HWPE: A CNN Accelerator for RISC-V

Author: Hao Chen, Qiang Chen
What can HWPE do?

**HWPE (CNN Hardware Processing Engines)**

- A CNN hardware acceleration coprocessor for RISC-V
- Support Convolution layer and ReLU layer
- Transform convolution into matrix multiplication (im2col on the fly)
- Kernel size from $3 \times 3$ to $11 \times 11$
- Support data type int8(uint8), exp4 (4 bits of exponential scale) and ternary
- 16 dot-product operations of 64 bits operands (8 int8, 16 exp4 or 32 ternary) per cycle
- Complete a convolution layer operation with one configuration
Convolution & ReLU

• Fully-Connected layer and pooling layer are still calculated by MCU
Architecture Diagram

- Data fetcher accepts the command from MCU and fetches data from Feature Map SRAM and pushes them into FIFO in the way of im2col
- FIFO is used as a buffer for the left matrix
- 16 PE, each one stores a 64 bits right matrix, performs multiply-accumulate and stores a column result matrix
- Two kernel SRAMs, each one broadcasts the data of the right matrix to 8 PE
Convolution / Matrix Multiplication

- Transform convolution into matrix multiplication
- Left Matrix: data-fetcher calculate memory address to access data in the way of im2col
- Right Matrix: store the kernel in the way of channel priority in advance
HWPE Instructions

- HWPE has only 7 instructions based on custom-0 opcode of RISC-V extended instruction set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>funct</th>
<th>rd</th>
<th>xd</th>
<th>rs1</th>
<th>xs1</th>
<th>rs2</th>
<th>xs2</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWPEWriteFmapAddrReg</td>
<td>1</td>
<td>addr_idx</td>
<td>0</td>
<td>addr1</td>
<td>1</td>
<td>addr2</td>
<td>1</td>
</tr>
<tr>
<td>HWPEWriteCfgReg</td>
<td>2</td>
<td>_____</td>
<td>0</td>
<td>cfg</td>
<td>1</td>
<td>cfg</td>
<td>1</td>
</tr>
<tr>
<td>HWPEMatrixMac</td>
<td>4</td>
<td>_____</td>
<td>0</td>
<td>W/H_count</td>
<td>1</td>
<td>H/W_stride</td>
<td>1</td>
</tr>
<tr>
<td>HWPEWriteAccReg</td>
<td>8</td>
<td>AccReg_ID</td>
<td>0</td>
<td>M(idx)</td>
<td>1</td>
<td>PE_ID</td>
<td>0</td>
</tr>
<tr>
<td>HWPEReadAccReg</td>
<td>16</td>
<td>M(idx)</td>
<td>1</td>
<td>Acc_Reg_ID</td>
<td>0</td>
<td>PE_ID</td>
<td>0</td>
</tr>
<tr>
<td>HWPEReluLUMemWriteAccReg</td>
<td>32</td>
<td>_____</td>
<td>0</td>
<td>M(addr)</td>
<td>1</td>
<td>AccReg_ID</td>
<td>0</td>
</tr>
<tr>
<td>HWPEReset</td>
<td>64</td>
<td>_____</td>
<td>0</td>
<td>_____</td>
<td>0</td>
<td>_____</td>
<td>0</td>
</tr>
</tbody>
</table>
## What do HWPE Instructions do?

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HWPEReset</td>
<td>perform a synchronous reset for the HWPE registers</td>
</tr>
<tr>
<td>2</td>
<td>HWPEWriteFmapAddrReg</td>
<td>configure 8 base address registers which point to the 8 initial points of the H-W plane</td>
</tr>
<tr>
<td>3</td>
<td>HWPEWriteCfgReg</td>
<td>configure control registers for convolution</td>
</tr>
<tr>
<td>4</td>
<td>HWPEWriteAccReg</td>
<td>reset or set AccRegs / adds the bias in convolution layers</td>
</tr>
<tr>
<td>5</td>
<td>HWPEMatrixMac</td>
<td>start first round of convolution operation</td>
</tr>
<tr>
<td>6</td>
<td>HWPEReadAccReg</td>
<td>read the AccRegs back to the MCU general registers</td>
</tr>
<tr>
<td>7</td>
<td>HWPEReLUMemWriteAccReg</td>
<td>execute ReLU operation and transfer the 32 bits AccRegs to 8 bits, then store the results into the specified memory address</td>
</tr>
</tbody>
</table>
How to run CNN: Config Registers

- A convolution task is determined by the configuration registers.

FmapConvAddr[i] = FmapAddrBase[i] + x·W_stride + y·H_stride
0 ≤ x ≤ H_count; 0 ≤ y ≤ W_count

FmapSramAddr[i] = FmapConvAddr[i] + m·Conv_W_stride + n·Conv_CH_stride
0 ≤ m ≤ Conv_CH_count; 0 ≤ n ≤ Kernel_size

Config Registers:
- FmapAddrBase[8]
- H_stride
- W_stride
- H_count
- W_count
- Conv_W_stride
- Kernel_size
- Conv_CH_count

Head address of convolution windows:
- FmapConvAddr[i] = FmapAddrBase[i] + x·W_stride + y·H_stride

SRAM memory access address:
- FmapSramAddr[i] = FmapConvAddr[i] + m·Conv_W_stride + n·Conv_CH_stride

8 Base Address
W_stride
H_stride
Conv_W_count
Conv_CH_count
Config Registers:
FmapAddrBase[8]
H_stride
W_stride
H_count
W_count
Conv_W_stride
Kernel_size
Conv_CH_count
How to run CNN

1. Configure the configuration registers with $\text{HWPEWriteFmapAddrReg}$ & $\text{HWPEWriteCfgReg}$
   • select 8 initial points on the H-W plane as base address, each one is the head address in the first convolution window.

2. Start the first round of convolution operations with $\text{HWPEMatrixMac}$. 128(8x16) output points are calculated.
   • In a round of computation, the data is fetched to perform dot multiplication. The address of the data is calculated according to registers $\text{Conv\_W\_stride}$ and $\text{Kernel\_size Conv\_CH\_count}$
   • Multiply 8 points in the H-W plane of feature map and 16 different kernels to obtain 128 output points.

3. Read the output back to the MCU general register for further operation with $\text{HWPEReadAccReg}$ or execute ReLU operation and transfer the 32 bits output to 8 bits, then store into the specified memory address with $\text{HWPEReLUMemWriteAccReg}$

4. HWPE continues the next round calculation until the last $\text{HWPEReadAccReg/ HWPEReLUMemWriteAccReg}$
   • Convolution windows move by the size of $\text{stride}$, first in the H direction and then in the W direction.
   • Head address of convolution window will be updated according to registers $\text{W\_stride H\_stride H\_count W\_count}$ every round. Then the data is fetched to perform dot multiplication in the way of im2col.

5. When calculate $\text{H\_count*W\_count*K\_count}$ rounds, the entire convolution task is completed.
   • A H-W plane needs $\text{H\_count*W\_count}$ rounds computation.
   • For K kernels, the entire convolution task needs $\text{H\_count*W\_count*K\_count}$ rounds.($\text{K\_count=K/16}$)
Convolution with One Configuration

• *In general, one configuration can complete a convolution layer operation.*

• Select 8 initial points and move according to the configuration registers to cover the entire H-W plane as far as possible.

• If not, reconfigure the configuration registers and select new 8 initial points to cover the entire H-W plane.

• MCU can also pad the input feature map before storage, making it easy to finish the convolution operation with one configuration.
## PE Utilization

<table>
<thead>
<tr>
<th>Type</th>
<th>Kernel size</th>
<th>PE utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal layer</td>
<td>any size</td>
<td>100%</td>
</tr>
<tr>
<td>input layer</td>
<td>3x3x3</td>
<td>84.38%</td>
</tr>
<tr>
<td></td>
<td>5x5x3</td>
<td>93.75%</td>
</tr>
<tr>
<td></td>
<td>7x7x3</td>
<td>87.50%</td>
</tr>
<tr>
<td></td>
<td>9x9x3</td>
<td>96.43%</td>
</tr>
<tr>
<td></td>
<td>11x11x3</td>
<td>82.50%</td>
</tr>
</tbody>
</table>
Conclusion

• Currently implemented on the EAI interface of hummingbird E200 MCU (not verified yet)
• Can be easily transplanted to other RISC-V coprocessor interfaces

• [https://github.com/chenhaoc/cnnhwpe/](https://github.com/chenhaoc/cnnhwpe/)
• Include Matlab Model / C Model / RTL

• Total equivalent gate count : 190K

• Peak performance (16PE): 256xFreq OPS (INT8)/512xFreq OPS (EXP4)
  /1024xFreq OPS (Ternary)

• HWPE architecture is flexible and efficient. It is suitable for CNNs acceleration on the resource-limited devices.
In the Coming Work

• Simulate with EAI interface of hummingbird E200
• software framework

• *Looking for volunteers to finish it!*
Thanks For Your Attention!