Accelerate Binarized Neural Networks with Processing-in-Memory Enabled by RISC-V Custom Instructions

Che-Chia Lin¹, Chao-Lin Lee¹, Jenq-Kuen Lee¹, Howard Wang², Ming-Yu Hung²

¹Department of Computer Science, National Tsing Hua University, Taiwan
²MediaTek Inc.
Outline

• Introduction & Background
• Processing-in-Memory Architecture and Custom Instructions
• TVM Compilation Flow
• Binarized Convolution of Processing-in-Memory
• Experiment Result
• Conclusion & Future Work
Introduction

• There are emerging technologies trying to take down the “Memory Wall” and one of the techniques is Processing-in-Memory (PIM).

• This work exploits PIM to accelerate Binarized Neural Networks (BNN).
  • We take convolution of BNNs as target application.
  • We model PIM in Gem5 for supporting bit-wise operations and population count (POPC).
  • PIM operations are supported as RISC-V Custom Instructions.
  • BNNs are compiled and deployed by TVM.
  • We propose a memory layout suitable for PIM operations.
Background - Processing-in-Memory

- Processing-in-Memory
  - Perform bit-wise operations on memory rows
  - Use sense amplifier to accomplish the operations

Background - Processing-in-Memory

- Processing-in-Memory
  - Perform bit-wise operations on memory rows
  - Use sense amplifier to accomplish the operations

Background - Processing-in-Memory

- Processing-in-Memory
  - Perform bit-wise operations on memory rows
  - Use sense amplifier to accomplish the operations

Background - TVM

• AI compiler framework
• Support different formats
• Relay IR (Intermediate Representation) represents graph composed of operators
• Bring Your Own Codegen (BYOC)

Our PIM Architecture
Our PIM Instructions

- 3 instructions
  - AND, NOT, POPC
  - Operands are memory addresses

Table 3.1: Specifications for PIM instructions.

<table>
<thead>
<tr>
<th>instruction</th>
<th>format</th>
<th>encoding space</th>
</tr>
</thead>
<tbody>
<tr>
<td>pim_and</td>
<td>pim_and rd rs1 rs2</td>
<td>custom0.rd.rs1.rs2</td>
</tr>
<tr>
<td>pim_not</td>
<td>pim_not rd rs1</td>
<td>custom0.rd.rs1</td>
</tr>
<tr>
<td>pim_popc</td>
<td>pim_popc rd rs1 imm12</td>
<td>custom0.rd</td>
</tr>
</tbody>
</table>
Our PIM Instructions

- AND, NOT
Our PIM Instructions

• POPC

```
\begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \ldots \\
3 &       \\
\end{array}
```

```
\texttt{pim.popc rd rs1 imm12}
```

Memory Row

1 byte
TVM Compilation Flow

- Binarized Convolution in TVM
  - bitpack
  - bitserial_conv2d

- New PIM_bconv2d
  - External C code utilizing PIM instructions
  - Use modified riscv-gnu-toolchain to compile
Binarized Convolution

Channels = 8
8 elements of int8

Channels = 1
1 elements of int8

Input

Weight
Binarized Convolution
Binarized Convolution

\[
\text{AND} = \begin{array}{c|c|c|c|c|c|c|c}
\hline
& & & & & & & \\
\hline
& & & & & & & \\
\hline
& & & & & & & \\
\hline
\end{array}
\]

\[
\text{POPC} = \begin{array}{c|c|c|c|c|c|c|c}
\hline
& & & & & & & \\
\hline
& & & & & & & \\
\hline
& & & & & & & \\
\hline
\end{array}
\]

\[
\text{SUM} = \begin{array}{c|c|c|c|c|c|c|c}
\hline
& & & & & & & \\
\hline
& & & & & & & \\
\hline
& & & & & & & \\
\hline
\end{array}
\]
PIM Version of Binarized Convolution

- Turn the layout to be suitable with row-based operations
PIM Version of Binarized Convolution
## Experiment Setting

- **Gem5**

<table>
<thead>
<tr>
<th>CPU Model</th>
<th>TimingSimpleCPU, 1GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>RISC-V</td>
</tr>
<tr>
<td>Mode</td>
<td>System Emulation</td>
</tr>
<tr>
<td>Caches</td>
<td>L1I 32KB, L1D 64KB, L2 2MB</td>
</tr>
<tr>
<td>Memory</td>
<td>DDR3-1600 8x8, 1KB row size, 8GB</td>
</tr>
</tbody>
</table>

- **BNN models from Riptide project (SqueezeNet, VGGNet, AlexNet)**
  - Models trained in Tensorflow
  - Use TVM to inference
  - TVM (0.6), riscv-gnu-toolchain (rvv-0.7.x)

---

Experiment Result

![Graph showing execution time results for different models: Riptide-unipolar-squeezenet, Riptide-unipolar-vggnet, Riptide-unipolar-alexnet. The x-axis represents the models, and the y-axis represents execution time in seconds. The bars indicate the execution time for PIM, Base, and Speedup.]
Experiment Result

<table>
<thead>
<tr>
<th>model</th>
<th>convolution</th>
<th>dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Riptide-unipolar-SqueezeNet</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>Riptide-unipolar-VGGNet</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Riptide-unipolar-AlexNet</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
Conclusion & Future Work

• Conclusion
  • We create a flow using PIM operations to accelerate the BNNs
    • PIM operations are modeled in Gem5 simulator.
    • We support TVM compilation for PIM implementation.
    • A memory layout suitable with PIM operations is proposed.
  • The results give speedup from 3.7x up to 57.3x.

• Future Work
  • Integrate the PIM_bconv in TVM
    • Optimization inside TVM can be applied.
  • Support more complex operation with this type of PIM
    • More types of AI model can be supported, like CNN models.
Thanks for Listening

Q&A