Variable bit-precision vector extension for a RISC-V based processor

Risikesh RK\textsuperscript{1}, Sharad Sinha\textsuperscript{2}, Nanditha Rao\textsuperscript{1}

\textsuperscript{1} International Institute of Information Technology Bangalore, Bangalore, India
\textsuperscript{2} Indian Institute of Technology Goa, Goa, India
Introduction

- DNNs Used in wide variety of applications
- Compute intensive task
- Specialised Hardware required
Neural Network Models

Optimisation Techniques:

- Pruning
- Quantisation
- Low-rank Approximation

Quantization

Low-Rank Approximation

Pruning
Objective

To Build an efficient RISC-V based processor which will be able to run Compressed DNN tasks in the processor:

1. To support compressed model memory’s Load/Store
2. To support arithmetic instruction to run the variable bit width Models
3. To support variable bit precision arithmetic and configurations through

<table>
<thead>
<tr>
<th>Network</th>
<th>Per Layer Neuron Precision in Bits</th>
<th>Ideal Speedup</th>
<th>Per Layer Neuron Precision in Bits</th>
<th>Ideal Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>3-3</td>
<td>5.33</td>
<td>2-3</td>
<td>7.33</td>
</tr>
<tr>
<td>ConvNet</td>
<td>4-8-8</td>
<td>2.89</td>
<td>4-5-7</td>
<td>3.53</td>
</tr>
<tr>
<td>AlexNet</td>
<td>9-8-5-5-7</td>
<td>2.38</td>
<td>9-7-4-5-7</td>
<td>2.58</td>
</tr>
<tr>
<td>NiN</td>
<td>8-8-8-9-7-8-8-8-8-9-8-8-8-8-8-8-8</td>
<td>1.91</td>
<td>8-8-7-9-7-8-8-9-8-9-8-7-8</td>
<td>1.93</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>10-8-10-9-8-10-9-8-9-10-9-10-7</td>
<td>1.76</td>
<td>10-8-9-8-8-8-9-10-8-9-10-8</td>
<td>1.80</td>
</tr>
<tr>
<td>VGG_M</td>
<td>7-7-7-8-7</td>
<td>2.23</td>
<td>6-8-7-7-7</td>
<td>2.34</td>
</tr>
<tr>
<td>VGG_S</td>
<td>7-8-9-7-9</td>
<td>2.04</td>
<td>7-8-9-7-9</td>
<td>2.04</td>
</tr>
</tbody>
</table>

TABLE I: Per Convolutional layer neuron precision profiles needed to maintain the same accuracy as in the baseline (100%) and to reduce it within 1% of the baseline (99%). Ideal: Potential speedup with Stripes over a 16-bit baseline.

*Data: Stripes: Bit-Serial Deep Neural Network Computing
Proposed Architecture
Proposed Architecture
PicoRV32 Microarchitecture

CPU STATES
1. Instruction Fetch
2. LD Rs1
3. LDMEM
4. STMEM
5. EXEC
6. SHIFT
7. TRAP
8. LDMEM_Vector
9. STMEM_Vector
10. EXEC_Vector

Newly Added States:
### Proposed Instruction & Implemented Instructions

<table>
<thead>
<tr>
<th>Proposed instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vset_precision, rs1, rs2</td>
<td>Vector set precision (Rs1, Rs2 to set Control Register Variable Arithmetic Precision and Element Offset)</td>
</tr>
<tr>
<td>vmul_varp vd, vs1, vs2</td>
<td>Vector multiplication on variable precision</td>
</tr>
<tr>
<td>vadd_varp vd, vs1, vs2</td>
<td>Vector addition on variable precision</td>
</tr>
<tr>
<td>vsub_varp vd, vs1, vs2</td>
<td>Vector subtract on variable precision</td>
</tr>
<tr>
<td>vleu_varp vd, rs1, vm</td>
<td>Vector load element unit-stride on variable precision</td>
</tr>
<tr>
<td>vles_varp vd, rs1, rs2, vm</td>
<td>Vector load element strided on variable precision</td>
</tr>
<tr>
<td>vseu_varp vs3, rs1, vm</td>
<td>Vector store element unit stride on variable precision</td>
</tr>
<tr>
<td>vses_varp vs3, rs1, rs2, vm</td>
<td>Vector store element strided on variable precision</td>
</tr>
</tbody>
</table>

### Control Registers

<table>
<thead>
<tr>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Arithmetic Precision</td>
<td>This is a 4 bit register which has the precision of the arithmetic / Load/store operation</td>
</tr>
<tr>
<td>Element Offset</td>
<td>This is a 32 bit register where the offset of elements is specified during vector load/stores</td>
</tr>
</tbody>
</table>

### Risc-V-Vector Instructions Implemented

- **Load/Store Instructions**
  - Configuration
  - Load/Store unit stride
  - Load/Store indexed
- **Other Instructions**
  - VAMO Instructions
  - Vector Integer Arithmetic
  - Vector Floating Point arithmetic
- **Vector Reduction Operation**
  - Vector Mask
  - Vector Permutation

---

**Proposed Instructions**

**Risc-V-Vector Instructions Implemented**
How data is stored in the Memory?

### Proposed instruction

<table>
<thead>
<tr>
<th>Proposed instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vleu_varp vd, rs1, vm</td>
<td>vector load element unit-stride on variable precision</td>
</tr>
<tr>
<td>vles_varp vd, rs1, rs2, vm</td>
<td>vector load element strided on variable precision</td>
</tr>
<tr>
<td>vseu_varp vs3, rs1 vm</td>
<td>vector store element unit stride on variable precision</td>
</tr>
<tr>
<td>vses_varp vs3, rs1, rs2, vm</td>
<td>vector store element strided on variable precision</td>
</tr>
</tbody>
</table>

### Descriptions

**Matrix B (5 bits)**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-3</td>
<td>3</td>
<td>-2</td>
<td>7</td>
</tr>
</tbody>
</table>

**Design 1 & 2 (a)**

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
```

**Design 3 (b)**

```
0 0 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0
```

- B[1] = 1
- Alignment padding
- 32b Word
Bit Serial Multiplication: Variable Bit precision Support

Operand A
Operand B

MUX

0

B<<1

A

MUX

0

B[15]

<<1

Left Shifter

Accumulator <<1

+    

Accumulator

<<1

Left Shifter

Output(AxB)
Bit Serial Multiplication: Variable Bit precision Support

How a 16bit integer and a 5 bit integer are multiplied?

A = 3
B = (01011) | MSB = 0 | Accumulator = 0
A = 3
B = (1011 ) | MSB = 1 | Accumulator = 3
A = 3
B = (011 ) | MSB = 0 | Accumulator = 6
A = 3
B = (11 ) | MSB = 1 | Accumulator = 15
A = 3
B = (1 ) | MSB = 1 | Accumulator = 33

(0 << 1) + (3*0)
(0 << 1) + (3*1)
(3 << 1) + (3*0)
(6 << 1) + (3*1)
(15 << 1) + (3*1)
VPU - Vector Processing Unit

Vector Register File (512b)

Vector Instruction Decode/Control Unit

Vector Register Read/Write Interface

Instruction Issue Interface

Instruction Fetch/PC

Instruction Decode

Control Signals Load/Store Vector Operation Exec

Control Status Register

Memory Controller

Matrix

Vector Processing Element 32 x 512bit wide register

Instruction control decoder

Load/Store Register access

Instruction issue/valid/wait

Vector Processing Element (Two 16 Bit Bit-Serial Multiplier)

Multiplexing

32b x16

Mem.Data

Mem.Addr
Memory Mapping to the Multiplier

How a 5 bit data is being mapped to the multiplier?
Software Support

Supported Instruction (JSON FILE)

Assembly Code

Assembler.py

Machine Code
Results

```
ll r1, 0x003 # setting 3 bit precision
ll r2, 0x003 # setting 3 as vector length
ll r3, 0 # offset to load from
ll r7 0x800 # variable bitwidth matrix A
ll r8 0x900 # matrix B
ll r9 0x008 # matrix C=AxB
vset_prec城里 r1,r8 #where r8 is element offset
vise_varp v1, (r7),r2
addi r3, r3, 0x001
vset_prec城里 r1, r3 #where r3 is element offset
vise_varp v2, (r7),r2
addi r3, r3, 0x001
vset_prec城里 r1, r3 #where r3 is element offset
vise_varp v3, (r7),r2
vset_prec城里 r1, r0
for( int i = 0; i<3;i++)
vsetvl r6, r2, E16
vsubv v8, v0, v0
vise.v v4, (r8), r0
addi r8, r8, 0x002
vise.v v5, (r8), r0
addi r8, r8, 0x002
vise.v v6, (r8), r0
addi r8, r8, 0x002
vmul_varp v7, v4, v1
vadd_varp v0, v0, v7
vmul_varp v7, v5, v2
vadd_varp v0, v0, v7
vmul_varp v7, v6, v3
vadd_varp v0, v0, v7
vse.v v8, (r9)
addi r9, r9, 0x006
```

Pseudo code for 3x3 matrix multiplication with variable precision support

<table>
<thead>
<tr>
<th>Design</th>
<th>Configuration</th>
<th>% LUT Used</th>
<th>% FlipFlop Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PICORV</td>
<td>VPU</td>
<td>PICORV</td>
</tr>
<tr>
<td>Design 1</td>
<td>PICORV32</td>
<td>1.7</td>
<td>NA</td>
</tr>
<tr>
<td>Design 2</td>
<td>PICORV32 + VPU (without Variable Bit Precision (Bitserial Multiplier))</td>
<td>1.9</td>
<td>14.51</td>
</tr>
<tr>
<td>Design 3</td>
<td>PICORV32 + VPU (With Variable Bit Precision (Bitserial Multiplier))</td>
<td>1.9</td>
<td>26.4</td>
</tr>
</tbody>
</table>

3x3 Matrix Multiplication Multiplication

Matrix A (16bit)
```
<table>
<thead>
<tr>
<th>20</th>
<th>11</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>15</td>
</tr>
</tbody>
</table>
```

Matrix B (N bit)
```
<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>5</td>
</tr>
<tr>
<td>-5</td>
<td>-3</td>
<td>7</td>
</tr>
</tbody>
</table>
```
## Results

<table>
<thead>
<tr>
<th>Bit width</th>
<th>Number of CPU CYCLES</th>
<th>Overall Speedup</th>
<th>Memory Space (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Design 1</td>
<td>Design 2</td>
<td>Design 3</td>
</tr>
<tr>
<td>1</td>
<td>1799</td>
<td>580</td>
<td>439</td>
</tr>
<tr>
<td>2</td>
<td>1799</td>
<td>580</td>
<td>448</td>
</tr>
<tr>
<td>3</td>
<td>1799</td>
<td>580</td>
<td>457</td>
</tr>
<tr>
<td>4</td>
<td>1799</td>
<td>580</td>
<td>468</td>
</tr>
<tr>
<td>5</td>
<td>1799</td>
<td>580</td>
<td>479</td>
</tr>
<tr>
<td>6</td>
<td>1799</td>
<td>580</td>
<td>486</td>
</tr>
<tr>
<td>7</td>
<td>1799</td>
<td>580</td>
<td>497</td>
</tr>
<tr>
<td>8</td>
<td>1799</td>
<td>580</td>
<td>506</td>
</tr>
<tr>
<td>9</td>
<td>1799</td>
<td>580</td>
<td>517</td>
</tr>
<tr>
<td>10</td>
<td>1799</td>
<td>580</td>
<td>530</td>
</tr>
<tr>
<td>11</td>
<td>1799</td>
<td>580</td>
<td>535</td>
</tr>
<tr>
<td>12</td>
<td>1799</td>
<td>580</td>
<td>544</td>
</tr>
<tr>
<td>13</td>
<td>1799</td>
<td>580</td>
<td>555</td>
</tr>
<tr>
<td>14</td>
<td>1799</td>
<td>580</td>
<td>566</td>
</tr>
<tr>
<td>15</td>
<td>1799</td>
<td>580</td>
<td>575</td>
</tr>
<tr>
<td>16</td>
<td>1799</td>
<td>580</td>
<td>580</td>
</tr>
</tbody>
</table>

### Design Configuration

- **Design 1**: PICORV32
- **Design 2**: PICORV32 + VPU (without Variable Bit Precision (Bitserial Multiplier))
- **Design 3**: PICORV32 + VPU (With Variable Bit Precision (Bitserial Multiplier))

![Graph showing speedup of CPU cycles against bitwidth](image)
Conclusion & Future Work

- Average Speedup of 1.14x and reduction in memory footprint by 1.88x for a 3x3 matrix multiplication.
- Integrating the CUSTOM Instructions to RiscV-GCC Compiler tool Chain
- Adding more ISA Support to 2d-Convolution, 3d-Convolution and other layers in DNN natively, also extending the rest of needed Risc-V-Vector extension.
Any questions/suggestions
Contact: Risikesh.Rk@iiitb.ac.in