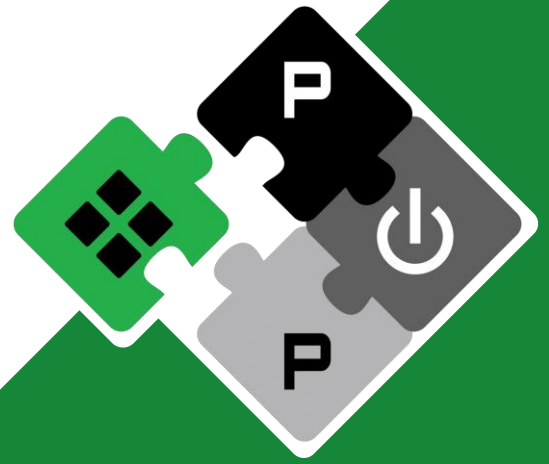
ETH zürich

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

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

Alessandro Nadalini [1], Georg Rutishauser [2], Alessio Burrello [1], Nazareno Bruschi [1], Angelo Garofalo [1], Luca Benini [1,2], Francesco Conti [1], Davide Rossi [1]

[1] University of Bologna, [2] ETH Zürich



neurSoC

TRISTAN
RISC-V

@pulp_platform

pulp-platform.org

youtube.com/pulp_platform

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**

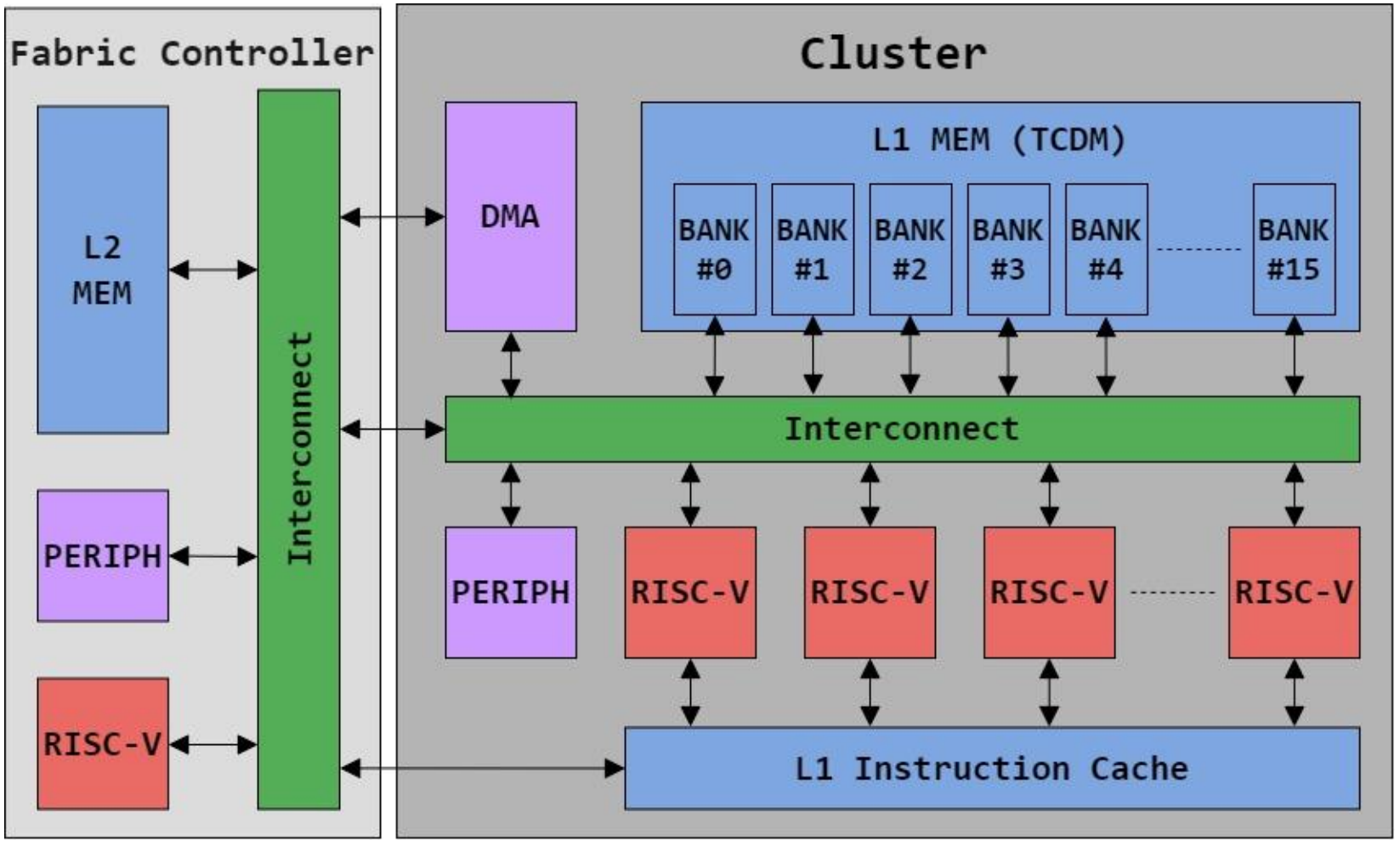
Introduction and Motivation

- **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

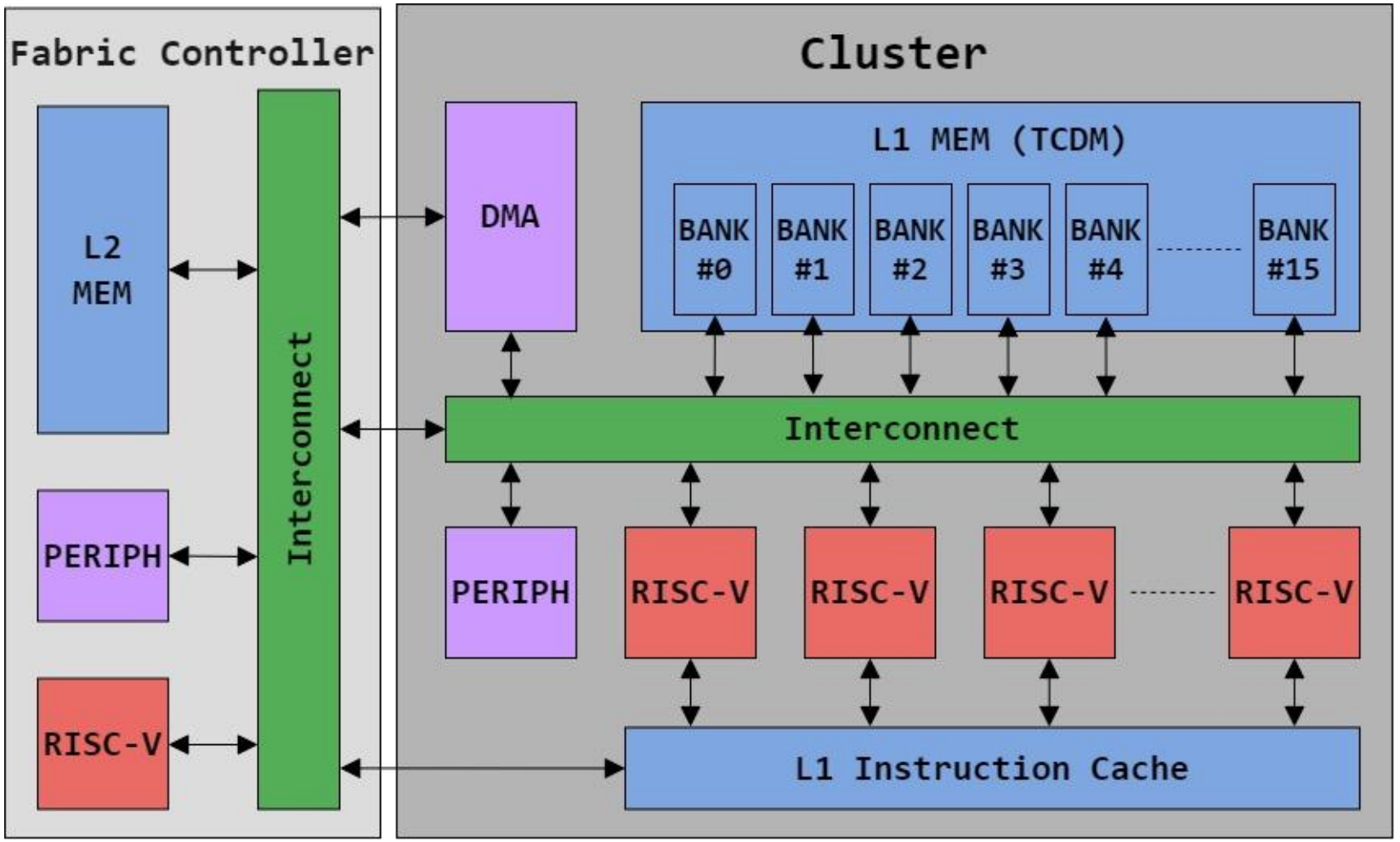
- **Emerging application areas for AI-enabled IoT**
  - Personalized Healthcare
  - Augmented Reality
  - Nano-Robotics
- **Challenges**
  - High computational demand from DNNs, other algorithms
  - Diverse computational patterns and requirements
- **Opportunities**
  - Bit-precision tolerance
  - Accelerable workloads

Reference architecture – PULP platform



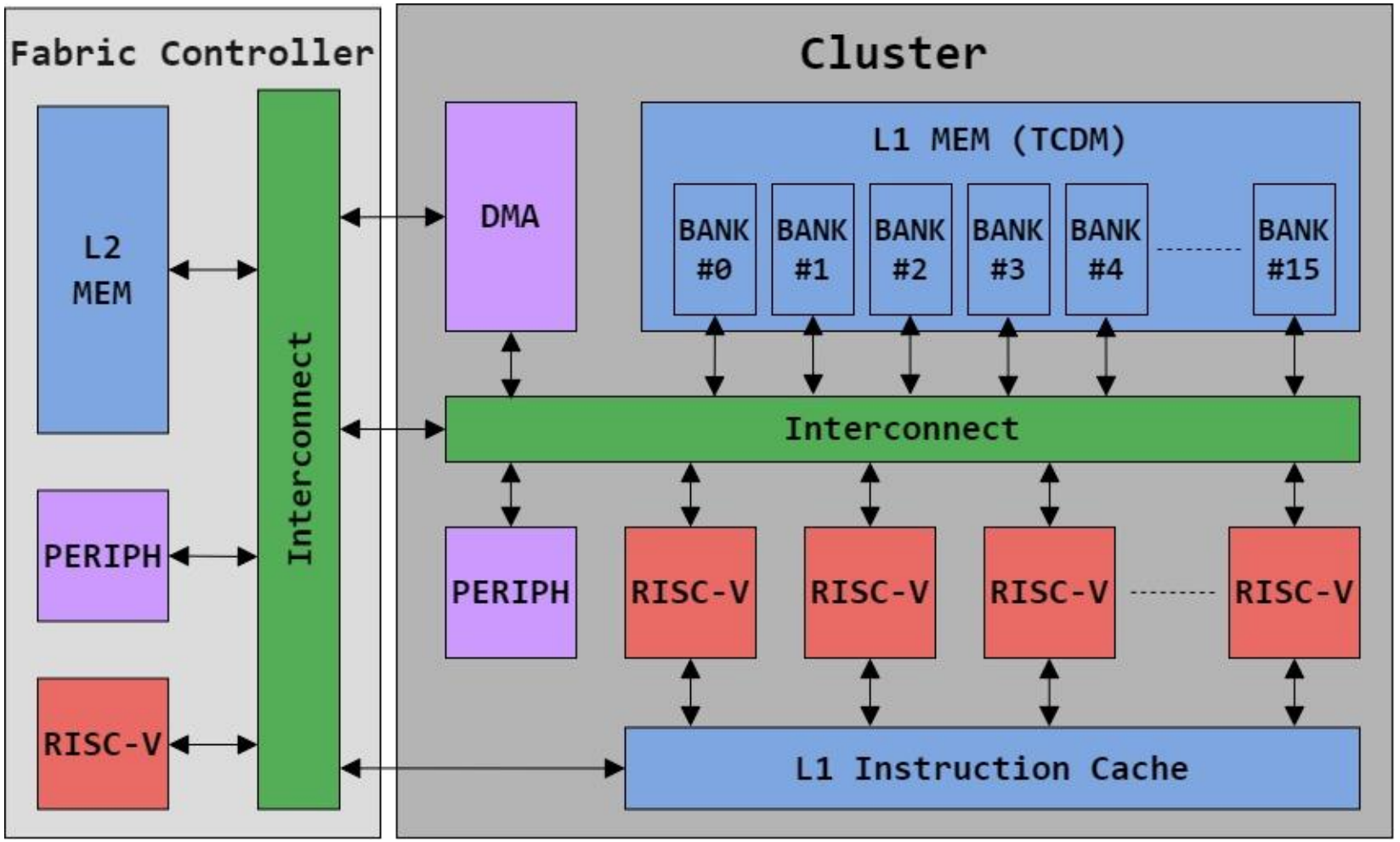
- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$
  - DMA controller

Reference architecture – PULP platform



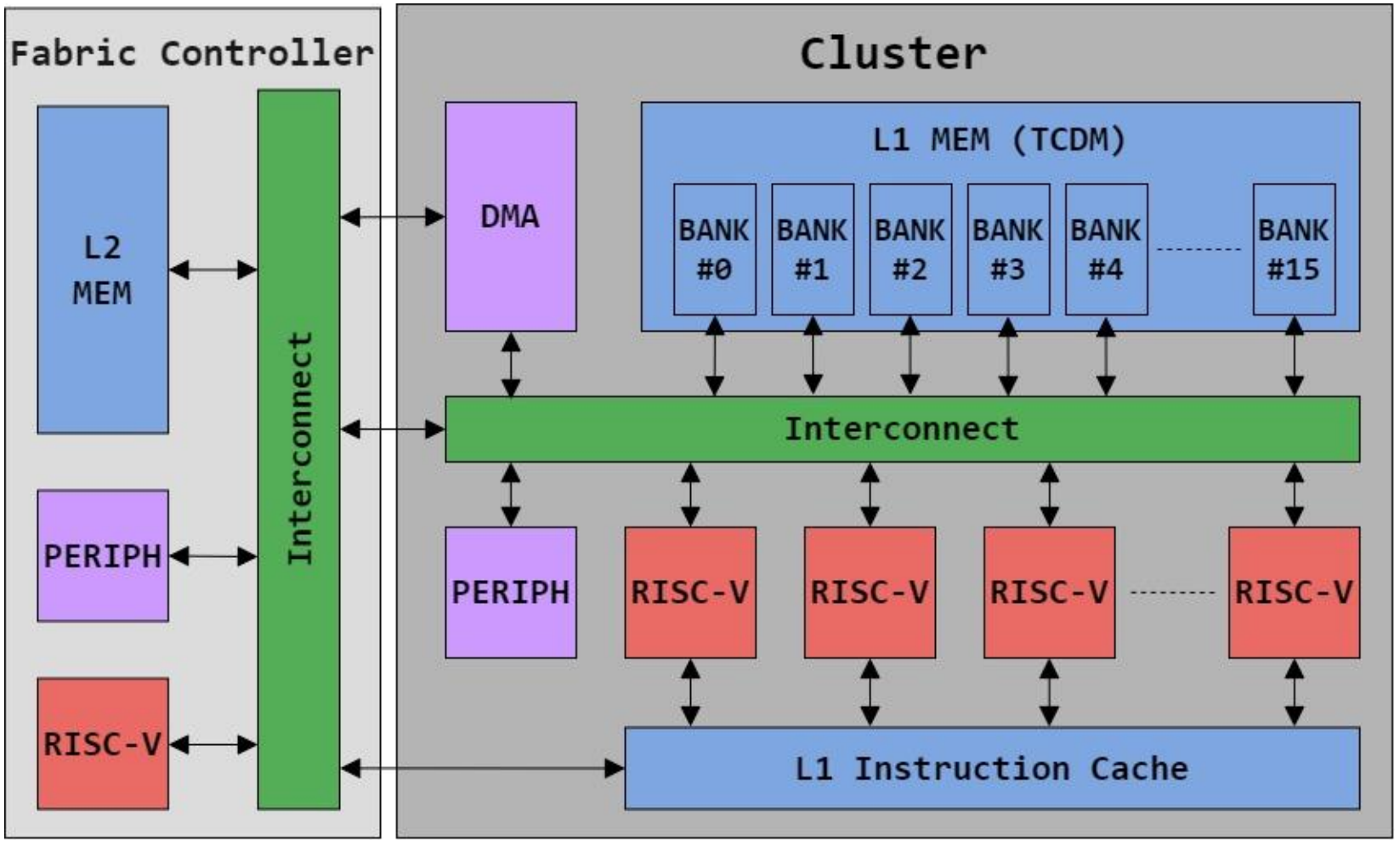
- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$
  - DMA controller

Reference architecture – PULP platform



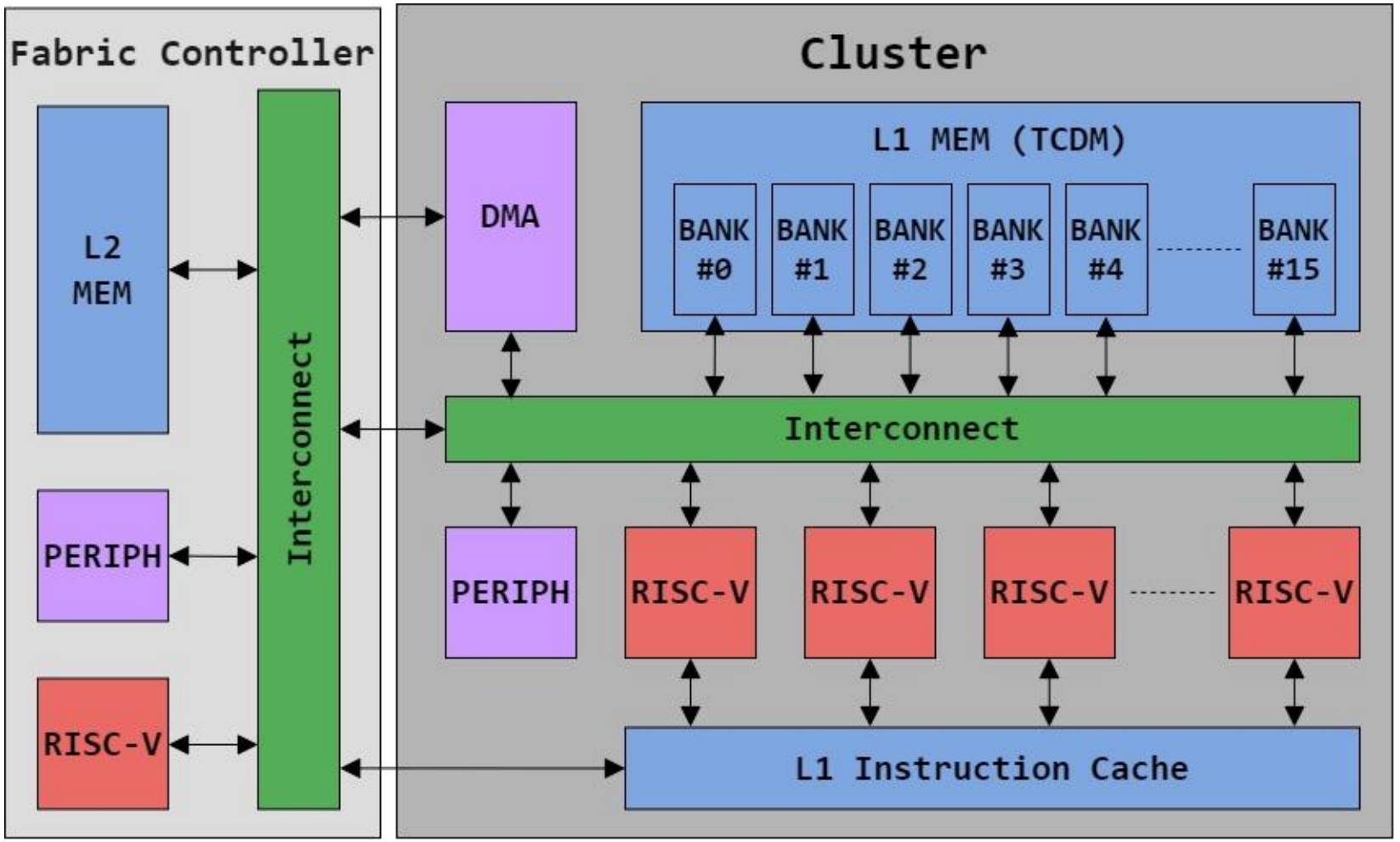
- Fabric Controller (FC)
- Cluster
  - 8 RISC-V processors
  - 128 kB of L1 – TCDM memory
  - L1 I$
  - DMA controller

Reference architecture – PULP platform



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

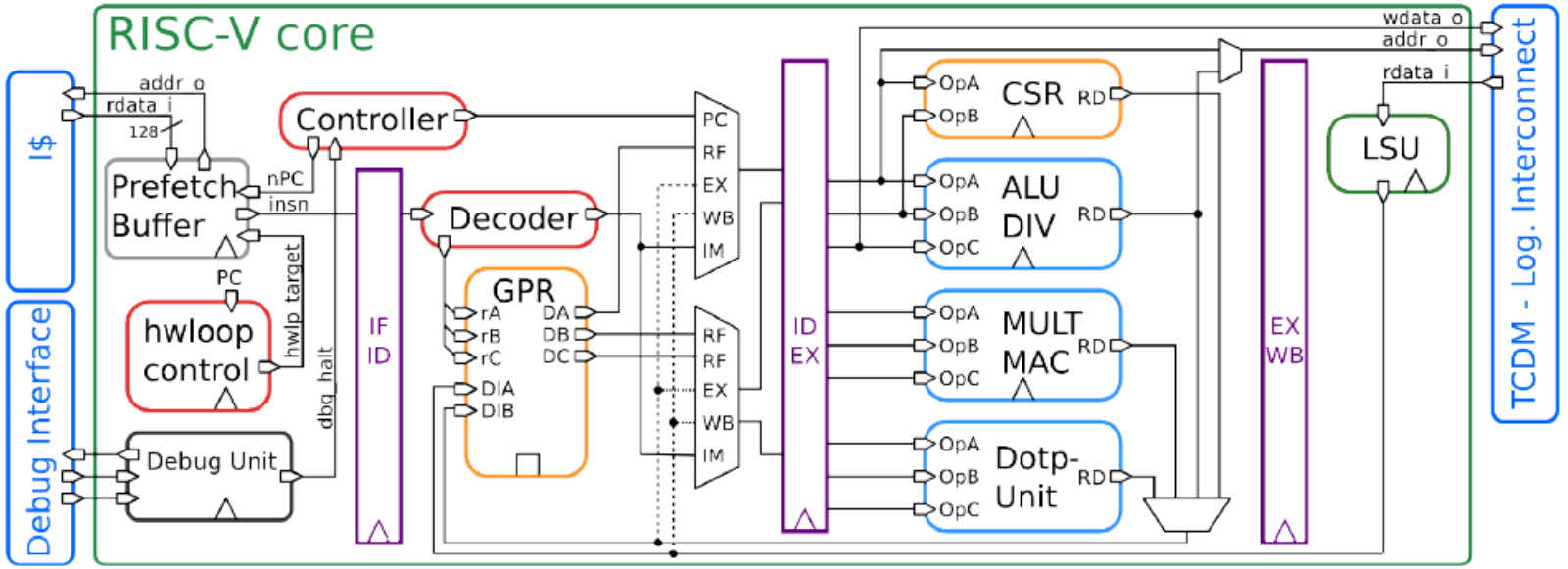
Reference architecture – PULP platform



- 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:

```
        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
```

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

```
INIT NN-RF
pv.nnsdotusp.h  zero, aw1,16
pv.nnsdotusp.h  zero, aw2,18
pv.nnsdotusp.h  zero, aw3,20
pv.nnsdotusp.h  zero, aw4,22
pv.nnsdotusp.h  zero, ax1,8
```

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

XpulpNN

```
INIT NN-RF  pv.nnsdotusp.h  zero, aw1,16
            pv.nnsdotusp.h  zero, aw2,18
            pv.nnsdotusp.h  zero, aw3,20
            pv.nnsdotusp.h  zero, aw4,22
            pv.nnsdotusp.h  zero, ax1,8
            lp.setup        l1, l2, end
            pv.nnsdotup.h   zero,ax2,9
            pv.nnsdotusp.b  s1, aw2, 0
            pv.nnsdotusp.b  s2, aw4, 2
            pv.nnsdotusp.b  s3, aw3, 4
            pv.nnsdotusp.b  s4, ax1, 14
            pv.nnsdotusp.b  s5, aw2, 17
            pv.nnsdotusp.b  s6, aw4, 19
            pv.nnsdotusp.b  s7, aw3, 21
end:        pv.nnsdotusp.b  s8, aw1, 23
```

Only one explicit load inside the innermost loop

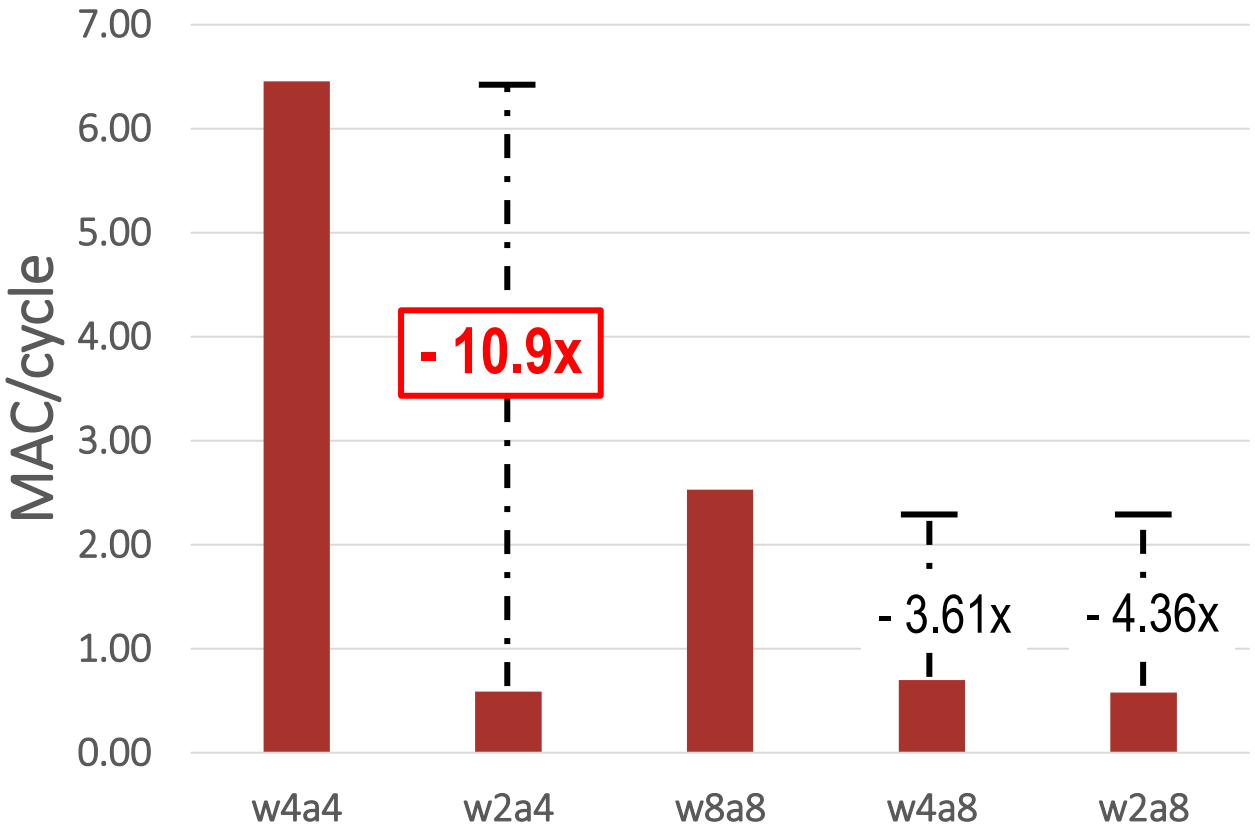
Single-cycle MAC + load instruction (**Mac&Load**) down to 2-bit width

XpulpNN

```
INIT    pv.nnsdotusp.h  zero, aw1,16
NN-RF   pv.nnsdotusp.h  zero, aw2,18
        pv.nnsdotusp.h  zero, aw3,20
        pv.nnsdotusp.h  zero, aw4,22
        pv.nnsdotusp.h  zero, ax1,8
        lp.setup        l1, l2, end
        pv.nnsdotup.h   zero,ax2,9
        pv.nnsdotusp.b  s1, aw2, 0
        pv.nnsdotusp.b  s2, aw4, 2
        pv.nnsdotusp.b  s3, aw3, 4
        pv.nnsdotusp.b  s4, ax1, 14
        pv.nnsdotusp.b  s5, aw2, 17
        pv.nnsdotusp.b  s6, aw4, 19
        pv.nnsdotusp.b  s7, aw3, 21
end:    pv.nnsdotusp.b  s8, aw1, 23
```

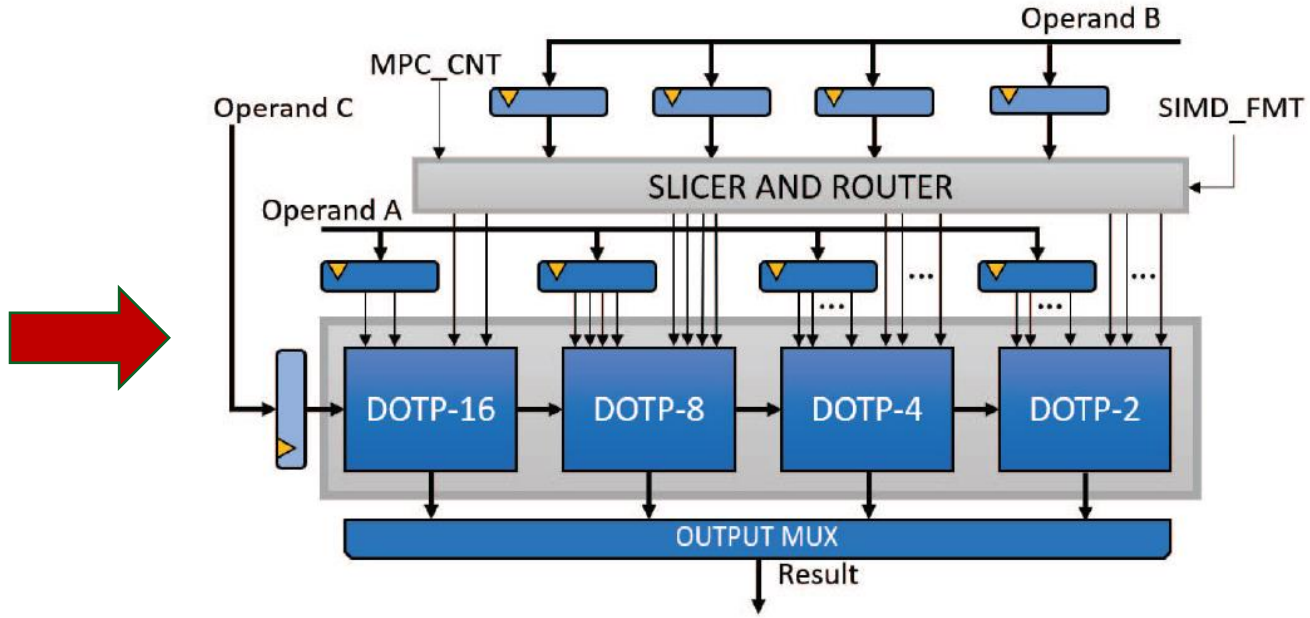
Only one explicit load inside the innermost loop

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



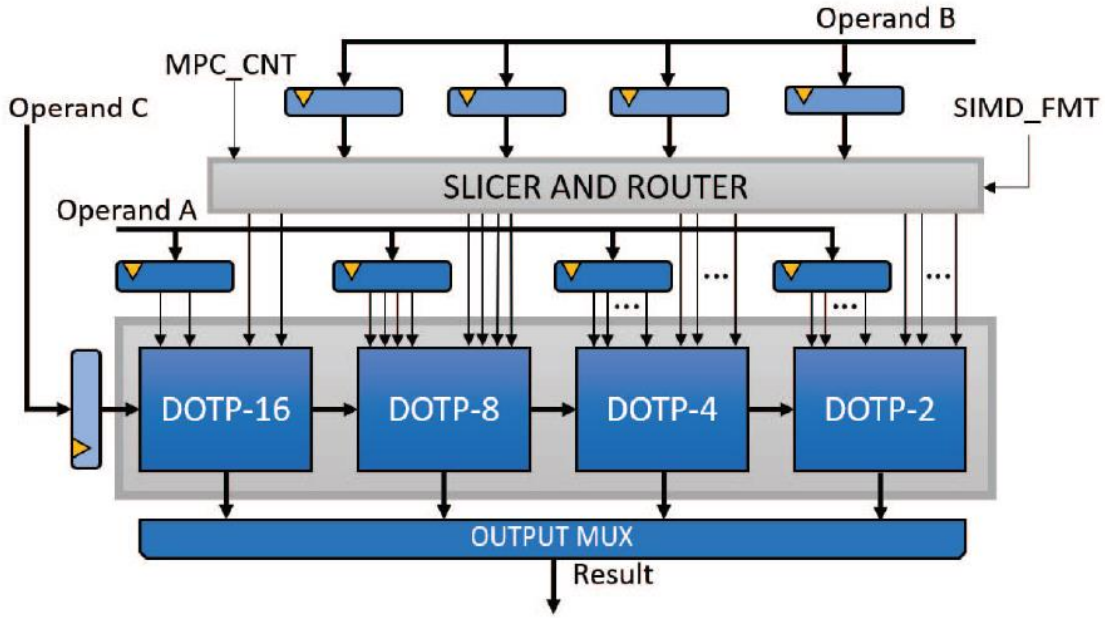
MPIC

- 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**

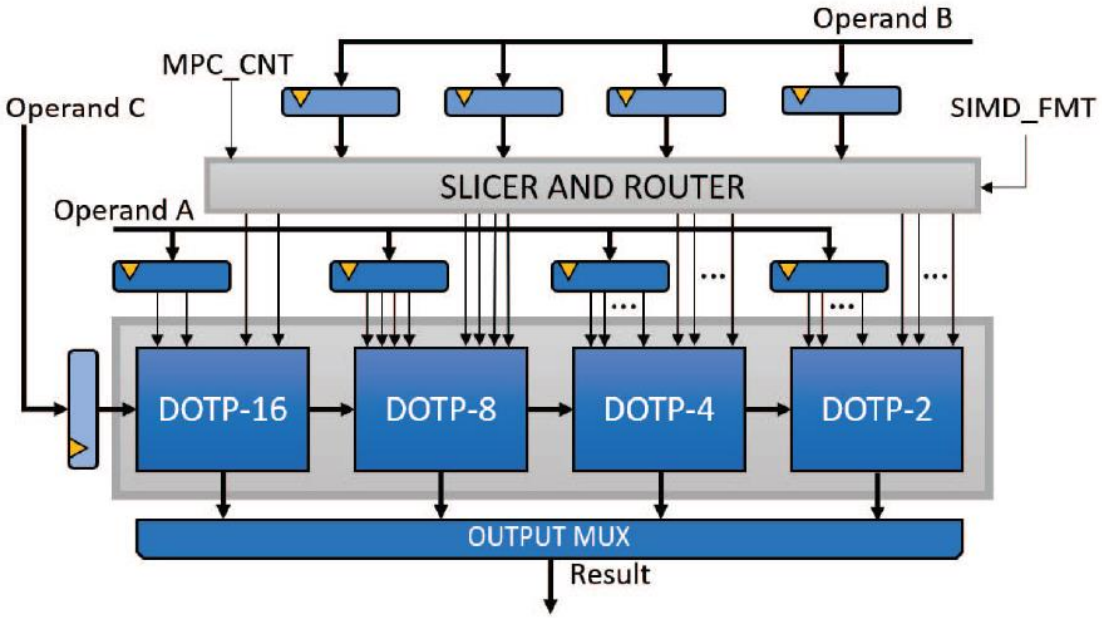


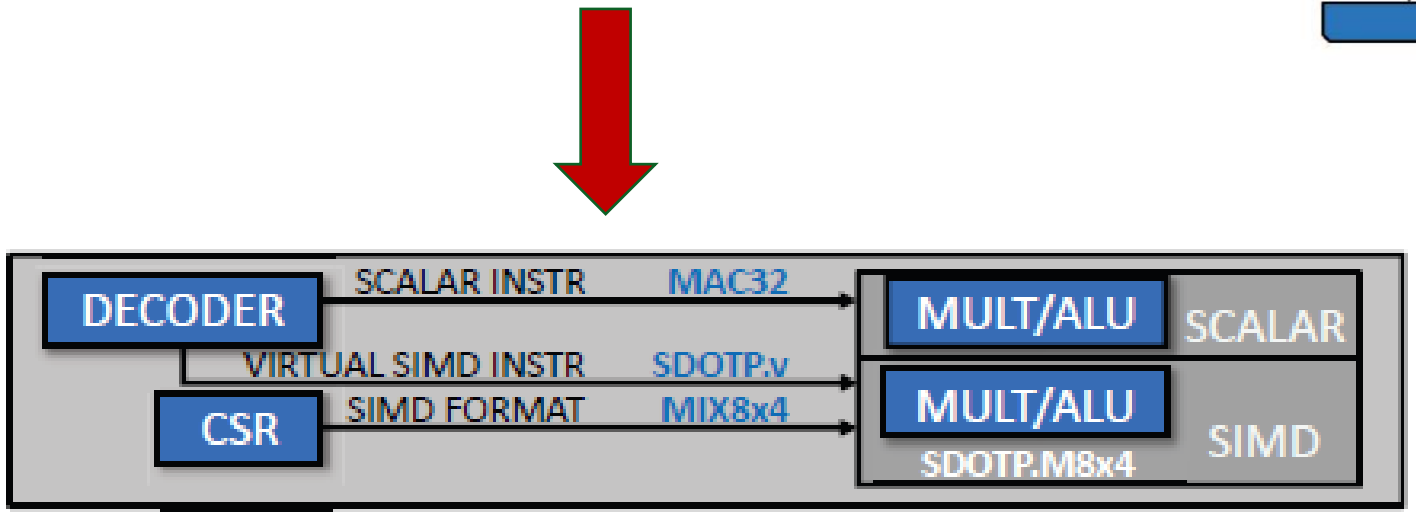
No more need for *packing* operations!

```
p.lw          x10,4(x4!)
p.lw          x11,4(x5!)
p.extract     x5,x11,4,0
p.extract     x6,x11,4,4
p.extract     x7,x11,4,8
p.extract     x8,x11,4,12
pv.packlo.b   x15,x5,x6
pv.packhi.b   x15,x7,x8
pv.sdotsp.b   x20,x15,x10
```

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!

```
p.lw          x10,4(x4!)
p.lw          x11,4(x5!)
p.extract     x5,x11,4,0
p.extract     x6,x11,4,4
p.extract     x7,x11,4,8
p.extract     x8,x11,4,12
pv.packlo.b   x15,x5,x6
pv.packhi.b   x15,x7,x8
pv.sdotsp.b   x20,x15,x10
```

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

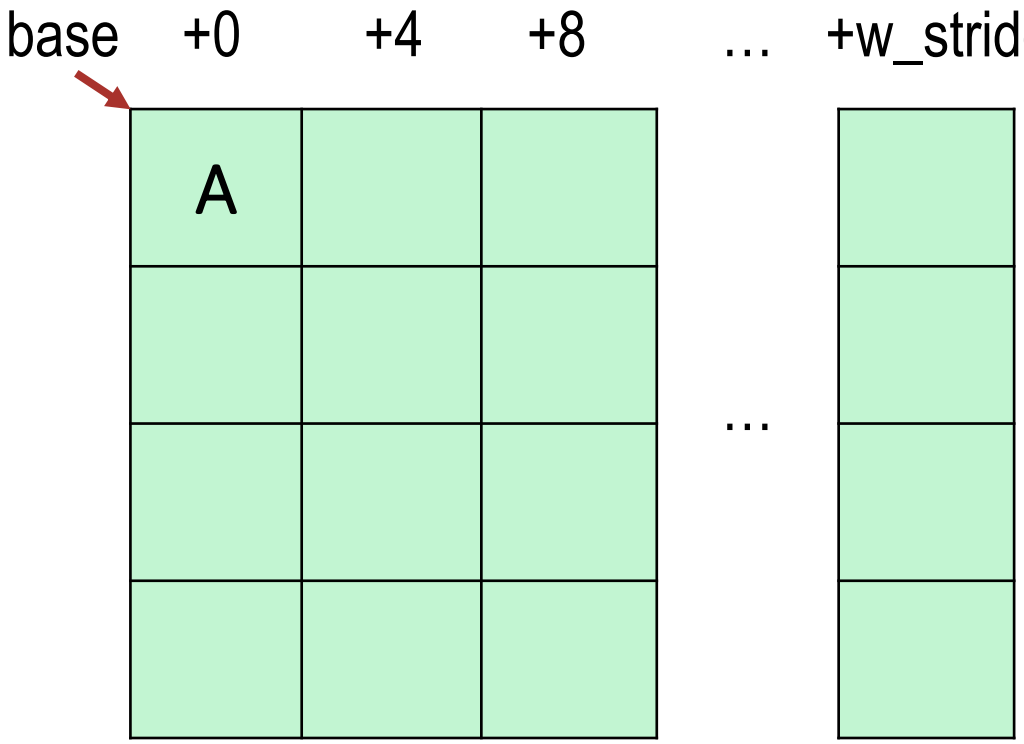
Optimized SW Library

HW support for mixed-precision

Our proposal for energy-efficient inference of DNNs

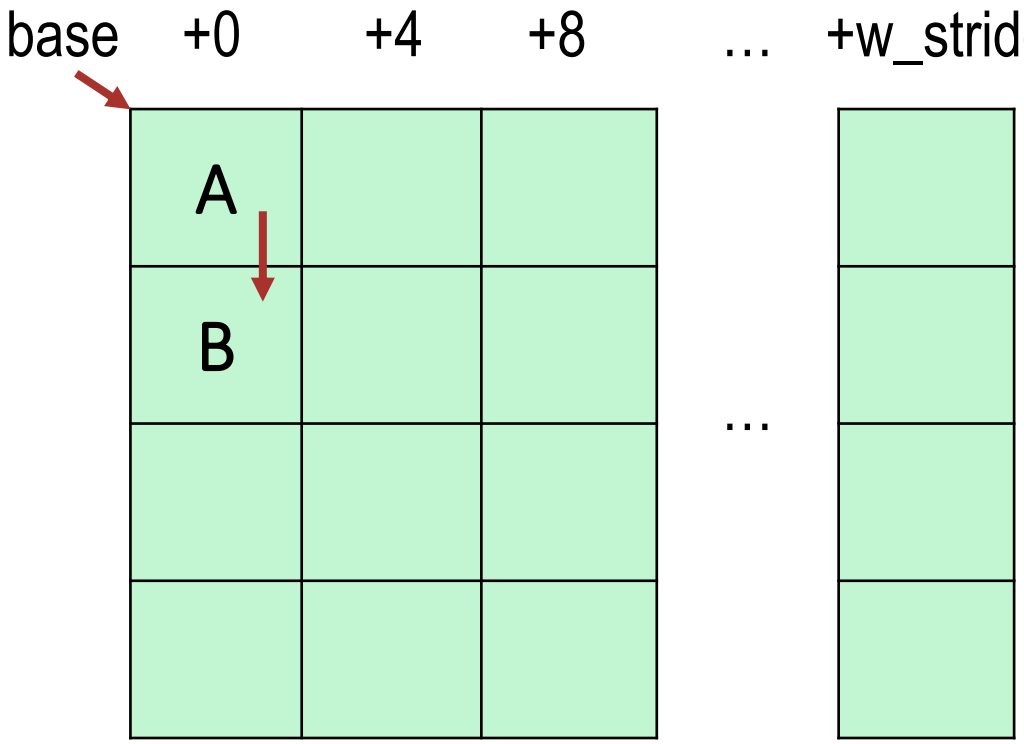
- Flex-V core
  - Performance of XpulpNN extensions
  - Flexibility of MPIC
  - Mixed-precision *Mac&Load* instructions
- Optimized SW library targeting well-known mixed-precision QNNs

MatMul – Memory access pattern



A: *addr* = *base* → *base*

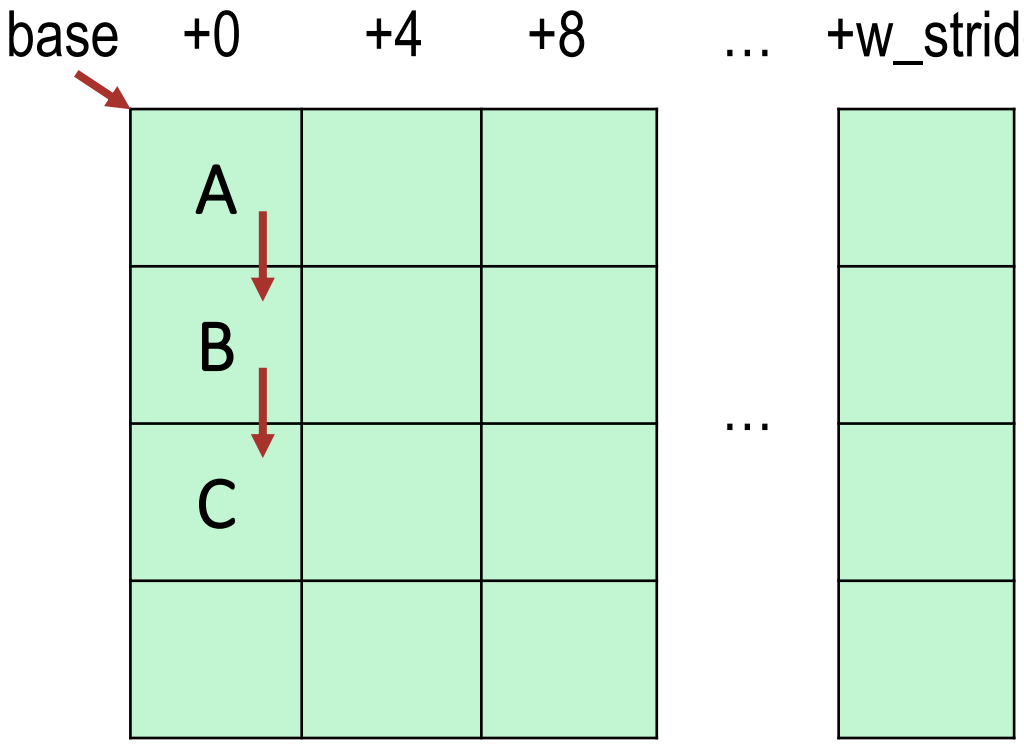
MatMul – Memory access pattern



A: *addr* = *base* → *base*

B: addr += w_stride → *base + w_stride*

MatMul – Memory access pattern

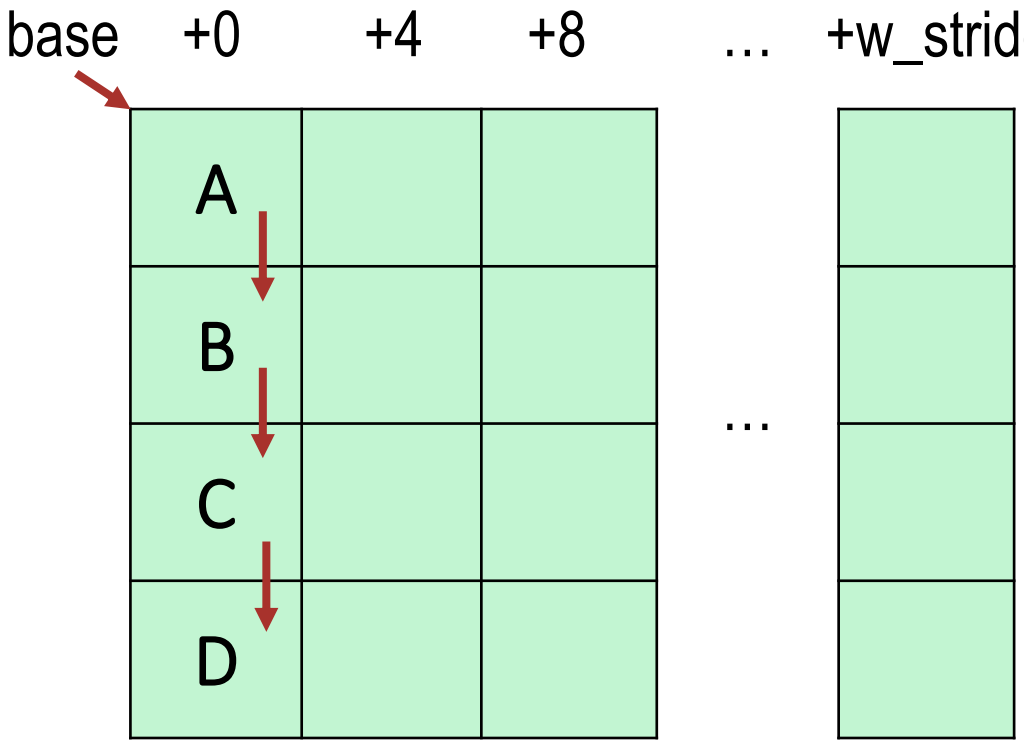


A: *addr = base* → *base*

B: addr += w_stride → *base + w_stride*

C: *addr += w_stride* → *base + 2*w_stride*

MatMul – Memory access pattern



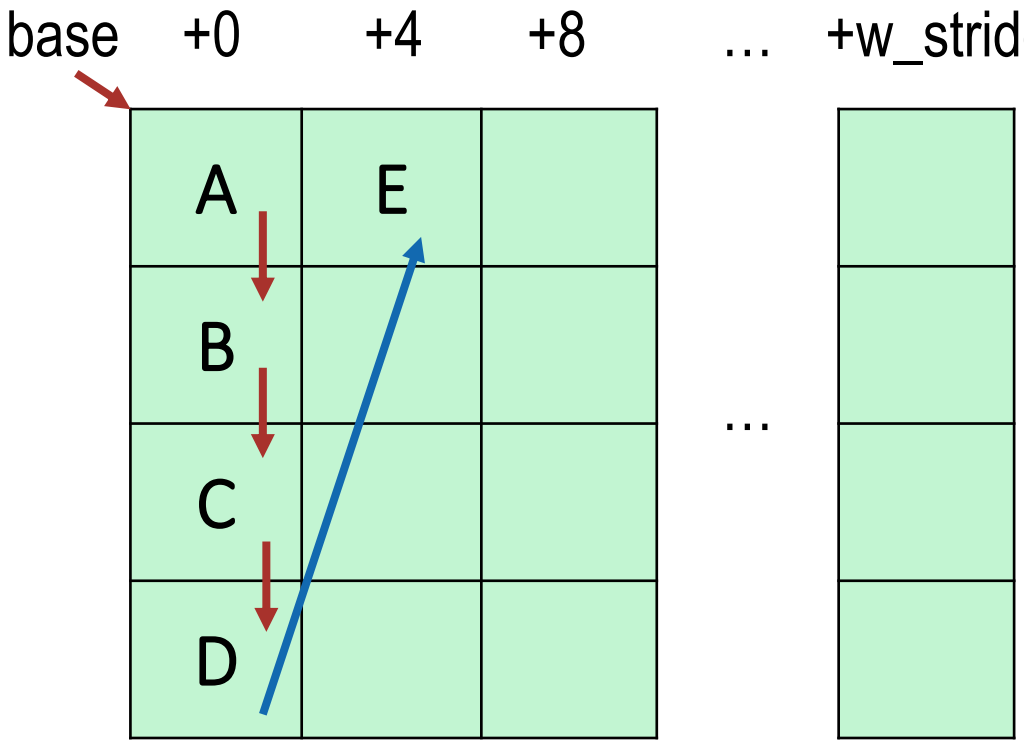
A: *addr = base* → *base*

B: addr += *w_stride* → *base + w_stride*

C: *addr* += *w_stride* → *base + 2*w_stride*

D: *addr* += *w_stride* → *base + 3*w_stride*

MatMul – Memory access pattern



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

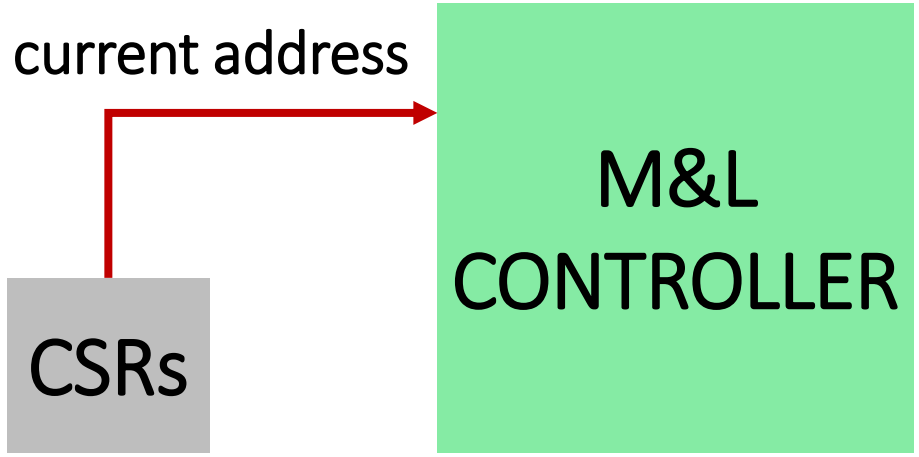
- Based on static information related to the kernel
- Needed invariant parameters are stored in CSRs
- Only one pointer for activations and one for weights
- Extensible to all 2D strided patterns, not only MatMuls!!

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

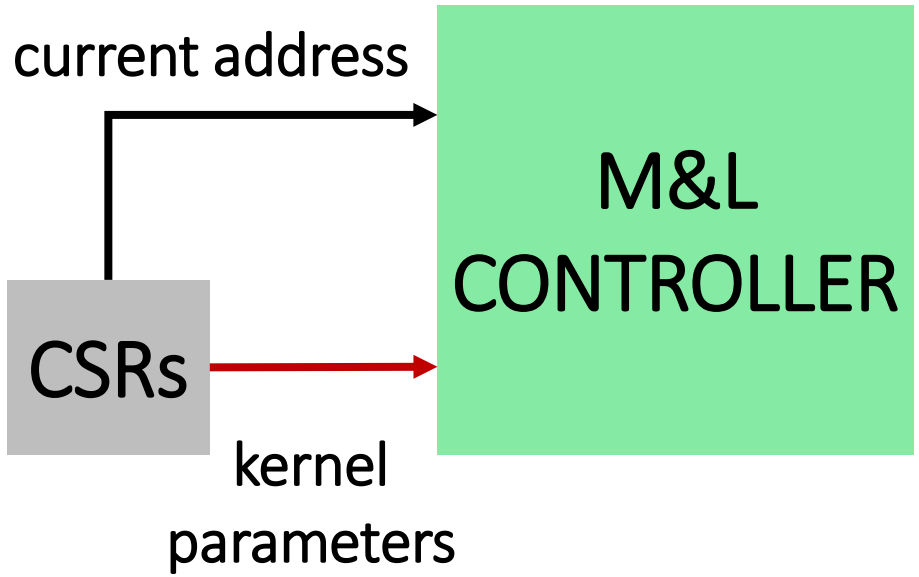


Automatic address generation

- 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

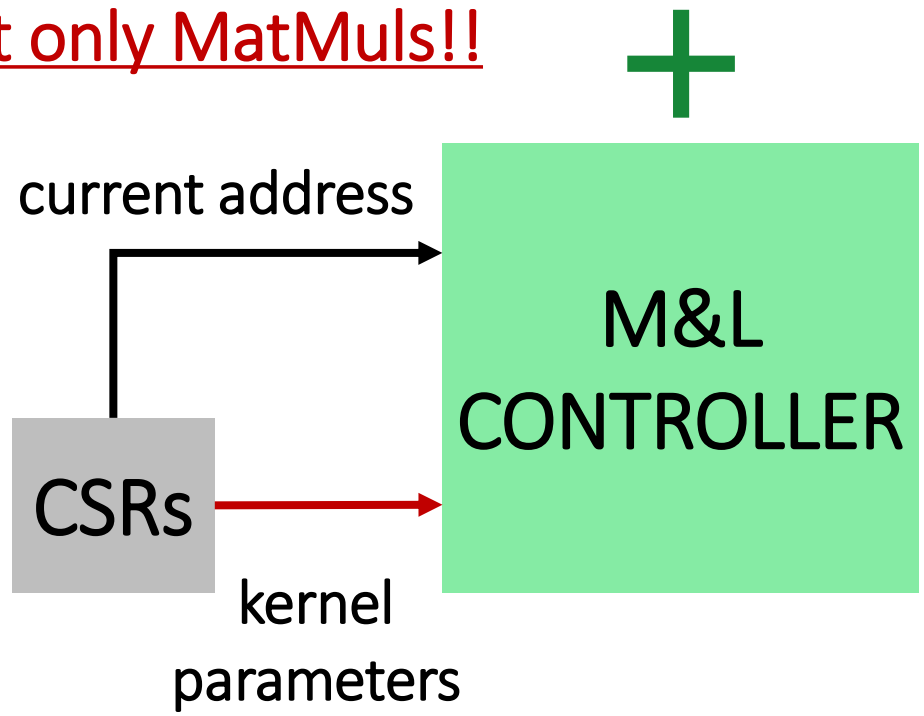


Automatic address generation

- 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

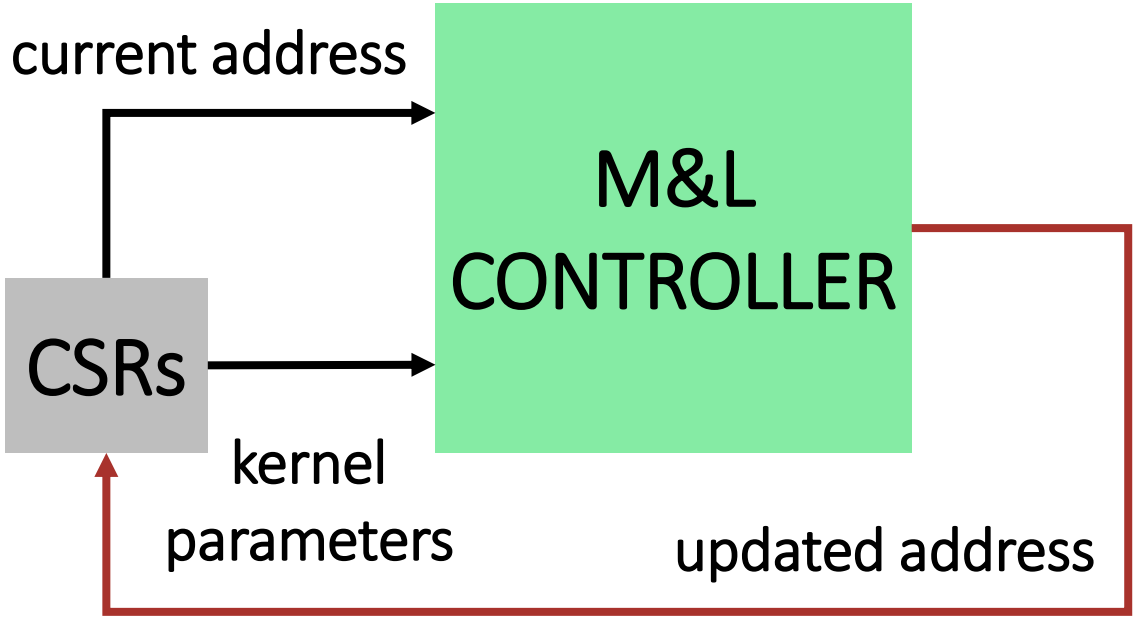


Automatic address generation

- 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.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 l1, l2, end
```

CSRs CONF. OUTSIDE THE KERNEL

CSRs CONF. INSIDE THE KERNEL

INIT THE NN-RF

```
                pv.mlsdotsp.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

```
csrwi sb_legacy,   0          ┐
csrwi simd_fmt,    8          │
csrwi mix_skip,    16         │  CSRs CONF.
csrw  a_stride,    A_STRIDE   ├  OUTSIDE
csrw  w_stride,    W_STRIDE   │  THE KERNEL
csrw  a_rollback,  A_ROLLB    │
csrw  w_rollback,  W_ROLLB    ┘
csrw  a_csr, A_BASE_ADDR      ┐  CSRs CONF.
csrw  w_csr, W_BASE_ADDR      ┘  INSIDE
                                 THE KERNEL
pv.mlsdotsp.h zero, aw, 16    ┐
pv.mlsdotsp.h zero, aw, 18    │  INIT THE
pv.mlsdotsp.h zero, aw, 20    ├  NN-RF
pv.mlsdotsp.h zero, aw, 22    │
pv.mlsdotsp.h zero, ax, 8     ┘
lp.setup l1, l2, end
```

```
                pv.mlsdotsp.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
- At the cost of simple writings to the CSRs outside the body of the loop

```
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 l1, l2, end
```

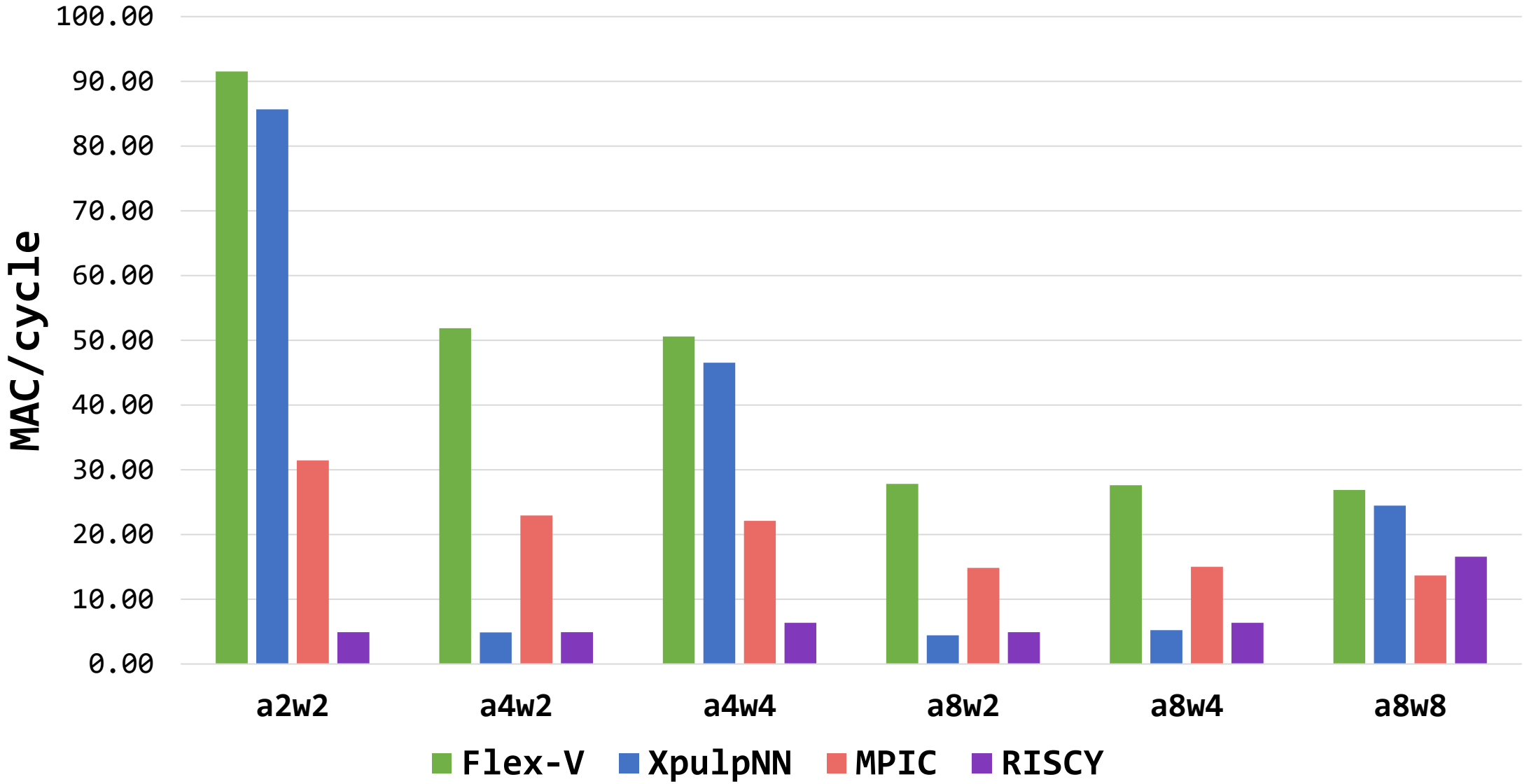
CSRs CONF. OUTSIDE THE KERNEL

CSRs CONF. INSIDE THE KERNEL

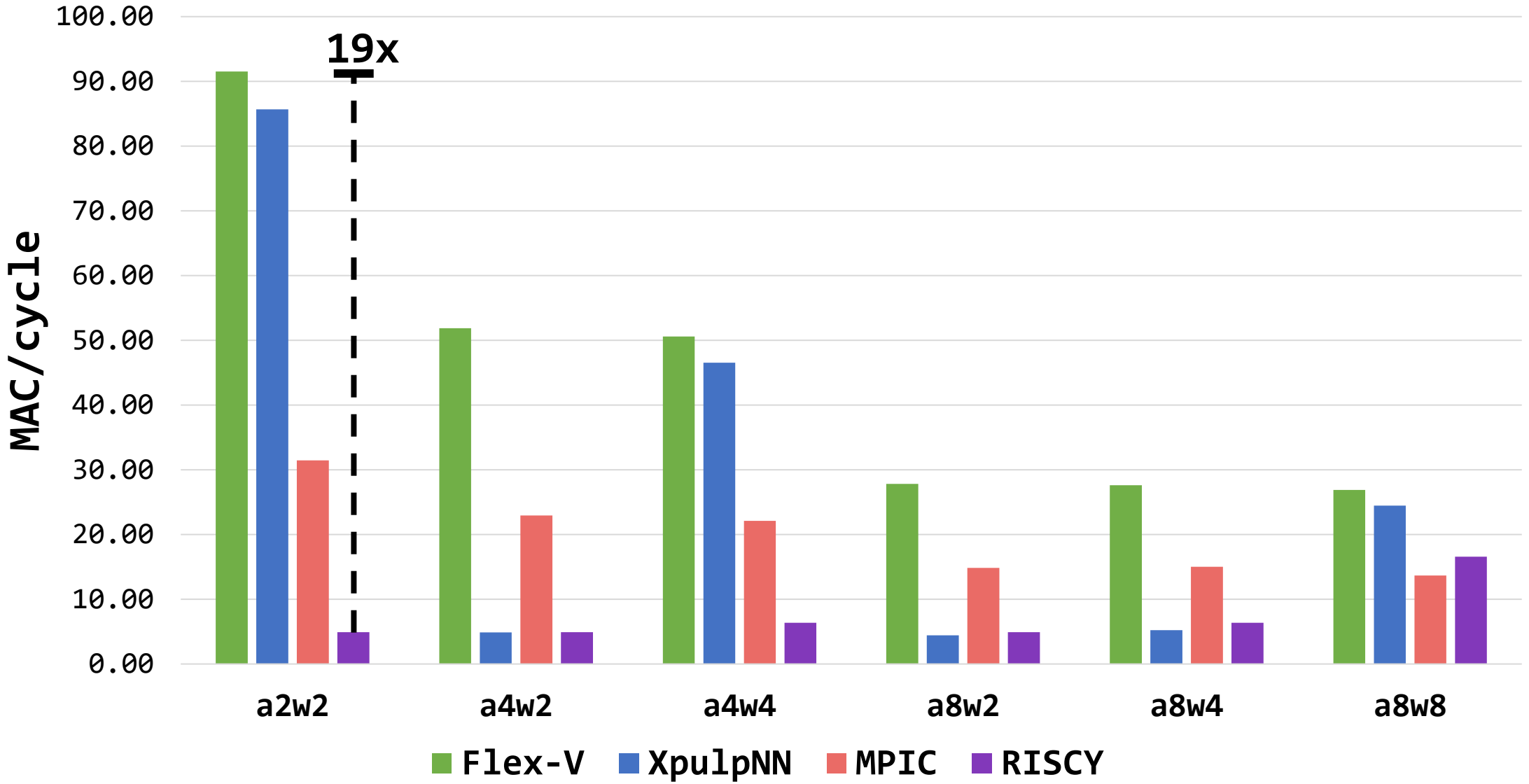
INIT THE NN-RF

```
      pv.mlsdotsp.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
```

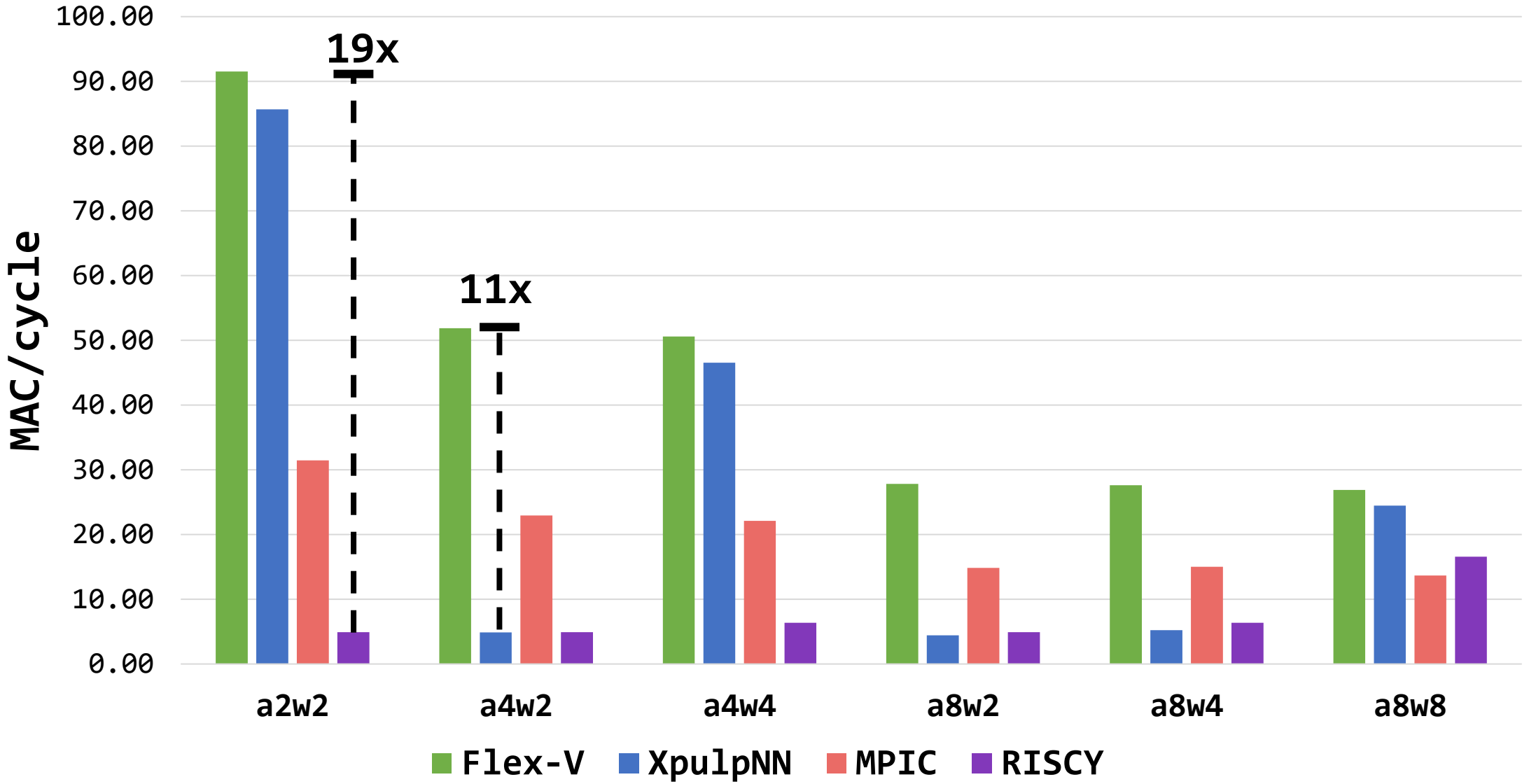
Results – Single kernels Performance



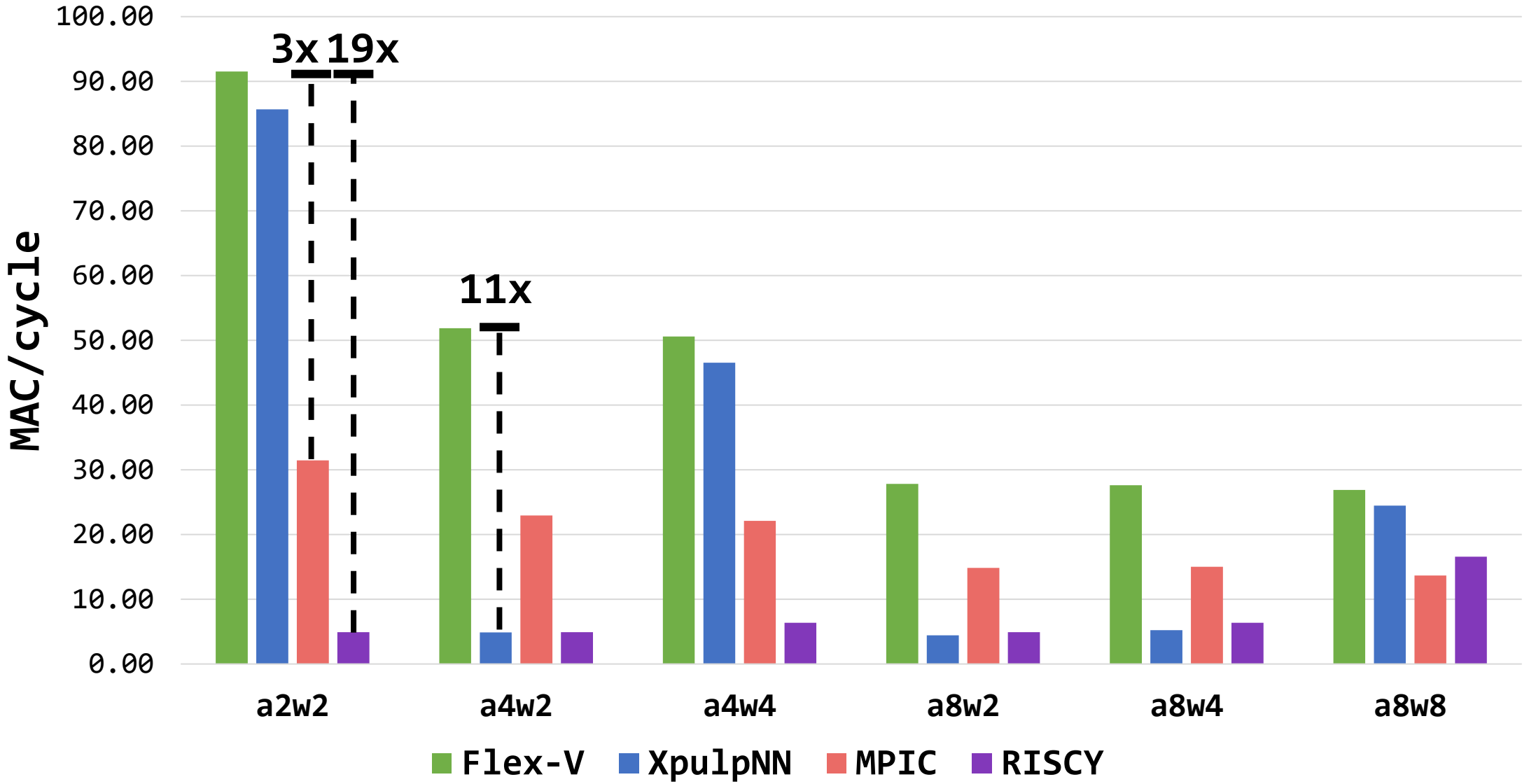
Results – Single kernels Performance



Results – Single kernels Performance

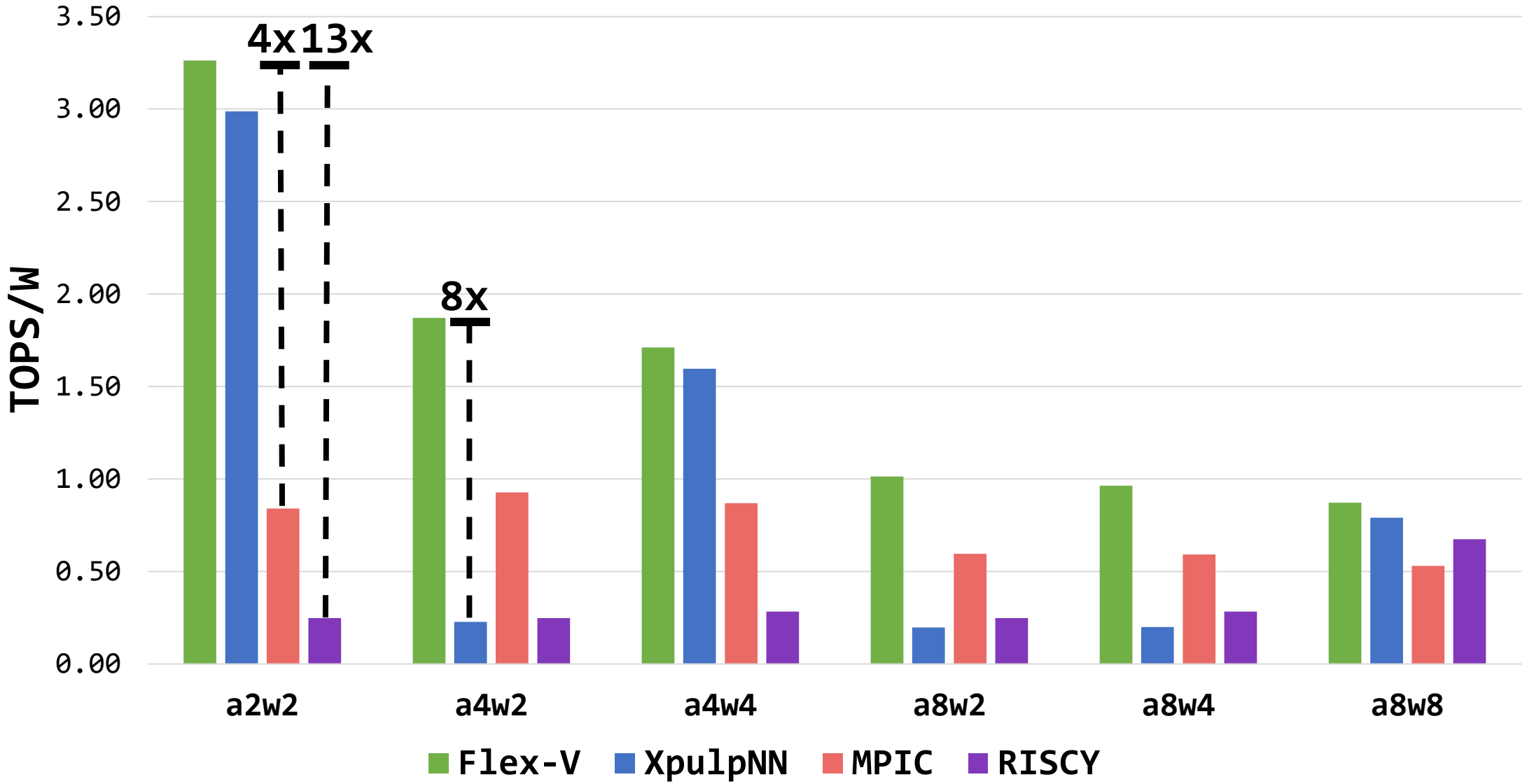


Results – Single kernels Performance

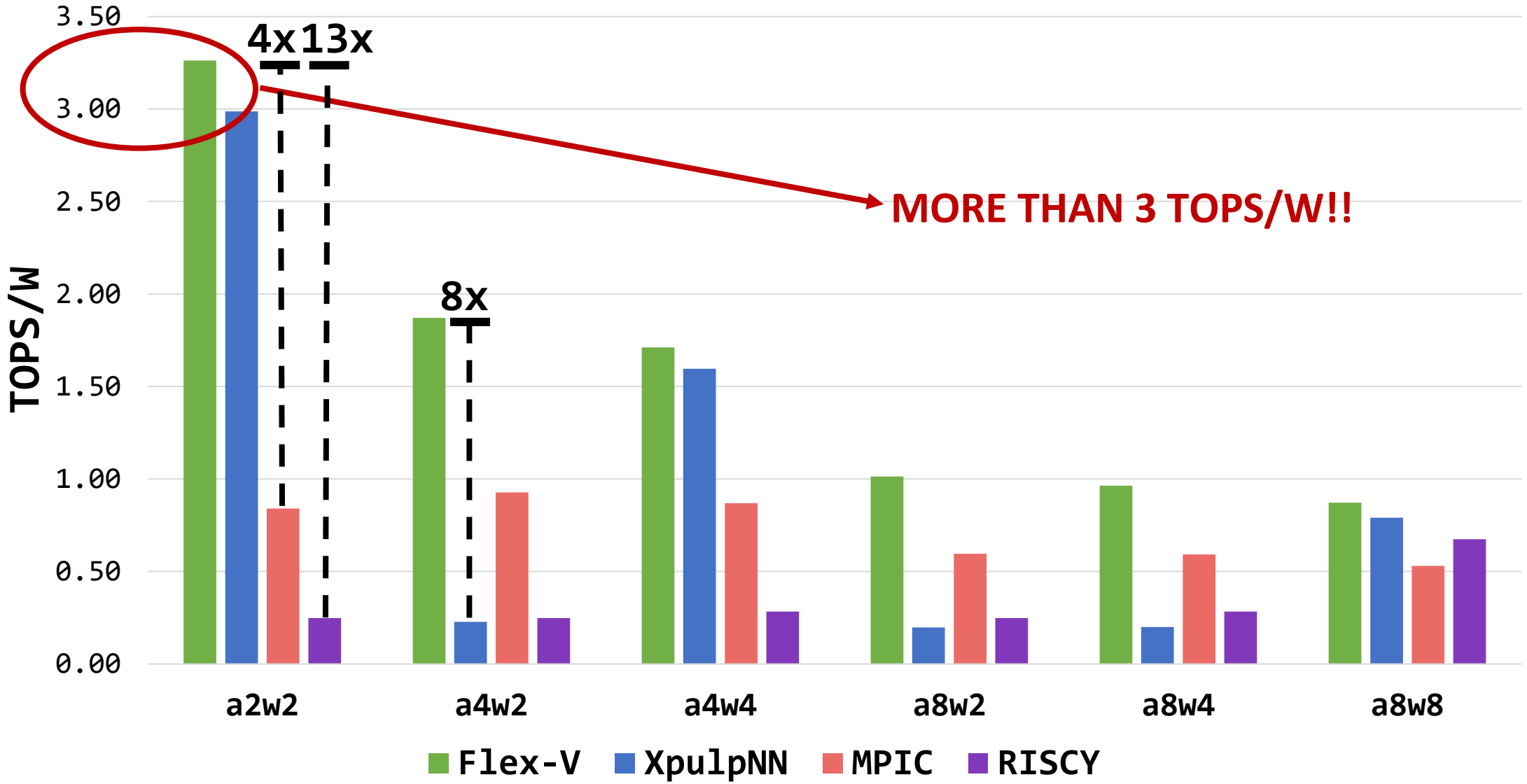


Results – Single kernels Energy Efficiency

Physical implementation
in GF-22nm technology



Results – Single kernels Energy Efficiency



Results – Full network

Network	MobileNetV1 (8b)	MobileNetV1 (8b4b)	ResNet-20 (4b2b)
Top-1 Accuracy	69.3 %	66.0 %	90.2 % [1]
Deg. W.r.t. 8b	-	3.3 %	0.15 %
Model size	1.9 MB	997 kB	142 kB
Memory saved	-	47 %	63 %
Performance (MAC/cycle)			
STM32H7	0.33	0.30	-
XpulpV2	5.6	3.2	4.8
XpulpNN	6.0	2.7	4.4
Flex-V	6.0	5.8	11.2

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

Results – Full network

Network	MobileNetV1 (8b)	MobileNetV1 (8b4b)	ResNet-20 (4b2b)
Top-1 Accuracy	69.3 %	66.0 %	90.2 % [1]
Deg. W.r.t. 8b	-	3.3 %	0.15 %
Model size	1.9 MB	997 kB	142 kB
Memory saved	-	47 %	63 %
Performance (MAC/cycle)			
STM32H7	0.33	0.30	-
XpulpV2	5.6	3.2	4.8
XpulpNN	6.0	2.7	4.4
Flex-V	6.0	5.8	11.2

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

Results – Full network

Network	MobileNetV1 (8b)	MobileNetV1 (8b4b)	ResNet-20 (4b2b)
Top-1 Accuracy	69.3 %	66.0 %	90.2 % [1]
Deg. W.r.t. 8b	-	3.3 %	0.15 %
Model size	1.9 MB	997 kB	142 kB
Memory saved	-	47 %	63 %
Performance (MAC/cycle)			
STM32H7	0.33	0.30	-
XpulpV2	5.6	3.2	4.8
XpulpNN	6.0	2.7	4.4
Flex-V	6.0	5.8	11.2

~ 20x

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

Results – Full network

Network	MobileNetV1 (8b)	MobileNetV1 (8b4b)	ResNet-20 (4b2b)
Top-1 Accuracy	69.3 %	66.0 %	90.2 % [1]
Deg. W.r.t. 8b	-	3.3 %	0.15 %
Model size	1.9 MB	997 kB	142 kB
Memory saved	-	47 %	63 %
Performance (MAC/cycle)			
STM32H7	0.33 ~ 20x	0.30	-
XpulpV2	5.6	3.2 ~ 2x	4.8
XpulpNN	6.0	2.7	4.4
Flex-V	6.0	5.8	11.2

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

Results – Full network

Network	MobileNetV1 (8b)	MobileNetV1 (8b4b)	ResNet-20 (4b2b)
Top-1 Accuracy	69.3 %	66.0 %	90.2 % [1]
Deg. W.r.t. 8b	-	3.3 %	0.15 %
Model size	1.9 MB	997 kB	142 kB
Memory saved	-	47 %	63 %
Performance (MAC/cycle)			
STM32H7	0.33 ~ 20x	0.30	-
XpulpV2	5.6	3.2 ~ 2x	4.8 ~ 2.5x
XpulpNN	6.0	2.7	4.4
Flex-V	6.0	5.8	11.2

[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!