A 3 TOPS/W RISC-V Parallel Cluster for Inference of Fine-Grain Mixed-Precision Quantized Neural Networks

Alessandro Nadalini\textsuperscript{1}, Georg Rutishauser\textsuperscript{2}, Alessio Burrello\textsuperscript{1}, Nazareno Bruschi\textsuperscript{1}, Angelo Garofalo\textsuperscript{1}, Luca Benini\textsuperscript{1,2}, Francesco Conti\textsuperscript{1}, Davide Rossi\textsuperscript{1}

\textsuperscript{1}University of Bologna, \textsuperscript{2}ETH Zürich
Introduction and Motivation

- Emerging application areas for AI-enabled IoT
  - Personalized Healthcare
  - Augmented Reality
  - Nano-Robotics

- Challenges
  - High computational demand from DNNs, other algorithms
  - Diverse computational patterns and requirements

- Opportunities
  - Bit-precision tolerance
  - Accelerable workloads
Introduction and Motivation

- Emerging application areas for AI-enabled IoT
  - Personalized Healthcare
  - Augmented Reality
  - Nano-Robotics

- Challenges
  - High computational demand from DNNs, other algorithms
  - Diverse computational patterns and requirements

- Opportunities
  - Bit-precision tolerance
  - Accelerable workloads
Introduction and Motivation

Emerging application areas for AI-enabled IoT
- Personalized Healthcare
- Augmented Reality
- Nano-Robotics

Challenges
- High computational demand from DNNs, other algorithms
- Diverse computational patterns and requirements

Opportunities
- Bit-precision tolerance
- Accelerable workloads
Reference architecture – PULP platform

- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$ cache
  - DMA controller
Reference architecture – PULP platform

- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$
Reference architecture – PULP platform

- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$ (Instruction Cache)
  - DMA controller
Reference architecture – PULP platform

- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$ cache
  - DMA controller
Reference architecture – PULP platform

- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$  
  - DMA controller
RISCY

• 32-bit in-order single-issue RISC-V processor
• 4 pipeline stages
• DSP extensions
• SIMD arithmetic instructions down to 8-bit precision
RISCY

- 32-bit in-order single-issue RISC-V processor
- 4 pipeline stages
- DSP extensions
- SIMD arithmetic instructions down to 8-bit precision

MatMul pseudo-assembly code:

```
l.p.setup       l1, l2, end
p.lw           w1, 4(aw1!)
p.lw           w2, 4(aw2!)
p.lw           w3, 4(aw3!)
p.lw           w4, 4(aw4!)
p.lw           x1, 4(ax1!)
p.lw           x2, 4(ax2!)
pv.sdotp.b     s1, x1, w1
pv.sdotp.b     s2, x1, w2
pv.sdotp.b     s3, x1, w3
pv.sdotp.b     s4, x1, w4
pv.sdotp.b     s5, x2, w1
pv.sdotp.b     s6, x1, w2
pv.sdotp.b     s7, x1, w3
```
end:  

```
pv.sdotp.b     s8, x1, w4
```
RISCY

- 32-bit in-order single-issue RISC-V processor
- 4 pipeline stages
- DSP extensions
- SIMD arithmetic instructions down to 8-bit precision

PERFORMANCE DEGRADATION DUE TO LOAD OPERATIONS WITHIN THE INNERMOST LOOP

MatMul pseudo-assembly code:

```
lp.setup   l1, l2, end
p.lw       w1, 4(aw1!)    
p.lw       w2, 4(aw2!)    
p.lw       w3, 4(aw3!)    
p.lw       w4, 4(aw4!)    
p.lw       x1, 4(ax1!)    
p.lw       x2, 4(ax2!)    
```

```
pv.sdotp.b s1, x1, w1
pv.sdotp.b s2, x1, w2
pv.sdotp.b s3, x1, w3
pv.sdotp.b s4, x1, w4
pv.sdotp.b s5, x2, w1
pv.sdotp.b s6, x1, w2
pv.sdotp.b s7, x1, w3
```

```
end:      pv.sdotp.b s8, x1, w4
```
### XpulpNN

<table>
<thead>
<tr>
<th>INIT</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN-RF</td>
<td>pv.nnssdotusp.h</td>
<td>zero, aw1,16</td>
</tr>
<tr>
<td></td>
<td>pv.nnssdotusp.h</td>
<td>zero, aw2,18</td>
</tr>
<tr>
<td></td>
<td>pv.nnssdotusp.h</td>
<td>zero, aw3,20</td>
</tr>
<tr>
<td></td>
<td>pv.nnssdotusp.h</td>
<td>zero, aw4,22</td>
</tr>
<tr>
<td></td>
<td>pv.nnssdotusp.h</td>
<td>zero, ax1,8</td>
</tr>
</tbody>
</table>

Single-cycle MAC + load instruction (Mac&Load) down to 2-bit width
XpulpNN

Single-cycle MAC + load instruction (Mac&Load) down to 2-bit width

Only one explicit load inside the innermost loop
Single-cycle MAC + load instruction (Mac&Load) down to 2-bit width

Only one explicit load inside the innermost loop

- 10.9x  - 3.61x  - 4.36x
MPIC

- HW sub-byte and mixed-precision sum of dot products (sdotp)
MPIC

- HW sub-byte and mixed-precision sum of dot products (sdotp)
- Virtual SIMD instructions
- Dynamic bit-scalable execution mode

No more need for packing operations!
MPIC

- HW sub-byte and mixed-precision sum of dot products (sdotp)
- Virtual SIMD instructions
- Dynamic bit-scalable execution mode

No more need for packing operations!
Our proposal for energy-efficient inference of DNNs

- **Flex-V core**
  - Performance of XpulpNN extensions
  - Flexibility of MPIC
  - *Mixed-precision Mac&Load instructions*

- Optimized SW library targeting well-known mixed-precision QNNs
Our proposal for energy-efficient inference of DNNs

- Flex-V core
  - Performance of XpulpNN extensions
  - Flexibility of MPIC
  - Mixed-precision *Mac&Load* instructions

- Optimized SW library targeting well-known mixed-precision QNNs
### MatMul – Memory access pattern

<table>
<thead>
<tr>
<th>base</th>
<th>+0</th>
<th>+4</th>
<th>+8</th>
<th>…</th>
<th>+w_stride - 4</th>
</tr>
</thead>
</table>

```
A
```

A: \( addr = base \) → \( base \)
MatMul – Memory access pattern

\[ \text{A: } \text{addr} = \text{base} \rightarrow \text{base} \]

\[ \text{B: } \text{addr} + = \text{w\_stride} \rightarrow \text{base} + \text{w\_stride} \]
MatMul – Memory access pattern

A: $addr = base$ → $base$
B: $addr += w_{\text{stride}}$ → $base + w_{\text{stride}}$
C: $addr += w_{\text{stride}}$ → $base + 2 \times w_{\text{stride}}$
MatMul – Memory access pattern

base +0 +4 +8 ... +w_stride - 4

A: \( \text{addr} = \text{base} \) \( \rightarrow \text{base} \)
B: \( \text{addr} += \ w_{\text{stride}} \) \( \rightarrow \text{base} + w_{\text{stride}} \)
C: \( \text{addr} += \ w_{\text{stride}} \) \( \rightarrow \text{base} + 2w_{\text{stride}} \)
D: \( \text{addr} += \ w_{\text{stride}} \) \( \rightarrow \text{base} + 3w_{\text{stride}} \)
MatMul – Memory access pattern

base +0 +4 +8 ... +w_stride - 4

A: addr = base → base
B: addr += w_stride → base + w_stride
C: addr += w_stride → base + 2*w_stride
D: addr += w_stride → base + 3*w_stride
E: addr += w_rollback → base + 4
Automatic address generation

• Based on static information related to the kernel
• Needed invariant parameters are stored in CSRs
• Only one pointer for activations and one for weights
• Extensible to all 2D strided patterns, not only MatMuls!!
Automatic address generation

- Based on static information related to the kernel
- Needed invariant parameters are stored in CSRs
- Only one pointer for activations and one for weights
- **Extensible to all 2D strided patterns, not only MatMuls!!**

**How does it work?**

1. Read current address

   ![Diagram](image)
Automatic address generation

• Based on static information related to the kernel
• Needed invariant parameters are stored in CSRs
• Only one pointer for activations and one for weights
• Extensible to all 2D strided patterns, not only MatMuls!!

How does it work?
1. Read current address
2. Check number of performed updates
Automatic address generation

- Based on static information related to the kernel
- Needed invariant parameters are stored in CSRs
- Only one pointer for activations and one for weights
- Extensible to all 2D strided patterns, not only MatMuls!!

How does it work?
1. Read current address
2. Check number of performed updates
3. Adds proper increment
Automatic address generation

- Based on static information related to the kernel
- Needed invariant parameters are stored in CSRs
- Only one pointer for activations and one for weights
- **Extensible to all 2D strided patterns, not only MatMuls!!**

**How does it work?**

1. Read current address
2. Check number of performed updates
3. Adds proper increment
4. New address stored back in related CSR
Mixed-precision + M&L + Automatic Address Generation

csrwi sb_legacy, 0
csrwi simd_fmt, 8
csrwi mix_skip, 16
csrw a_stride, A_STRIDE
csrw w_stride, W_STRIDE
csrw a_rollback, A_ROLLB
csrw w_rollback, W_ROLLB
csrw a_csr, A_BASE_ADDR
csrw w_csr, W_BASE_ADDR
pv.mlsdotp.h zero, aw, 16
pv.mlsdotp.h zero, aw, 18
pv.mlsdotp.h zero, aw, 20
pv.mlsdotp.h zero, aw, 22
pv.mlsdotp.h zero, ax, 8
lp.setup 11, 12, end

pv.mlsdotp.h zero, ax, 9
pv.mlsdotusp.b s1, aw, 0
pv.mlsdotusp.b s2, aw, 2
pv.mlsdotusp.b s3, aw, 4
pv.mlsdotusp.b s4, ax, 14
...
pv.mlsdotusp.b s13, aw, 1
pv.mlsdotusp.b s14, aw, 3
pv.mlsdotusp.b s15, aw, 5
pv.mlsdotusp.b s16, ax, 15
pv.mlsdotusp.b s1, aw, 0
...
pv.mlsdotusp.b s13, aw, 17
pv.mlsdotusp.b s14, aw, 19
pv.mlsdotusp.b s15, aw, 21
(end): pv.mlsdotusp.b s16, aw, 23
Mixed-precision + M&L + Automatic Address Generation

- NO extraction/packing operation within the innermost loop of MatMuls
- Masked load operations
- Extension of the MatMul unrolling factor

```assembly
csrwi sb_legacy, 0
csrwi simd_fmt, 8
csrwi mix_skip, 16
csrw a_stride, A_STRIDE
csrw w_stride, W_STRIDE
csrw a.rollback, A_ROLLB
csrw w.rollback, W_ROLLB
csrw a_csr, A_BASE_ADDR
csrw w_csr, W_BASE_ADDR
pv.mlsdotsp.h zero, aw, 16
pv.mlsdotsp.h zero, aw, 18
pv.mlsdotsp.h zero, aw, 20
pv.mlsdotsp.h zero, aw, 22
pv.mlsdotsp.h zero, ax, 8
lp.setup 11, 12, end

<table>
<thead>
<tr>
<th>Outside the Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv.mlsdotsp.h zero, ax, 9</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s1, aw, 0</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s2, aw, 2</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s3, aw, 4</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s4, ax, 14</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s13, aw, 1</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s14, aw, 3</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s15, aw, 5</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s16, ax, 15</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s1, aw, 0</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s13, aw, 17</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s14, aw, 19</td>
</tr>
<tr>
<td>pv.mlsdotsp.b s15, aw, 21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inside the Kernel</th>
</tr>
</thead>
</table>

| pv.mlsdotsp.h zero, aw, 16 |
| pv.mlsdotsp.b s1, aw, 0 |
| pv.mlsdotsp.b s2, aw, 2 |
| pv.mlsdotsp.b s3, aw, 4 |
| pv.mlsdotsp.b s4, ax, 14 |
| ... |
| pv.mlsdotsp.b s13, aw, 1 |
| pv.mlsdotsp.b s14, aw, 3 |
| pv.mlsdotsp.b s15, aw, 5 |
| pv.mlsdotsp.b s16, ax, 15 |
| pv.mlsdotsp.b s1, aw, 0 |
| ... |
| pv.mlsdotsp.b s13, aw, 17 |
| pv.mlsdotsp.b s14, aw, 19 |
| pv.mlsdotsp.b s15, aw, 21 |

(end): pv.mlsdotsp.b s16, aw, 23
```
• NO extraction/packing operation within the innermost loop of MatMuls
• Masked load operations
• Extension of the MatMul unrolling factor
• At the cost of simple writings to the CSRs outside the body of the loop
Results – Single kernels Performance

MAC/cycle

- a2w2
- a4w2
- a4w4
- a8w2
- a8w4
- a8w8

- Flex-V
- XpulpNN
- MPIC
- RISCY
Results – Single kernels Performance
Results – Single kernels Performance

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Flex-V</th>
<th>XpulpNN</th>
<th>MPIC</th>
<th>RISCY</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2w2</td>
<td>19x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a4w2</td>
<td>11x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a4w4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a8w2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a8w4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a8w8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MAC/cycle
Results – Single kernels Performance

- **a2w2**: 11x improvement
- **a4w2**: 3x improvement
- **a4w4**: 19x improvement
- **a8w2**: 3x improvement
- **a8w4**: 19x improvement
- **a8w8**: 19x improvement

- Flex-V
- XpulpNN
- MPIC
- RISCY
Results – Single kernels Energy Efficiency

Physical implementation in GF-22nm technology

![Bar chart showing energy efficiency comparisons for different kernels and architectures (Flex-V, XpulpNN, MPIC, RISCY). The chart illustrates the efficiency of each kernel in TOPS/W.](image-url)
Results – Single kernels Energy Efficiency

MORE THAN 3 TOPS/W!!
## Results – Full network

<table>
<thead>
<tr>
<th>Network</th>
<th>MobileNetV1 (8b)</th>
<th>MobileNetV1 (8b4b)</th>
<th>ResNet-20 (4b2b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 Accuracy</td>
<td>69.3 %</td>
<td>66.0 %</td>
<td>90.2 % [1]</td>
</tr>
<tr>
<td>Deg. W.r.t. 8b</td>
<td>-</td>
<td>3.3 %</td>
<td>0.15 %</td>
</tr>
<tr>
<td>Model size</td>
<td>1.9 MB</td>
<td>997 kB</td>
<td>142 kB</td>
</tr>
<tr>
<td>Memory saved</td>
<td>-</td>
<td>47 %</td>
<td>63 %</td>
</tr>
</tbody>
</table>

### Performance (MAC/cycle)

<table>
<thead>
<tr>
<th></th>
<th>STM32H7</th>
<th>MobileNetV1 (8b4b)</th>
<th>ResNet-20 (4b2b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XpulpV2</td>
<td>5.6</td>
<td>3.2</td>
<td>4.8</td>
</tr>
<tr>
<td>XpulpNN</td>
<td>6.0</td>
<td>2.7</td>
<td>4.4</td>
</tr>
<tr>
<td>Flex-V</td>
<td>6.0</td>
<td>5.8</td>
<td>11.2</td>
</tr>
</tbody>
</table>

[1] Z. Dong et al., «HAWQ, Hessian Aware Quantization of Neural Networks With Mixed-Precision»
## Results – Full network

<table>
<thead>
<tr>
<th>Network</th>
<th>MobileNetV1 (8b)</th>
<th>MobileNetV1 (8b4b)</th>
<th>ResNet-20 (4b2b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 Accuracy</td>
<td>69.3 %</td>
<td>66.0 %</td>
<td>90.2 % [1]</td>
</tr>
<tr>
<td>Deg. W.r.t. 8b</td>
<td>-</td>
<td>3.3 %</td>
<td>0.15 %</td>
</tr>
<tr>
<td>Model size</td>
<td>1.9 MB</td>
<td>997 kB</td>
<td>142 kB</td>
</tr>
<tr>
<td>Memory saved</td>
<td>-</td>
<td>47 %</td>
<td>63 %</td>
</tr>
</tbody>
</table>

### Performance (MAC/cycle)

<table>
<thead>
<tr>
<th>Platform</th>
<th>STM32H7</th>
<th>XpulpV2</th>
<th>XpulpNN</th>
<th>Flex-V</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32H7</td>
<td>0.33</td>
<td>5.6</td>
<td>6.0</td>
<td>6.0</td>
</tr>
<tr>
<td>XpulpV2</td>
<td>0.30</td>
<td>3.2</td>
<td>2.7</td>
<td>5.8</td>
</tr>
<tr>
<td>XpulpNN</td>
<td>-</td>
<td>4.8</td>
<td>4.4</td>
<td>11.2</td>
</tr>
</tbody>
</table>

[1] Z. Dong et al., «HAWQ, Hessian Aware Quantization of Neural Networks With Mixed-Precision»
## Results – Full network

<table>
<thead>
<tr>
<th>Network</th>
<th>MobileNetV1 (8b)</th>
<th>MobileNetV1 (8b4b)</th>
<th>ResNet-20 (4b2b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 Accuracy</td>
<td>69.3 %</td>
<td>66.0 %</td>
<td>90.2 % [1]</td>
</tr>
<tr>
<td>Deg. W.r.t. 8b</td>
<td>-</td>
<td>3.3 %</td>
<td>0.15 %</td>
</tr>
<tr>
<td>Model size</td>
<td>1.9 MB</td>
<td>997 kB</td>
<td>142 kB</td>
</tr>
<tr>
<td>Memory saved</td>
<td>-</td>
<td>47 %</td>
<td>63 %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance (MAC/cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32H7</td>
</tr>
<tr>
<td>XpulpV2</td>
</tr>
<tr>
<td>XpulpNN</td>
</tr>
<tr>
<td>Flex-V</td>
</tr>
</tbody>
</table>

[1] Z. Dong et al., «HAWQ, Hessian Aware Quantization of Neural Networks With Mixed-Precision»
## Results – Full network

<table>
<thead>
<tr>
<th>Network</th>
<th>MobileNetV1 (8b)</th>
<th>MobileNetV1 (8b4b)</th>
<th>ResNet-20 (4b2b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 Accuracy</td>
<td>69.3 %</td>
<td>66.0 %</td>
<td>90.2 % [1]</td>
</tr>
<tr>
<td>Deg. W.r.t. 8b</td>
<td>-</td>
<td>3.3 %</td>
<td>0.15 %</td>
</tr>
<tr>
<td>Model size</td>
<td>1.9 MB</td>
<td>997 kB</td>
<td>142 kB</td>
</tr>
<tr>
<td>Memory saved</td>
<td>-</td>
<td>47 %</td>
<td>63 %</td>
</tr>
</tbody>
</table>

### Performance (MAC/cycle)

<table>
<thead>
<tr>
<th></th>
<th>STM32H7</th>
<th>XpulpV2</th>
<th>XpulpNN</th>
<th>Flex-V</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 20x</td>
<td>0.33</td>
<td>5.6</td>
<td>6.0</td>
<td>6.0</td>
</tr>
<tr>
<td>0.30</td>
<td></td>
<td>3.2</td>
<td>2.7</td>
<td>5.8</td>
</tr>
<tr>
<td>~ 2x</td>
<td>4.8</td>
<td></td>
<td>4.4</td>
<td>11.2</td>
</tr>
</tbody>
</table>

[1] Z. Dong et al., «HAWQ, Hessian Aware Quantization of Neural Networks With Mixed-Precision»
## Results – Full network

<table>
<thead>
<tr>
<th>Network</th>
<th>MobileNetV1 (8b)</th>
<th>MobileNetV1 (8b4b)</th>
<th>ResNet-20 (4b2b)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Top-1 Accuracy</strong></td>
<td>69.3 %</td>
<td>66.0 %</td>
<td>90.2 % [1]</td>
</tr>
<tr>
<td><strong>Deg. W.r.t. 8b</strong></td>
<td>-</td>
<td>3.3 %</td>
<td>0.15 %</td>
</tr>
<tr>
<td><strong>Model size</strong></td>
<td>1.9 MB</td>
<td>997 kB</td>
<td>142 kB</td>
</tr>
<tr>
<td><strong>Memory saved</strong></td>
<td>-</td>
<td>47 %</td>
<td>63 %</td>
</tr>
</tbody>
</table>

### Performance (MAC/cycle)

<table>
<thead>
<tr>
<th></th>
<th>STM32H7</th>
<th>~ 20x</th>
<th>0.30</th>
<th>~ 2x</th>
<th>~ 2.5x</th>
<th>XpulpV2</th>
<th>3.2</th>
<th>4.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>XpulpNN</td>
<td>6.0</td>
<td></td>
<td>2.7</td>
<td>~ 2x</td>
<td>~ 2.5x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flex-V</td>
<td>6.0</td>
<td></td>
<td>5.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.2</td>
</tr>
</tbody>
</table>

[1] Z. Dong et al., «HAWQ, Hessian Aware Quantization of Neural Networks With Mixed-Precision»
Conclusion

- In this work we proposed a full stack to optimize the inference of fine-grain QNNs
- We designed new RISC-V ISA extensions:
  - Starting from XpulpV2 baseline
  - Mixed-precision *Mac&Load* instructions through SIMD virtual instructions
  - Automatic address generation
- Optimized key kernels for the inference of mixed-precision QNNs
- Outperformed the baseline core by 19x with 91.5 MAC/cycle
- Reached an energy efficiency of 3.3 TOPS/W, likely HW accelerators
- Benchmarked our ISA extensions on full networks
  - Obtaining a speedup against all reference architectures
  - Low Top-1 accuracy loss against a huge reduction of the memory footprint
Thank you!