL Interface</td>
</tr>
</tbody>
</table>
Performance

**Latency** (1 input image)
- 18.3x against CPU
- 7.7x against GPU

**Throughput** (max batch size)
- 12.4x against CPU
- 2.2x against GPU
Energy Efficiency

- Half the power usage of CPU/GPU
- 37.1x more efficient than CPU
- 16.7x better than GPU
Execution Breakdown and Area Cost

- ~65% of execution time for data movement
- Area overhead:
  - ~7% per SRAM array
  - ~2.5% per whole CPU
Conclusion

• By adding additional circuitry it is possible to perform addition, multiplication and reduction in SRAM
• The Neural Cache architecture can use these to accelerate CNNs by reducing data movement and exploit parallelism
• Faster and more energy efficient than GPU or CPU
Questions
Strengths

• High potential for more applications using PIM in SRAM
• High demand for CNN accelerators
• Possibly well scalable
• General processing performance not affected
Weaknesses

• Data movement bottleneck not solved
• Modifying cache requires redesigning the whole CPU
• Evaluation is lacking
  • No comparison with other CNN accelerators
  • No comparison with other PIM solutions (DRAM based)
• Claiming that PIM in DRAM is a bad idea without evidence
Similar Works

• Vivek Seshadri +, “Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology”, 2017
• Fei Gao +, “Compute DRAM: In-Memory Compute Using Off-the-Shelf DRAMs”, 2019 (Last weeks presentation)
• N. P. Joupp +, “In-Datacenter performance analysis of a tensor processing unit”, 2017
• Vivek Seshadri +, “RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization”, 2013
Discussion

Could you see the neural cache replace a GPU?

• Is probably cheaper
• Potentially more energy efficient
• How easy can we map programming models to the neural cache model?
• Reducing CPU power when using Neural Cache
Discussion

As we saw, all the computation is done in the L3 Cache. Why not build our CPU out of just Caches? Any problems you can see?

• Sequential computations are slow
• How to handle control flow?
• Memory Gap is still a problem if data does not fit into cache
• Probably many more...
Discussion

Repurposing the CPU Cache has some disadvantages. What if we used a standalone Neural Cache instead?

• All advantages remain
• CPU can run in parallel at full performance
• Could be made with higher capacity
• Neural Cache not only shares SRAM cells, but also memory controller and memory bus. The performance depends highly on how fast one can get data from main memory
Human Brain Model

- Computation in place (basically PIM)
- Very specialized regions widely distributed
- Good idea for a new model?

How good is a human in computing?

- Very low throughput
- Very high latency
- Unreliable

My conclusion: Not what we want to accelerate these kind of computations
Thanks for Your Attention