Mondays in Memory (MiM) Webinar

A 28nm 16.9-300TOPS/W Computing-in-Memory Processor Supporting Floating-Point NN Inference/Training with Intensive-CIM Sparse-Digital Architecture

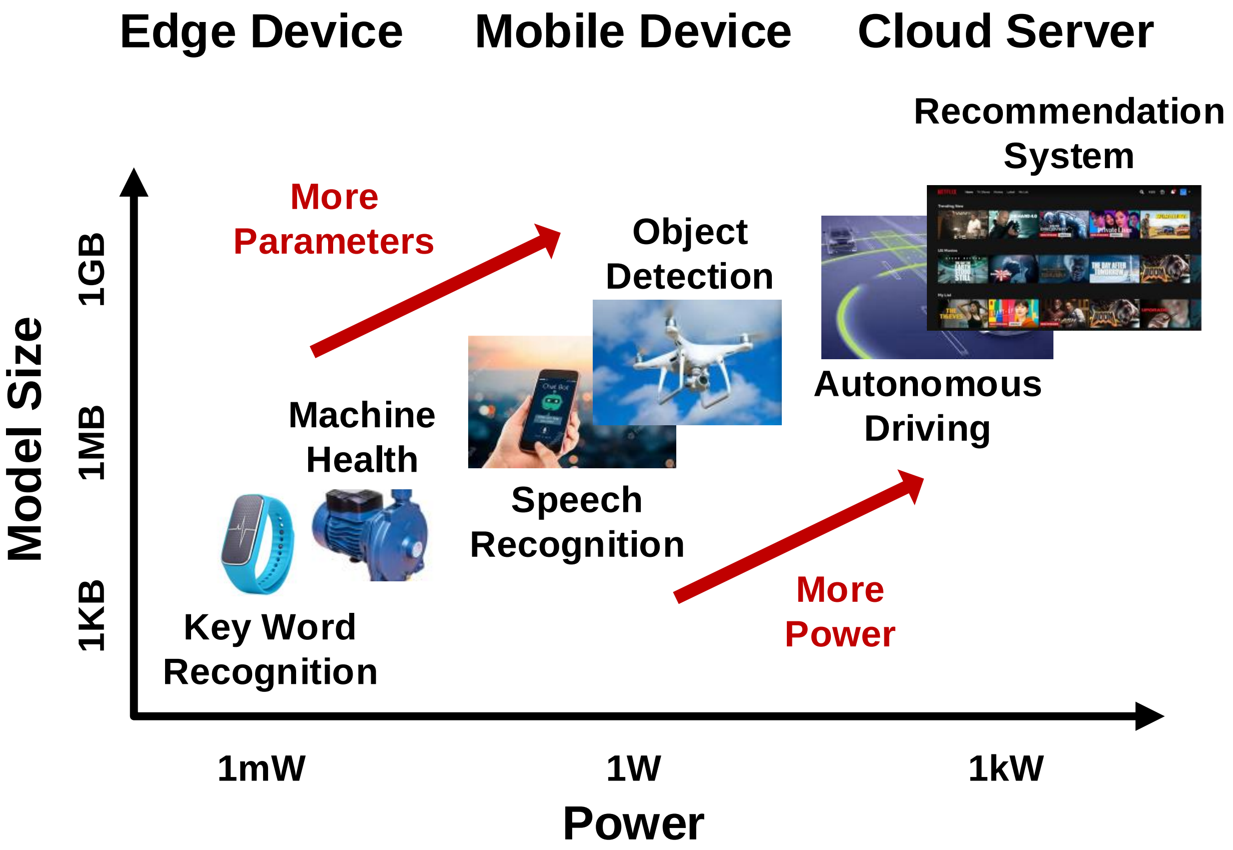
Jinshan Yue[1], Chaojie He[1], Zi Wang[1], Zhaori Cong[1], Yifan He[2], Mufeng Zhou[2], Wenyu Sun[2], Xueqing Li[2], Chunmeng Dou[1], Feng Zhang[1], Huazhong Yang[2], Yongpan Liu[2], Ming Liu[1]

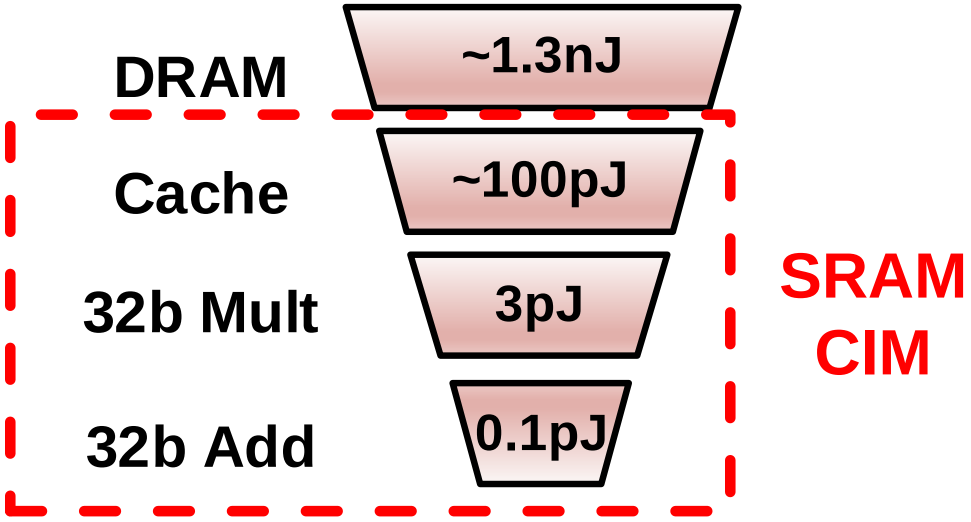
[1] Institute of Microelectronics of the Chinese Academy of Sciences
[2] Tsinghua University

Outline

- Motivation & Challenges
- Proposed FP CIM Processor
  - Efficient FP-to-INT CIM workflow for intensive FP operations
  - Flexible sparse digital core for sparse FP operations
  - Low-MACV CIM macro for random sparsity
- Measurement Results
- Conclusion

Computing-in-Memory (CIM)





Sources: M. Horowitz, ISSCC’14 1.1

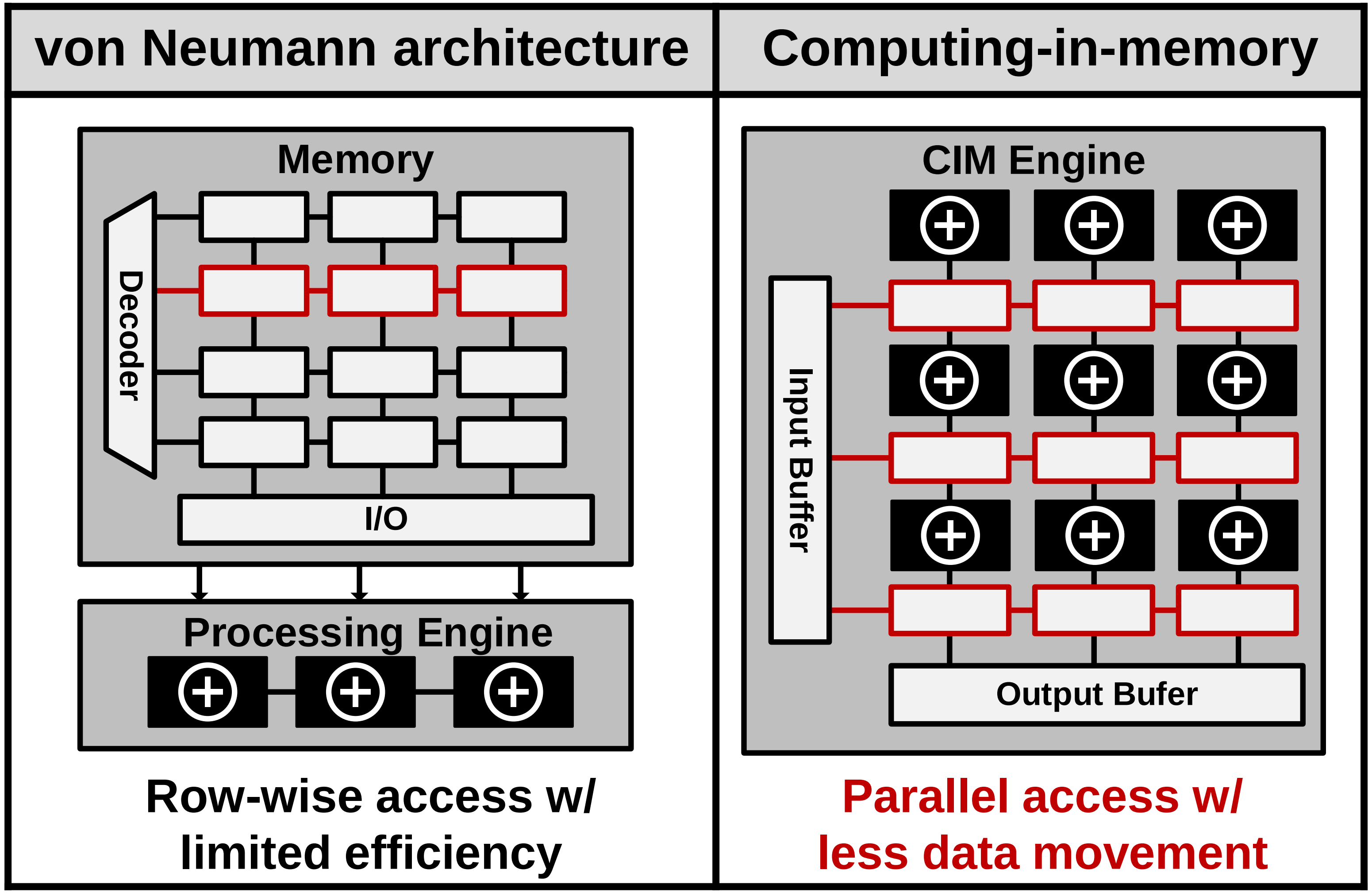
Memory access dominates neural network inference

Source: ISSCC2023 7.3.

Computing-in-Memory (CIM)

Matrix-vector multiplication (MVM)

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,N} \\ \vdots & \ddots & \vdots \\ w_{M,1} & \cdots & w_{M,N} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}$$



Row-wise access w/ limited efficiency

Parallel access w/ less data movement

Source: ISSCC2023 7.3.

CIM reduces data access energy and provides high computation bandwidth

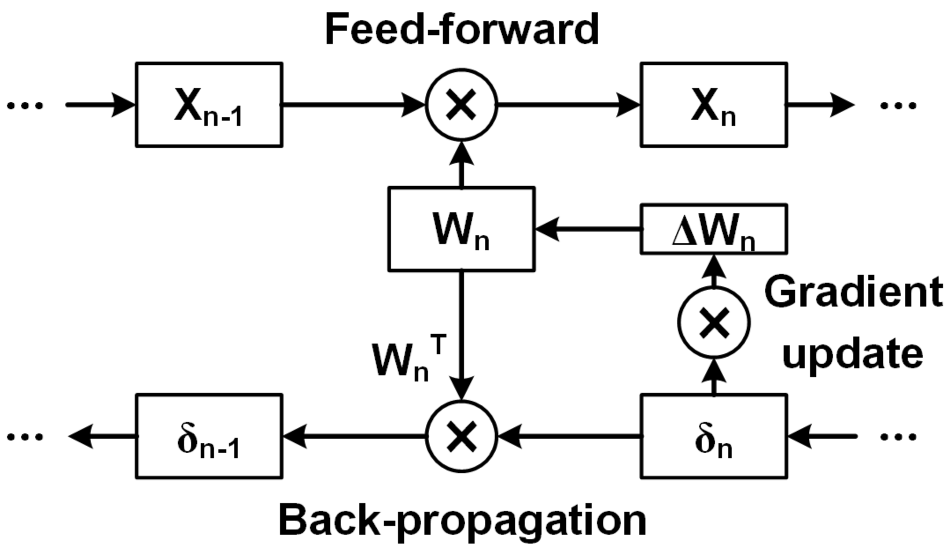
Floating-Point (FP) CIM

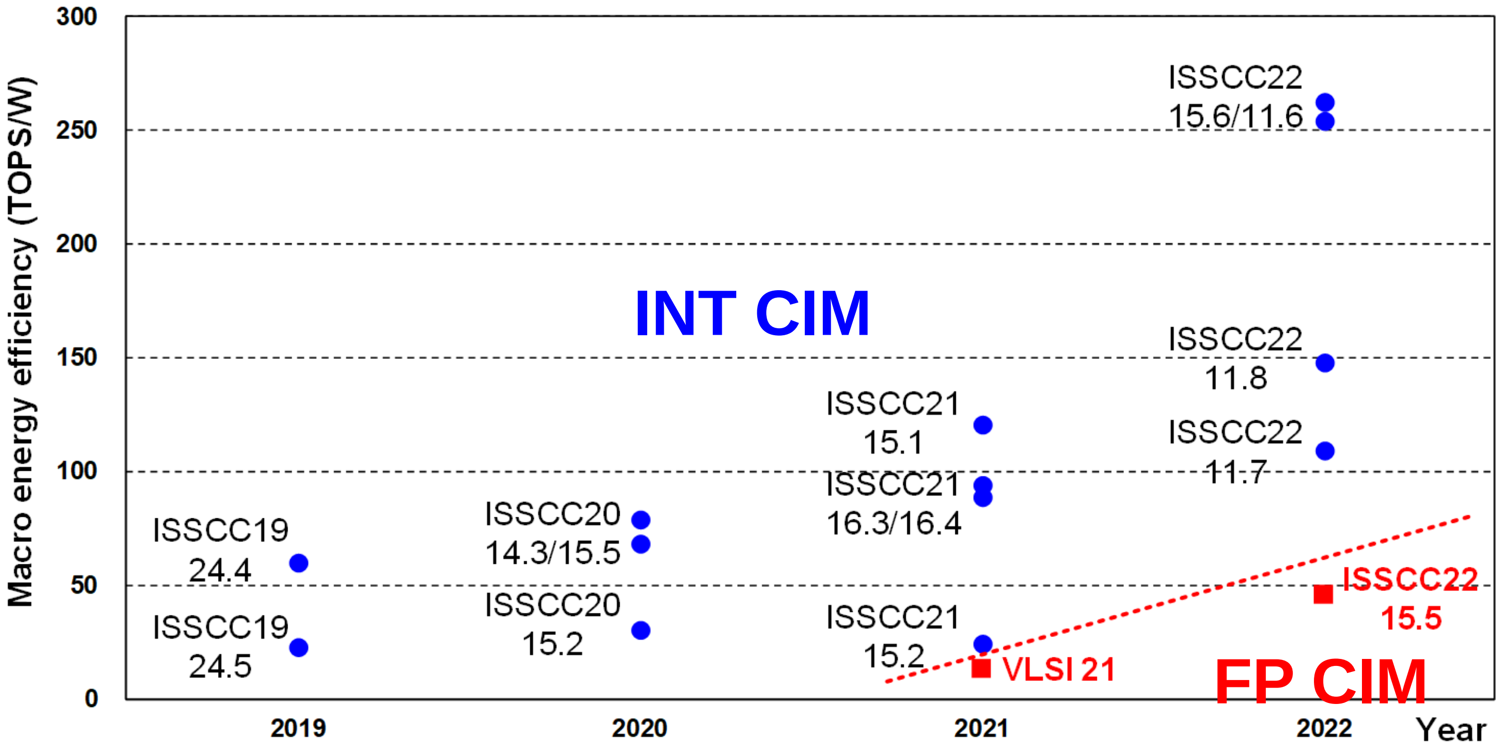
- Why FP CIM: Higher accuracy, training
  - **Integer (INT) CIM** : High energy efficiency
  - **Floating-point (FP) CIM** : Lack of research, limited efficiency

FP NN scenarios:

① Higher accuracy

② Training tasks

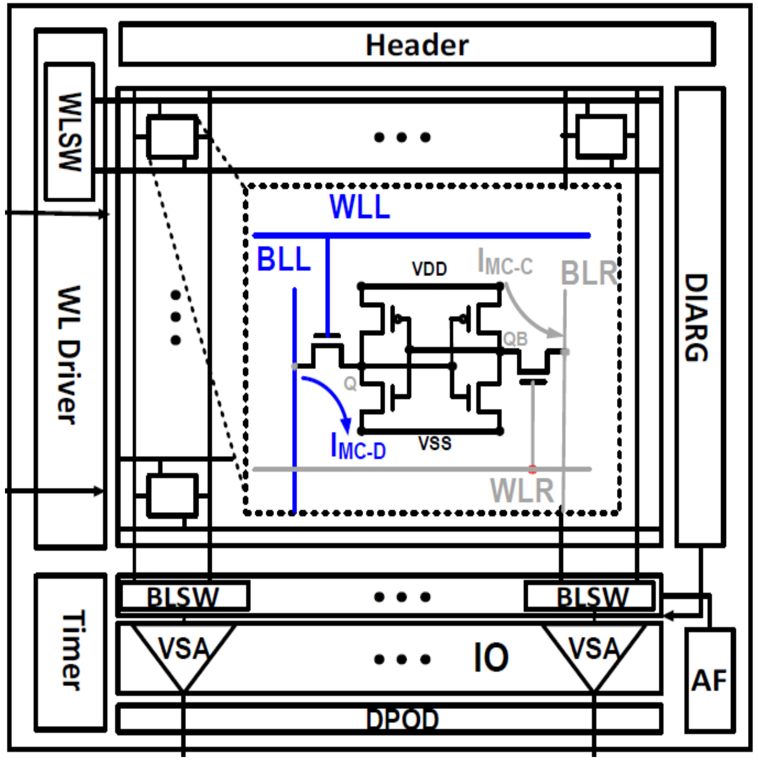




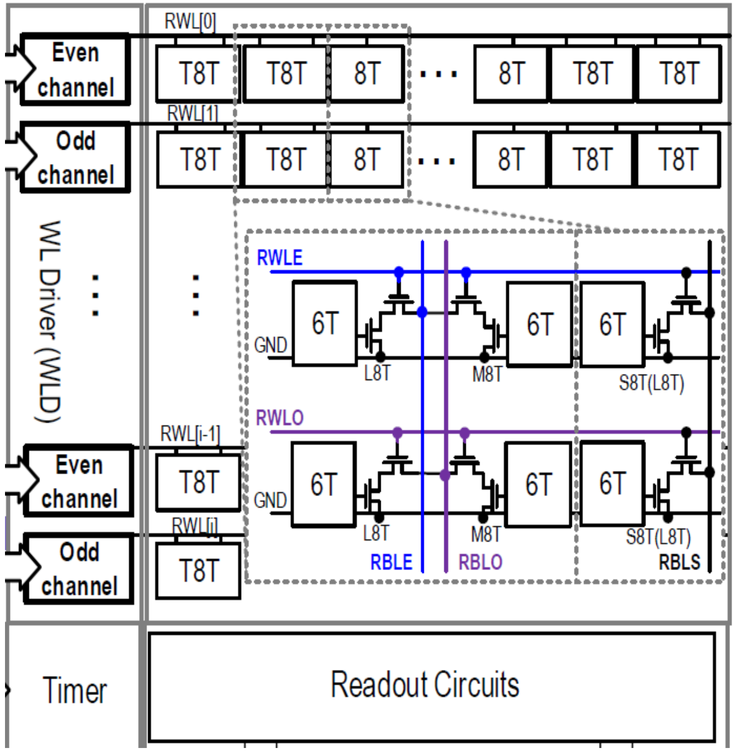
Scaled to INT4 or FP16 precision

Fixed-Point (INT) CIM

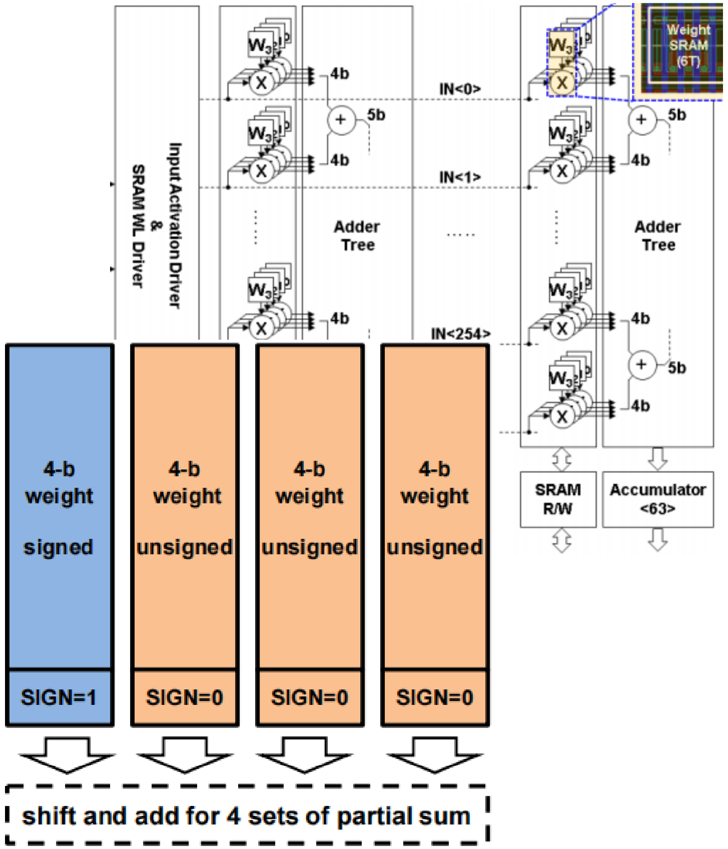
- INT computation naturally suit the CIM circuits design
- Operations on each column are similar



Binary CIM



Multi-bit analog CIM



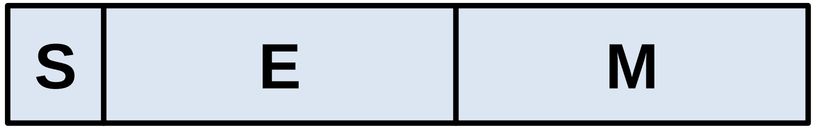
Multi-bit digital CIM

Source: ISSCC 2019, 2020, 2021.

Floating-Point (FP) CIM

- How are FP operations executed?

FP format: Sign + Exponent + Mantissa



FP16: 1b	5b	10b
BF16: 1b	8b	7b
FP32: 1b	8b	23b

$$Data = (-1)^{S} \cdot 2^{E-E_0} \cdot (\underbrace{1}_{\text{Hidden bit}} + M)$$

- FP multiplication steps

$$\boldsymbol{Shift}(\boldsymbol{E_A} + \boldsymbol{E_B}), \boldsymbol{Norm}(\boldsymbol{M_A} \cdot \boldsymbol{M_B})$$

① Exponent addition

② Mantissa multiplication

③ Mantissa normalization

④ Exponent shift

- FP addition steps*

$$\boldsymbol{Shift}(\boldsymbol{E_A}), \boldsymbol{Norm}(\boldsymbol{M_A} + (\boldsymbol{M_B} \gg \boldsymbol{E_{A-B}}))$$

① Exponent compare and subtraction

② Mantissa shift

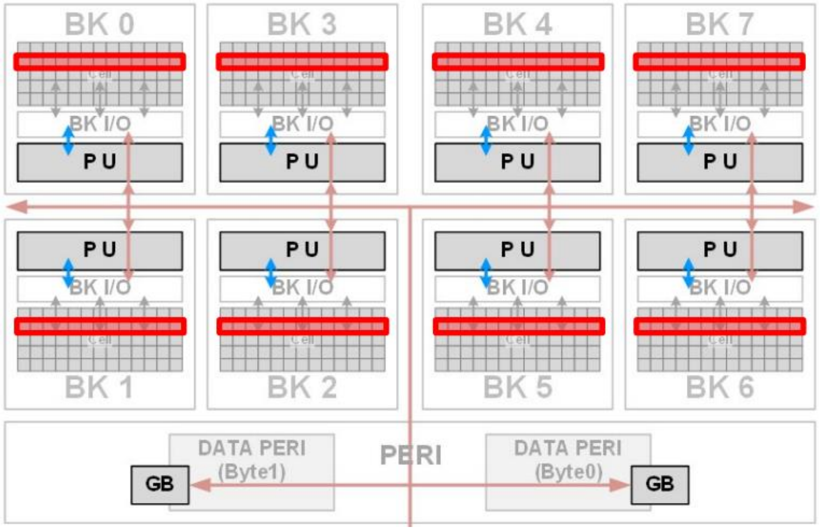
③ Mantissa addition

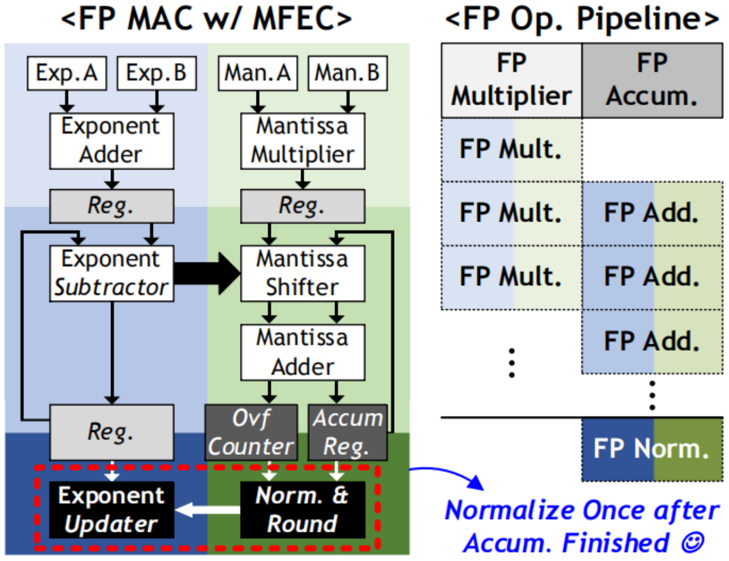
④ Mantissa normalization

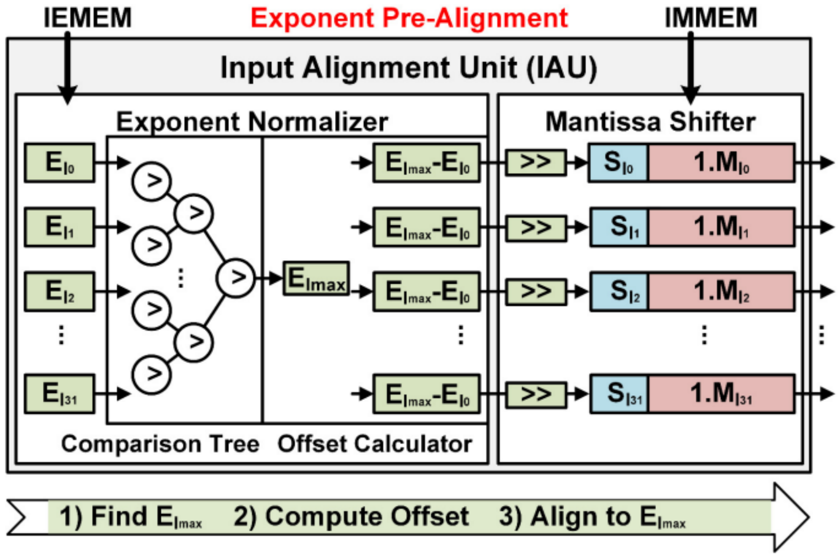
⑤ Exponent shift

* Assume $E_A > E_B$

Existing FP CIM Solutions







Solution 1:

Direct FP/Boolean logic (near memory)

☺ FP CIM execution

☹ Limited row parallelism

Solution 2:

Separated exp./man. circuits

☺ FP CIM execution

☹ Limited parallelism

Solution 3:

Alignment: Compare exp. & shift FP as INT MAC

☺ FP CIM execution

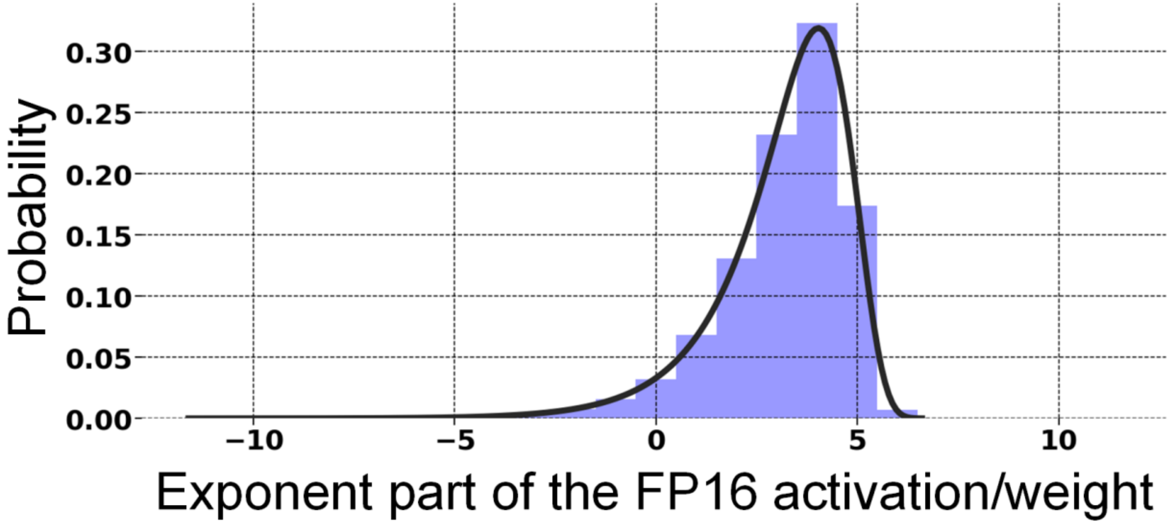
☺ INT/FP configurable

☹ Many execution cycles

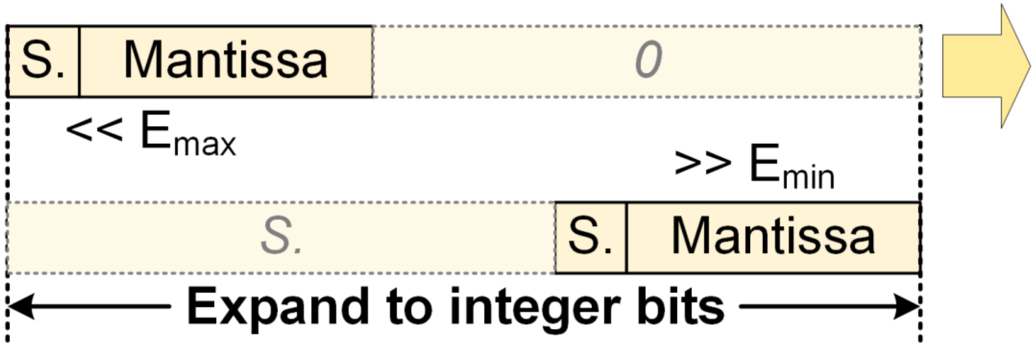
Source: ISSCC2022 11.1/15.5, VLSI2021 JFS2-1.

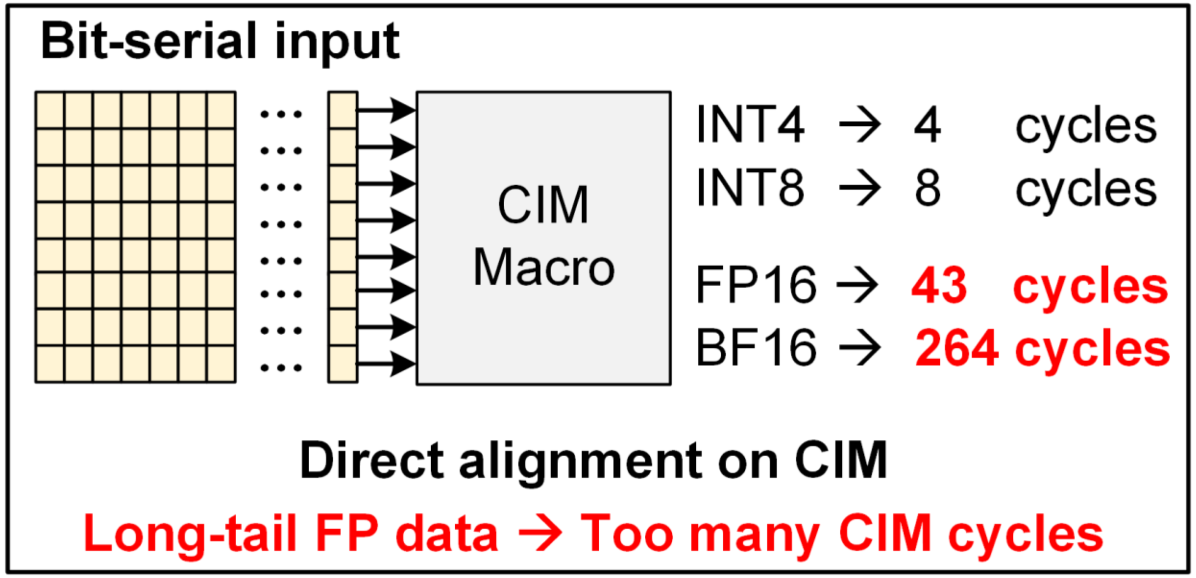
Challenge 1: Long-Tail FP CIM

- Direct FP expansion (alignment)
  - Left/right shift according to the exponent value
  - Exponent: Long-tail distribution
  - Many CIM execution cycles



	Sign	Exponent	Mantissa
FP16	1	**5**	10
BF16	1	**8**	7





Bit-serial input

INT4 → 4 cycles

INT8 → 8 cycles

FP16 → **43 cycles**

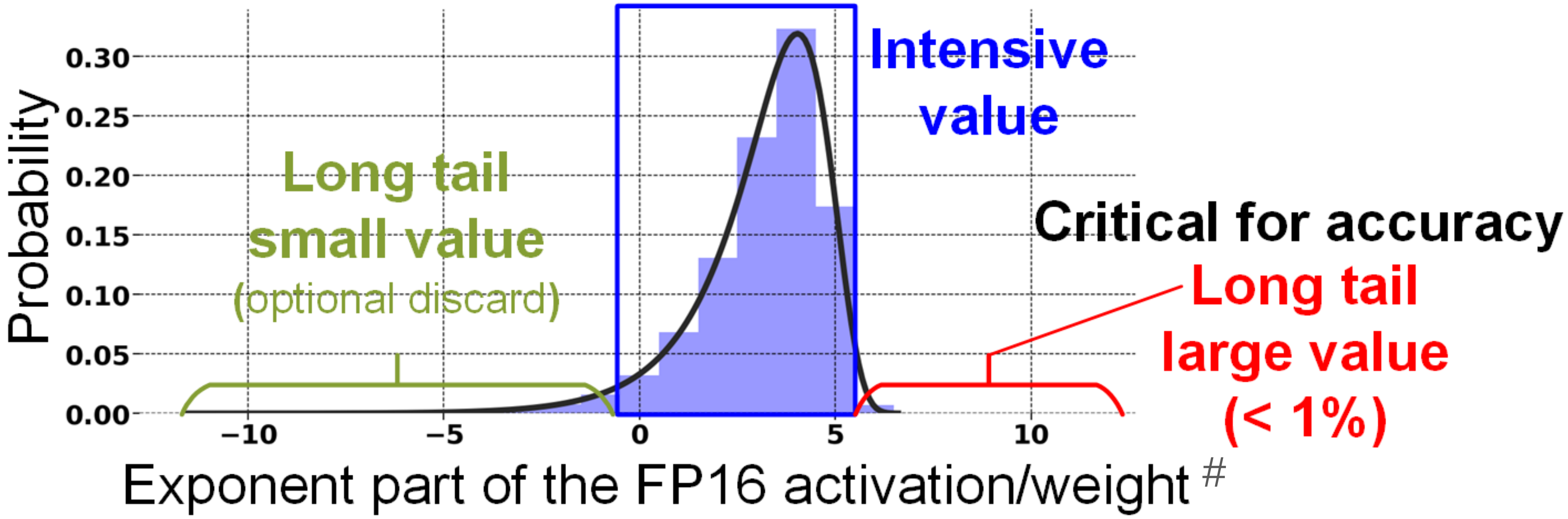
BF16 → **264 cycles**

Direct alignment on CIM

Long-tail FP data → Too many CIM cycles

Challenge 1: Long-Tail FP CIM

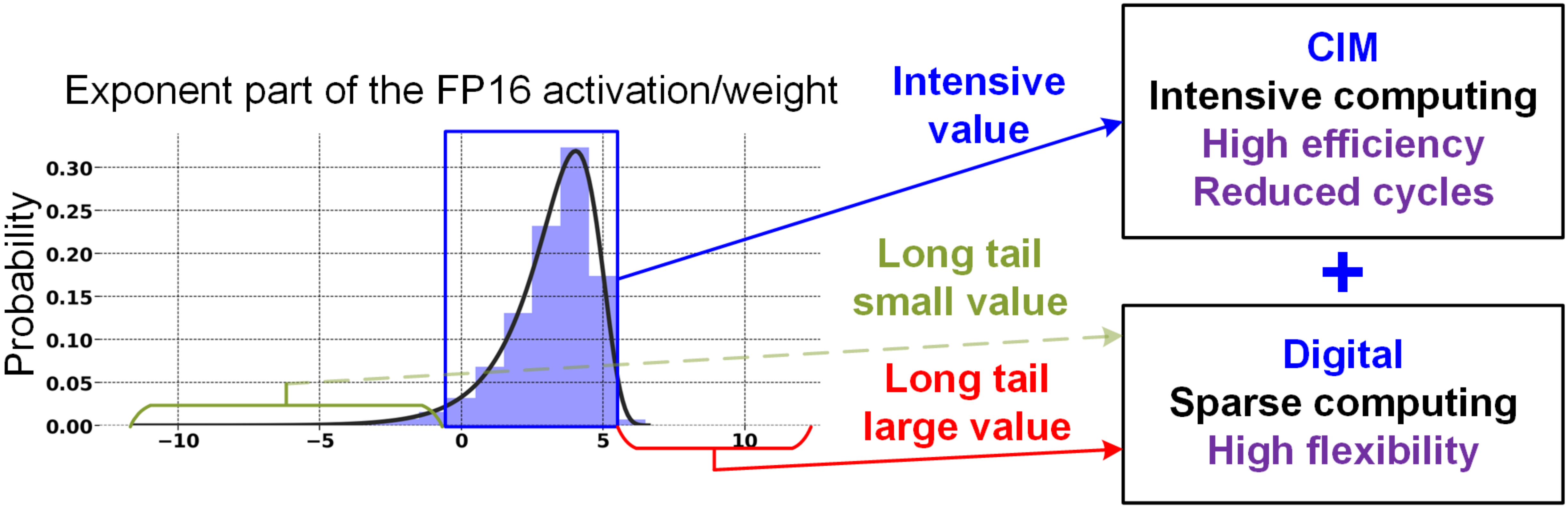
- Observation: Long-tail FP values take a small proportion
- Intensive FP values in a small alignment range (reduced CIM cycles)
- Long-tail FP values: Small proportion, critical for accuracy



# Presented is the activation data distribution from a specific convolutional layer on ResNet50, ImageNet. The activation/weight data in other layers show similar distribution.

Challenge 1: Long-Tail FP CIM

- Solution: Divide FP data into intensive CIM + sparse digital parts



Intensive value

Long tail small value

Long tail large value

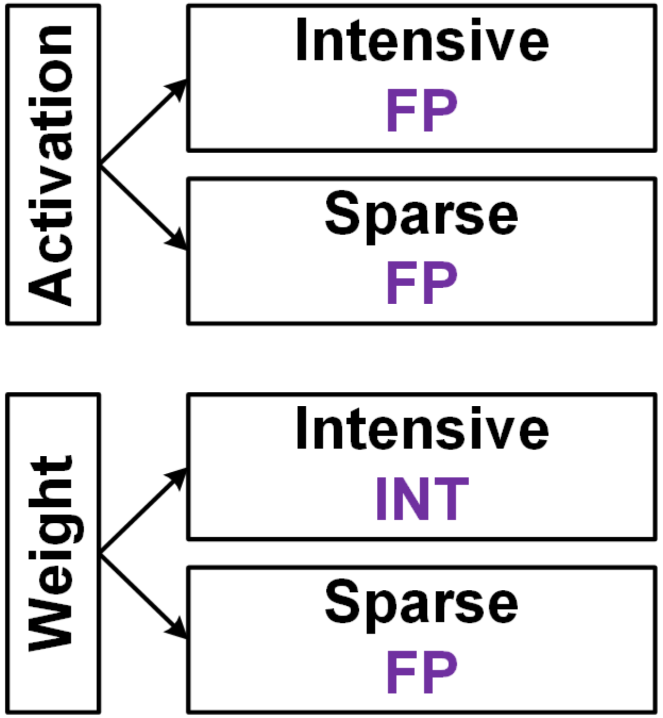
CIM
Intensive computing
High efficiency
Reduced cycles

+

Digital
Sparse computing
High flexibility

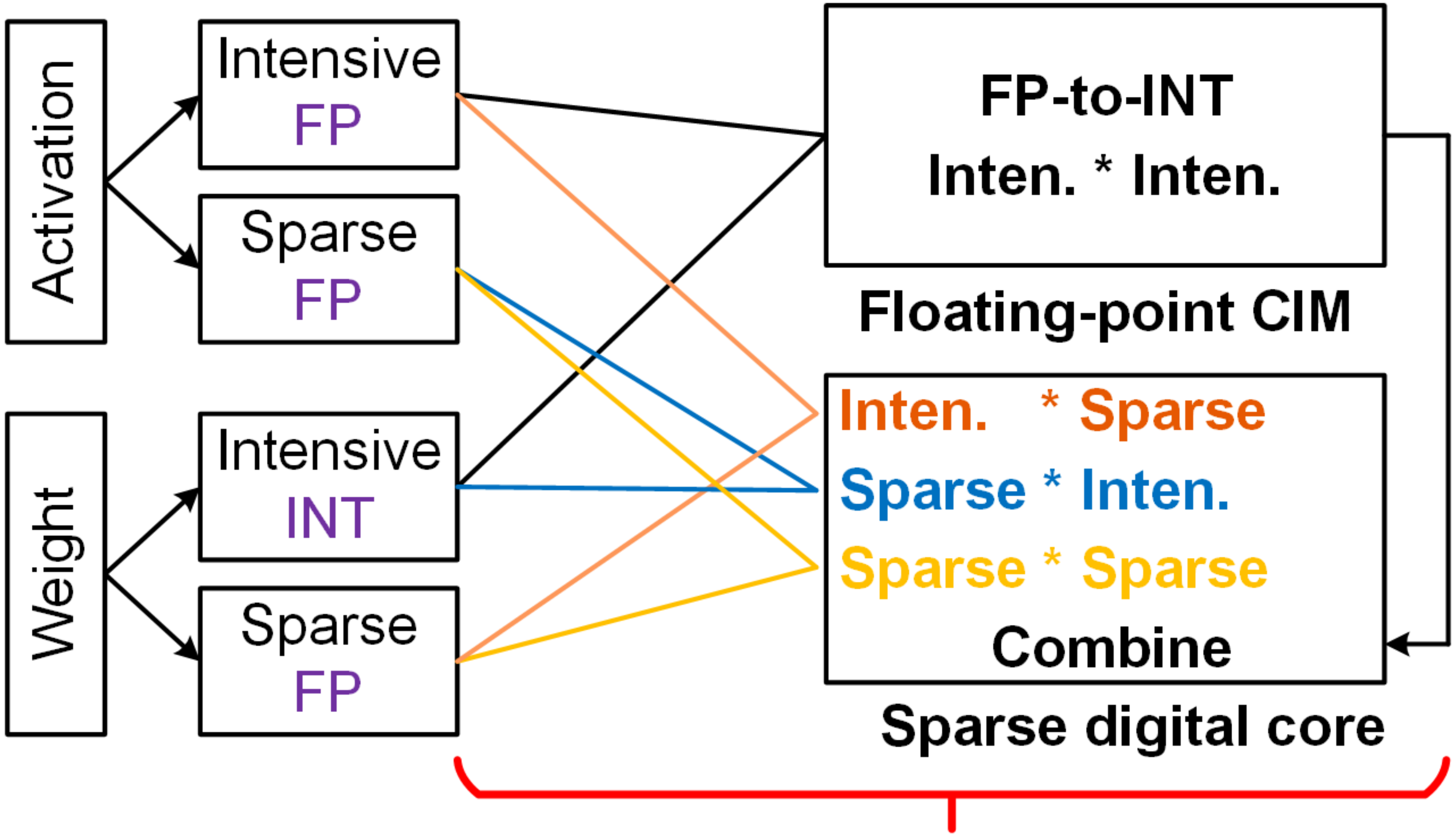
Challenge 2: Efficient Intensive-Sparse Workflow

- Divide both FP activation/weight data into intensive and sparse parts



Challenge 2: Efficient Intensive-Sparse Workflow

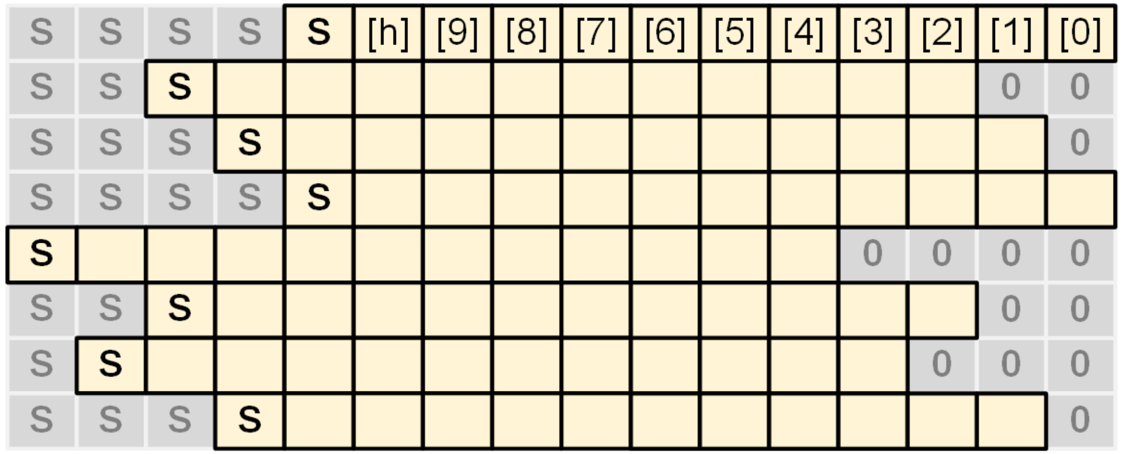
- Intensive CIM core : Efficient FP-to-INT transfer
- Sparse digital core : Efficient flexible intensive/sparse operations



Need to support flexible intensive/sparse processing

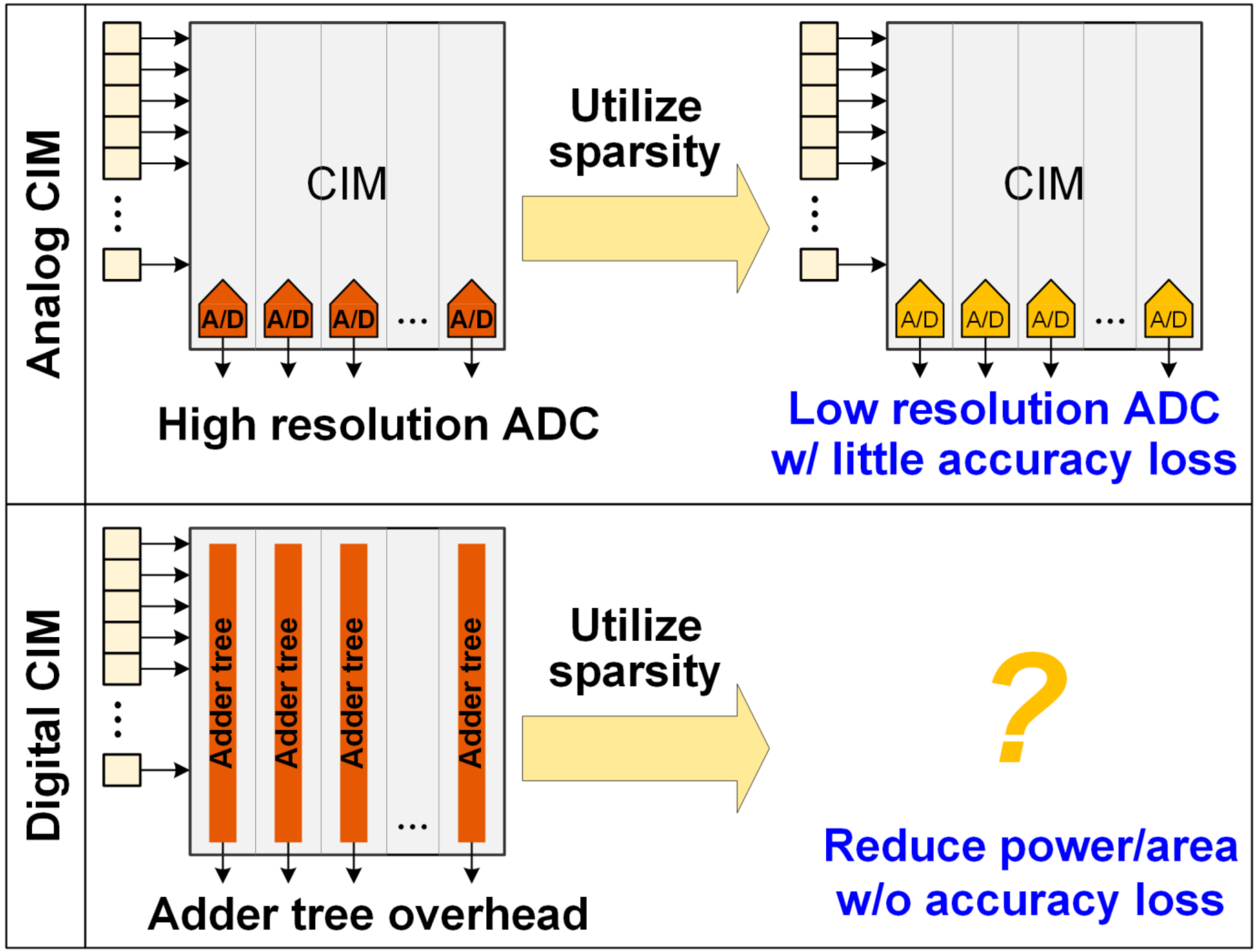
Challenge 3: Sparsity in Digital CIM

- Structural (block-wise) sparsity
  – skip zero blocks
- Random cell/bit-wise sparsity
  – ①Natural / pruning ②FP alignment
  – Analog: Reduce current / resolution
  – Digital: Reduce toggle rate and ?



S	S	S	S	S	[h]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
S	S	S												0	0
S	S	S	S												0
S	S	S	S	S											
S												0	0	0	0
S	S	S												0	0
S	S												0	0	0
S	S	S	S												0

FP alignment → more random sparsity



Outline

- **Motivation & Challenges**
- **Proposed FP CIM Processor**
  - Efficient FP-to-INT CIM workflow for intensive FP operations
  - Flexible sparse digital core for sparse FP operations
  - Low-MACV CIM macro for random sparsity
- **Measurement Results**
- **Conclusion**

Proposed FP CIM Architecture

- Intensive-CIM sparse-digital architecture
  - – Workload breakdown

$$\begin{aligned} W * A &= (W_{intensive} + W_{sparse}) * (A_{intensive} + A_{sparse}) \\ &= W_{intensive} * A_{intensive} + \left(W_{intensive} + W_{sparse}\right) * A_{sparse} + W_{sparse} * A_{intensive} \\ &= \boxed{W_{intensive} * A_{intensive}} + \boxed{W_{all} * A_{sparse}} + \boxed{W_{sparse} * A_{intensive}} \end{aligned}$$

Heavy computation

Intensive CIM: High efficiency

Light sparse computation

Sparse digital: High flexibility

$W_{all}, A_{intensive}$ **: on-chip storage**

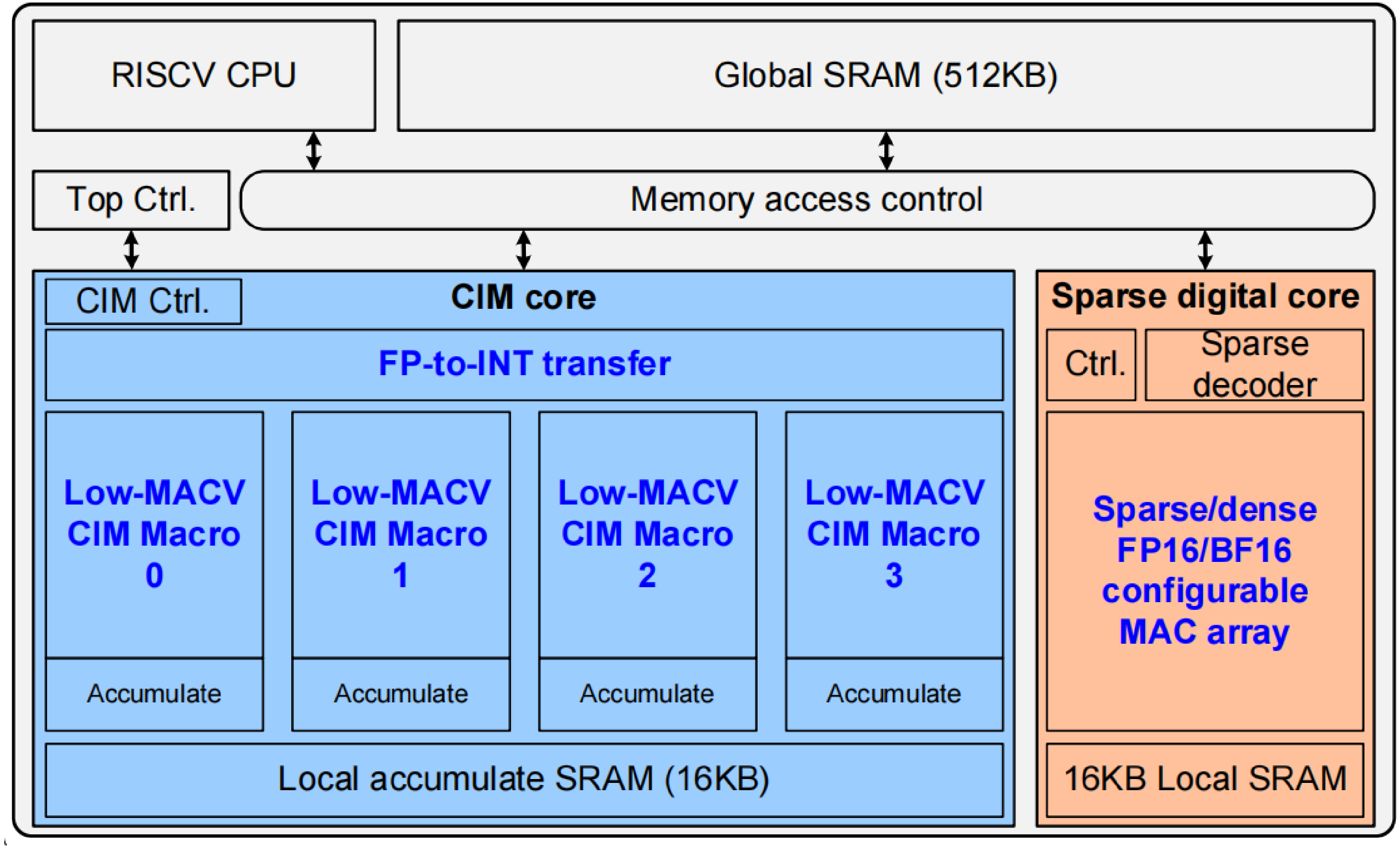
W_{sparse}, A_{sparse} **: on-chip storage, small overhead**

$W_{intensive}$ **: on-chip generated from** W_{all}

Proposed FP CIM Architecture

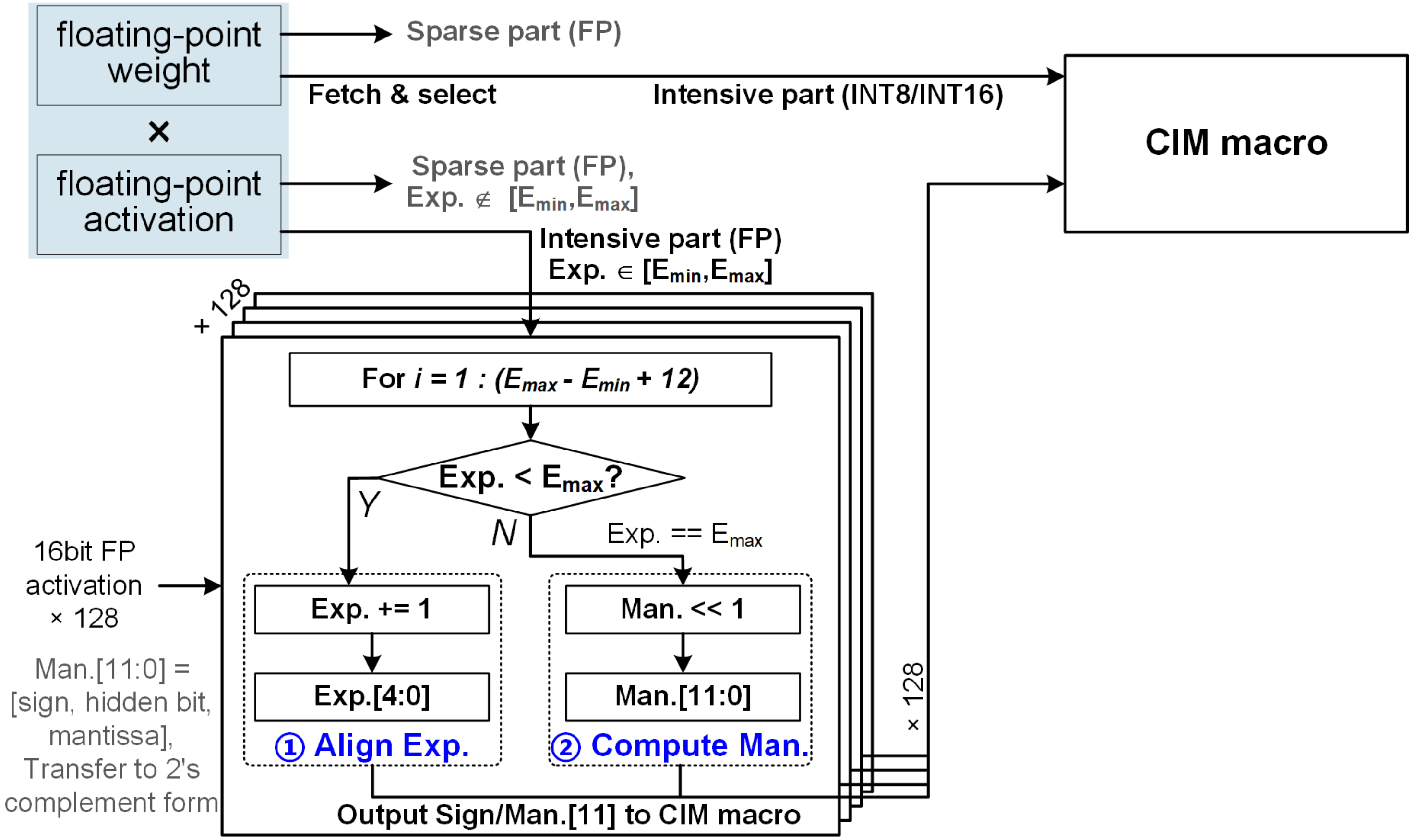
- Intensive-CIM sparse-digital architecture for FP operations
  - – CPU + CIM + digital core
  - – CIM: Intensive FP ops
  - – Digital: Sparse FP ops
  - – Four CIM macros:

    Low-MACV adder tree



FP CIM Core

- FP weight/activation storage
- Fetch the intensive part: weight ($\boldsymbol{W_{intensive}}$), activation (FP in $[E_{min}, E_{max}]$)

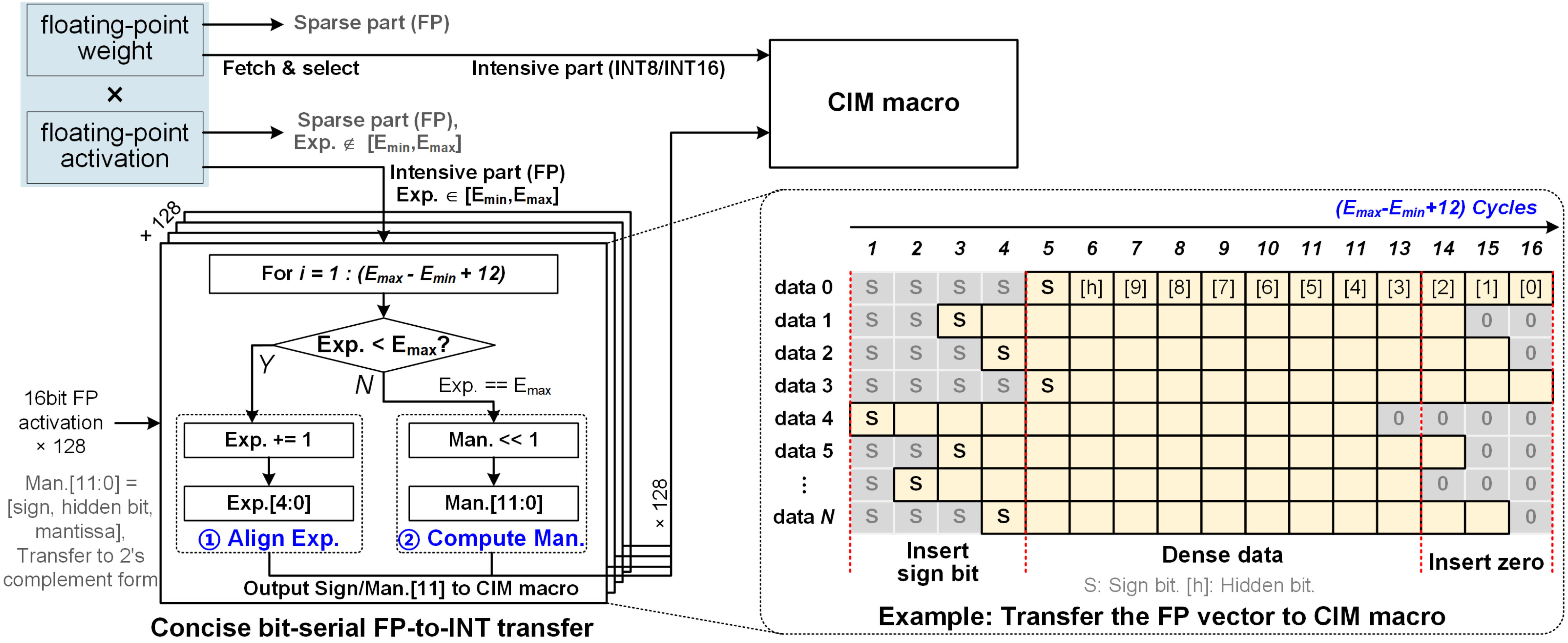


floating-point weight
×
floating-point activation
Sparse part (FP)
Fetch & select
Intensive part (INT8/INT16)
CIM macro
Sparse part (FP),
Exp. ∉ [E_{min},E_{max}]
Intensive part (FP)
Exp. ∈ [E_{min},E_{max}]
+ 128
For $i = 1 : (E_{max} - E_{min} + 12)$
Exp. < E_{max}?
Y
N
Exp. == E_{max}
16bit FP activation × 128
Man.[11:0] = [sign, hidden bit, mantissa], Transfer to 2's complement form
Exp. += 1
Exp.[4:0]
① Align Exp.
Man. << 1
Man.[11:0]
② Compute Man.
× 128
Output Sign/Man.[11] to CIM macro

Concise bit-serial FP-to-INT transfer

FP CIM Core

- Bit-serial FP-to-INT transfer
  - ① Exponent alignment ② Mantissa shift


floating-point weight
×
floating-point activation
Sparse part (FP)
Fetch & select
Intensive part (INT8/INT16)
CIM macro
Sparse part (FP), Exp. ∉ [E_{min},E_{max}]
Intensive part (FP) Exp. ∈ [E_{min},E_{max}]
+ 128
For $i = 1 : (E_{max} - E_{min} + 12)$
Exp. < E_{max}?
Y
N
Exp. == E_{max}
16bit FP activation × 128
Man.[11:0] = [sign, hidden bit, mantissa], Transfer to 2's complement form
Exp. += 1
Exp.[4:0]
① Align Exp.
Man. << 1
Man.[11:0]
② Compute Man.
× 128
Output Sign/Man.[11] to CIM macro
(E_{max}-E_{min}+12) Cycles
Insert sign bit
Dense data
Insert zero

	1	2	3	4	5	6	7	8	9	10	11	11	13	14	15	16
data 0	S	S	S	S	S	[h]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
data 1	S	S	S												0	0
data 2	S	S	S	S												0
data 3	S	S	S	S	S											
data 4	S												0	0	0	0
data 5	S	S	S												0	0
⋮	S	S												0	0	0
data *N*	S	S	S	S												0

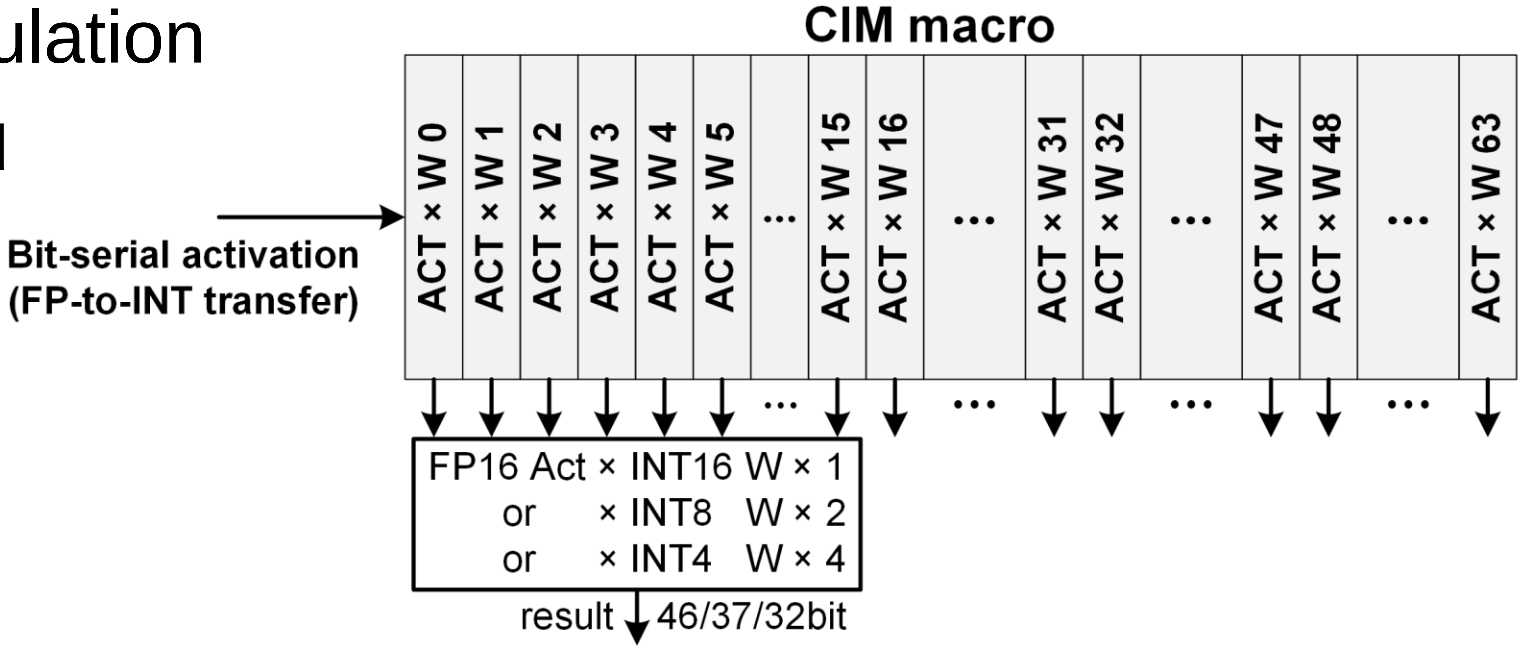
S: Sign bit. [h]: Hidden bit.

Concise bit-serial FP-to-INT transfer

Example: Transfer the FP vector to CIM macro

FP CIM Core

- INT MAC operations in the CIM macro
- Reconfigurable INT accumulation
  - From bit-serial activation and INT4/INT8/INT16 weight
  - Intensive FP weight stored as INT format



FP CIM Core

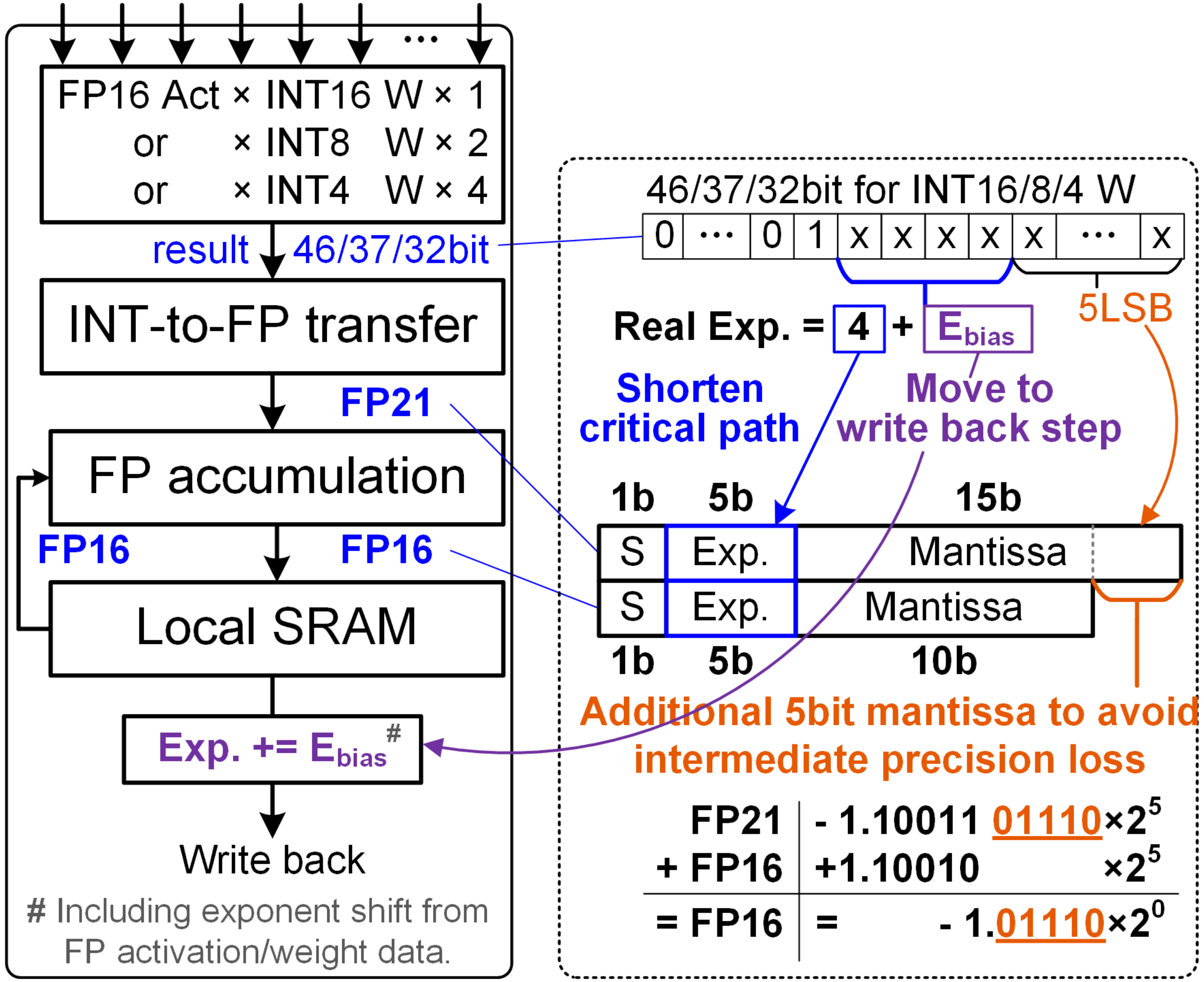
- INT-to-FP accumulation
  - Self-designed 21bit FP format

    Move FP normalization to the write-back step

    Simplify the INT-to-FP transfer and FP accumulation circuits
  - Additional 5bit mantissa

    Avoid intermediate precision loss
  - Write-back step

    Process the activation/weight exp. bias and FP normalization together

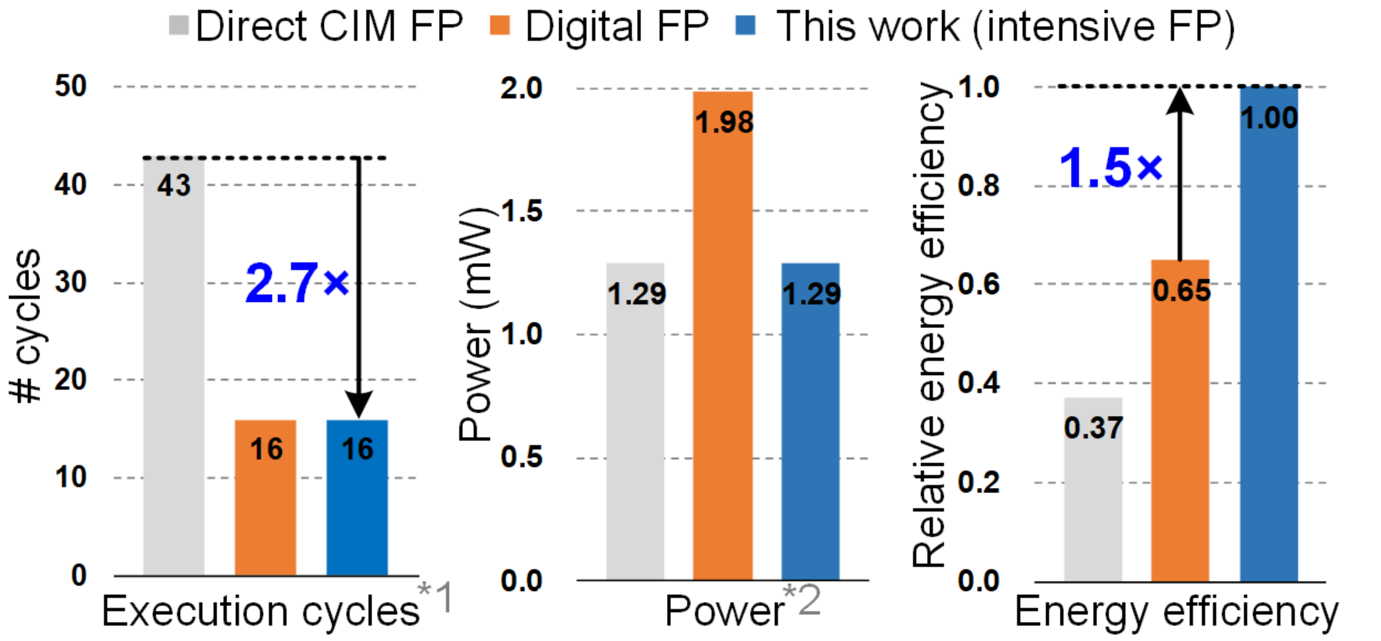


Including exponent shift from FP activation/weight data.

INT-to-FP accumulation & data format

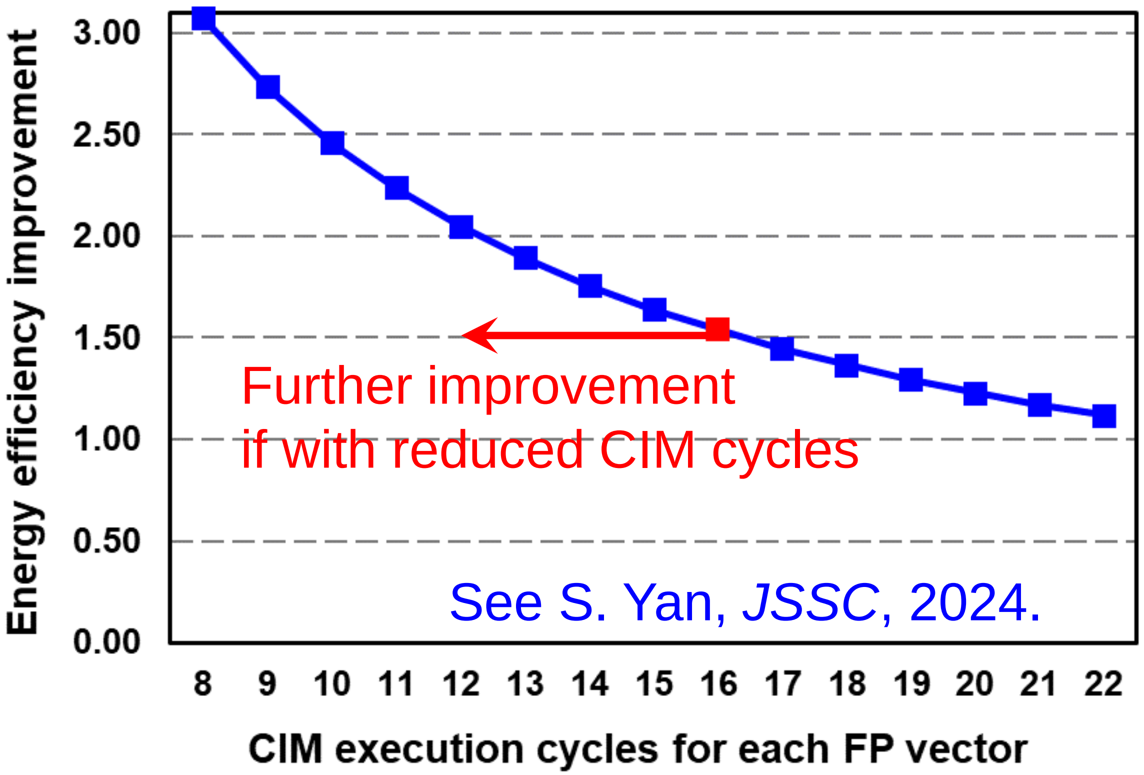
FP CIM Core

- Improvement of the FP CIM core
  - Avoid many execution cycles
  - Further improvement if FP alignment bits are reduced (block-wise/training)



Comparison with direct CIM FP alignment & digital FP unit

*1: Assume Emax-Emin=4. *2: Including power of the input buffer, CIM macros, and accumulation units, at the same performance.

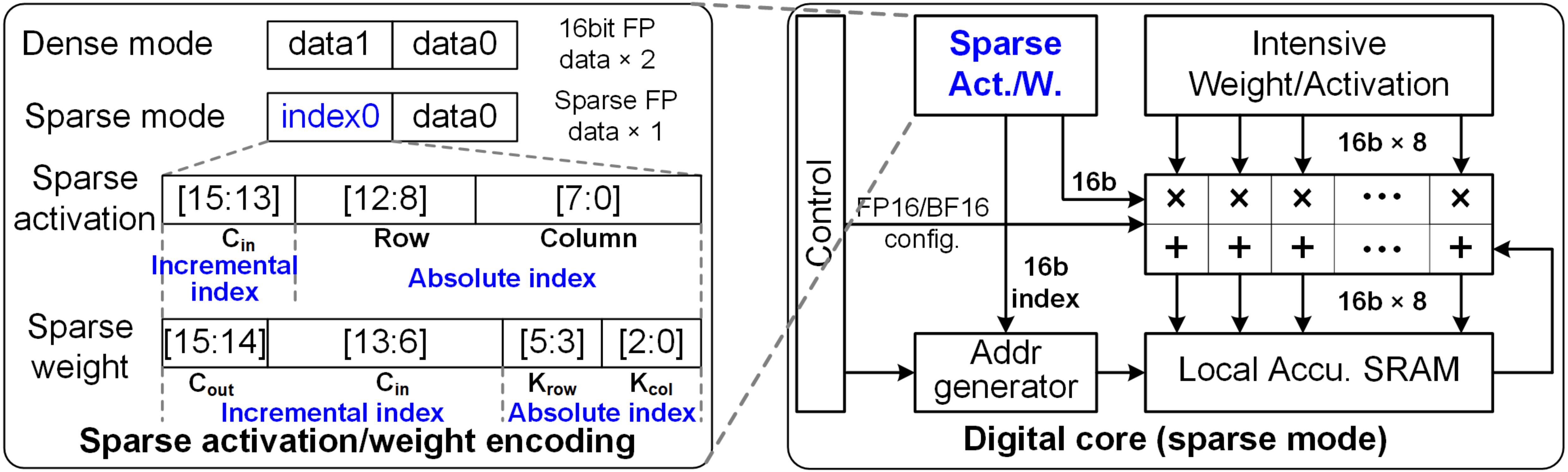


Further improvement if with reduced CIM cycles

See S. Yan, *JSSC*, 2024.

Sparse Digital Core

- Support both intensive/sparse SIMD execution
- Sparse encoding with hybrid incremental & absolute indexes
  - Channel (C_{in} /C_{out}): Incremental index
  - Row/column: Absolute index

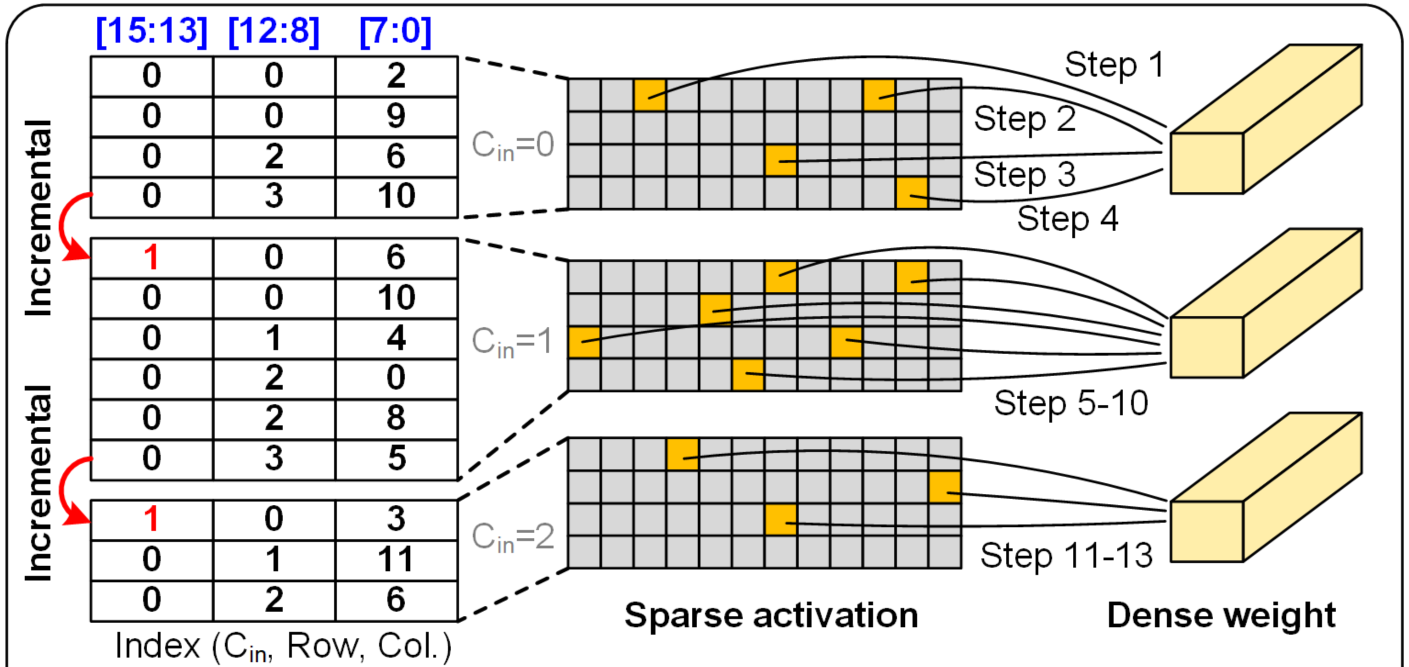


Sparse activation/weight encoding

Digital core (sparse mode)

Sparse Digital Core

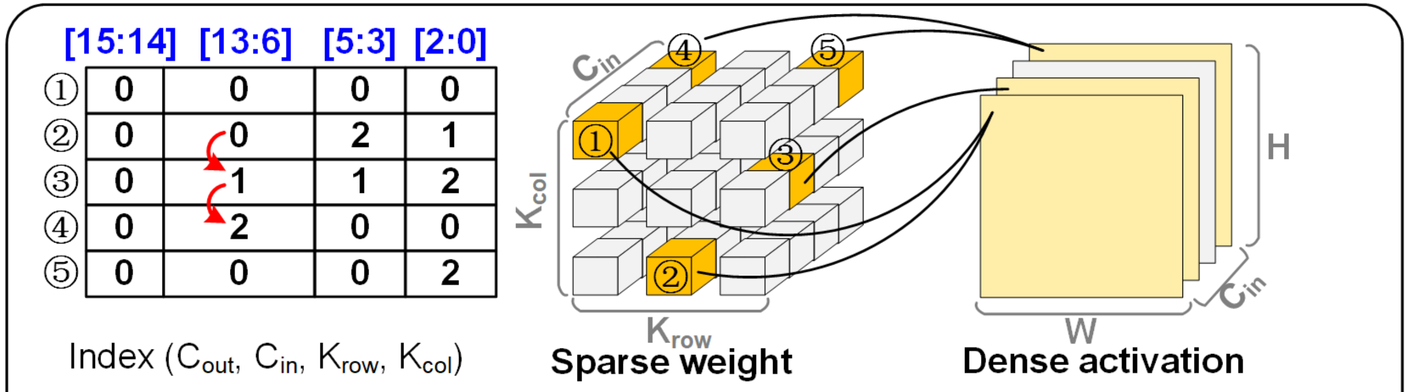
- Sparse encoding examples
  - $A_{sparse} * W_{all}$
  - $W_{sparse} * A_{intensive}$
- Absolute row/column indexes
  - Reduce the decoder complexity
- Incremental channel indexes
  - Reduce index bits
  - 16 bits are enough for sparse activation & weight



[15:13]	[12:8]	[7:0]
0	0	2
0	0	9
0	2	6
0	3	10
1	0	6
0	0	10
0	1	4
0	2	0
0	2	8
0	3	5
1	0	3
0	1	11
0	2	6

Index (C_{in}, Row, Col.)

Sparse activation encoding example



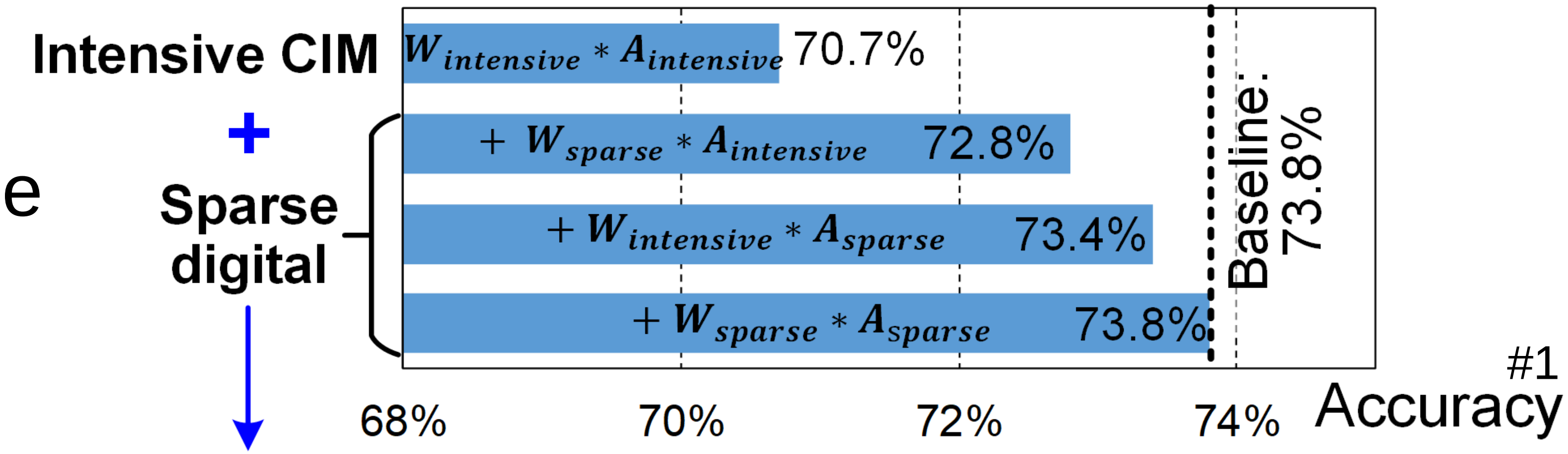
	[15:14]	[13:6]	[5:3]	[2:0]
①	0	0	0	0
②	0	0	2	1
③	0	1	1	2
④	0	2	0	0
⑤	0	0	0	2

Index (C_{out}, C_{in}, K_{row}, K_{col})

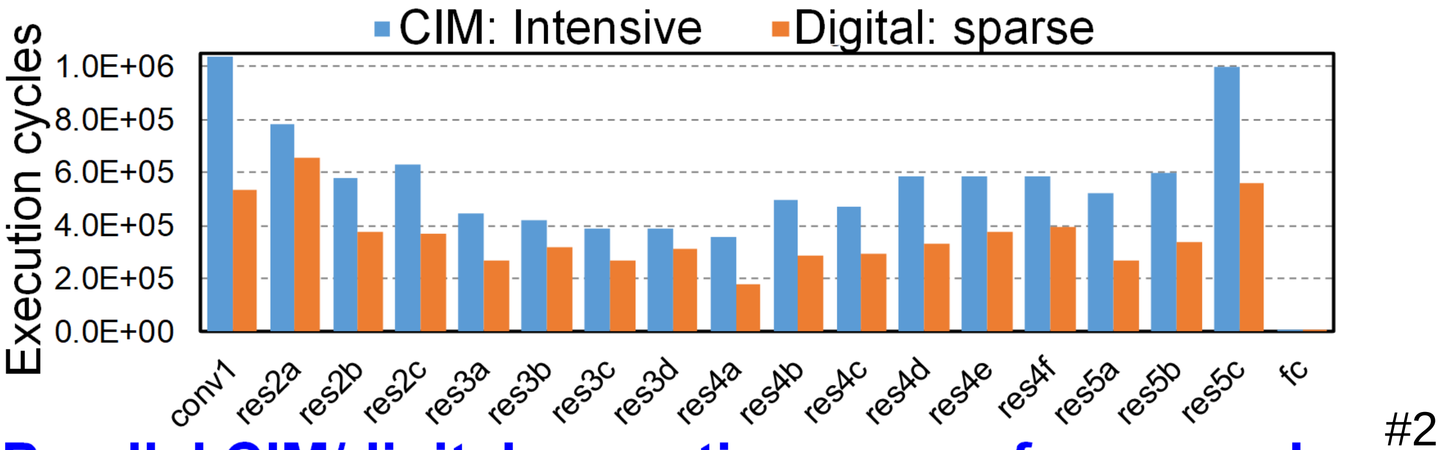
Sparse weight encoding example

Sparse Digital Core

- No accuracy loss compared with FP baseline
- No performance loss by parallel CIM/digital execution



Achieve FP16 accuracy w/ only 4.59% additional energy

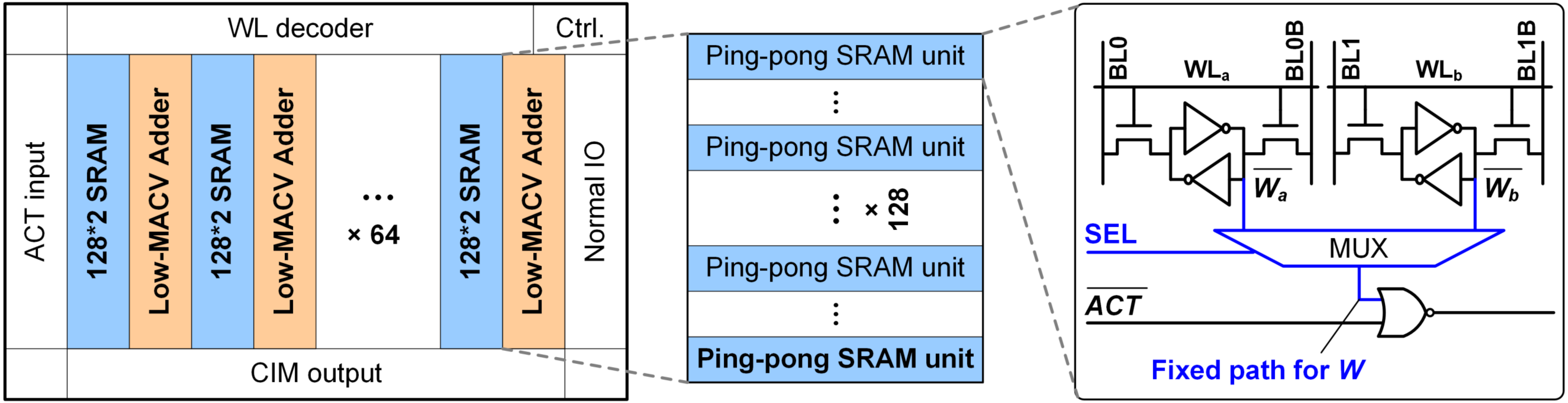


Parallel CIM/digital execution, no performance loss

#1: On ImageNet, ResNet50 (3.21x block-wise sparsity). Sparse activation ratio: 5%, sparse weight ratio: 0.2%. #2: On ImageNet, ResNet50.

Low-MACV CIM Macro

- Ping-pong SRAM unit
  - Fixed path from the stored weight to NOR gate if *SEL* is fixed
  - Support simultaneous CIM and weight update

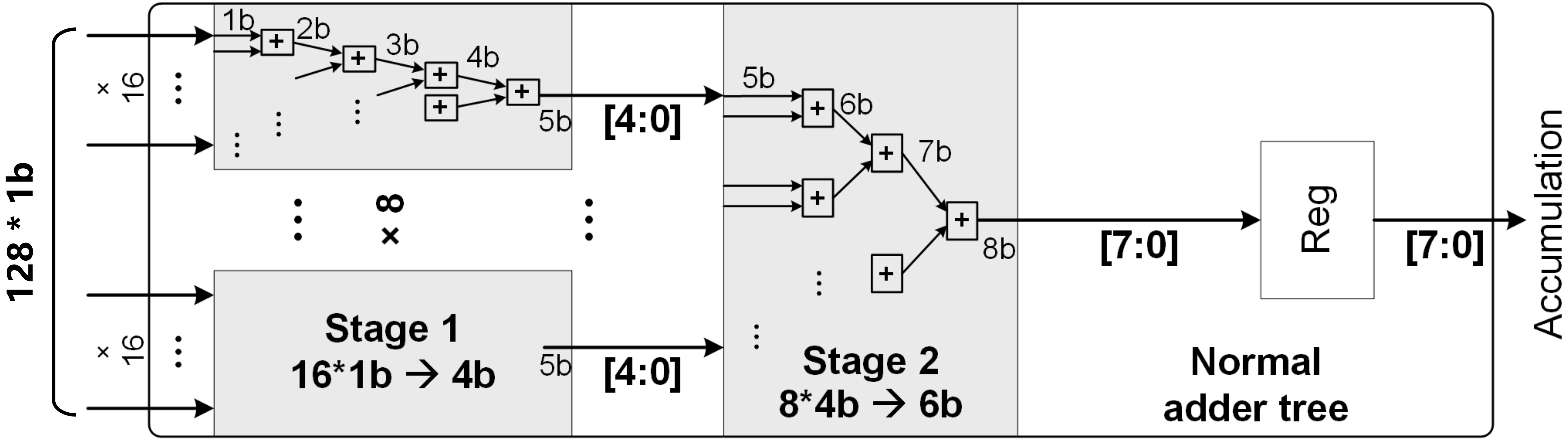


Low-MACV CIM macro **Ping-pong SRAM unit**

MACV: multiply-accumulation value.

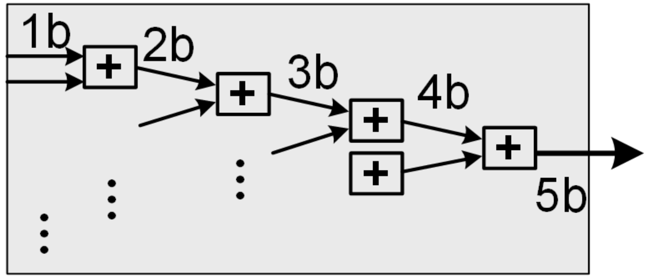
Low-MACV CIM Macro

- Normal digital adder tree
  - Requires full-precision multiply-accumulation values (MACV)
  - 8bit result for 128×1b input

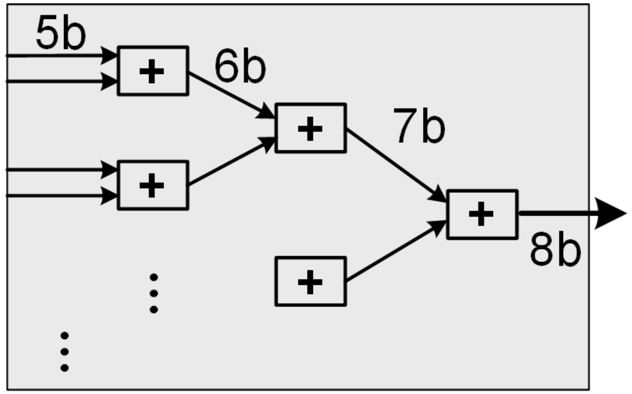


Low-MACV CIM Macro

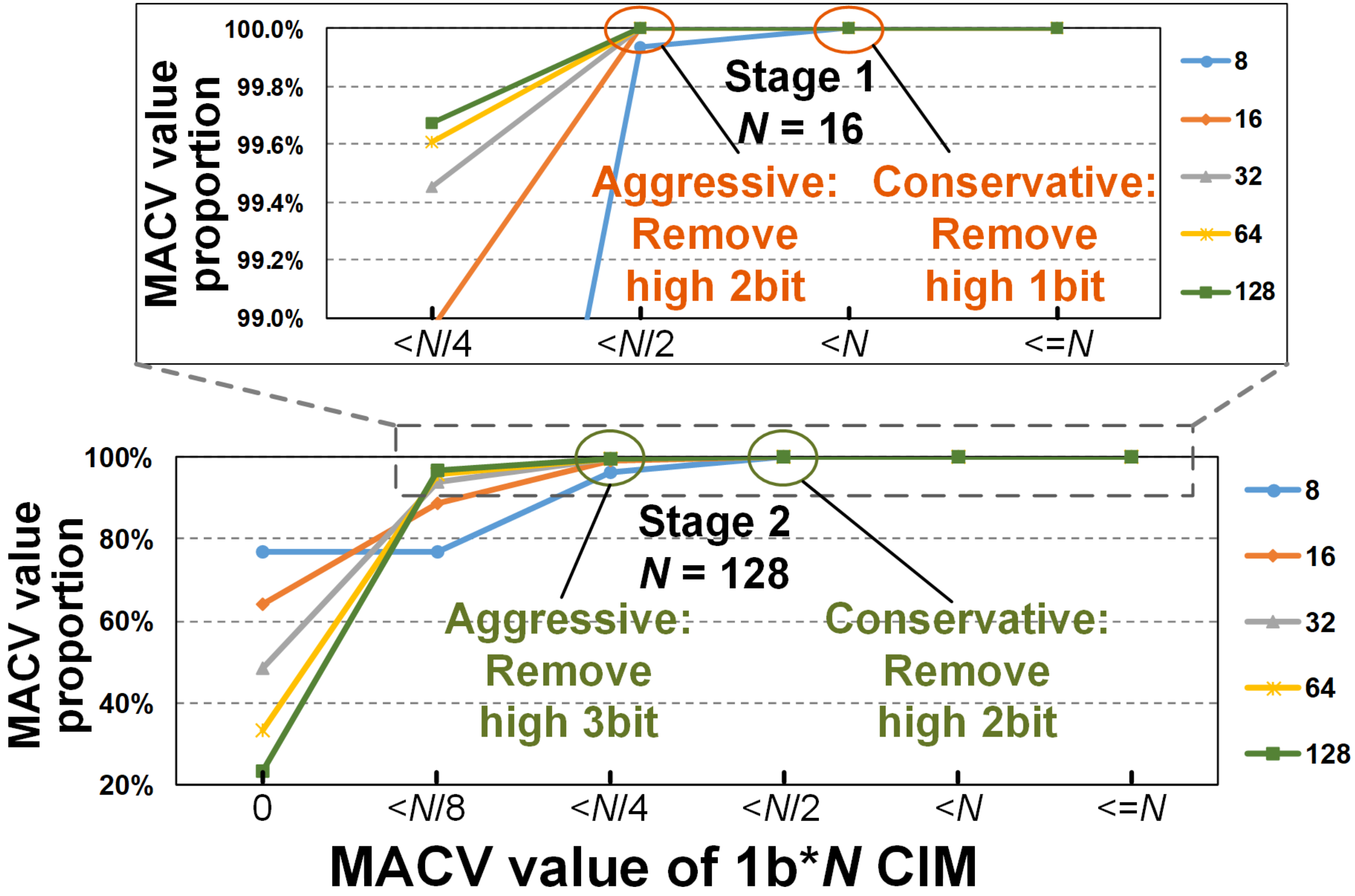
- Large CIM MACV results barely not arise
  - Remove high bit positions



Stage 1: 16*1b → 4b



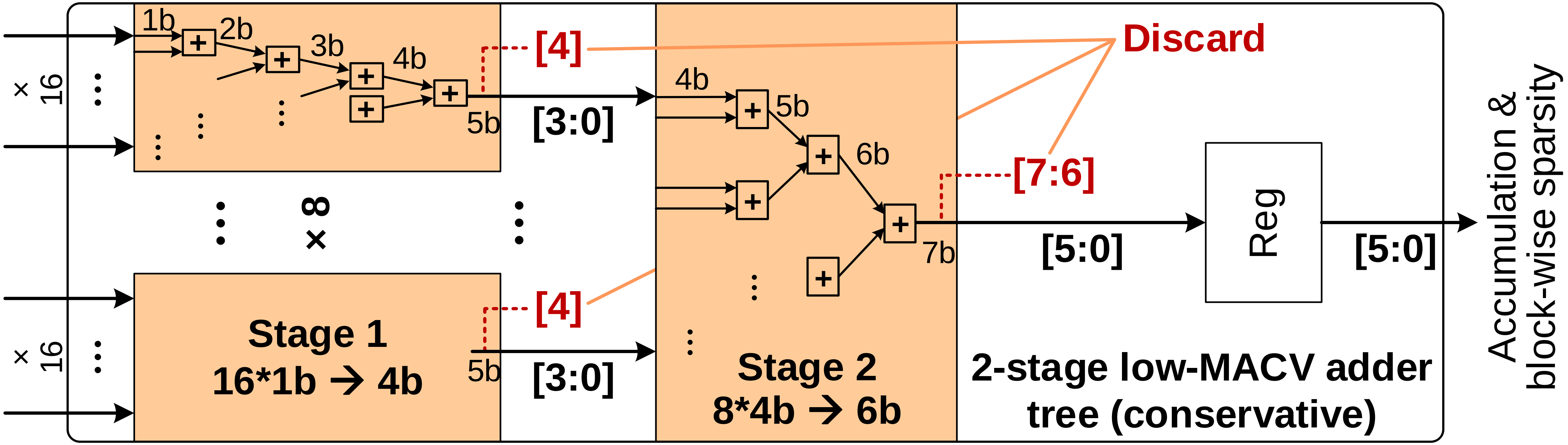
Stage 2: 8*5b → 8b



On ResNet50, ImageNet w/o sparsity training.

Low-MACV CIM Macro

- Large CIM MACV results barely not arise
  - Two-stage low-MACV adder tree (conservative mode)
  - Omit computation for the high-bit positions in the MAC result



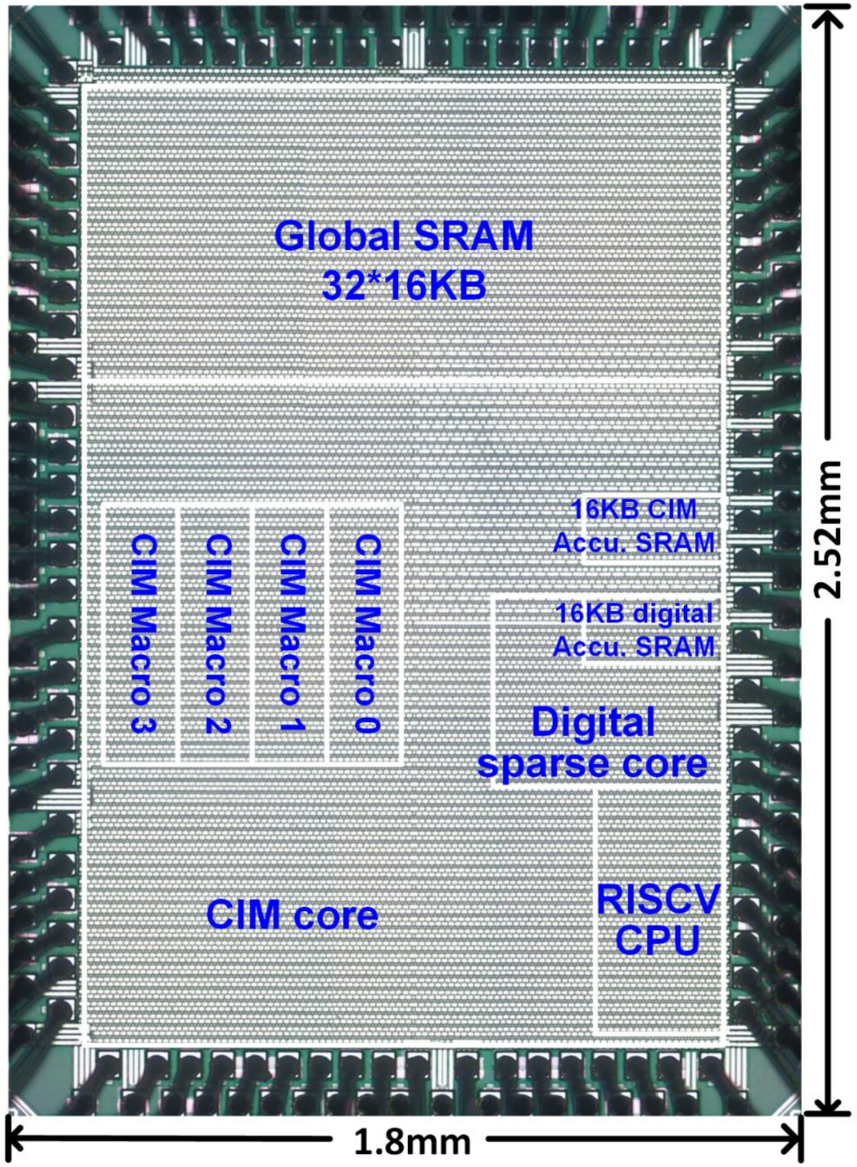
9.0% and 13.9% power reduction in conservative/aggressive modes w/o accuracy loss

Outline

- **Motivation & Challenges**
- **Proposed FP CIM Processor**
  - Efficient FP-to-INT CIM workflow for intensive FP operations
  - Flexible sparse digital core for sparse FP operations
  - Low-MACV CIM macro for random sparsity
- **Measurement Results**
- **Conclusion**

Chip Photograph

Technology	28nm
Chip area	2.52mm×1.8mm
CIM macro area	0.56mm×0.12mm ×4
Weight prec.	INT4/8, FP16/BF16
Activation prec.	INT4/8, FP16/BF16
Voltage (CIM)	0.397 - 0.90V
Voltage (digital)	0.469 - 0.90V
Frequency	10 - 400MHz
CIM power *1	0.13 - 13.7mW
System power *1	0.87 - 74.9mW
Performance *2	1.64 - 9.63TOPS (INT4/INT4)
CIM macro energy efficiency *2	275 - 1615TOPS/W (INT4/INT4) 68.7 - 403TOPS/W (INT8/INT8) 17.2 - 91.3TOPS/W (FP16/FP16)
System energy efficiency *2	51 - 300TOPS/W (INT4/INT4) 12.8 - 75.0TOPS/W (INT8/INT8) 3.2 - 16.9TOPS/W (FP16/FP16)



*1: At 10-400MHz w/ 0.469-0.79V (digital) and 0.397-0.78V (CIM).

*2: from dense to average block-wise sparsity on test models. Assume average FP cycles = 16 for the dense situation.

Results on Different NN Models

Model	VGG16 inference		ResNet50 inference		ConvNeXt-T inference	ResNet50 training
Dataset	Cifar-10		ImageNet			
Activation	**4**	**FP16**	**8**	**FP16**	**FP16**	**FP16**
Weight	**4**	**FP16**	**8**	**FP16**	**FP16**	**FP16**
Average FP cycles (E_{max}-E_{min}+12)	**--**	**17.93**	**--**	**18.53**	**21.52**	**19.60**
Baseline accuracy	94.20%		80.86%		82.10%	80.86%
Chip accuracy (1)	90.03%	91.57%	76.92%	77.85%	77.98%	80.86%
Execution time	0.523ms	9.35ms	97.7ms	413ms	263ms	2.00s
CIM macro energy effi. (TOPS/W)	**1615**	**91.3**	**106**	**22.7**	**43.9**	**14.1** (2)
System energy effi. (TOPS/W)	**300**	**16.9**	**19.7**	**4.22**	**8.17**	**2.61** (2)

At the best system energy efficiency point. (1) Pre-train with block-wise sparsity for the inference tasks. Sparse acceleration of VGG16 / Resnet50 / ConvNeXt-T: 5.88x / 1.54x / 3.87x. (2) Dense training w/o block-wise sparsity.

Comparison with State-of-the-Art

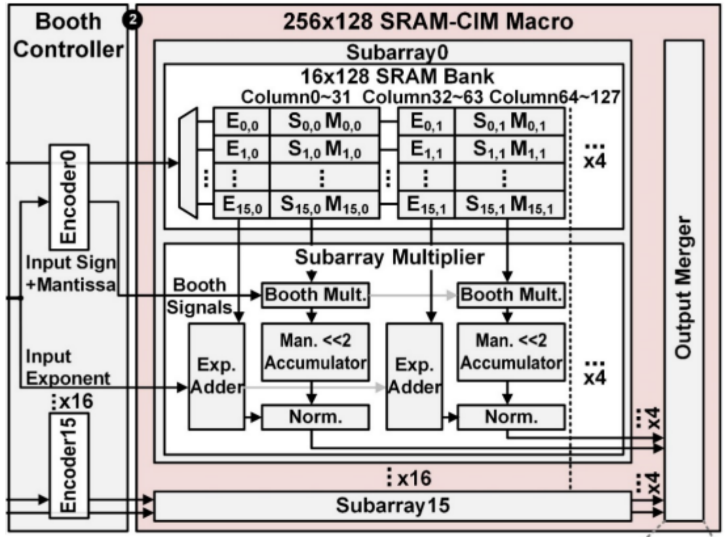
	ISSCC22 [1]	ISSCC22 [2]	ISSCC22 [3]	VLSI21 [4]	ISSCC22 [5]	**This work**
Technology	28nm	5nm	22nm	28nm	28nm	**28nm**
Area (mm^2)	0.033	0.013	10.2	5.8	6.7	**4.54**
Activation	1b	INT4/8	7b (Analog) 2/4/8b (Digital)	BF16	INT8/16, BF16/FP32	**INT4/8, FP16/BF16**
Weight	1b	INT4/8	Ternary (Analog) 2/4/8b (Digital)	BF16	INT8/16, BF16/FP32	**INT4/8, FP16/BF16**
Sparsity	Approximate adder	N/A	N/A	Activation sparsity	N/A	**Low-MACV adder + block-wise**
FP support	N/A	N/A	N/A	Exponent CIM	Pure CIM (long-tail FP cost unmentioned)	**Intensive CIM + sparse digital**
(1) **Macro energy effi. (TOPS/W)**	128 (INT4 eq.)	254 (INT4)	600 (7b/Ternary)	(2) 13.7 (BF16)	231 (INT4) (3) 57.8 (INT8) 46.2 (BF16)	**275 - 1615 (INT4)** (4) **68.7 - 403 (INT8)** **17.2 - 91.3 (FP16)**

(1) Maximum reported value in [1-5] for fair comparison. One operation (OP) represents one multiplication or addition. (2) Including external control and SIMD modules. (3) The processing of the long-tail floating-point data is not mentioned. (4) From dense models to average of test sparse NN models.

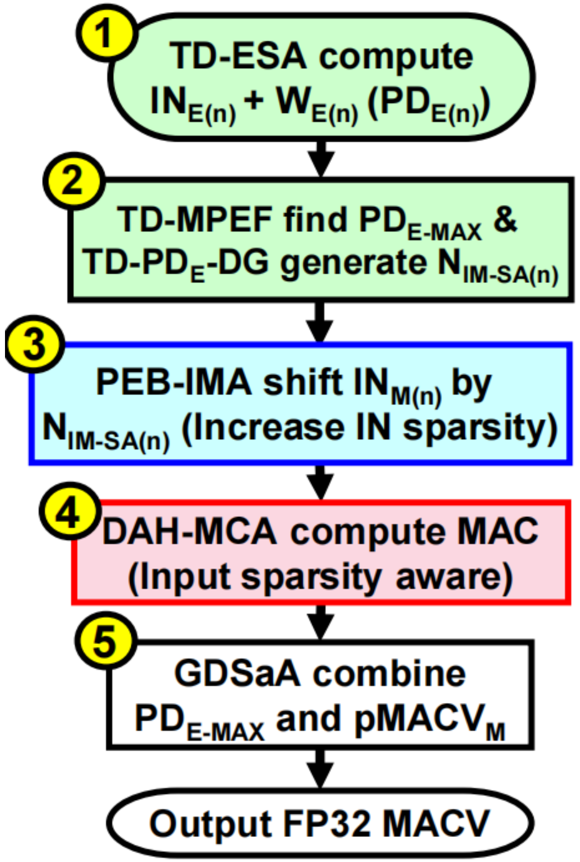
Outline

- **Motivation & Challenges**
- **Proposed FP CIM Processor**
  - Efficient FP-to-INT CIM workflow for intensive FP operations
  - Flexible sparse digital core for sparse FP operations
  - Low-MACV CIM macro for random sparsity
- **Measurement Results**
- **Further Discussion & Conclusion**

Recent Progress in ISSCC



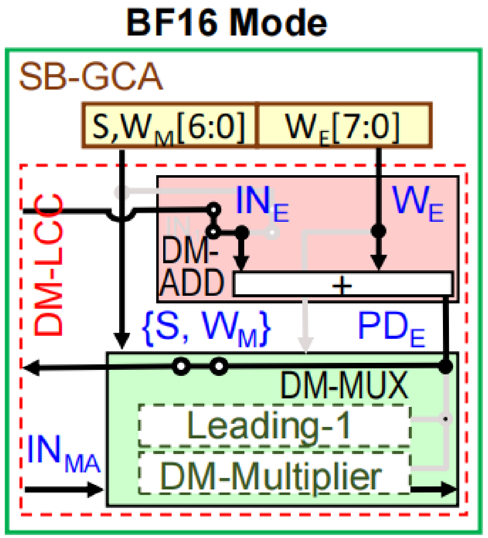
Exp. computation inside CIM (digital)



1. TD-ESA compute $IN_{E(n)} + W_{E(n)}$ $(PD_{E(n)})$
2. TD-MPEF find $PD_{E\text{-}MAX}$ & TD-PD_E-DG generate $N_{IM\text{-}SA(n)}$
3. PEB-IMA shift $IN_{M(n)}$ by $N_{IM\text{-}SA(n)}$ (Increase IN sparsity)
4. DAH-MCA compute MAC (Input sparsity aware)
5. GDSaA combine $PD_{E\text{-}MAX}$ and $pMACV_M$

Output FP32 MACV

Analog + digital Hybrid

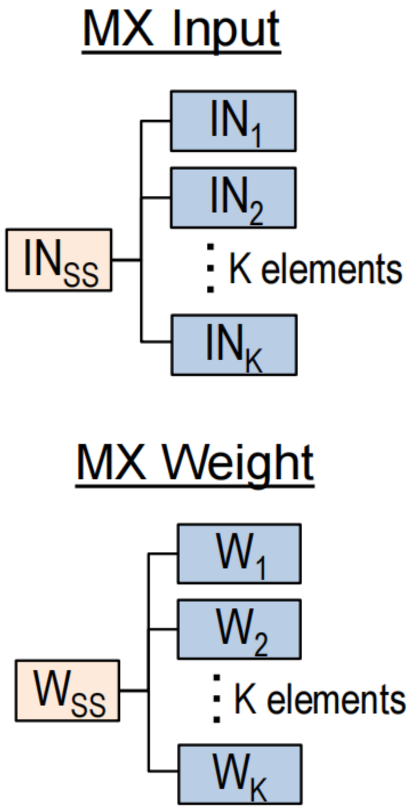


SB-GCA:
- INT8 ➔ $\{W_0[7:0], W_1[7:0]\}$
- BF16 ➔ $\{S, W_M[6:0], W_E[7:0]\}$

DM-ADD:
- INT8 ➔$(8bW_0 + 8bW_1)$
- BF16 ➔ $(8bIN_E + 8bW_E)$

DM-MUX:
- INT8 ➔ $(8bIN_0 \times 8bW_0) + (8bIN_1 \times 8bW_1)$
- BF16 ➔ $(8bIN_M \times 8bW_M)$

FP / INT reuse



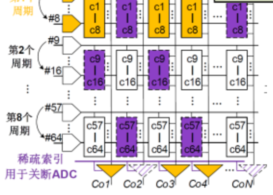
Grouped Exp.

Compatible to Our intensive-CIM sparse-digital solution

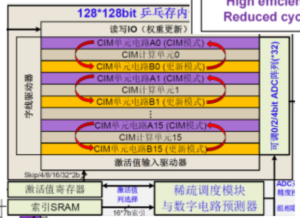
Further Discussion

- One observation (Might not be totally accurate):
  - Most solutions are actually using INT CIM to realize Floating-point CIM
  - Even with accurate floating-point CIM circuits:
  - **Not identical to the baseline results (e.g. IEEE754 FP standards)**
  - Reasons: Bit truncation, integral vector addition before normalization
  - Require further research on the FP CIM formats/standards for result evaluation

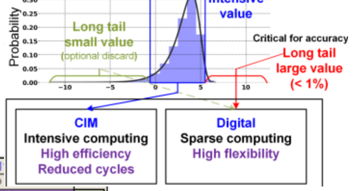
Our Roadmap: From Effi. Circuits to Effi. System



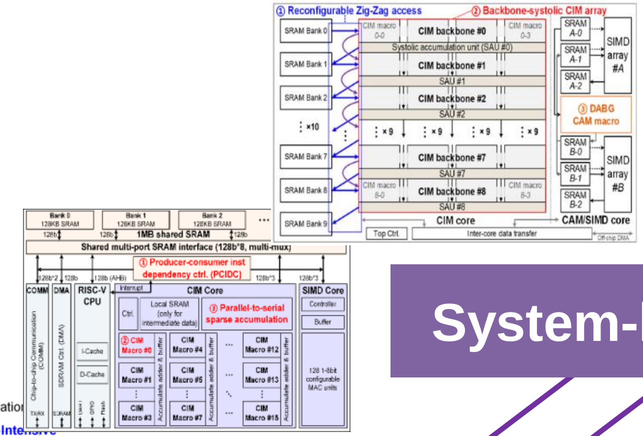
Efficient CIM circuits



Sparsity + CIM



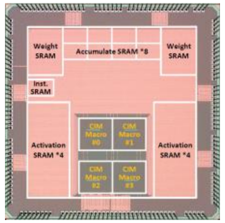
Floating-point

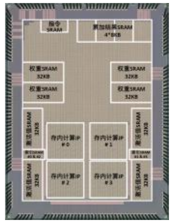


Hetero. SoC

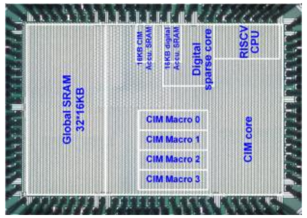
System-level Opt.

Multi-device/chiplets

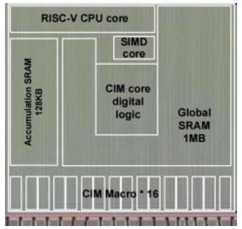




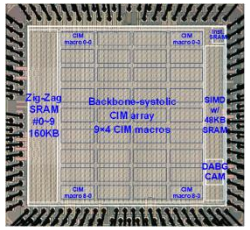
ISSCC 2020/2021
JSSC 2022/2023



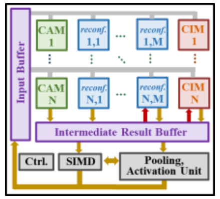
ISSCC 2023
JSSC 2024



VLSI 2023



VLSI 2024



DAC 2024
SCIS 2024

High metrics

⇧

Functionality

⇧

System Effi.

⇧

Scalability

Conclusion

❑ **Energy-Efficient INT/FP CIM Processor (Intensive CIM + Sparse Digital)**

- ➢ **Efficient FP-to-INT CIM workflow**

  Reduced FP CIM execution cycles

- ➢ **Flexible sparse digital core**

  Sparse activation/weight formats, no performance loss

- ➢ **Low-MACV CIM macro**

  More random sparsity in FP CIM alignment, no accuracy loss

An INT/FP CIM processor with intensive-CIM sparse-digital architecture to achieve 16.9TOPS/W@FP16 and 300TOPS/W@INT4 system energy efficiency

Thanks!

yuejinshan@ime.ac.cn