Vecim: A 289.13GOPS/W RISC-V Vector Co-Processor with Compute-in-Memory Vector Register File for Efficient High-Performance Computing

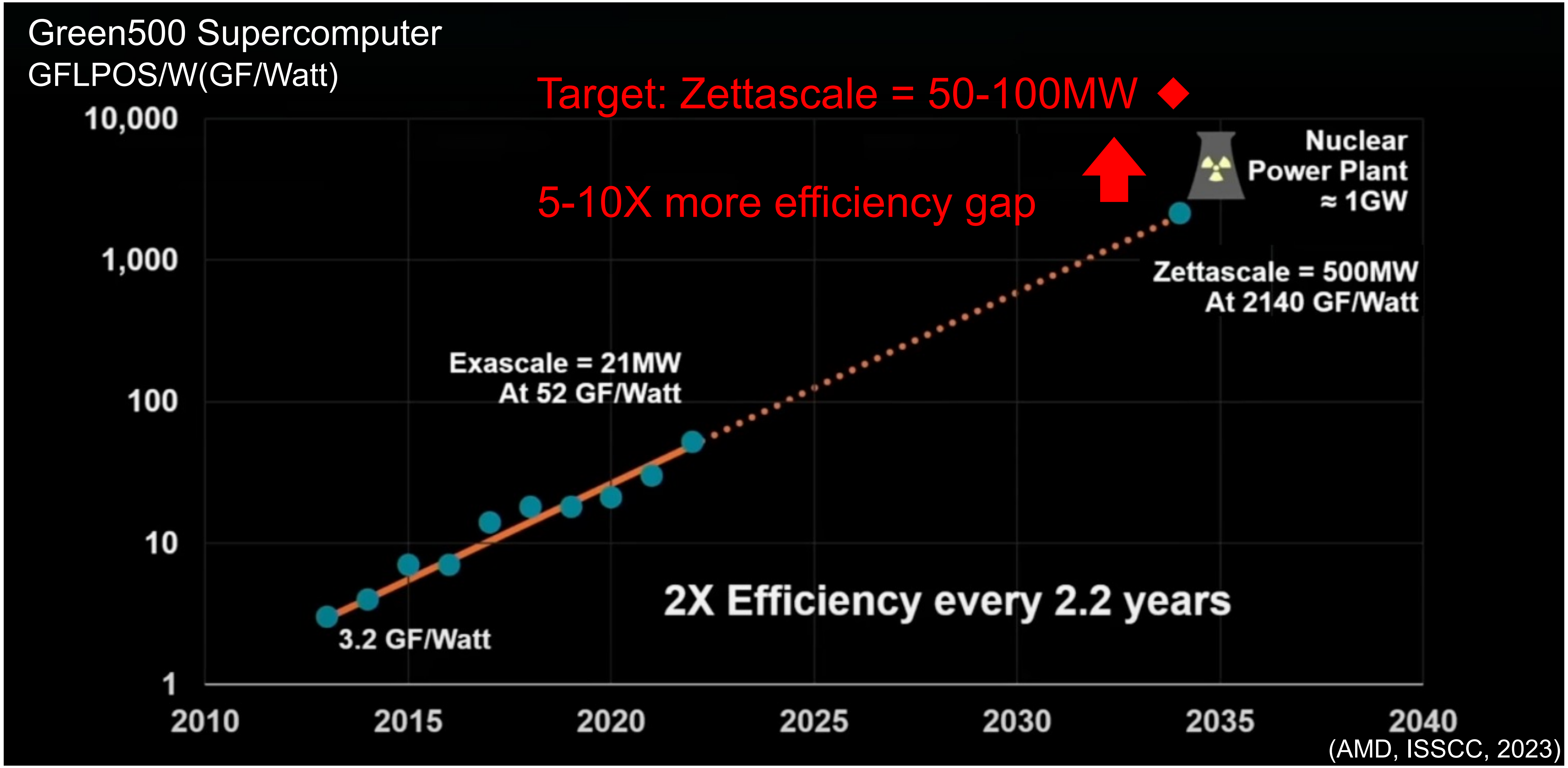
Yipeng Wang, Mengtian Yang, Chieh-pu Lo, Jaydeep P. Kulkarni

University of Texas at Austin, Circuit Research Lab

Outline

- **Motivation**
  - Efficiency gap of HPC
  - SRAM Compute-in-memory(CIM) application challenges
- **Proposed Vecim Architecture**
  - Overall architecture
  - CIM vector register file (VRF) and multiplication scheme
  - Data path and data flow
- **Silicon prototype measurements**
- **Summary**

Motivation 1: Efficiency Gap of HPC



(AMD, ISSCC, 2023)

Motivation 1: Efficiency Gap of HPC

$$Efficiency = OPS/W = \frac{OP}{E_{compute} + E_{others}} \uparrow$$

$$= \frac{E_{compute}}{E_{compute} + E_{others}} \uparrow \Big/ \frac{E_{compute}}{OP} \downarrow$$

Increase the proportion of Compute's energy

- Specialized instruction
- Domain specific hardware
- Datapath and memory optimization
- …

Improve the average Energy consumption of compute OPeration

- Technology scaling
- Lower precision
- Sparsity
- …

Motivation 1: Efficiency Gap of HPC

$$Efficiency = OPS/W = \frac{OP}{E_{compute} + E_{others}} \uparrow$$

$$= \frac{E_{compute}}{E_{compute} + E_{others}} \uparrow / \frac{E_{compute}}{OP} \downarrow$$

Increase the proportion of Compute's energy

- Specialized instruction
- Domain specific hardware
- Datapath and memory optimization
- In memory vector processing

This work

Improve the average energy consumption of compute operation

- Technology scaling
- Lower precision
- Sparsity
- Reusing SRAM for compute

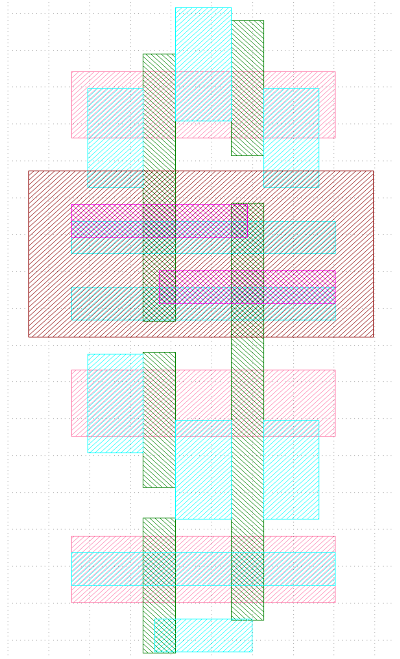
Explore architectural opportunity CIM provide for general purpose HW.

Motivation 2: SRAM CIM application challenges

Large area footprint

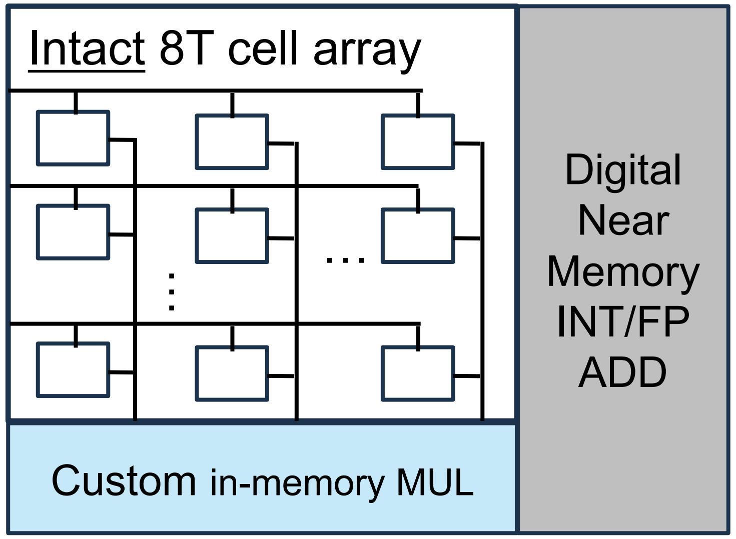
Low robustness: PVT variation of custom cells, IR drop
Low frequency: Longer WL/BL, large adder tree
Accuracy loss: ADC, FP conversion/limited window

This work:



Use foundry 8T cell

(Modified layout only for illustration purpose)



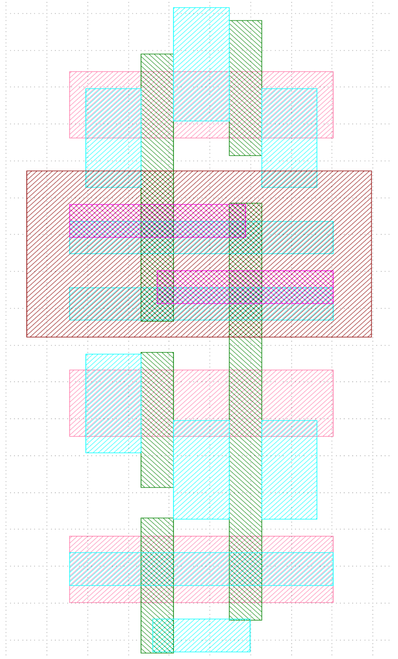
Intact cell array
Pipelined digital CIM
Near memory FP support

Motivation 2: SRAM CIM application challenges

Large area footprint

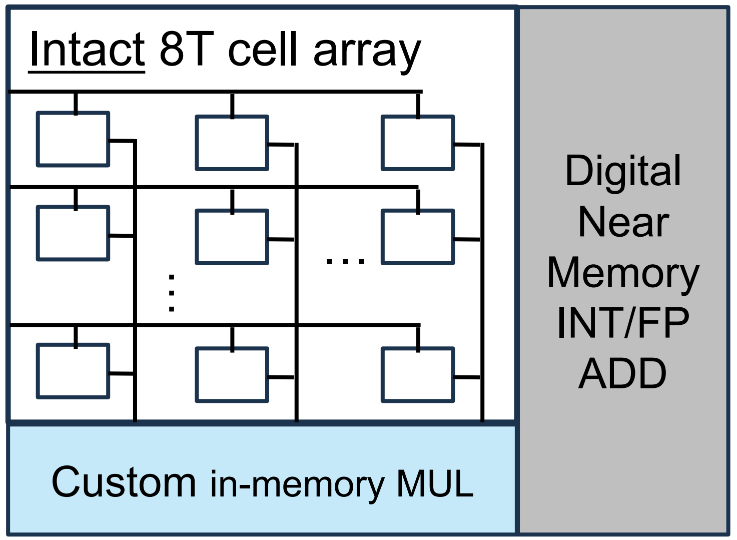
Low robustness
Low frequency
Accuracy loss

Low programmability



Use foundry 8T cell

(Modified layout only for illustration purpose)



Intact cell array
Pipelined digital CIM
Near memory FP support

Embed CIM in general purpose architecture / ISA;
Show efficiency improvement

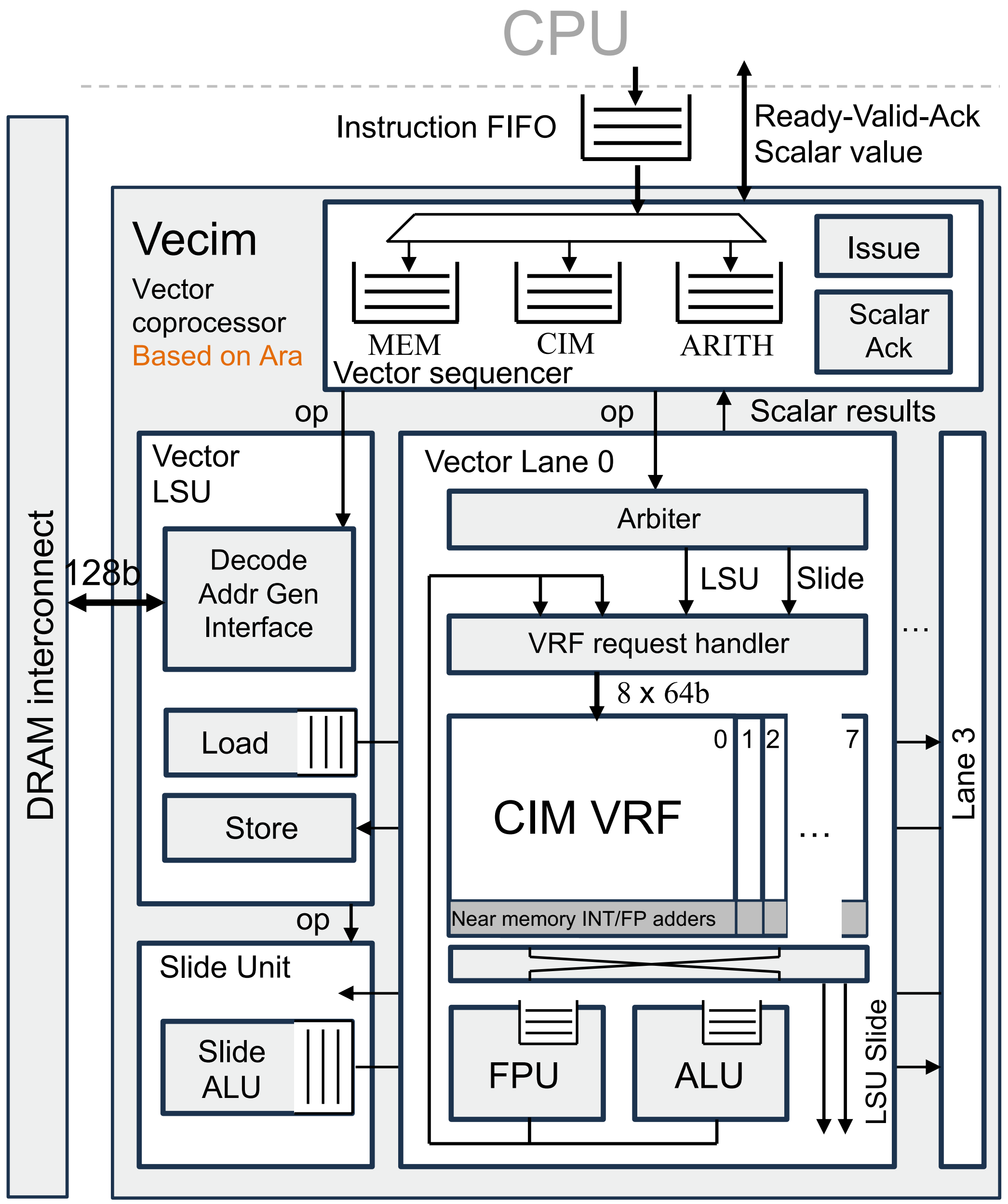
→ RISCV vector processor
→ large memory capacity in register files
→ Target matrix multiplication
→ Our key contribution

Outline

- Motivation
  - Efficiency gap of HPC
  - SRAM Compute-in-memory(CIM) application challenges
- **Proposed Vecim Architecture**
  - Overall architecture
  - CIM vector register file (VRF) and multiplication scheme
  - Data path and data flow
- Silicon prototype measurements
- Summary

Vecim Overall Architecture

- ❑ Based on open sourced [Ara, TVLSI, 2020]
- ❑ Instructions from scalar CPU
- ❑ 64bit/lane/cycle bandwidth DRAM
- ❑ Vector Load-Store Unit (LSU)
- ❑ Vector Slide Unit

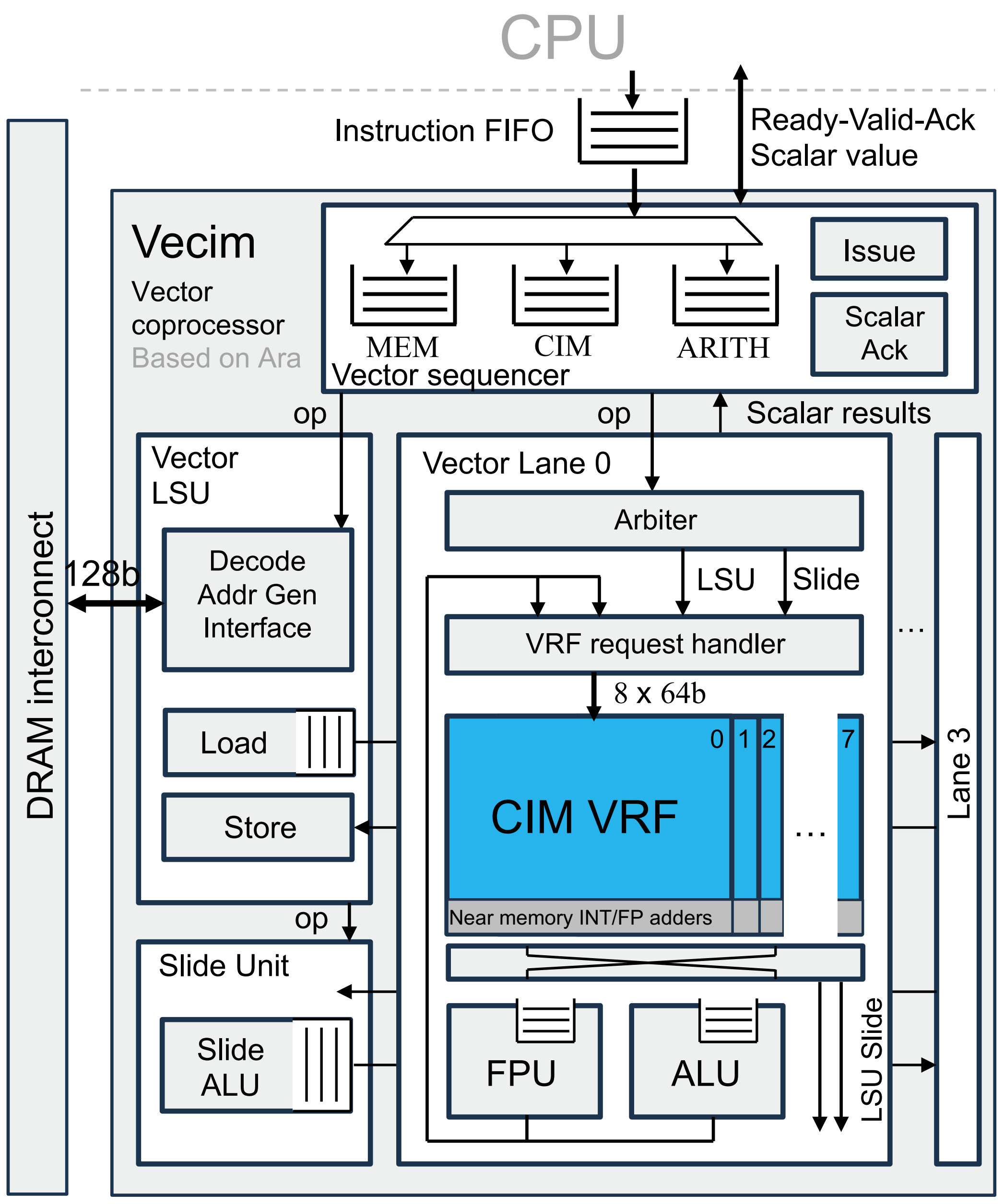


Vecim Overall Architecture

Our Innovations

❑A 1R1W SRAM Vector Register File (VRF)

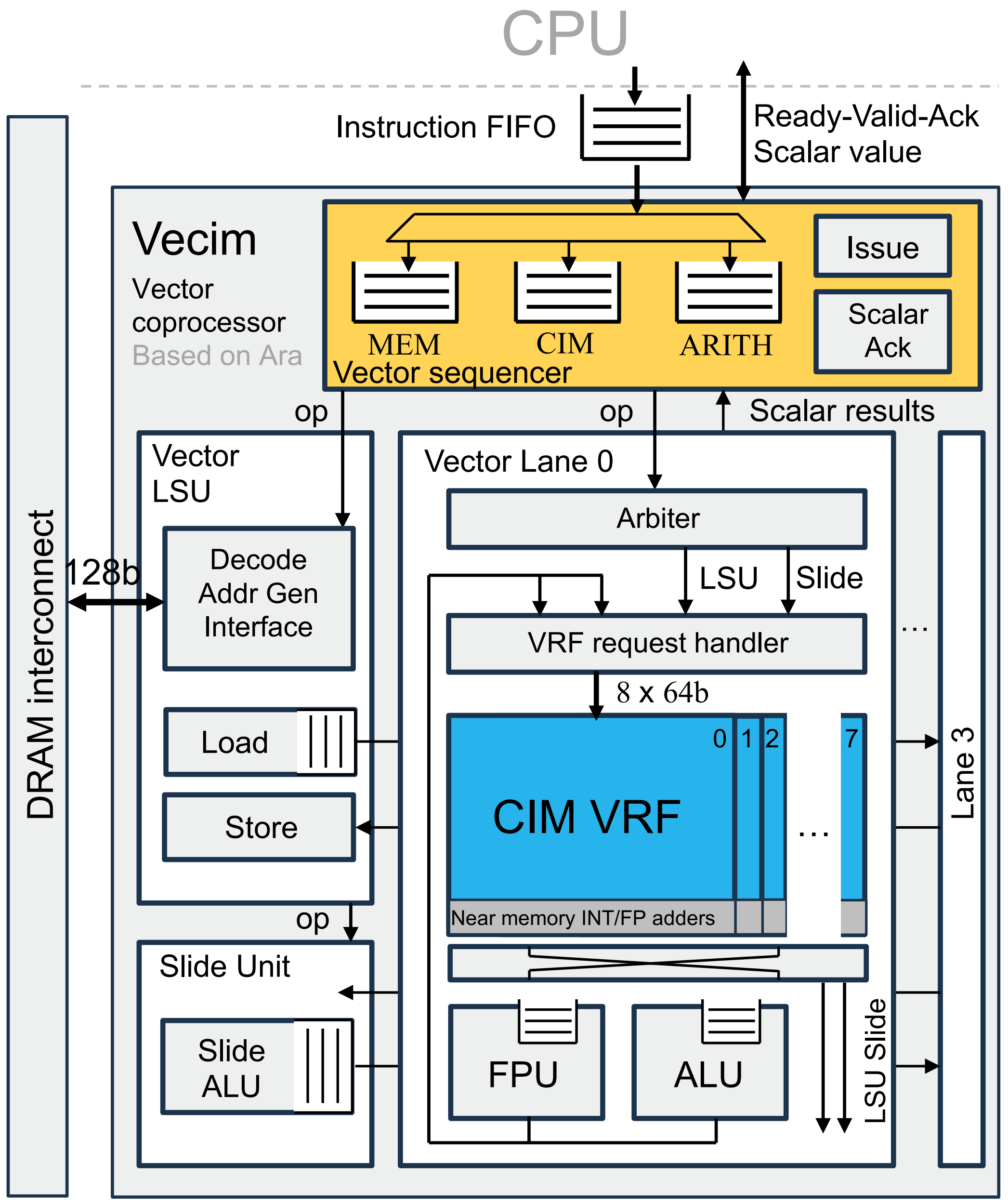
- INT8/BF16/FP16 all-digital in-memory multiplication and near-mem addition
- Double-rate-bit-parallel multiplication
- Specialized instruction extension



Vecim Overall Architecture

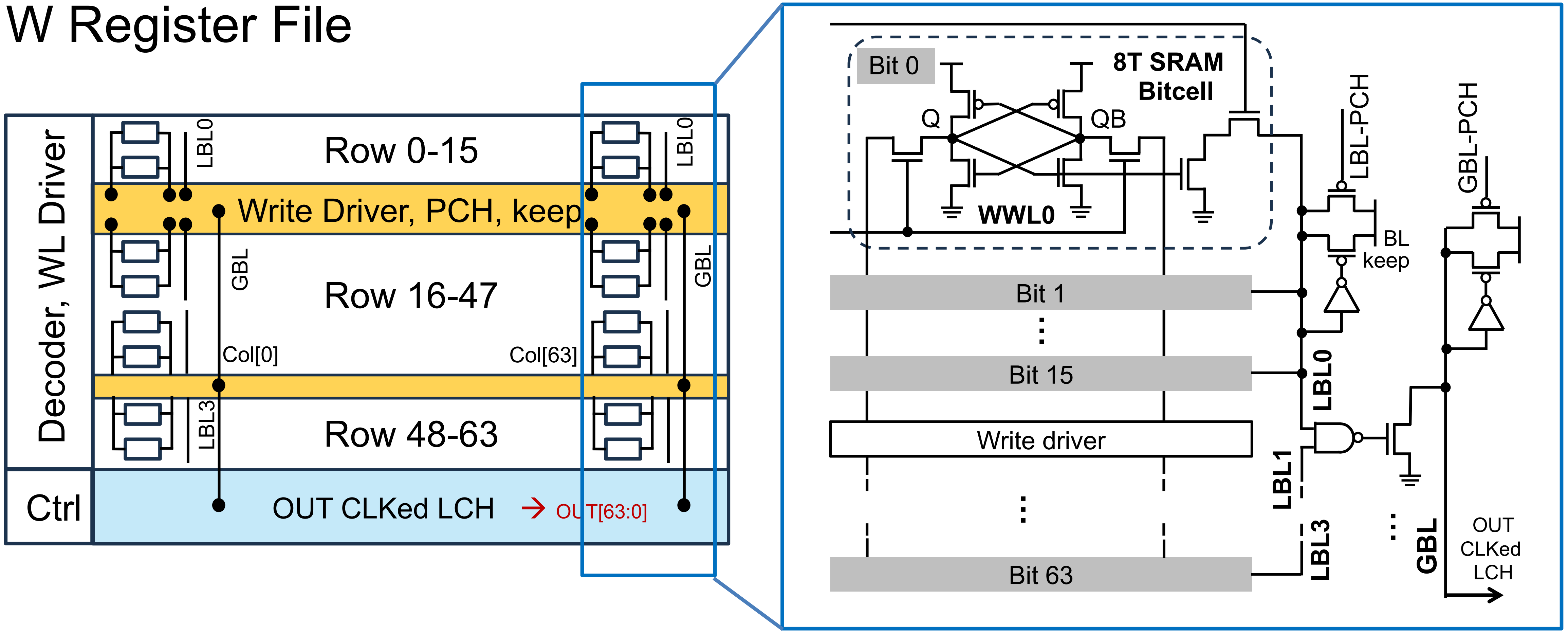
Our Innovations

- ❑ A 1R1W SRAM Vector Register File (VRF)
  - INT8/BF16/FP16 all-digital in-memory multiplication and addition
  - Double-rate-bit-parallel multiplication
  - Specialized instruction extension
- ❑ A dedicated vector sequencer
  - Light-weight out-of-order execution



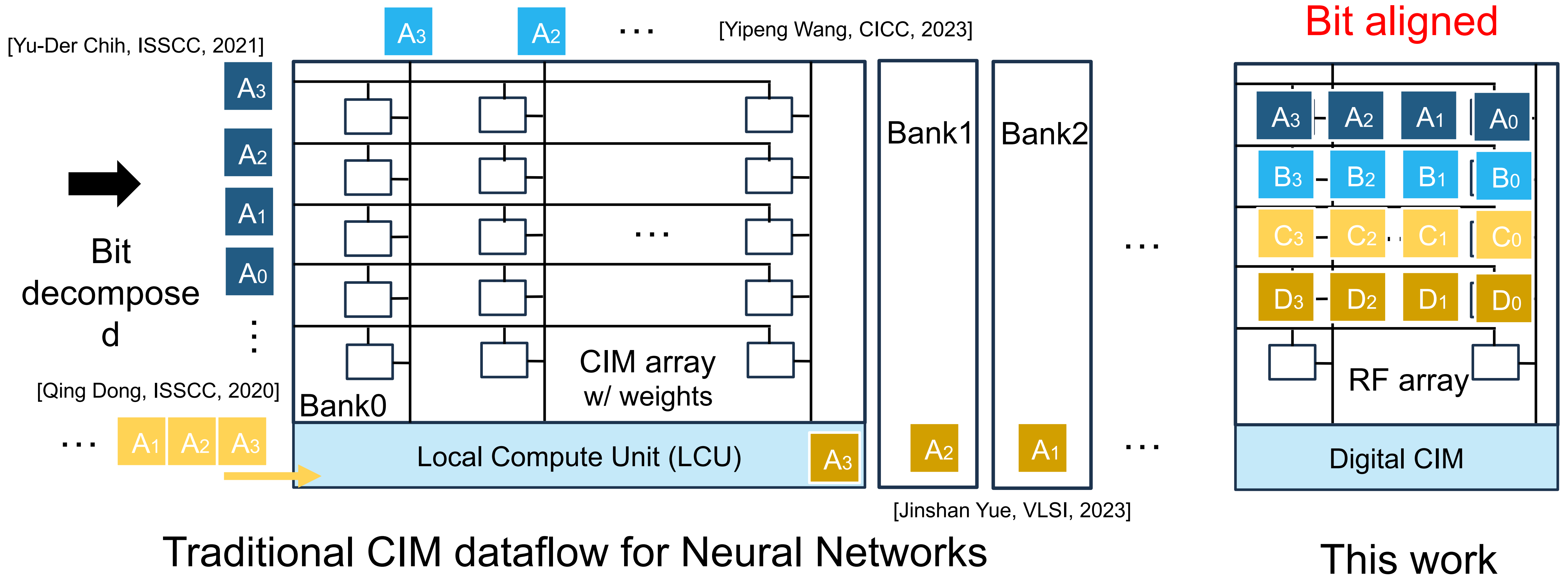
CIM Vector Register File

1R1W Register File



- Decoupled read/write ports for higher throughput
- Lower Vmin to be compatible with core logic

CIM Vector Register File

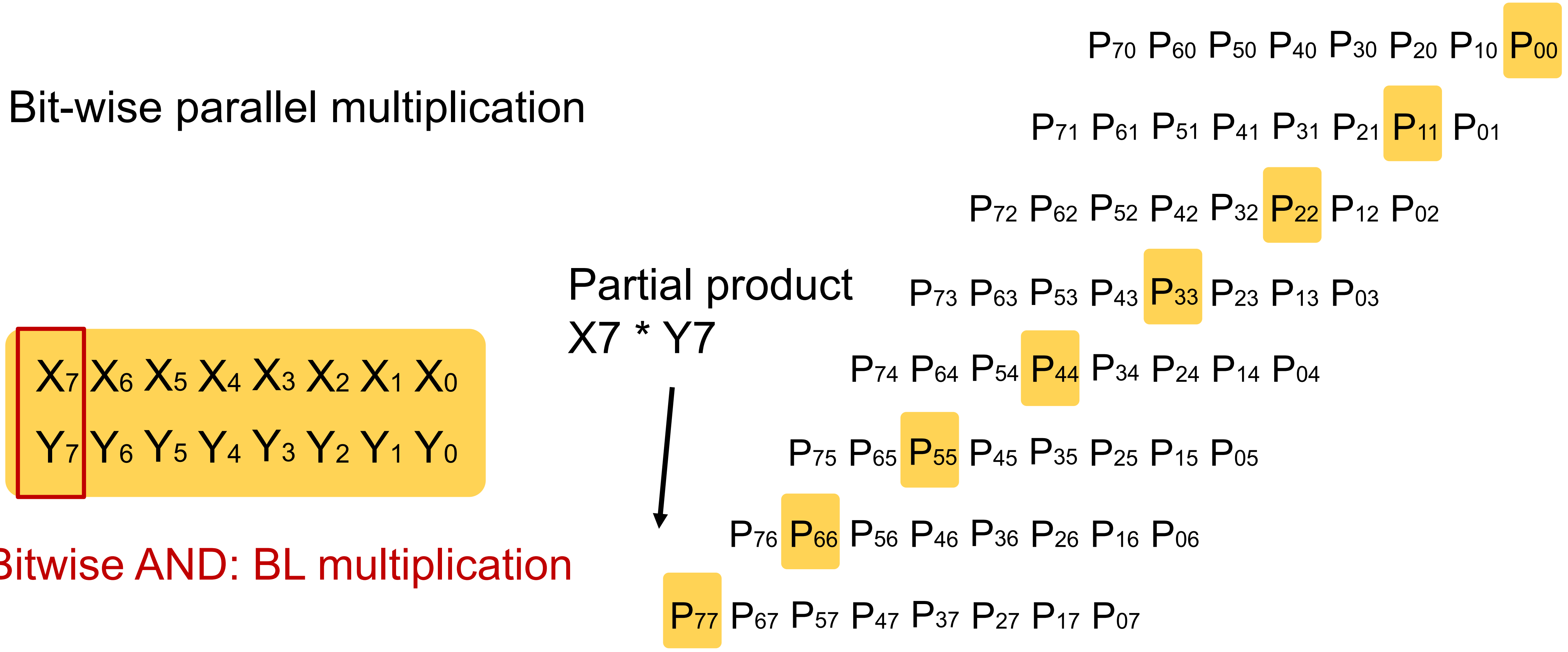


- Activations are bit decomposed to WL/BL/LCU/Banks, either bit-serial or bit-parallel.
- Vector processor only support RF with bit-aligned data layout. New dataflow needed.

CIM Vector Register File

Double-rate-bit-parallel multiplication

- ❑ Bit-wise parallel multiplication



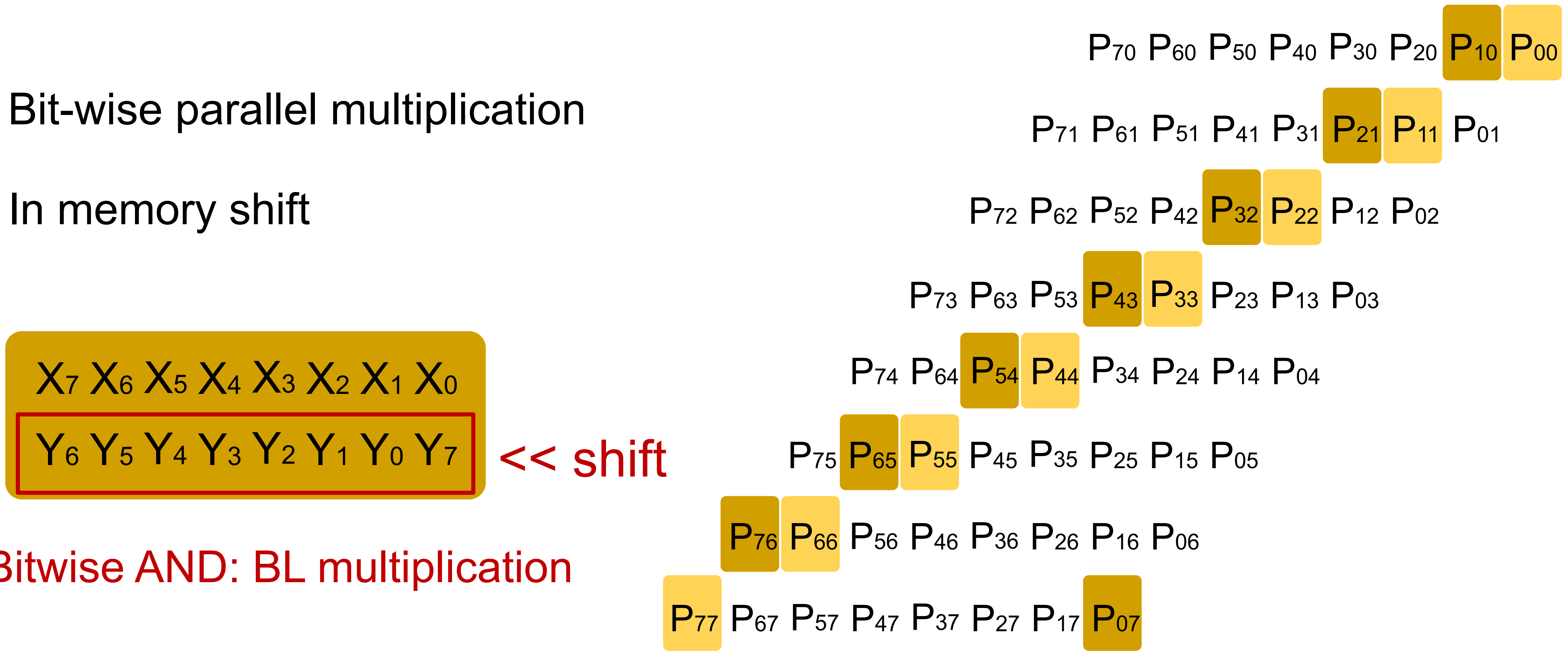
$X_7\ X_6\ X_5\ X_4\ X_3\ X_2\ X_1\ X_0$
$Y_7\ Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0$
Partial product
X7 * Y7
$P_{70}\ P_{60}\ P_{50}\ P_{40}\ P_{30}\ P_{20}\ P_{10}\ P_{00}$
$P_{71}\ P_{61}\ P_{51}\ P_{41}\ P_{31}\ P_{21}\ P_{11}\ P_{01}$
$P_{72}\ P_{62}\ P_{52}\ P_{42}\ P_{32}\ P_{22}\ P_{12}\ P_{02}$
$P_{73}\ P_{63}\ P_{53}\ P_{43}\ P_{33}\ P_{23}\ P_{13}\ P_{03}$
$P_{74}\ P_{64}\ P_{54}\ P_{44}\ P_{34}\ P_{24}\ P_{14}\ P_{04}$
$P_{75}\ P_{65}\ P_{55}\ P_{45}\ P_{35}\ P_{25}\ P_{15}\ P_{05}$
$P_{76}\ P_{66}\ P_{56}\ P_{46}\ P_{36}\ P_{26}\ P_{16}\ P_{06}$
$P_{77}\ P_{67}\ P_{57}\ P_{47}\ P_{37}\ P_{27}\ P_{17}\ P_{07}$

Bitwise AND: BL multiplication

CIM Vector Register File

Double-rate-bit-parallel multiplication

- ❑ Bit-wise parallel multiplication
- ❑ In memory shift



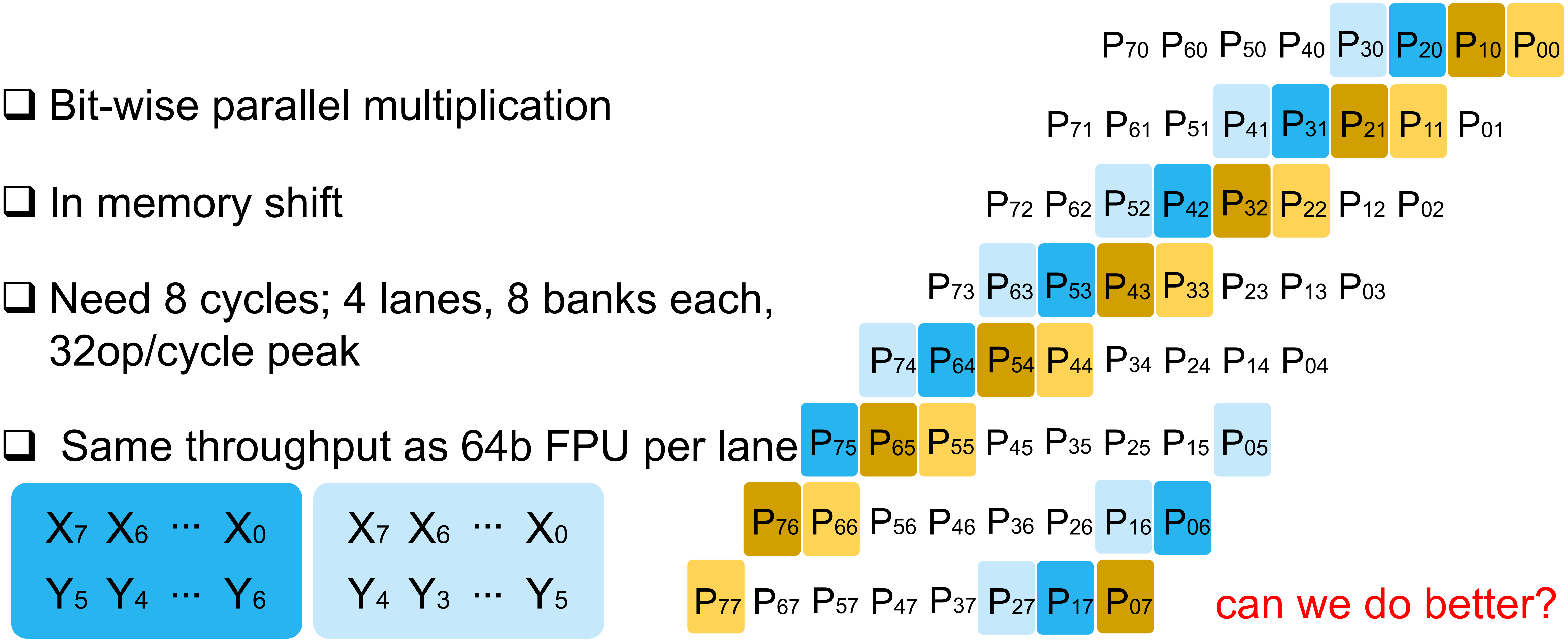
$X_7\ X_6\ X_5\ X_4\ X_3\ X_2\ X_1\ X_0$
$Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0\ Y_7$
<< shift
$P_{70}\ P_{60}\ P_{50}\ P_{40}\ P_{30}\ P_{20}\ P_{10}\ P_{00}$
$P_{71}\ P_{61}\ P_{51}\ P_{41}\ P_{31}\ P_{21}\ P_{11}\ P_{01}$
$P_{72}\ P_{62}\ P_{52}\ P_{42}\ P_{32}\ P_{22}\ P_{12}\ P_{02}$
$P_{73}\ P_{63}\ P_{53}\ P_{43}\ P_{33}\ P_{23}\ P_{13}\ P_{03}$
$P_{74}\ P_{64}\ P_{54}\ P_{44}\ P_{34}\ P_{24}\ P_{14}\ P_{04}$
$P_{75}\ P_{65}\ P_{55}\ P_{45}\ P_{35}\ P_{25}\ P_{15}\ P_{05}$
$P_{76}\ P_{66}\ P_{56}\ P_{46}\ P_{36}\ P_{26}\ P_{16}\ P_{06}$
$P_{77}\ P_{67}\ P_{57}\ P_{47}\ P_{37}\ P_{27}\ P_{17}\ P_{07}$

Bitwise AND: BL multiplication

CIM Vector Register File

Double-rate-bit-parallel multiplication

- ❑ Bit-wise parallel multiplication
- ❑ In memory shift
- ❑ Need 8 cycles; 4 lanes, 8 banks each, 32op/cycle peak
- ❑ Same throughput as 64b FPU per lane

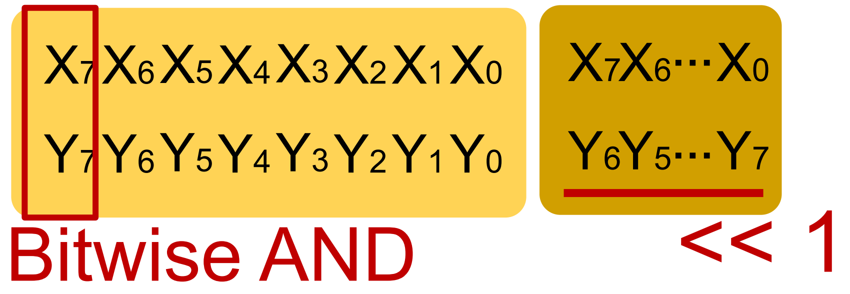

X_7 X_6 ⋯ X_0
Y_5 Y_4 ⋯ Y_6
X_7 X_6 ⋯ X_0
Y_4 Y_3 ⋯ Y_5
P_{70} P_{60} P_{50} P_{40} P_{30} P_{20} P_{10} P_{00}
P_{71} P_{61} P_{51} P_{41} P_{31} P_{21} P_{11} P_{01}
P_{72} P_{62} P_{52} P_{42} P_{32} P_{22} P_{12} P_{02}
P_{73} P_{63} P_{53} P_{43} P_{33} P_{23} P_{13} P_{03}
P_{74} P_{64} P_{54} P_{44} P_{34} P_{24} P_{14} P_{04}
P_{75} P_{65} P_{55} P_{45} P_{35} P_{25} P_{15} P_{05}
P_{76} P_{66} P_{56} P_{46} P_{36} P_{26} P_{16} P_{06}
P_{77} P_{67} P_{57} P_{47} P_{37} P_{27} P_{17} P_{07}

can we do better?

CIM Vector Register File

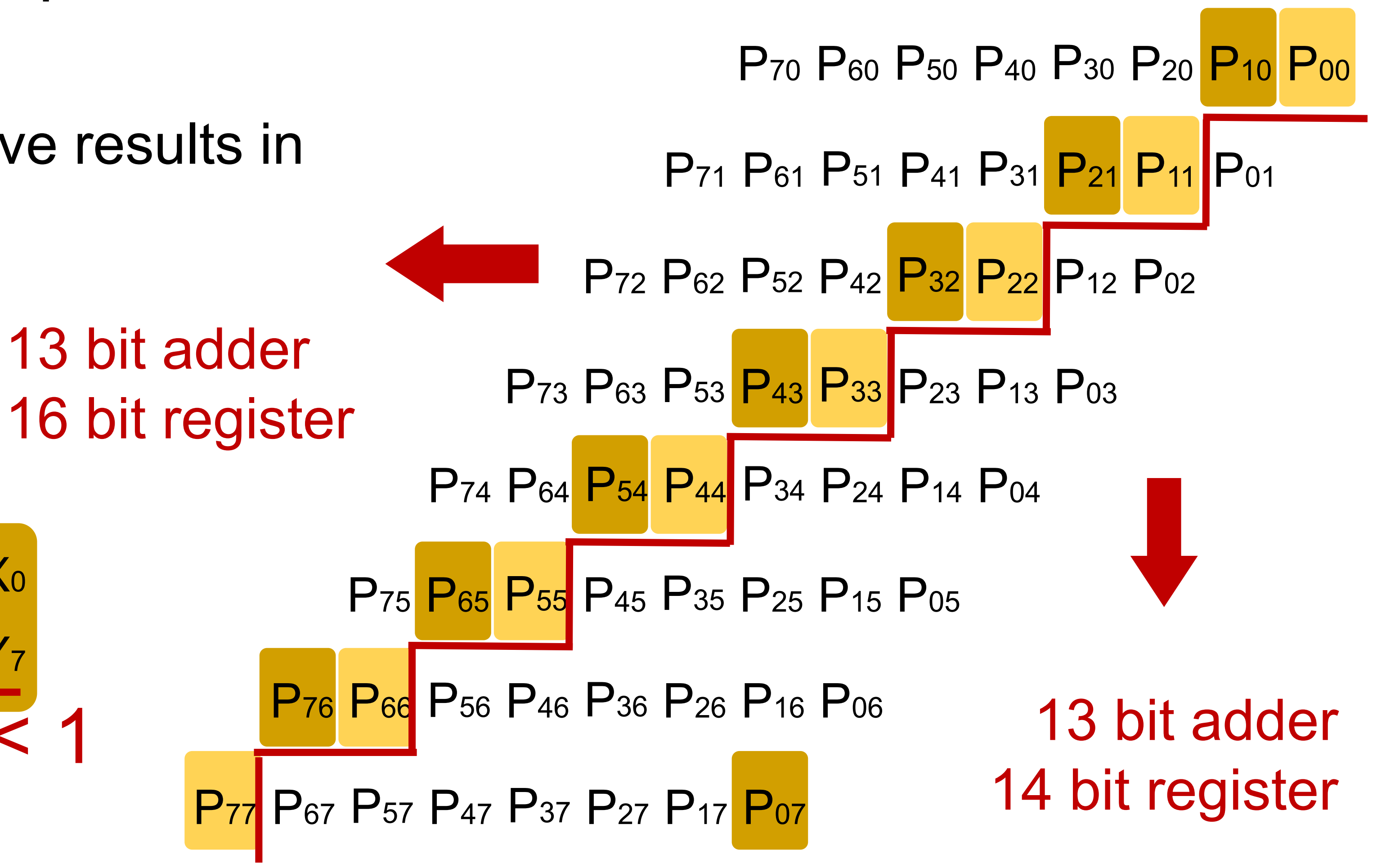
Double-rate-bit-parallel multiplication

- ❑ Accumulate two consecutive results in one cycle.



$X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0$
$Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0$
$X_7 X_6 \cdots X_0$
$Y_6 Y_5 \cdots Y_7$
Bitwise AND
<< 1

Cycle 1

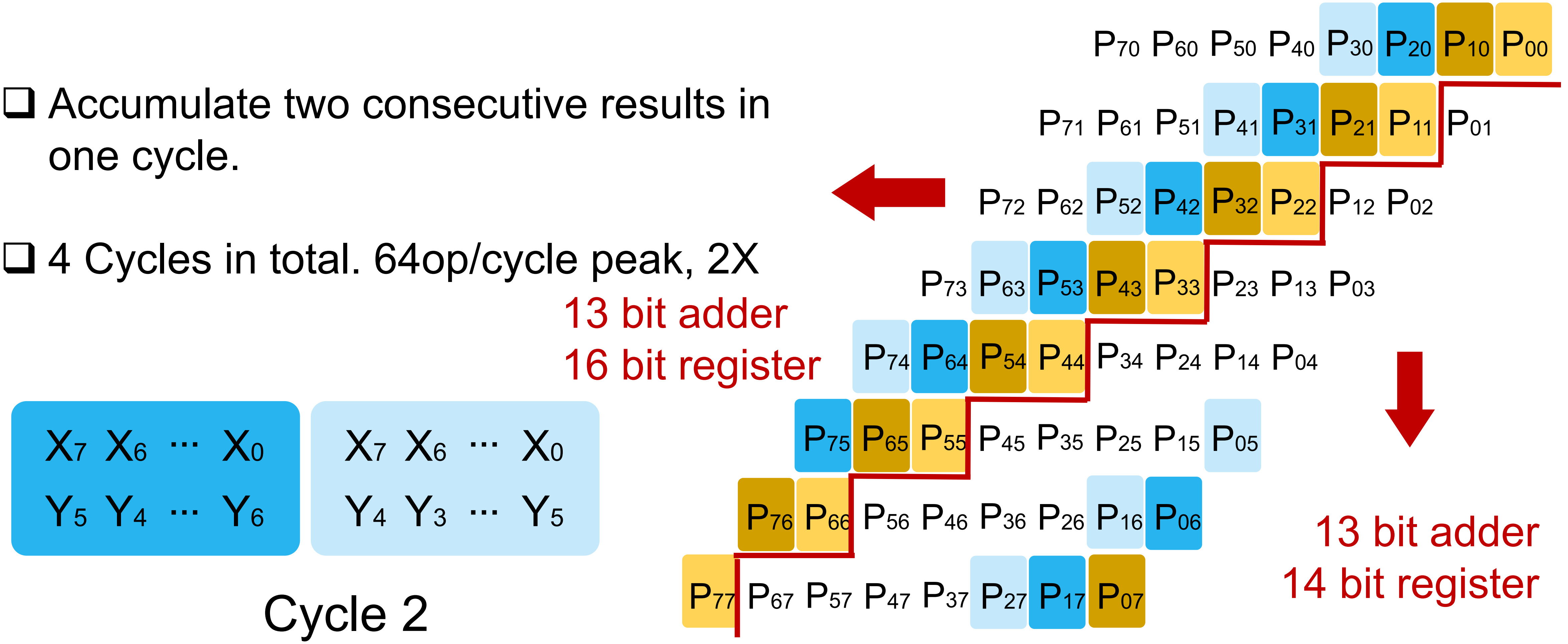


13 bit adder
16 bit register
P_{70} P_{60} P_{50} P_{40} P_{30} P_{20} P_{10} P_{00}
P_{71} P_{61} P_{51} P_{41} P_{31} P_{21} P_{11} P_{01}
P_{72} P_{62} P_{52} P_{42} P_{32} P_{22} P_{12} P_{02}
P_{73} P_{63} P_{53} P_{43} P_{33} P_{23} P_{13} P_{03}
P_{74} P_{64} P_{54} P_{44} P_{34} P_{24} P_{14} P_{04}
P_{75} P_{65} P_{55} P_{45} P_{35} P_{25} P_{15} P_{05}
P_{76} P_{66} P_{56} P_{46} P_{36} P_{26} P_{16} P_{06}
P_{77} P_{67} P_{57} P_{47} P_{37} P_{27} P_{17} P_{07}
13 bit adder
14 bit register

CIM Vector Register File

Double-rate-bit-parallel multiplication

- ❑ Accumulate two consecutive results in one cycle.
- ❑ 4 Cycles in total. 64op/cycle peak, 2X



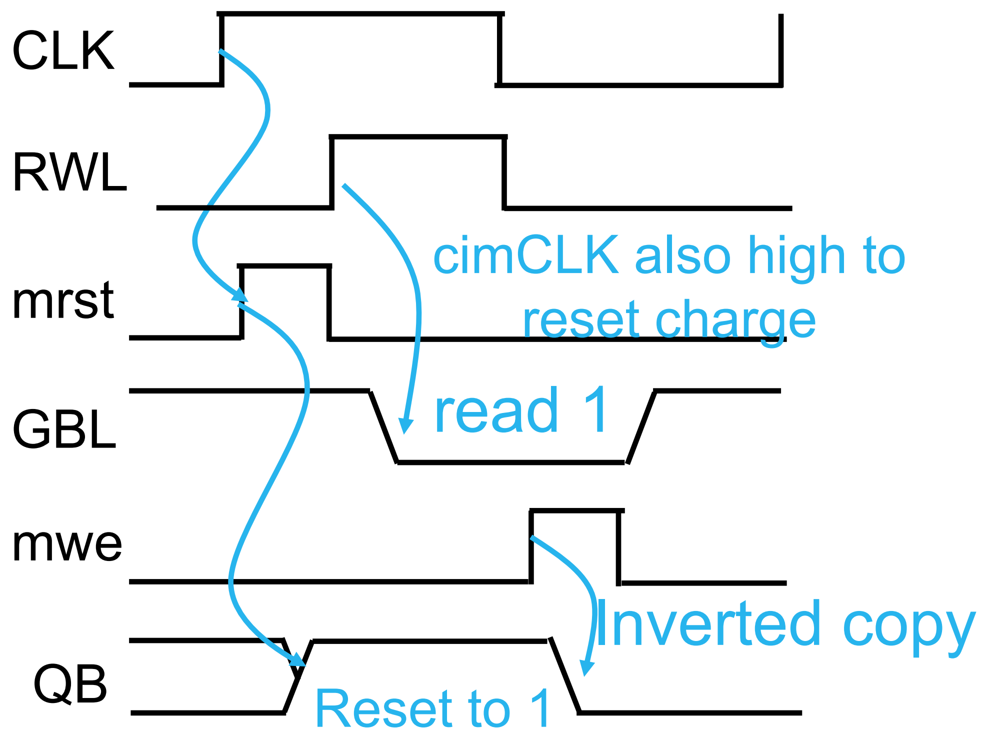
13 bit adder
16 bit register
X_7 X_6 ··· X_0
Y_5 Y_4 ··· Y_6
X_7 X_6 ··· X_0
Y_4 Y_3 ··· Y_5
P_{70} P_{60} P_{50} P_{40} P_{30} P_{20} P_{10} P_{00}
P_{71} P_{61} P_{51} P_{41} P_{31} P_{21} P_{11} P_{01}
P_{72} P_{62} P_{52} P_{42} P_{32} P_{22} P_{12} P_{02}
P_{73} P_{63} P_{53} P_{43} P_{33} P_{23} P_{13} P_{03}
P_{74} P_{64} P_{54} P_{44} P_{34} P_{24} P_{14} P_{04}
P_{75} P_{65} P_{55} P_{45} P_{35} P_{25} P_{15} P_{05}
P_{76} P_{66} P_{56} P_{46} P_{36} P_{26} P_{16} P_{06}
P_{77} P_{67} P_{57} P_{47} P_{37} P_{27} P_{17} P_{07}
13 bit adder
14 bit register

Cycle 2

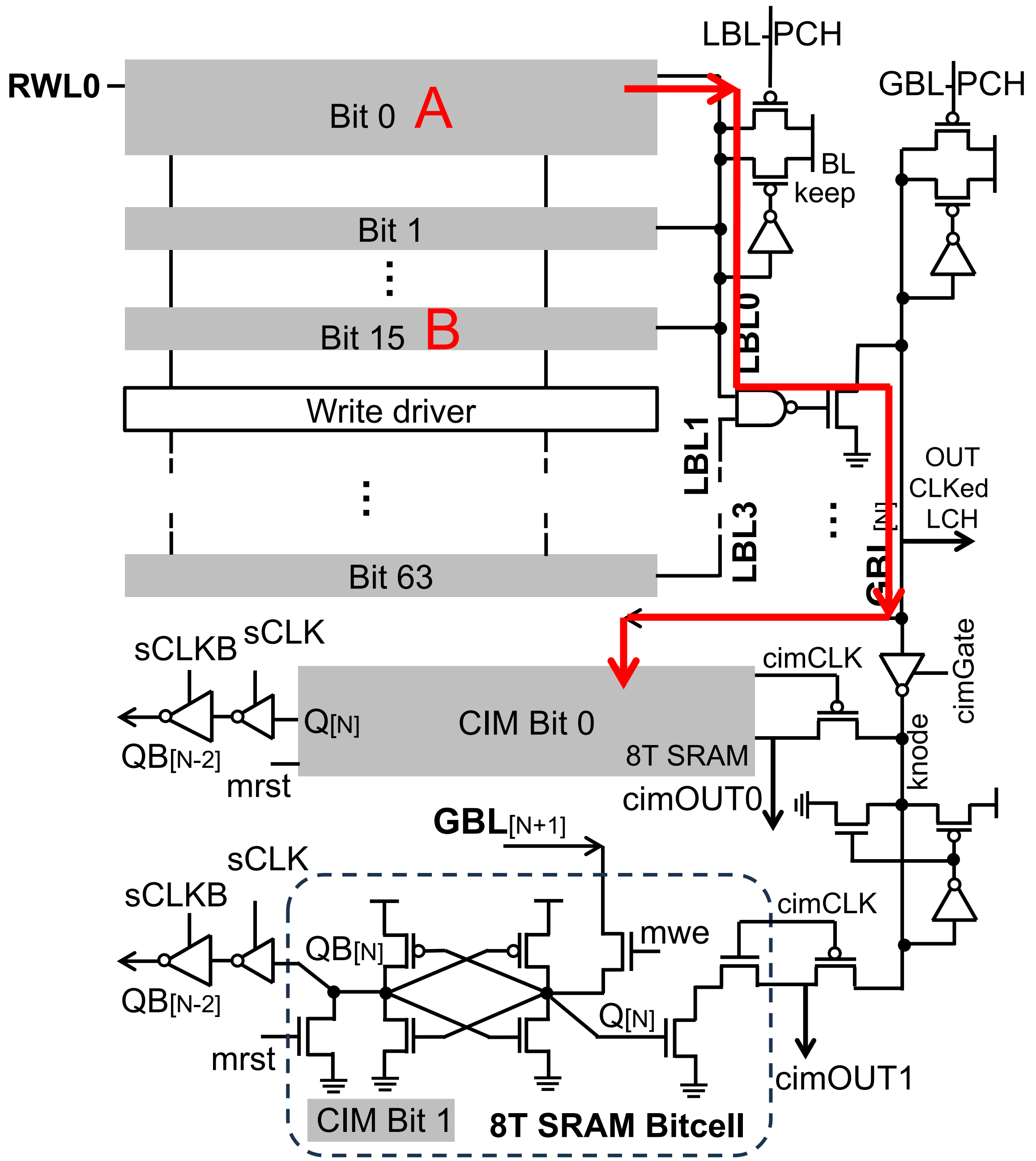
CIM Vector Register File

CIM VRF Circuit and dataflow

- ❑ Copy operand A to CIM BIT 0
  - ▪ In-memory inverted copy operation



Longer delay than SRAM read

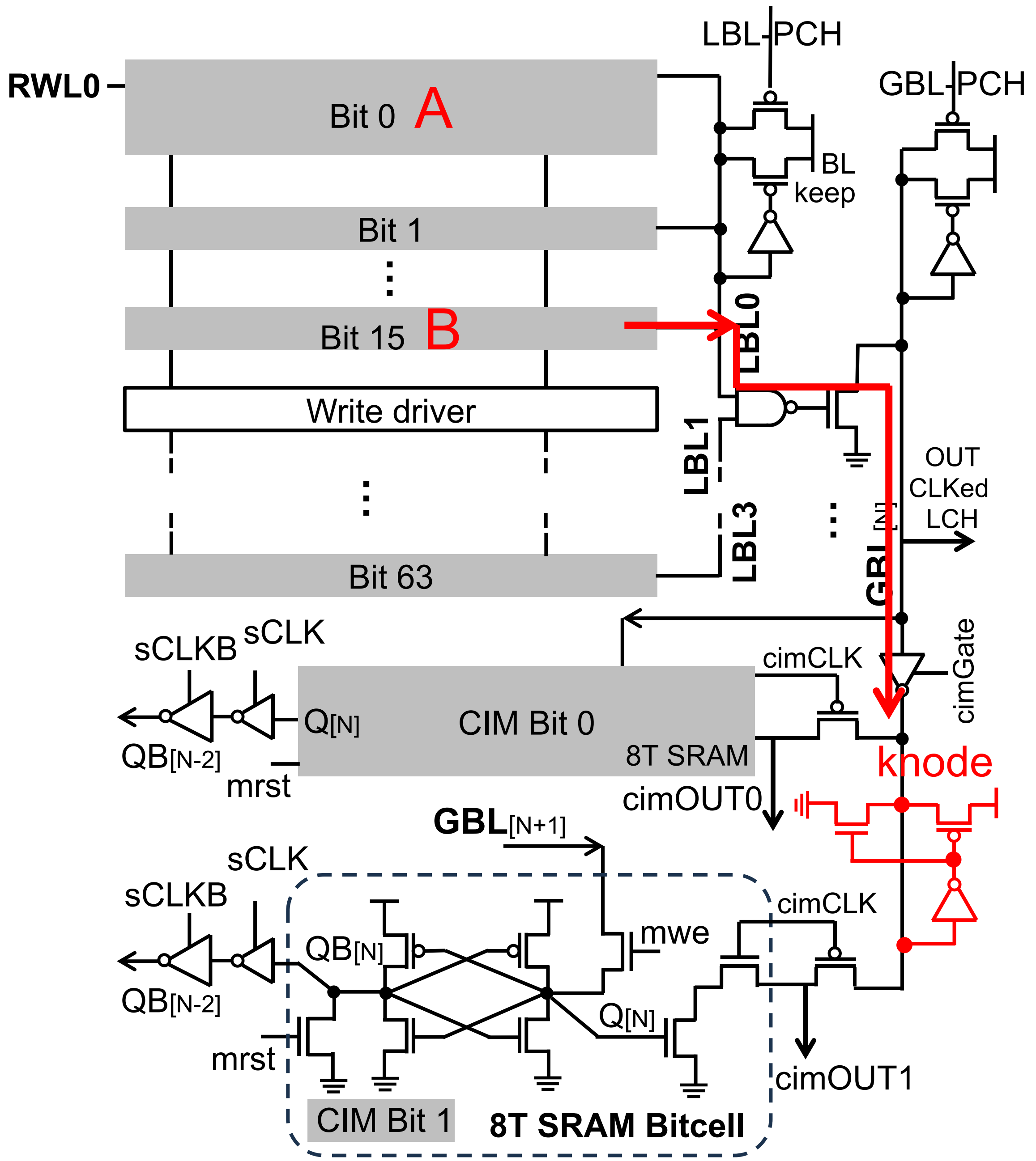


CIM Vector Register File

CIM VRF Circuit and dataflow

- ❑ Copy operand A to CIM BIT 0
    - ▪ In-memory inverted copy operation

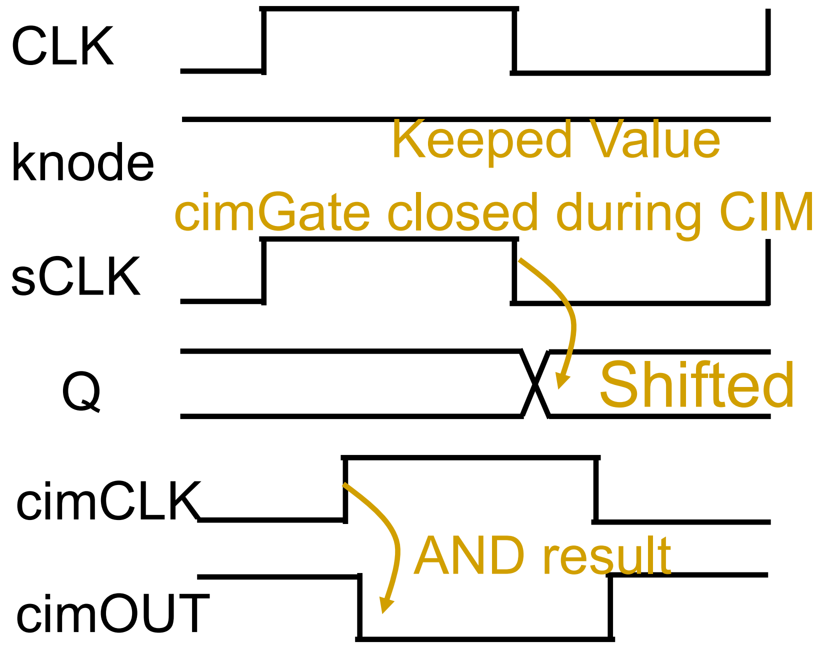
- ❑ Keep operand B at knode

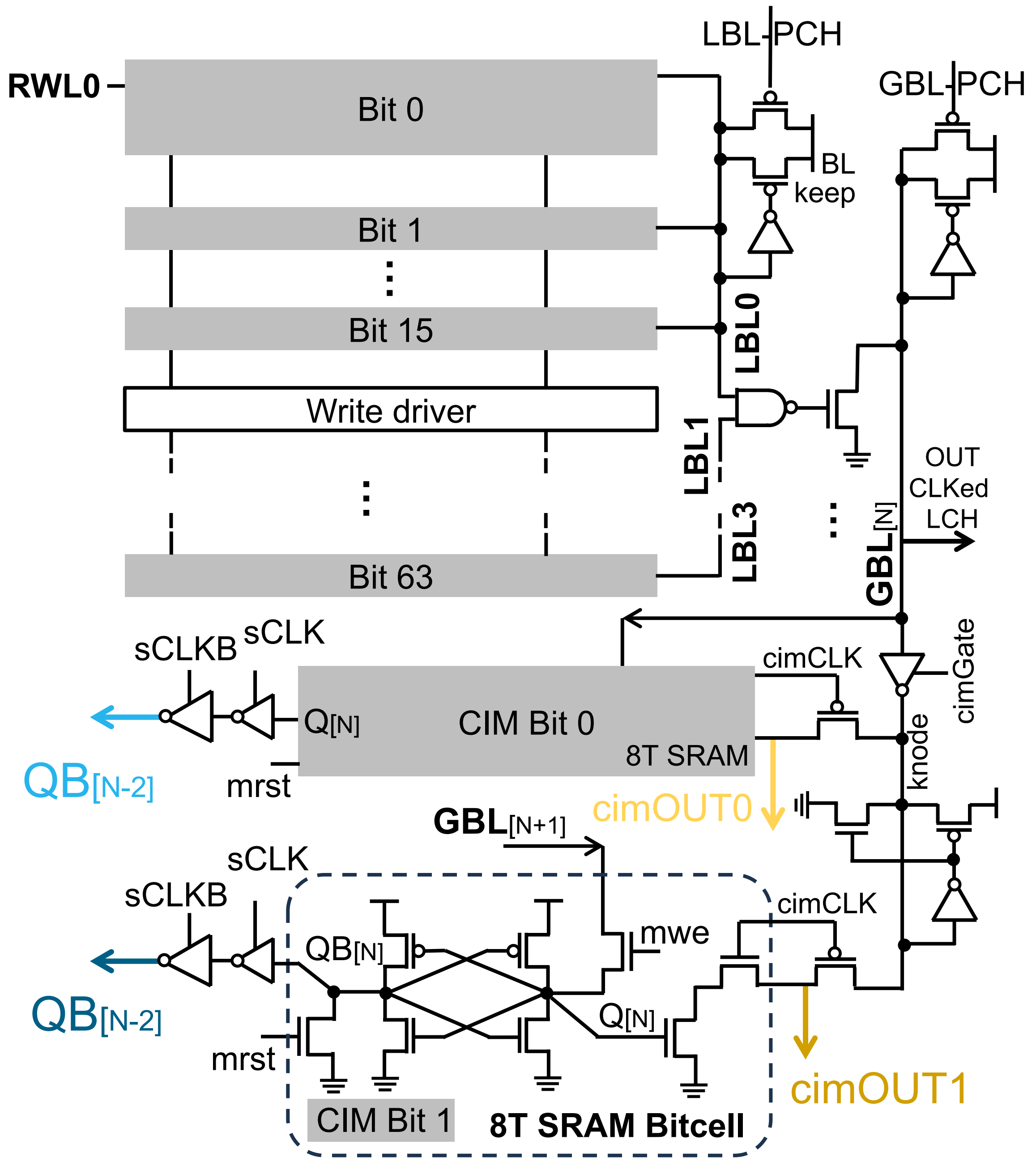


CIM Vector Register File

CIM VRF Circuit and dataflow

- ❑ Double-rate-bit-parallel multiplication
  - In-memory shift
  - BL multiplication



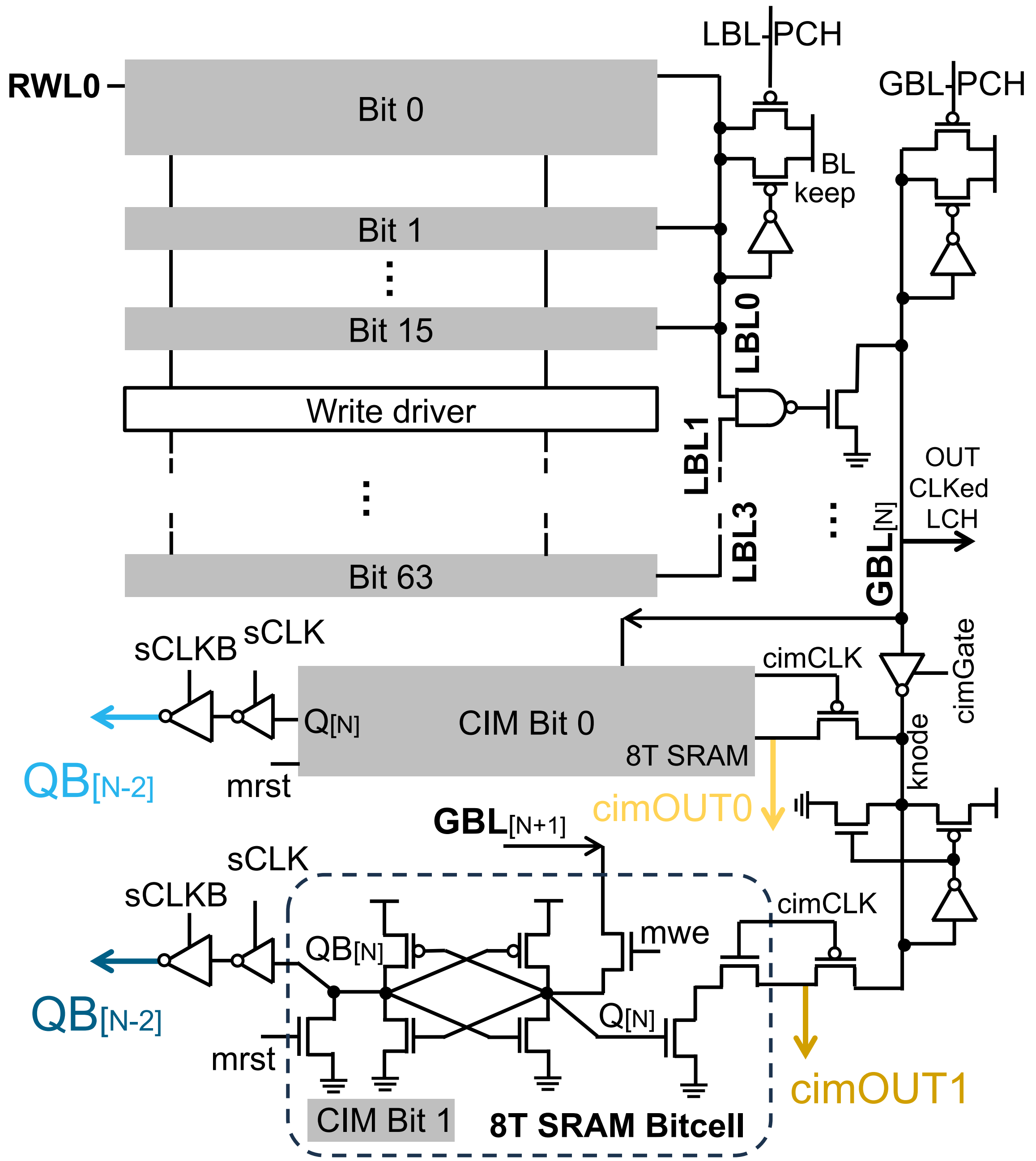


CIM Vector Register File

CIM VRF Circuit and dataflow

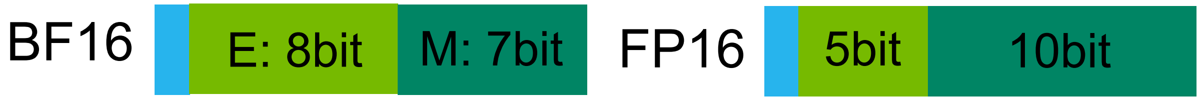
- ❑ Double-rate-bit-parallel multiplication
  - In-memory shift
  - BL multiplication

Critical delay path similar with SRAM read.
Similar IR drop.



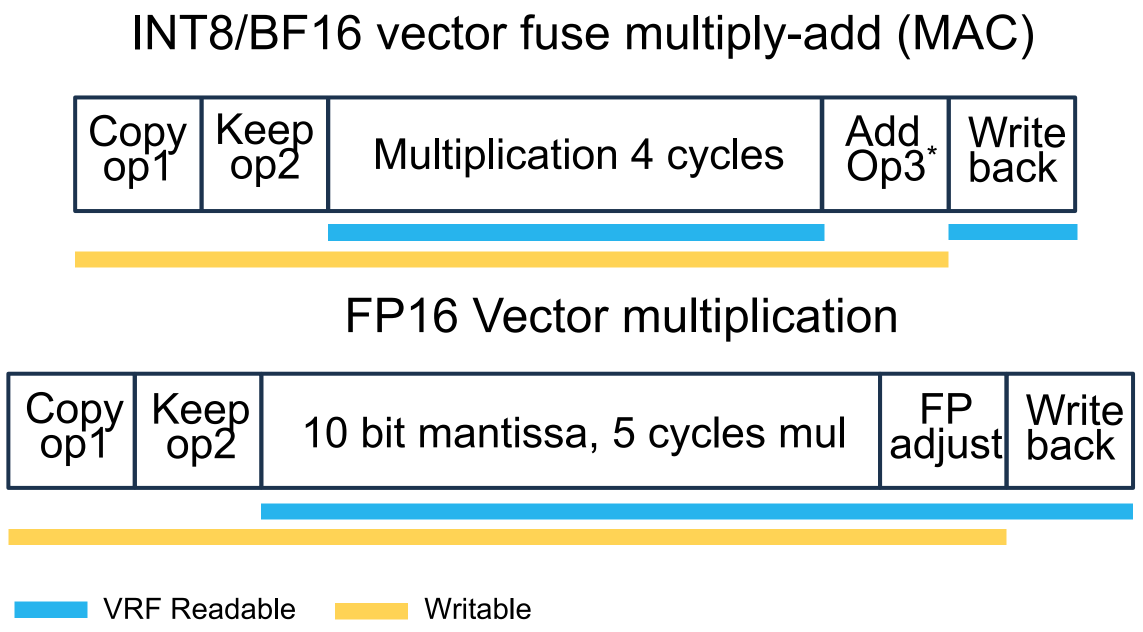
CIM Vector Register File

Floating point support with near memory adders



Mul E: Near mem INT add M: 8bit/10bit in-VRF Multiplication

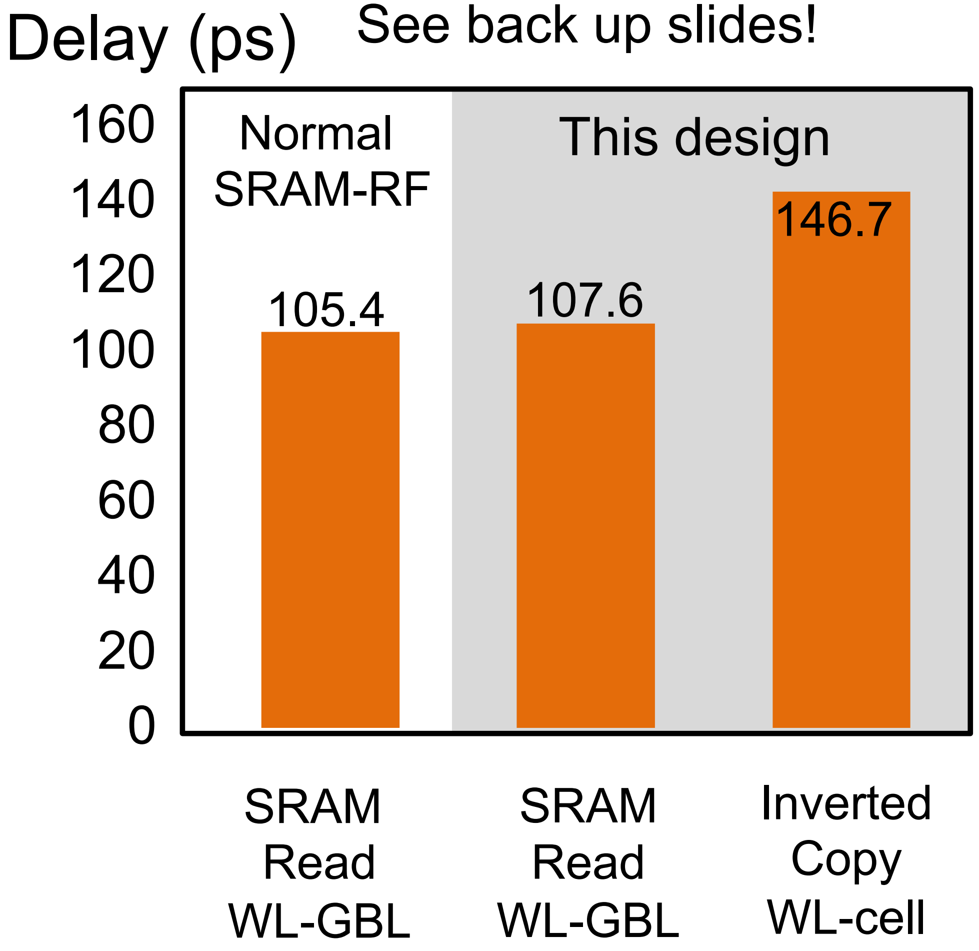
Add Near memory FP16/BF16 add

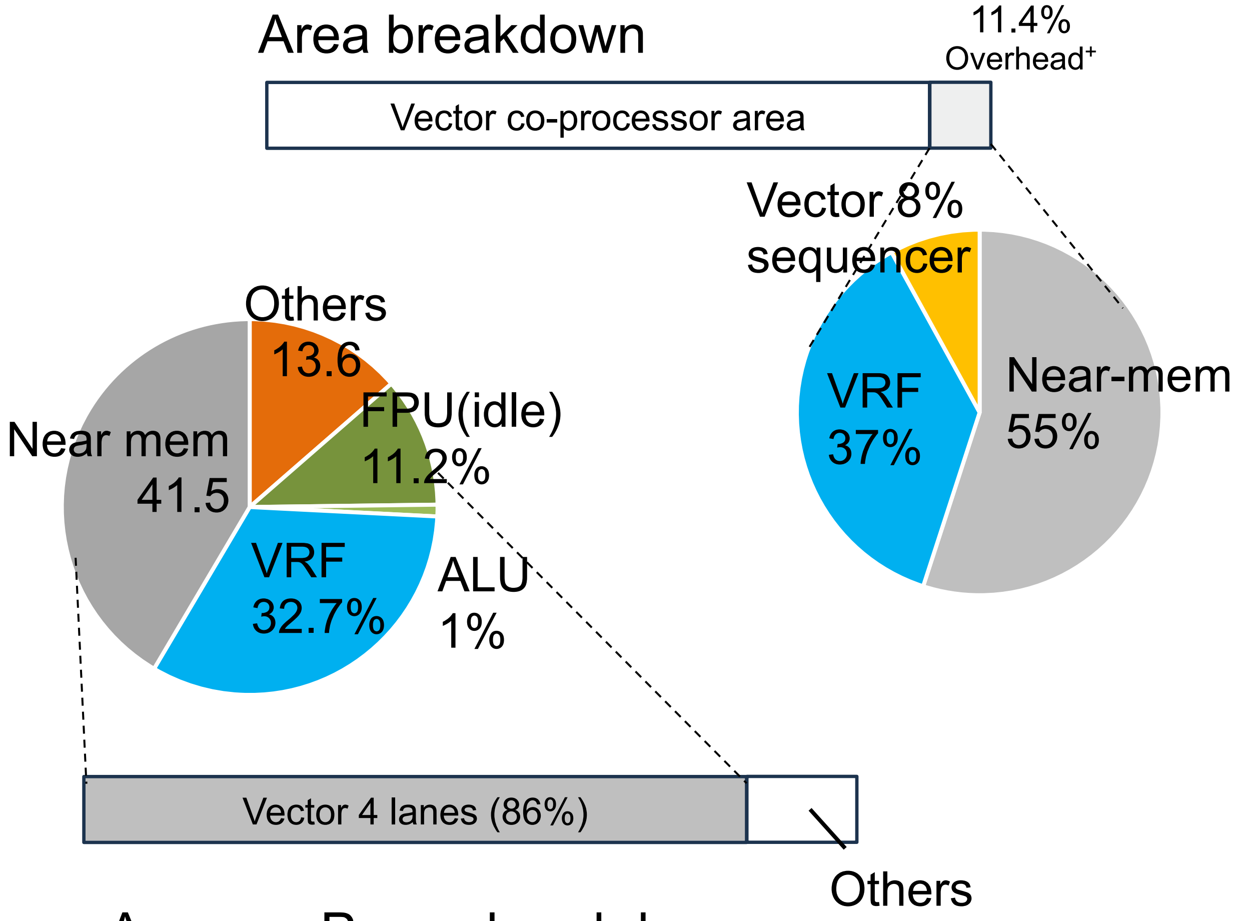


CIM Vector Register File

PPA and area overhead

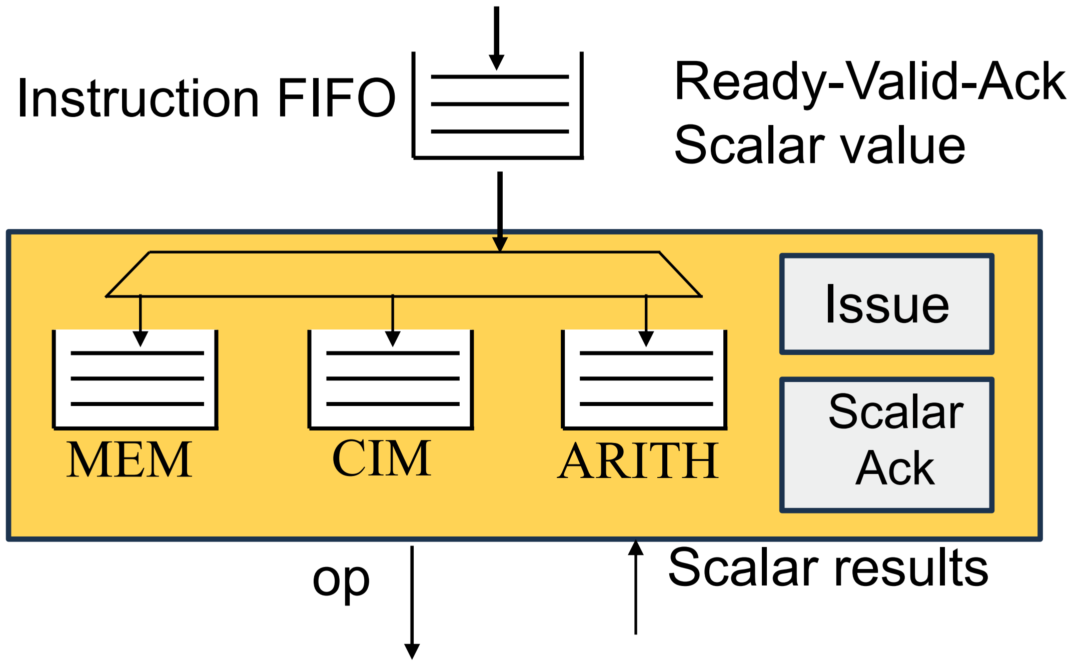
~40% delay overhead solution:
See back up slides!





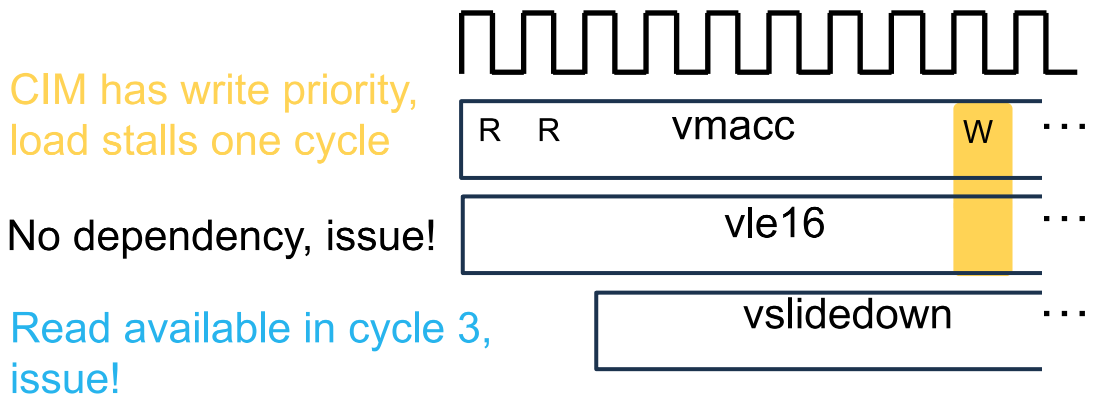
Average Power breakdown
Running matrix multiplication
Based on measurement and simulation

Vector Sequencer



asm capture for conv2d

```
 ...
vmacc.vv        v2 , v8,  v12   [CIM]
vle16.v         v16, addr       [MEM]w-RF
vslidedown.vi   v20, v8,  1     [ARITH]r-RF
 ...
```



CIM has write priority, load stalls one cycle
R
R
vmacc
W
...
No dependency, issue!
vle16
...
Read available in cycle 3, issue!
vslidedown
...

- 3 queues for memory load/store, CIM related, and other arithmetic instruction
- Light-weight out-of-order execution

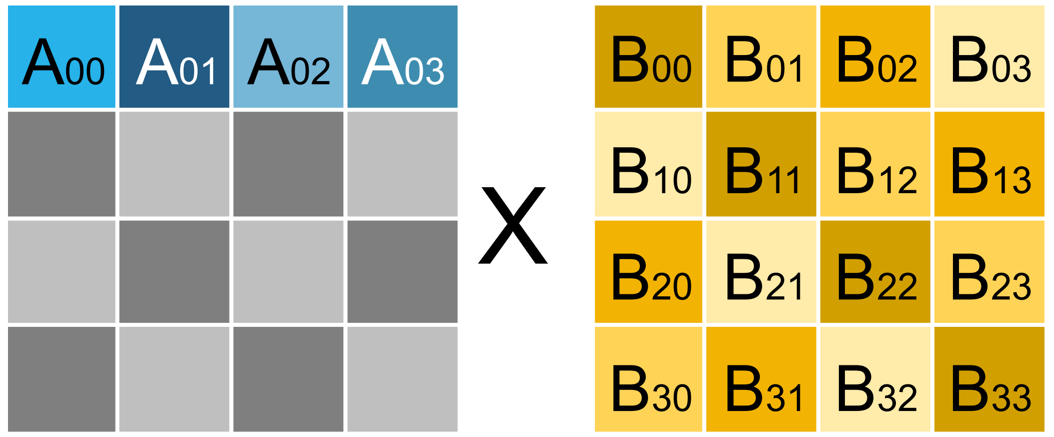
Outline

Emerging Matrix Multiplication Application

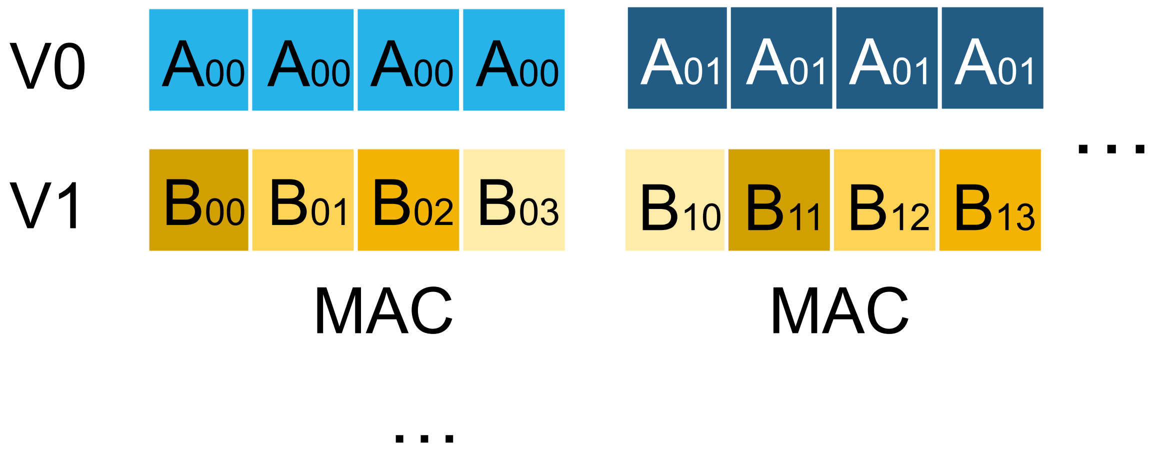
- ❑Deep learning: CNN, Transformer
- ❑Combinatorial optimization: SAT, Ising, ILP
  - Solve Max-SAT using matrix mul. [David Warde-Farley, Deepmind, Arxiv, 2023]
- ❑Security: Kyber, CKKS, TFHE
  - Vector-matrix multiplication in RLWE; TFHE key switching
- ❑Graphics: NeRF, 3DGS
  - Matrix multiplication in NeRF: NLP; 3DGS: View transformation

GEMM / MVM algorithm and Instruction extension

Matrix multiplication:



Reuse A:


V0
V1
A_{00} A_{00} A_{00} A_{00}
A_{01} A_{01} A_{01} A_{01}
B_{00} B_{01} B_{02} B_{03}
B_{10} B_{11} B_{12} B_{13}
MAC
MAC
…
…

```
add    addrA + 8
ld     t0 <- A[]
vmacc  t0,vB,vC
```

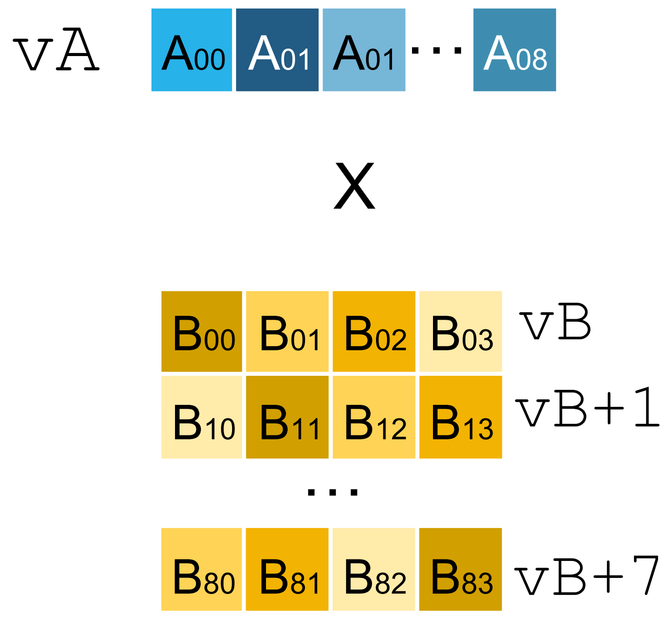
1 mac per 3 instructions

- Reuse A: Instructions bottleneck

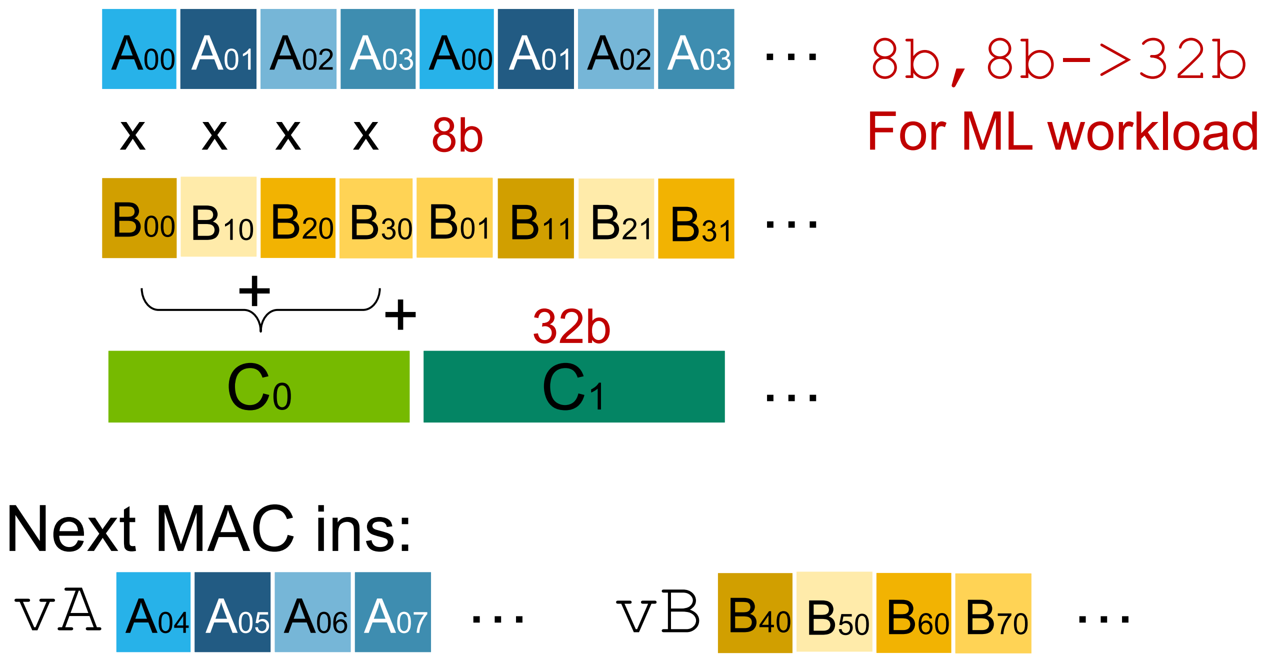
GEMM / MVM algorithm and Instruction extension

Instruction extension

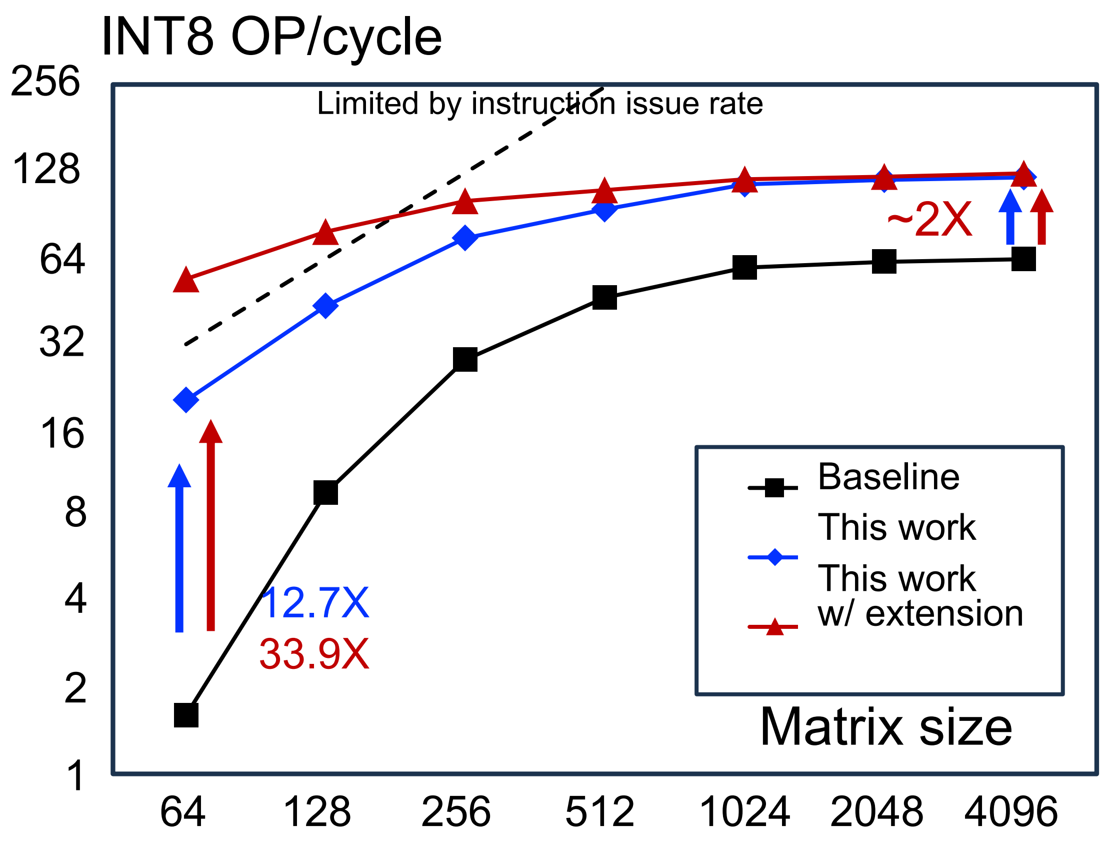
FP16/BF16:

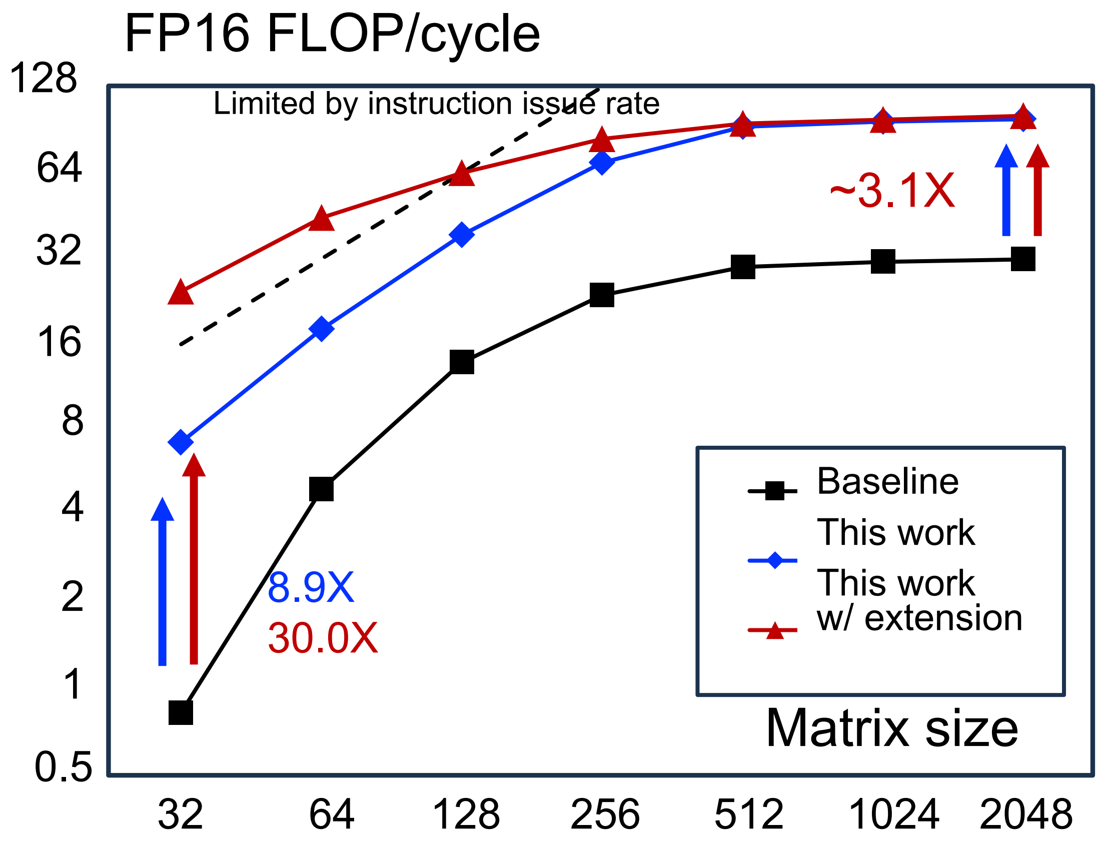


INT8:



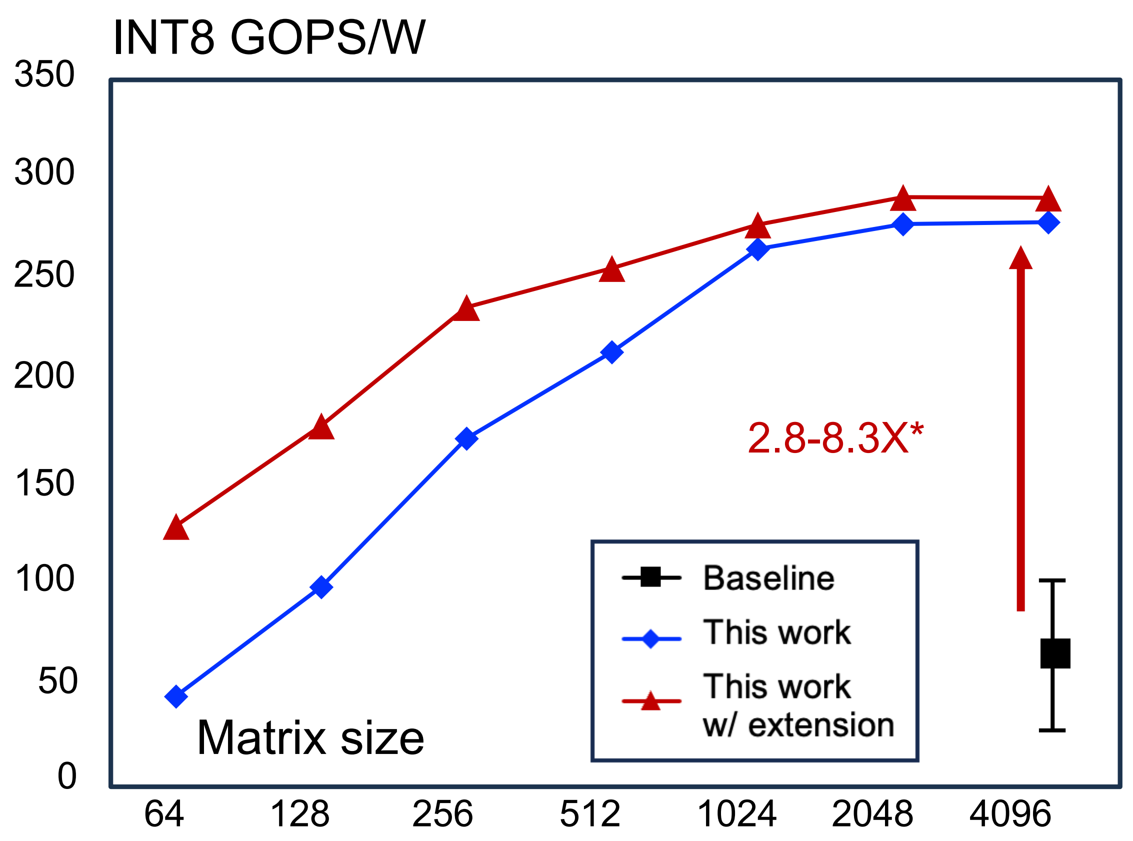
Throughput measurement

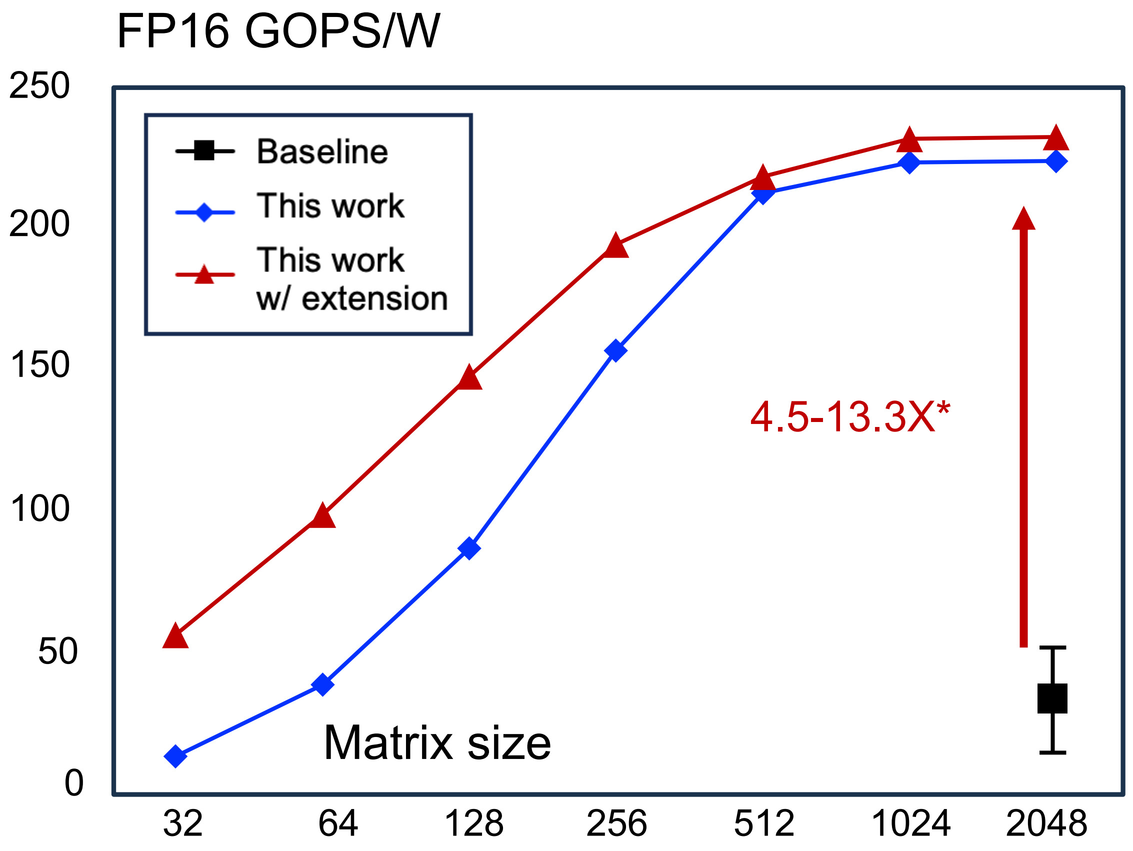




Average throughput measurements running matrix multiplication tasks.

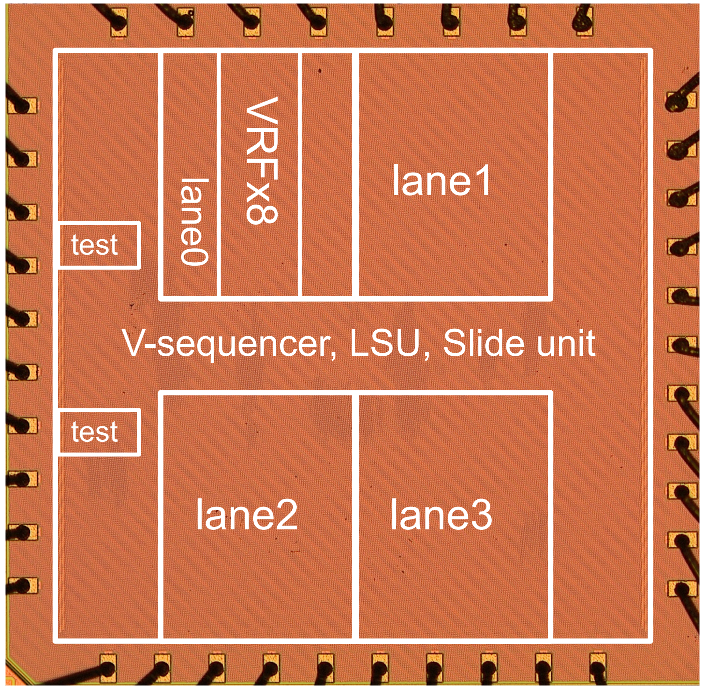
Efficiency measurement with average power





*the min and max point corresponding to power ∝ tech^2 (pessimistic) and ∝ tech (optimistic) normalization.
Power measurement is the average of the power curve running matrix multiplication tasks.
This work does not count CPU power.

Die shot and chip summary



Chip summary	
Technology	65nm
Supply voltage	1V
Die size	2x2 mm^2
Frequency	250MHz
Precision	All (INT8/BF16/FP16 in memory)
VRF size	4 lanes x 8 banks x 4kb
Bit cell area	1.658 um^2
Performance	31.8 / 25.3 GOPS
Energy efficiency	289.13 / 230.10 G(FL)OPS/W
Area efficiency	7.95 / 6.33 GOPS/mm^2

Comparison table

*1: 256x256 size matrix multiplication, unless noted; *2: Use power ∝ tech^2, this may be pessimistic for advanced nodes.
*3: This is optimistic since there's no pads. *4: calculated by 1:1 mul/add efficiency x reported CPU mode power.

	Ara [2]	ISSCC 2019 [5]	VLSI 2023 [6]	This work
Technology	GF 22nm	TSMC 28nm	TSMC 65nm	TSMC 65nm
ISA	RISCV	Customed	Customed	RISCV
Category	General purpose processor	Customed CIM design with general purpose operations		General purpose processor
CIM type	-	Custom 8T	Custom 8T/9T	Foundry 8T
Bit precision	All	All (potentially)	INT8/32	All (INT8/BF16/FP16 enhanced)
Design level	Processor (simulation)	Macro + ctrl	Macro + ctrl	Co-processor
Peak performance INT8/FP16[*1] (CLK frequency is different)	78.4 / 39.2 G(FL)OPS	40.5 / ~0.5 G(FL)OPS	2.3[*4] / X GOPS	31.8 / 25.3 G(FL)OPS
Throughput (GOPS/MHz)	0.063 / 0.031	0.085 / ~	0.012 / X	0.127 / 0.101
Energy efficiency INT8/FP16 (power normalized to 65nm[*2])	34.64 / 17.32 G(FL)OPS/W	439.78 / ~5 G(FL)OPS/W	473.35 (GP mode) 7620 (DNN mode) / X GOPS/W	289.13/230.10 G(FL)OPS/W

Outline

Summary

- Demonstrate SRAM Compute-in-memory in RISCV vector processor register file.
- The 1R1W 8T SRAM register file uses foundry cell with digital CIM and near memory compute unit.
- Achieves 289.13GOPS/W and 7.95GOPS/mm^2 for INT8, 230.10GFLOPS/W and 6.33 GOPS/mm^2 for FP16 precision.

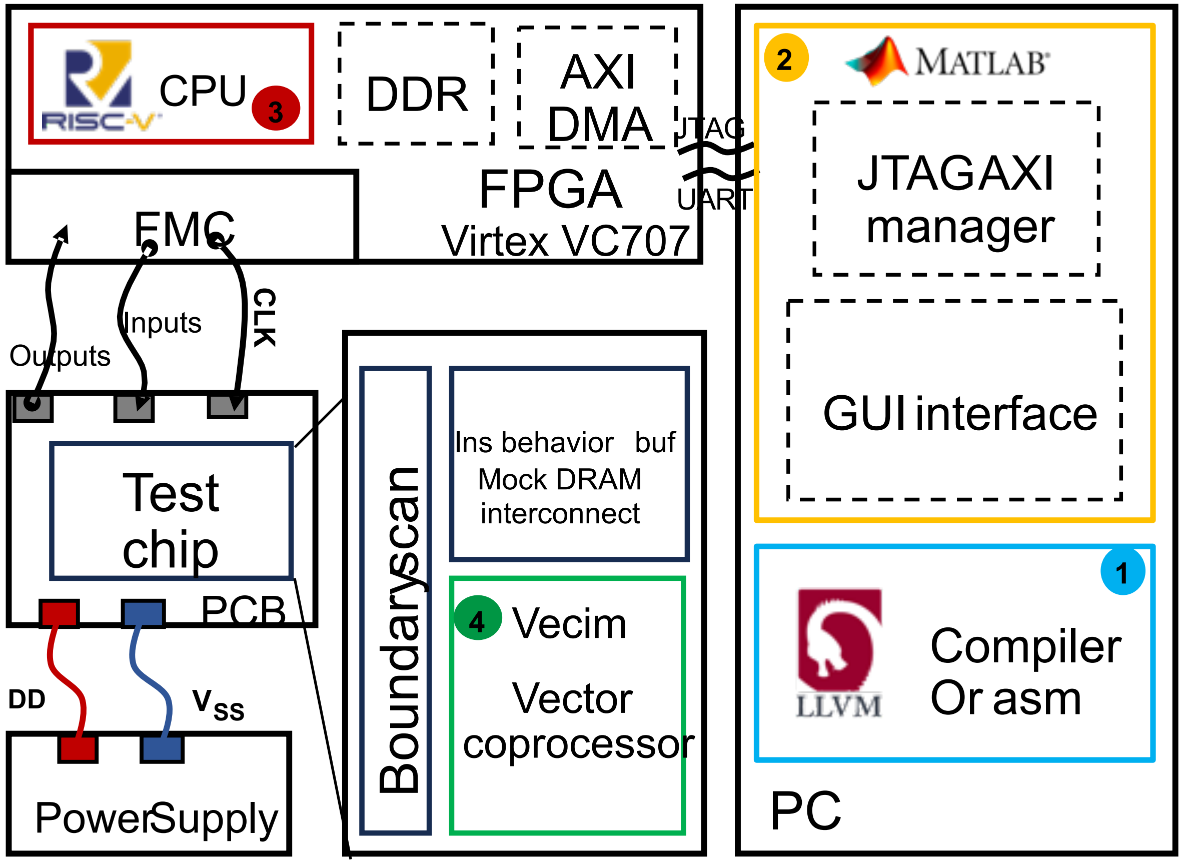
Acknowledgments

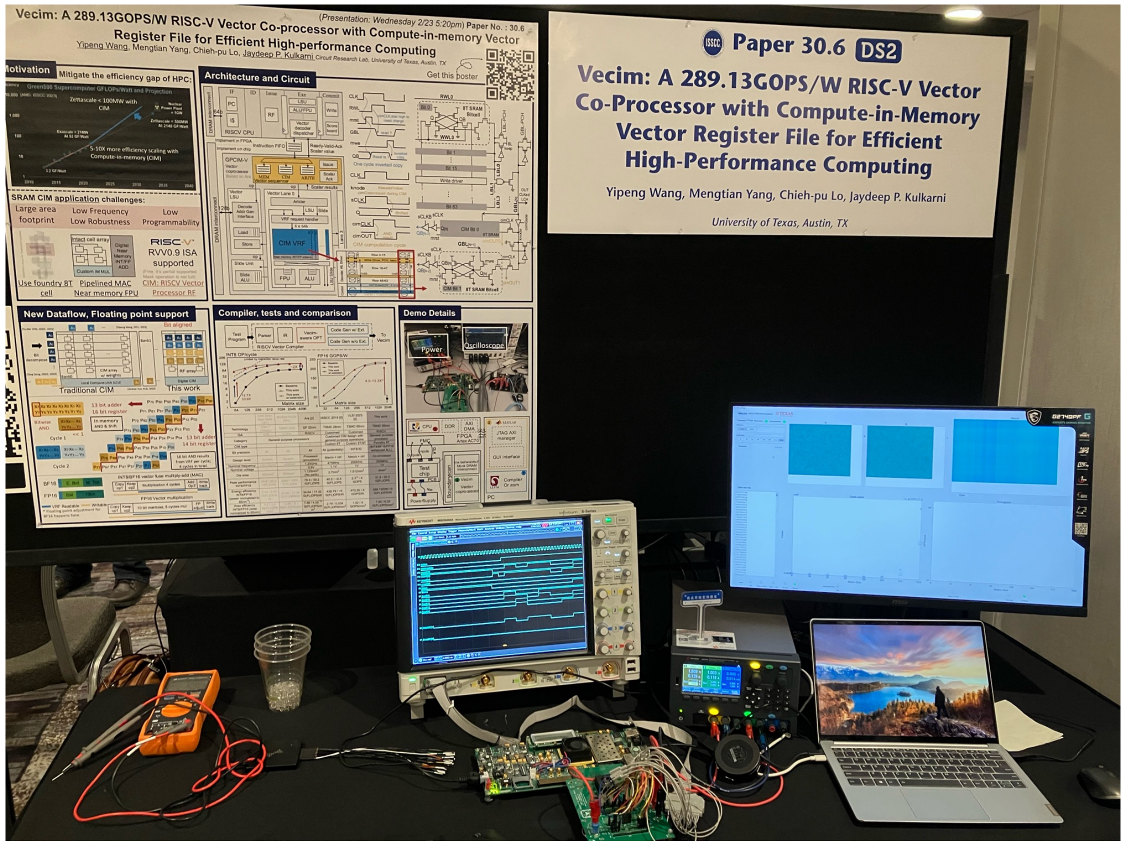
- TSMC university shuttle support
- UT ECE iMAGINE consortium

Thank you for your attention!

Demo system

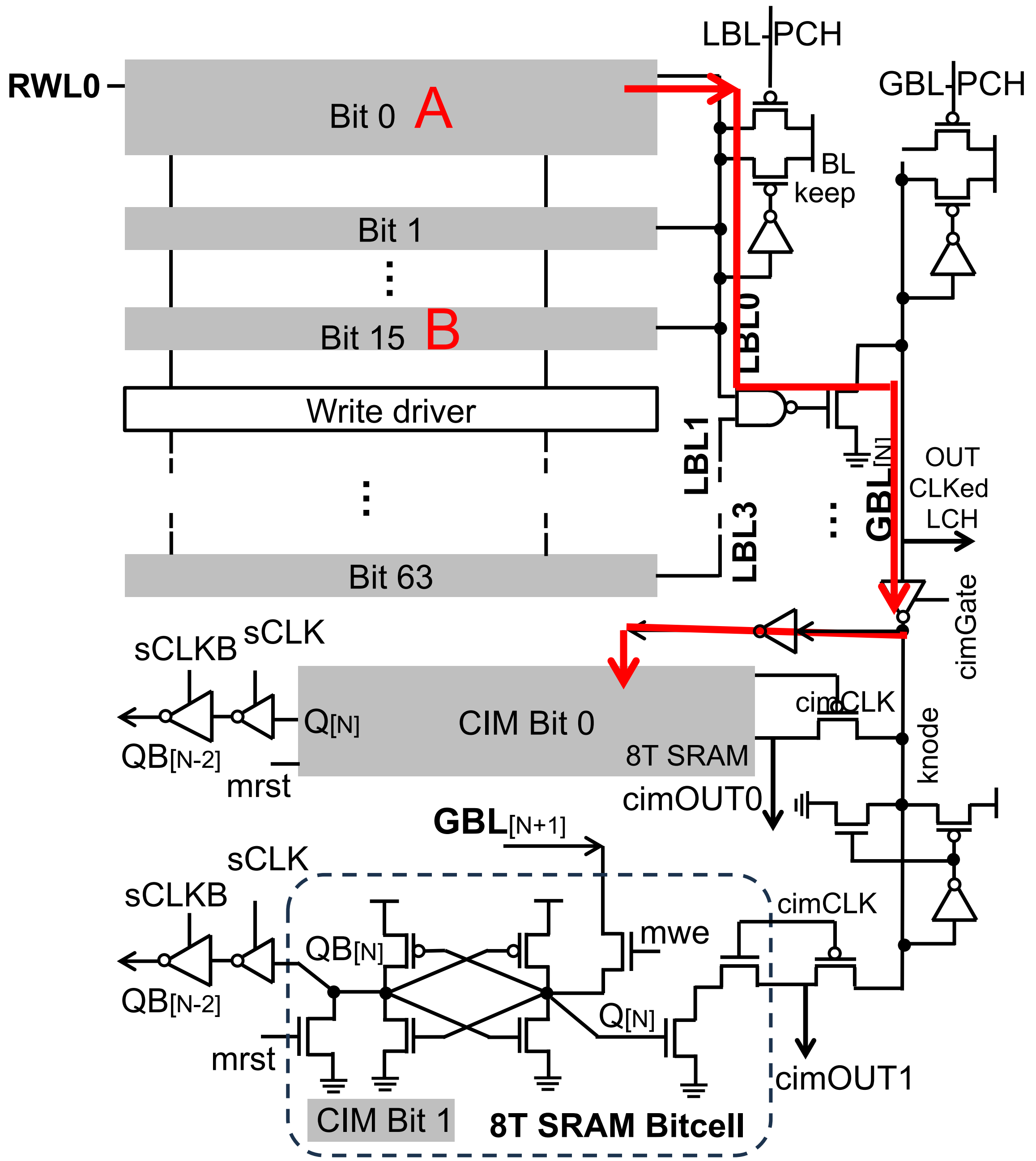
Get the poster



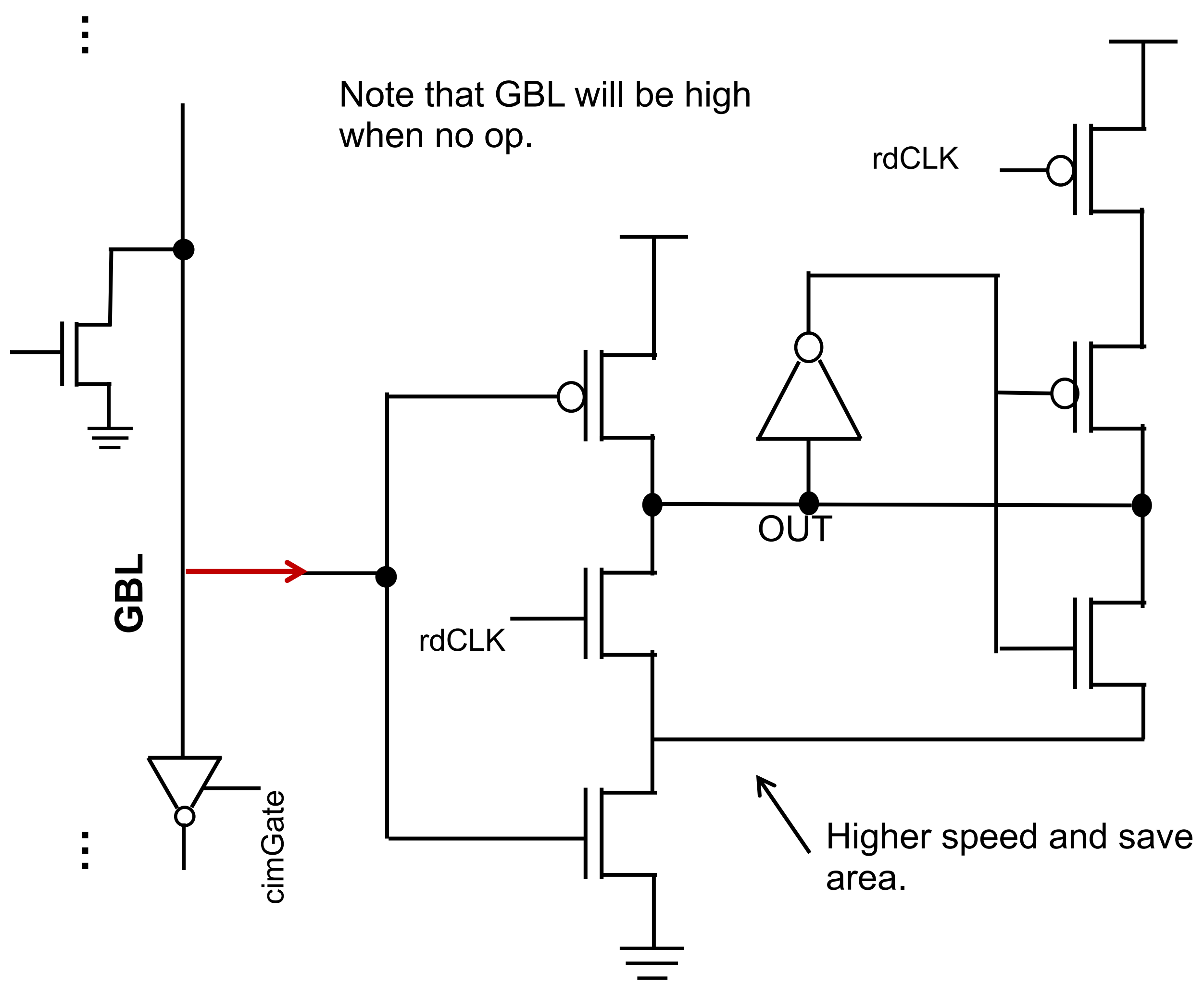


Solution for slides 24

- Write to CIM bit happens next cycle
- Inverted copy is guaranteed not happen in two consecutive cycles.
- No timing overhead now.
- Some area overhead.



Output CLKed latch

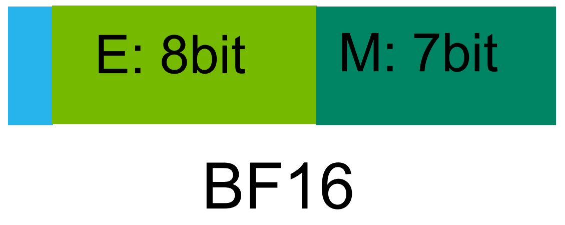


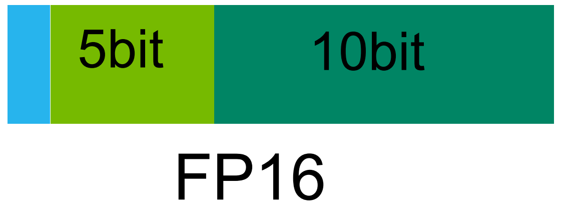
Note that GBL will be high when no op.

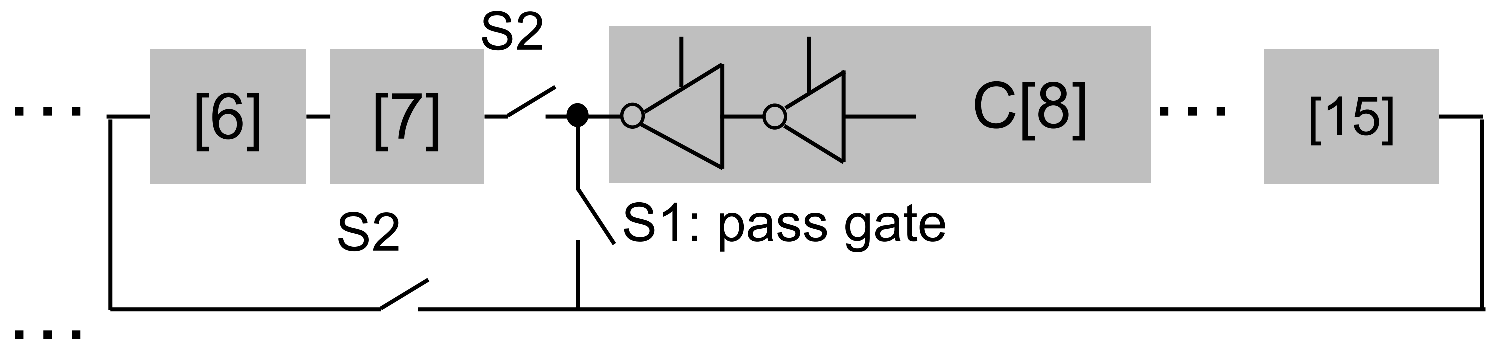
Higher speed and save area.

BL / shifter MUX pattern for INT8/BF16/FP16

FP16 needs reconfigure 8bit ring shifter to 10bit.
We show the MUX pattern here. (GBL is similar)



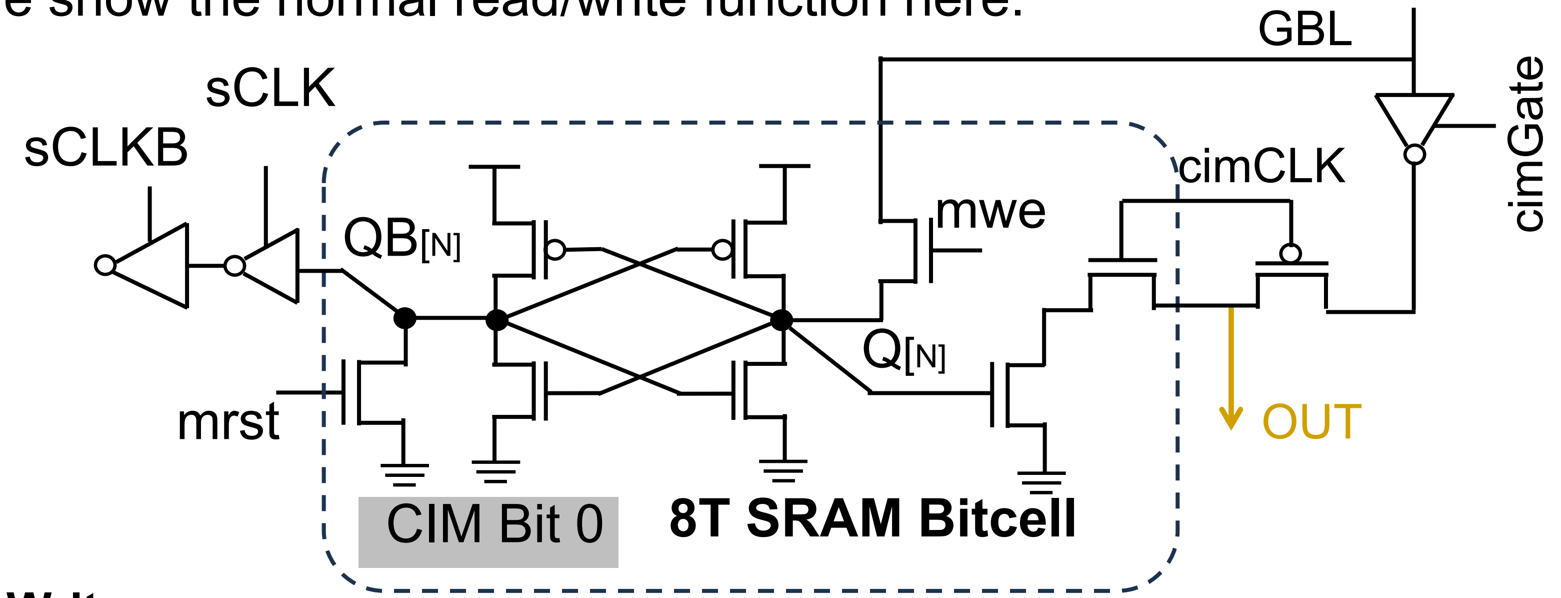




BF16: S1 closed S2 open, C[8] out disabled;
FP16: S1 open S2 closed, E read out for add.

Reuse CIM Bits as renaming physical register

The CIM Bits can be used as physical register for renaming,
We show the normal read/write function here.



Write

1. Assert mrst, reset to 1
2. Discharge GBL based on data and open mwe

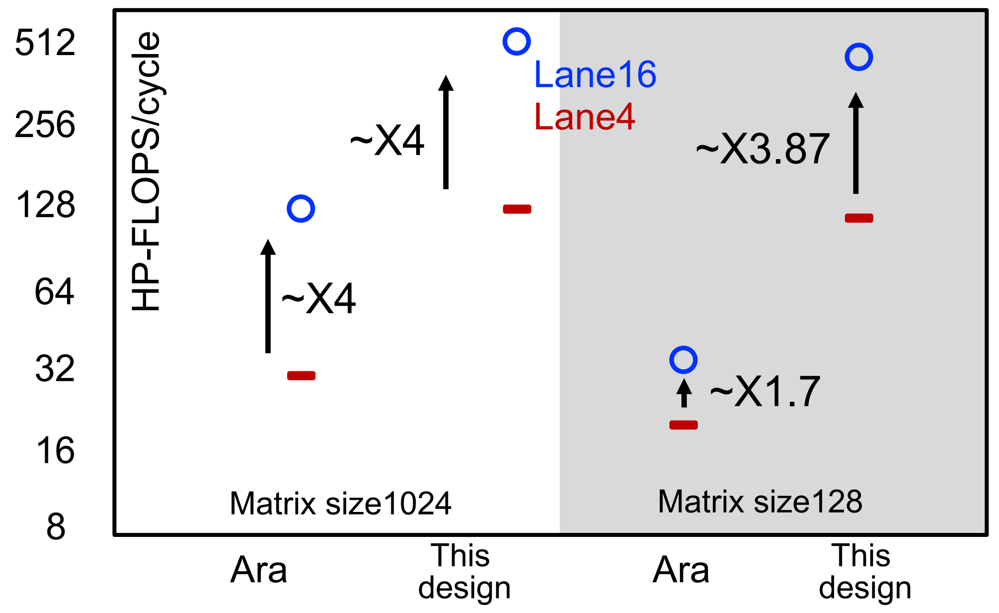
Read

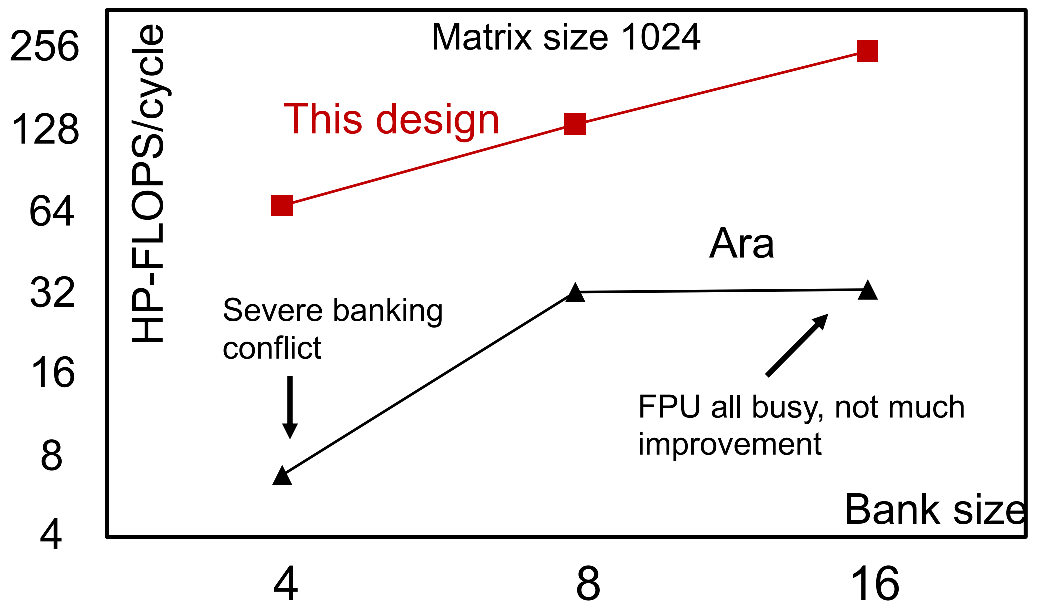
1. Discharge GBL and open cimGate
2. Toggle cimCLK

Read Write CIM Bits has lower energy and higher performance.
This design did not implement register renaming.

Performance scaling simulation

We show the scalability of this design by increasing the lane numbers and increasing the VRF capacity. The multicore design evaluation is out of our scope.





This design scales good with number of lanes on both large and small matrix size.

This design scales simply with VRF size with enough memory bandwidth.