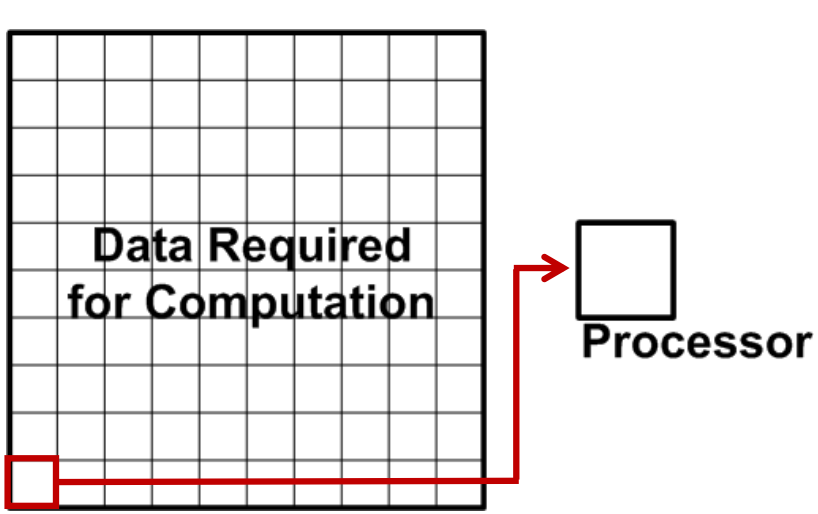
Advances and Prospects for In-memory Computing

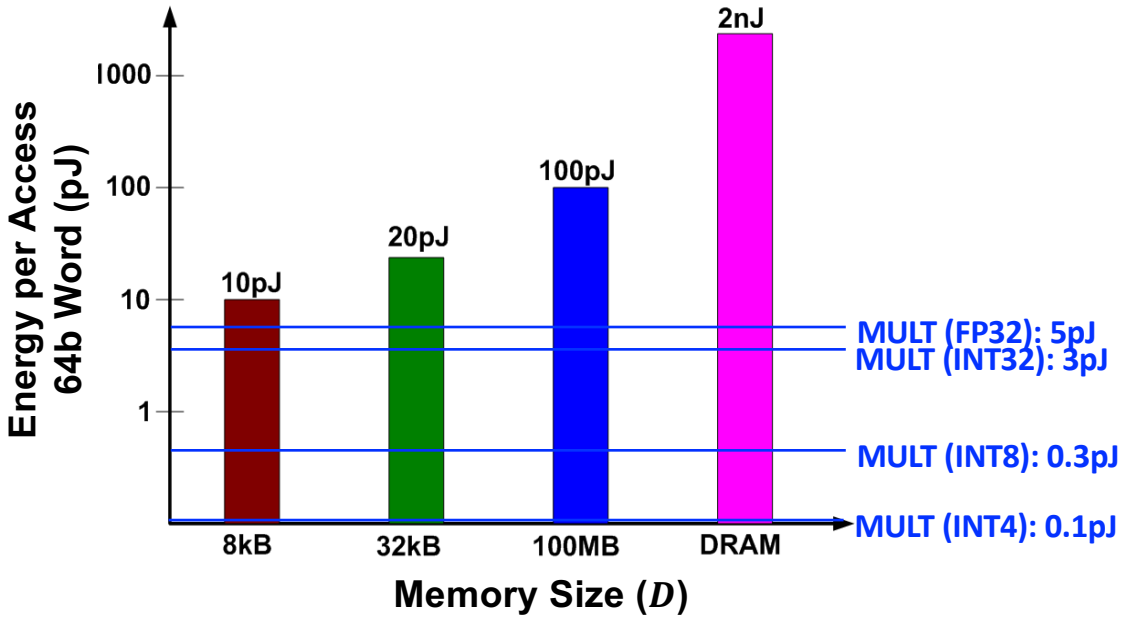
Naveen Verma (nverma@princeton.edu),
L.-Y. Chen, P. Deaville, H. Jia, J. Lee, M. Ozatay, R. Pathak,
Y. Tang, H. Valavi, B. Zhang, J. Zhang

Dec. 13, 2019

The memory wall

- **Separating memory from compute fundamentally raises a communication cost**

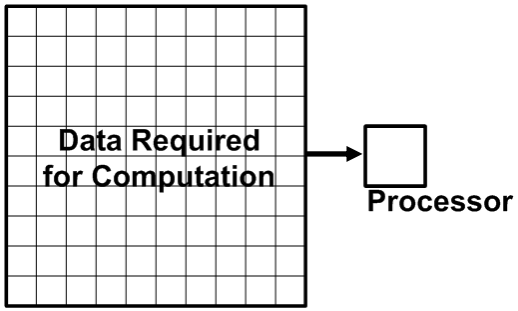


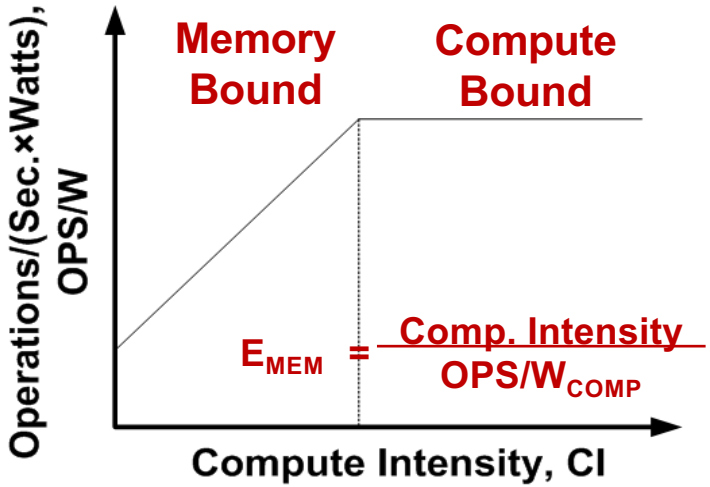


More data → bigger array → larger comm. distance → more comm. energy

So, we should amortize data movement

- **Reuse accessed data for compute operations**

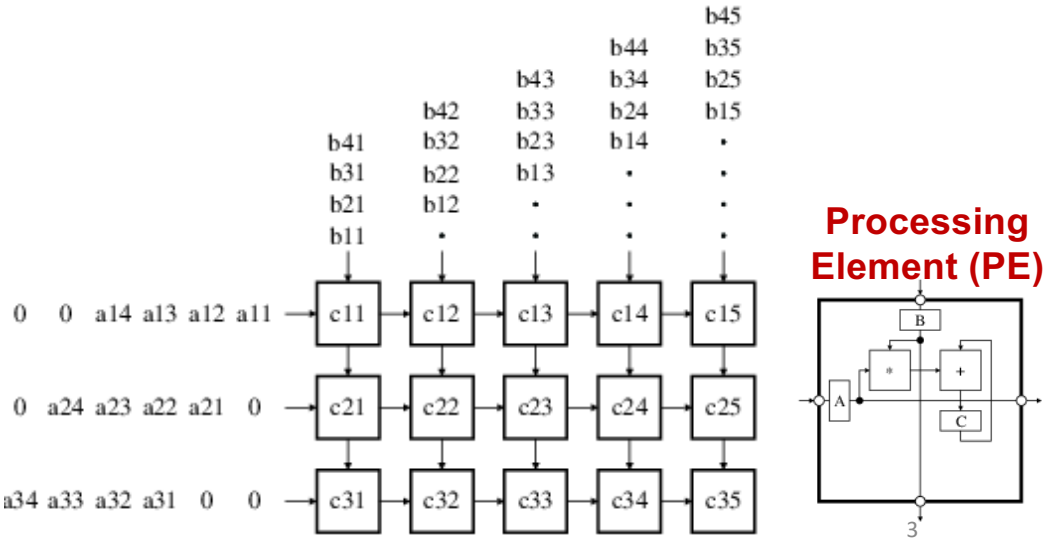




- **Specialized (memory-compute integrated) architectures**

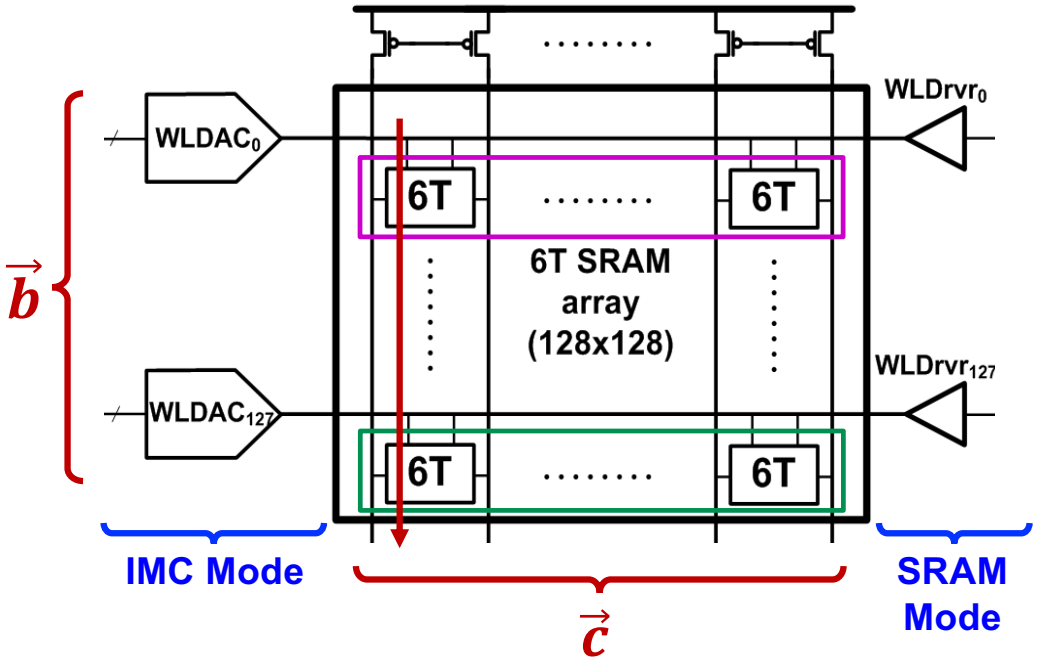
$$\vec{c} = A \times \vec{b}$$

$$\begin{bmatrix} c_1 \\ \vdots \\ c_M \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{M,1} & \cdots & a_{M,N} \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix}$$



In-memory computing (IMC)

$$\vec{c} = A\vec{b} \quad \Rightarrow \quad \begin{bmatrix} c_1 \\ \vdots \\ c_M \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{M,1} & \cdots & a_{M,N} \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix}$$

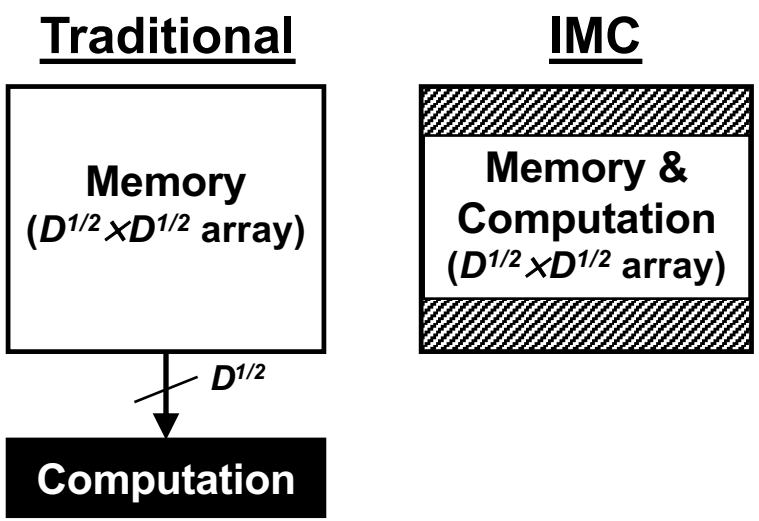


- **In SRAM mode, matrix *A* stored in bit cells <u>row-by-row</u>**
- **In IMC mode, many WLs driven <u>simultaneously</u>**
  → *amortize comm. cost inside array*
- **Can apply to diff. mem. Technologies**
  → *enhanced scalability*
  → *embedded non-volatility*

[J. Zhang, *VLSI*'16][J. Zhang, *JSSC*'17]

The basic tradeoffs

CONSIDER: Accessing D bits of data associated with computation, from array with $\sqrt{D}$ columns × $\sqrt{D}$ rows.



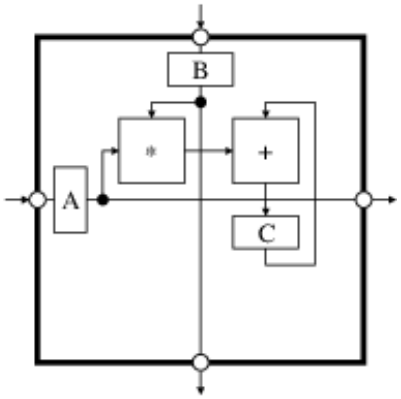
Metric	Traditional	In-memory
Bandwidth	$1/D^{1/2}$	1
Latency	D	1
Energy	$D^{3/2}$	$\sim D$
SNR	1	$\sim 1/D^{1/2}$

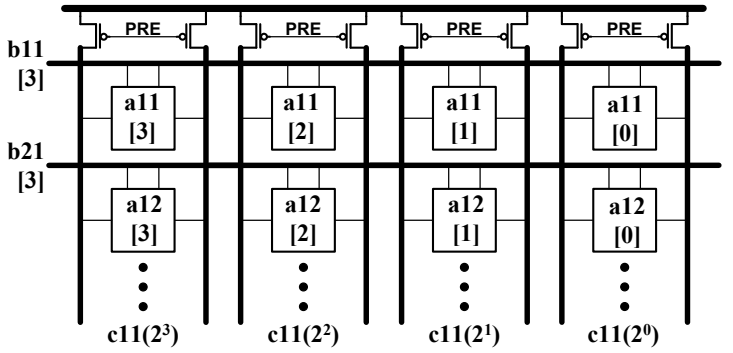
- IMC benefits energy/delay at cost of SNR
- SNR-focused systems design is critical (circuits, architectures, algorithms)

IMC as a spatial architecture

Assume:

- 1k dimensionality
- 4-b multiplies
- 45nm CMOS

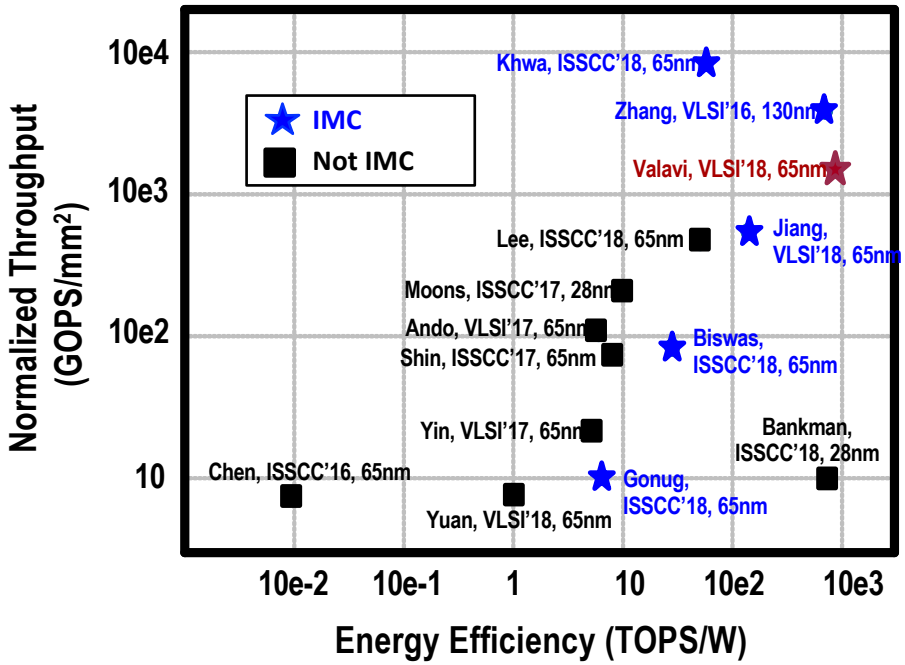


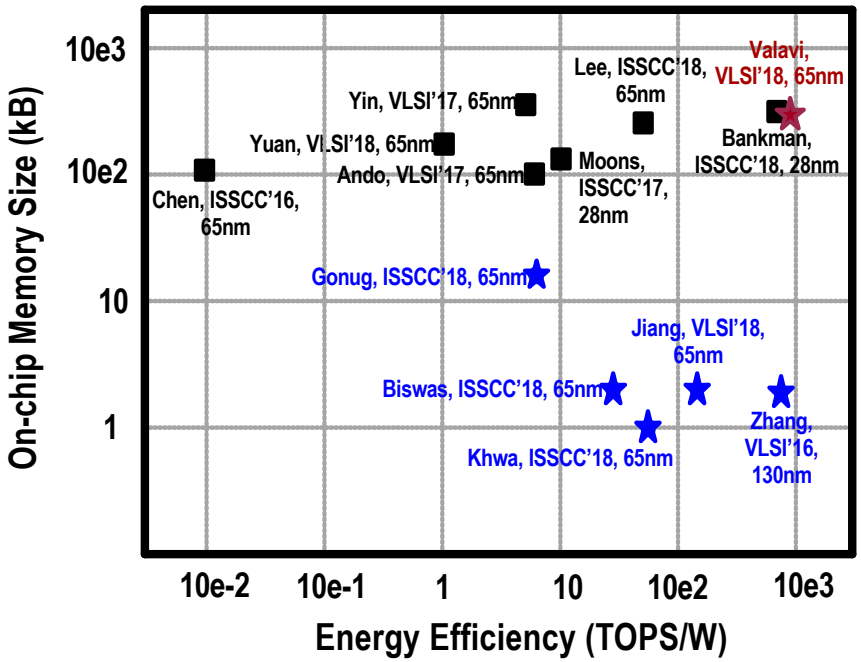


Operation	Digital-PE Energy (fJ)	Bit-cell Energy (fJ)
Storage	250	50
Multiplication	100	
Accumulation	200	
Communication	40	5
Total	**590**	**55**

Where does IMC stand today?

- **Potential for 10× higher efficiency & throughput**
- **Limited scale, robustness, configurability**

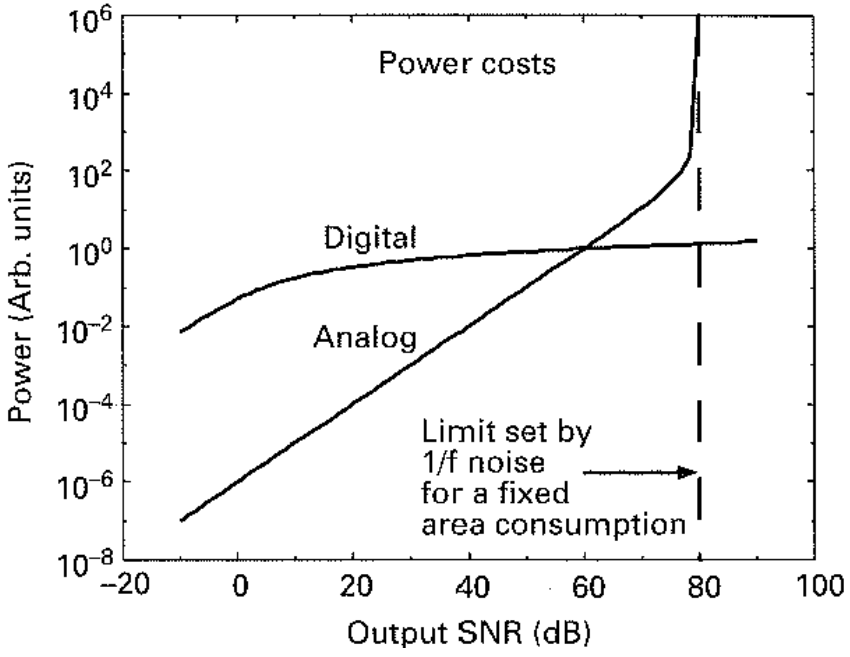




IMC challenge (1): analog computation

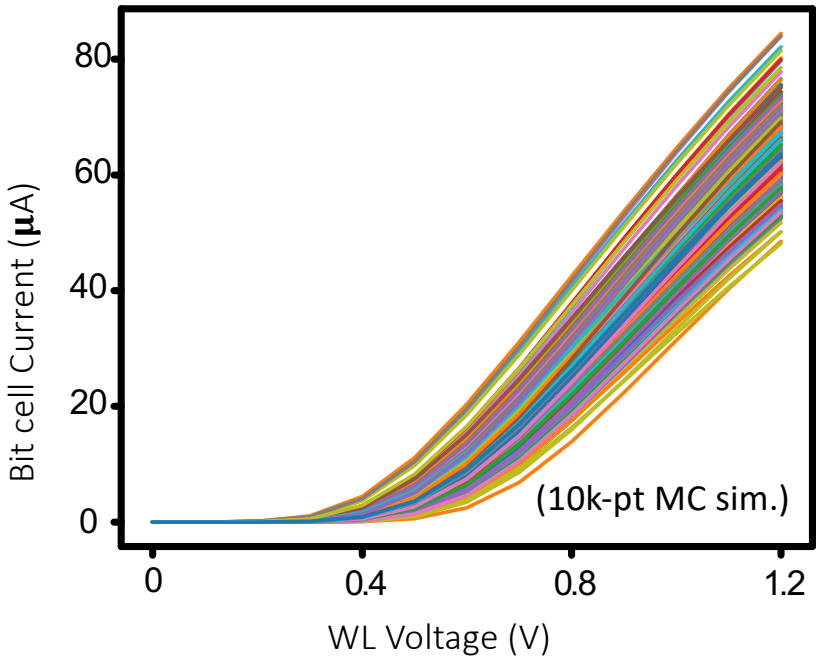
- **Need analog to 'fit' compute in bit cells (SNR limited by analog non-idealities)**
  → Must be feasible/competitive @ 16/12/7nm

Noise Limited



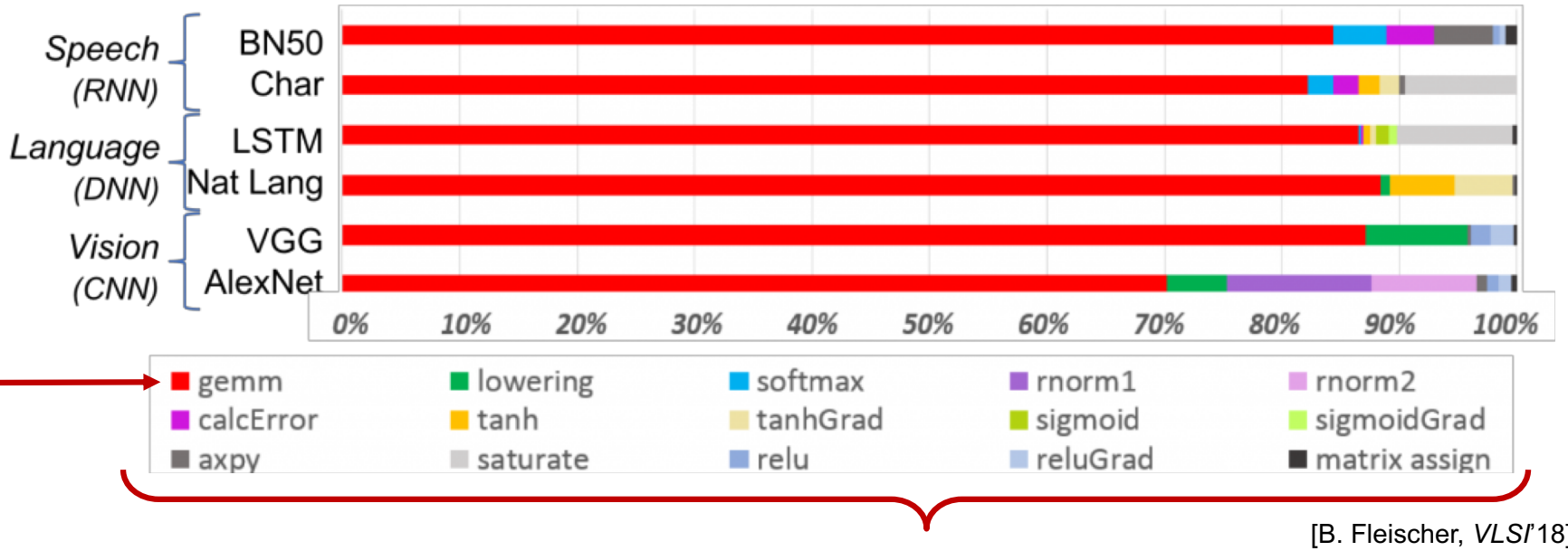
[R. Sarpeshkar, *Ultra Low Power Bioelectronic*]

Linearity/variation Limited



IMC Challenge (2): heterogeneous computing

- **Matrix-vector multiply is only 70-90% of operations**
  ⟶ IMC must integrate in programmable, heterogenous architectures



[B. Fleischer, *VLSI*'18]

General Matrix Multiply (~256×2300=590k elements)

Single/few-word operands (traditional, near-mem. acceleration)

IMC Challenge (3): efficient application mappings

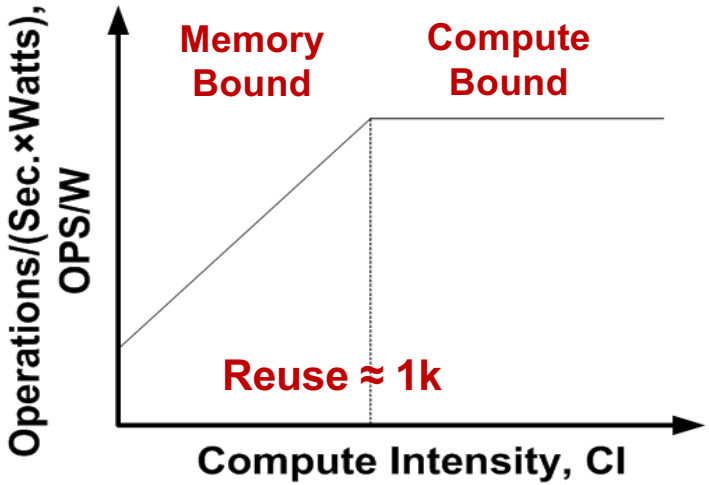
- **IMC engines must be 'virtualized'**
  ⟶ IMC amortizes MVM costs, not weight loading. But…
  ⟶ Need new mapping algorithms (physical tradeoffs very diff. than digital engines)

Activation Accessing

- $E_{DRAM \rightarrow IMC}$/4-bit: 40pJ
- Reuse: $N \times I \times J$ (10-20 lyrs)
- $E_{MAC,4\text{-}b}$: 50fJ

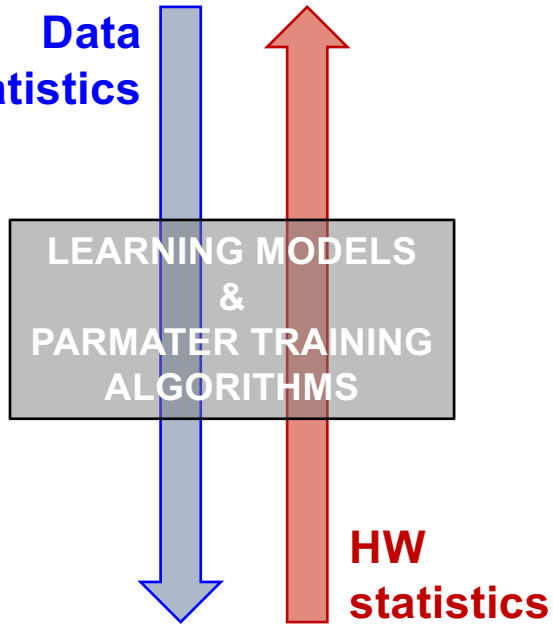
Weight Accessing

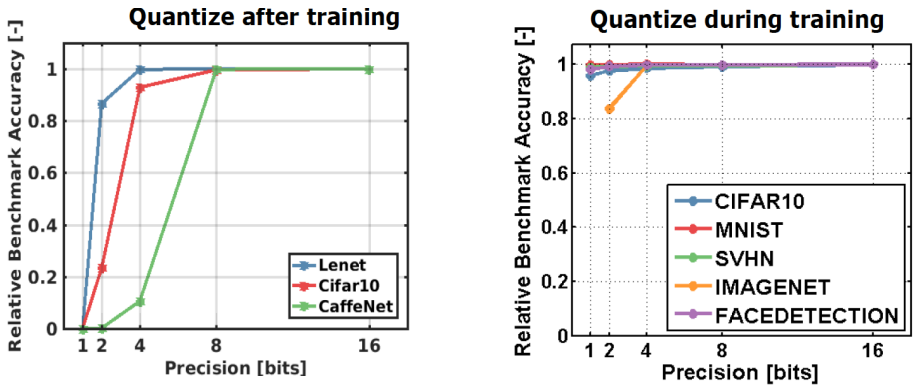
- $E_{DRAM \rightarrow IMC}$/4-bit: 40pJ
- Reuse: $X \times Y$
- $E_{MAC,4\text{-}b}$:50fJ



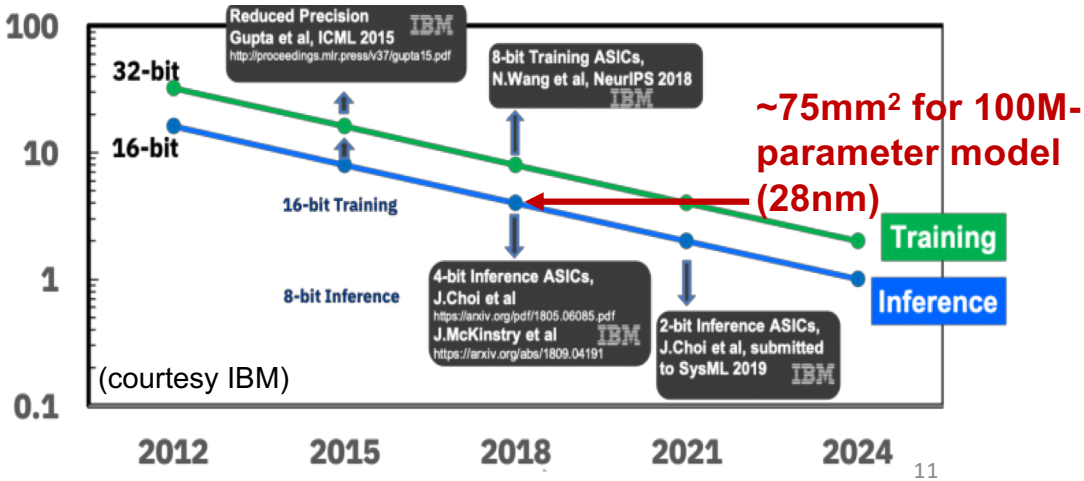
Low-SNR computation via algorithmic co-design

Opportunity for top-down
and bottom-up design



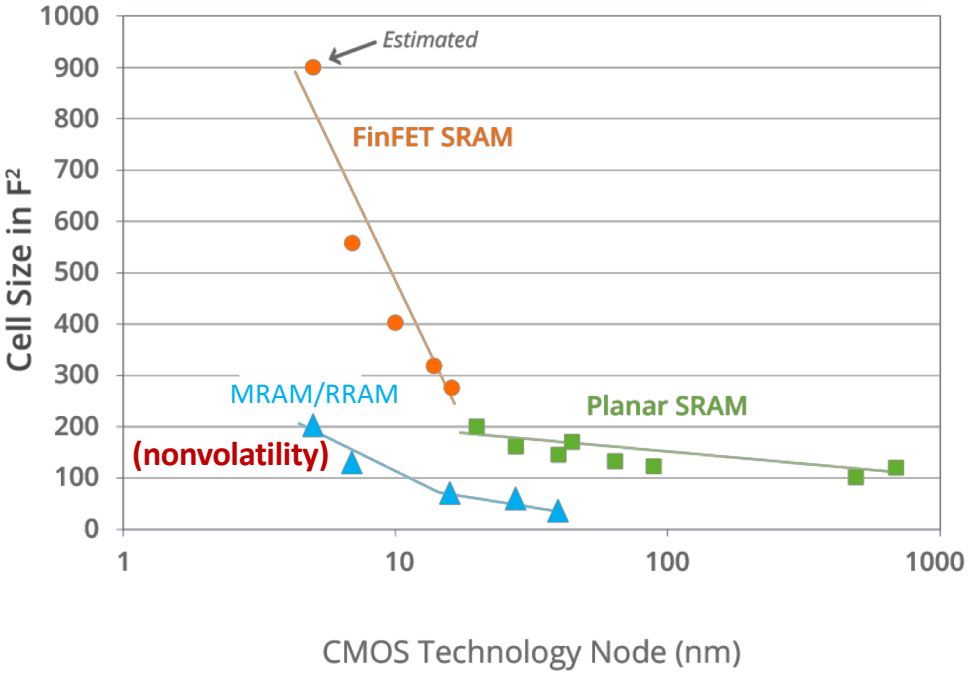


[M. Verhelst, *ISSCC SC on Machine Learning (2018)*]



Emerging non-volatile memory (NVM)

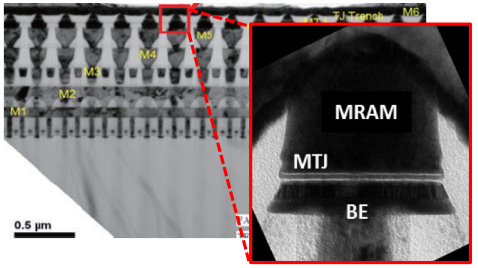
- **2-terminal resistive memory provides better scaling at advanced nodes**

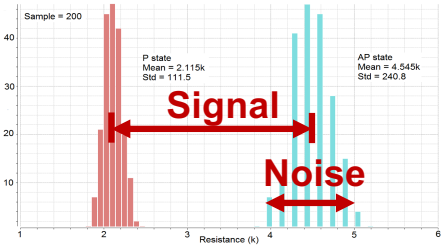


https://www.spinmemory.com/technologies/mram-overview/

- **… BUT, leads to significantly reduced SNR**

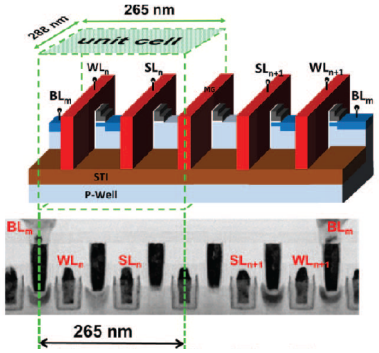
Magnetic RAM (MRAM)

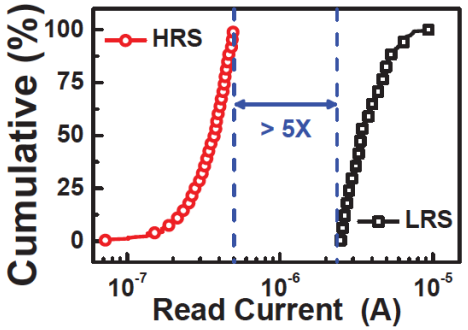




Globalfoundries, 22nm [D. Shum, VLSI'17]

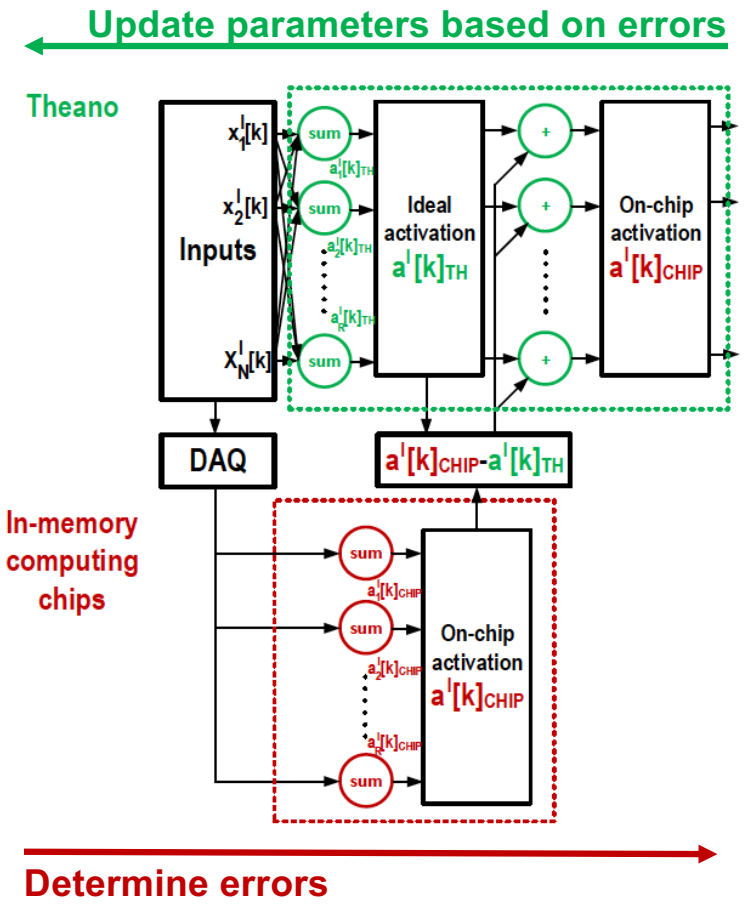
Resistive RAM (RRAM)





TSMC 16nm [H. W. Pan, IEDM'15]

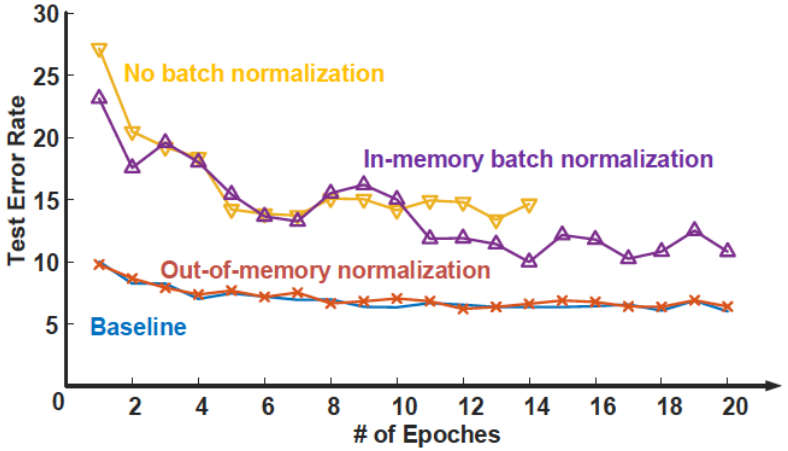
Chip-specific parameter training

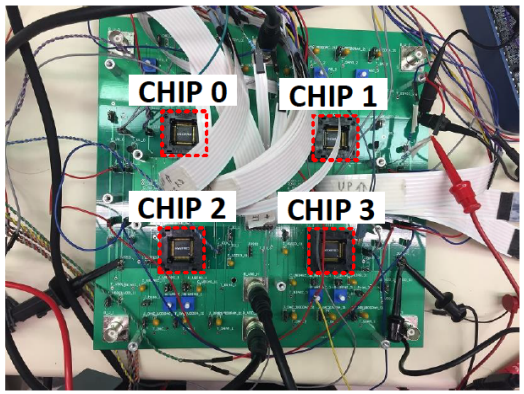


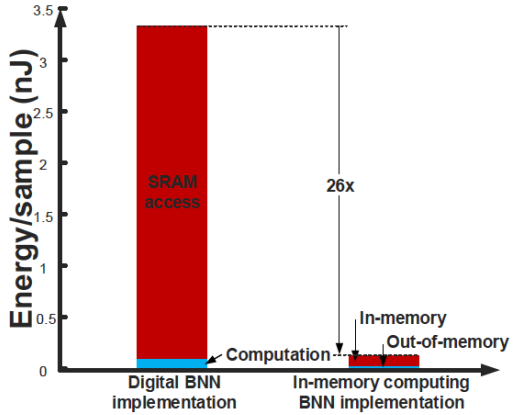
[J. Zhang, IEEE JETCAS'19]

Ex.: MNIST Classification with Binarized NN

(implemented on chip)

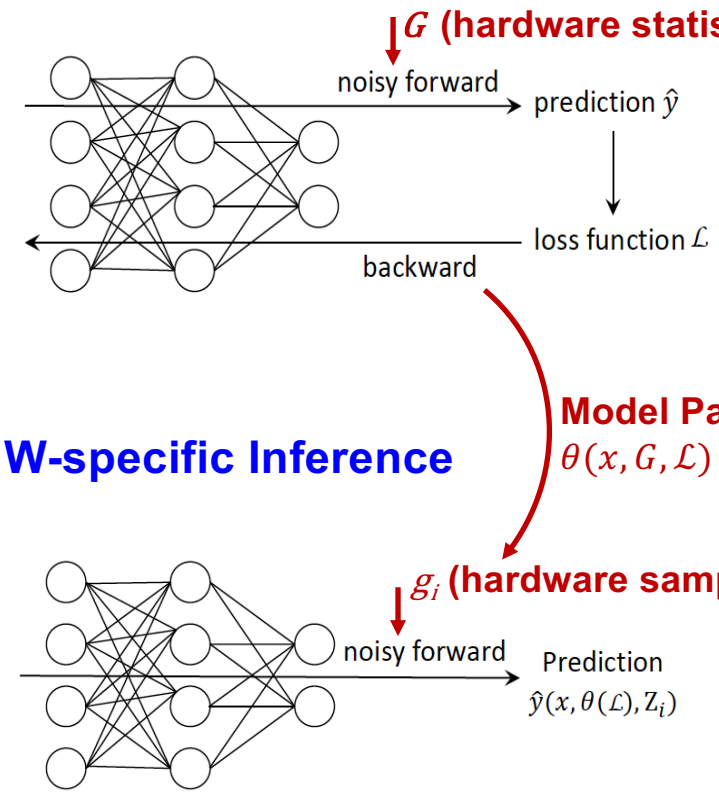






Chip-generalized parameter training

- **HW-generalized Stochastic Training**

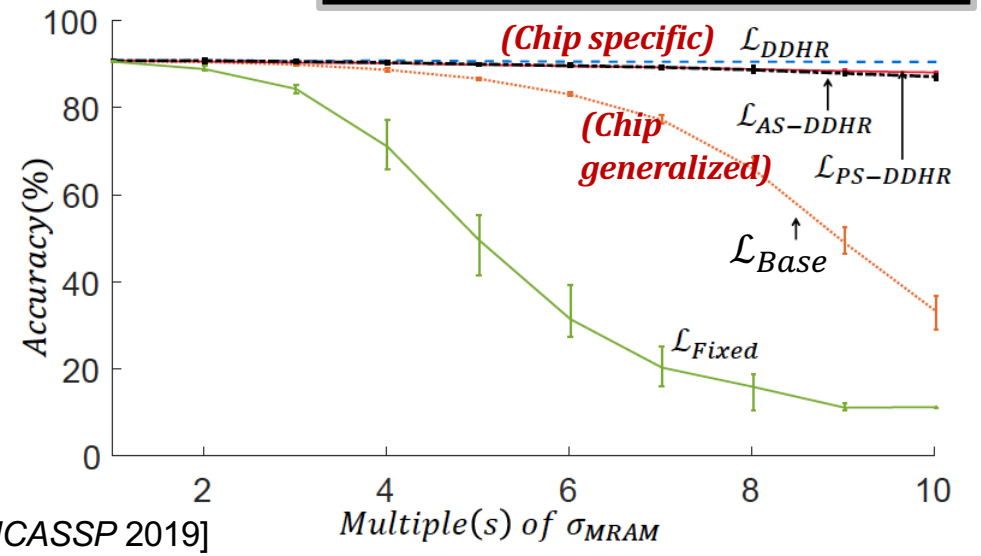


- **HW-specific Inference**

CIFAR-10 Classification using Binarized NN
(simulated)

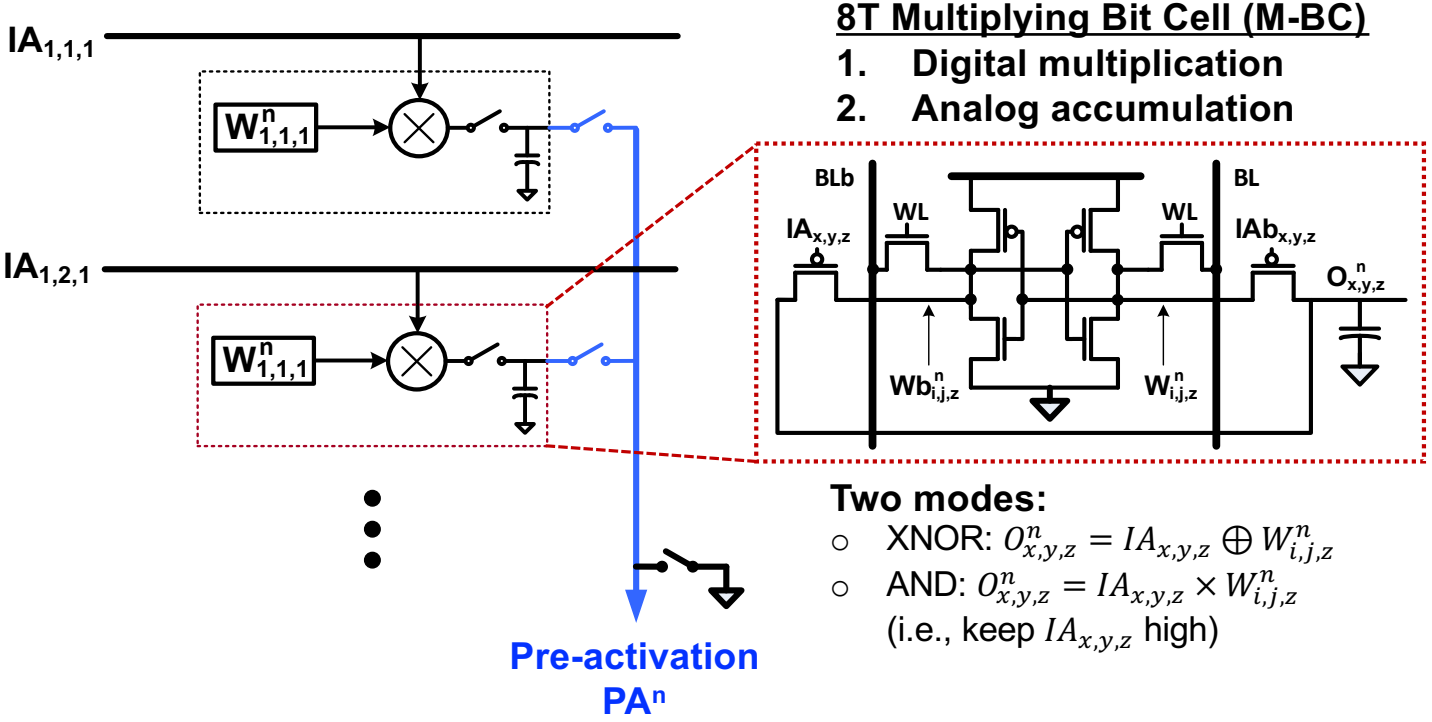
Statistical model in PyTorch, Keras

```
def model_stochastic(y, num_cell):
    # determine # of 1 and -1
    common_fac = (num_cell - abs(y))/2
    indicator_pos = (y > 0).type(torch.cuda.FloatTensor) * y
    indicator_neg = (y < 0).type(torch.cuda.FloatTensor) * abs(y)

    # compute variation statistics (mean & variance of Gaussian)
    common_mean = ms.cpM + ms.capM
    common_var = ms.cpV + ms.capV
    mean =  indicator_pos * ms.cpM + indicator_neg * ms.capM + common_fac * common_mean
    var = indicator_pos * ms.cpV + indicator_neg * ms.capV + common_fac * common_var

    # include variation
    y = torch.mul(torch.sqrt(var), torch.cuda.FloatTensor(y.size()).normal_()) + mean

    # ADC
    y = (y - num_cell * ms.capM + 0.5*ms.md) / ms.md
    y = torch.floor(y).type(torch.cuda.FloatTensor)
    y = -num_cell + 2*y
    y = torch.clamp(y, -num_cell, num_cell)

    return y
```



[B. Zhang, *ICASSP* 2019]

High-SNR, charge-domain IMC

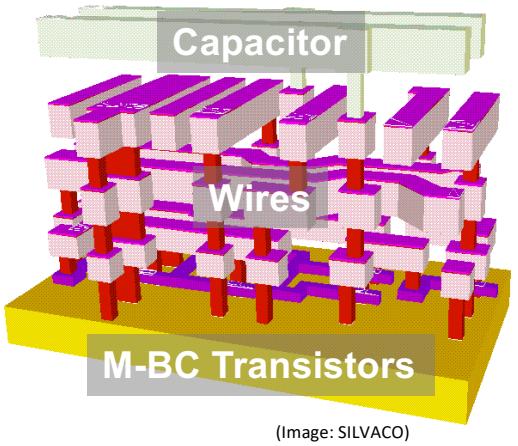


8T Multiplying Bit Cell (M-BC)

1. **Digital multiplication**
2. **Analog accumulation**

Two modes:

- XNOR: $O^n_{x,y,z} = IA_{x,y,z} \oplus W^n_{i,j,z}$
- AND: $O^n_{x,y,z} = IA_{x,y,z} \times W^n_{i,j,z}$
  (i.e., keep $IA_{x,y,z}$ high)



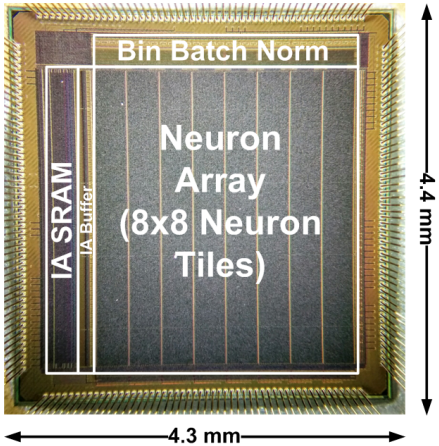
(Image: SILVACO)

(approx. 80% area overhead compared to standard 6T bit cell)

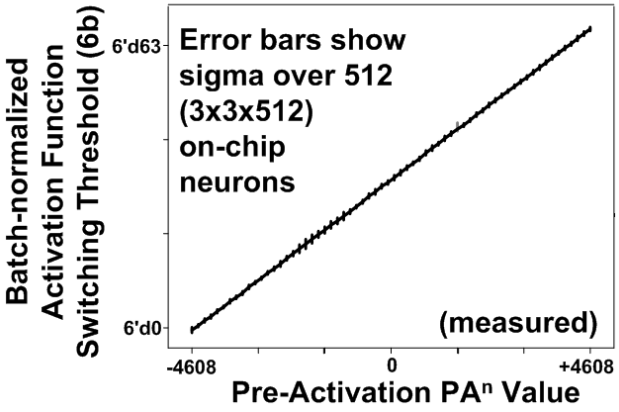
- **Capacitors provide much better controllability & technology scalability**
  ⟶ 10's of thousands of IMC rows before SNR is capacitor limited

[H. Valavi, *VLSI*'18] [H. Valavi, *JSSC*'19]

2.4Mb, 64-tile IMC



Neuron Transfer Function

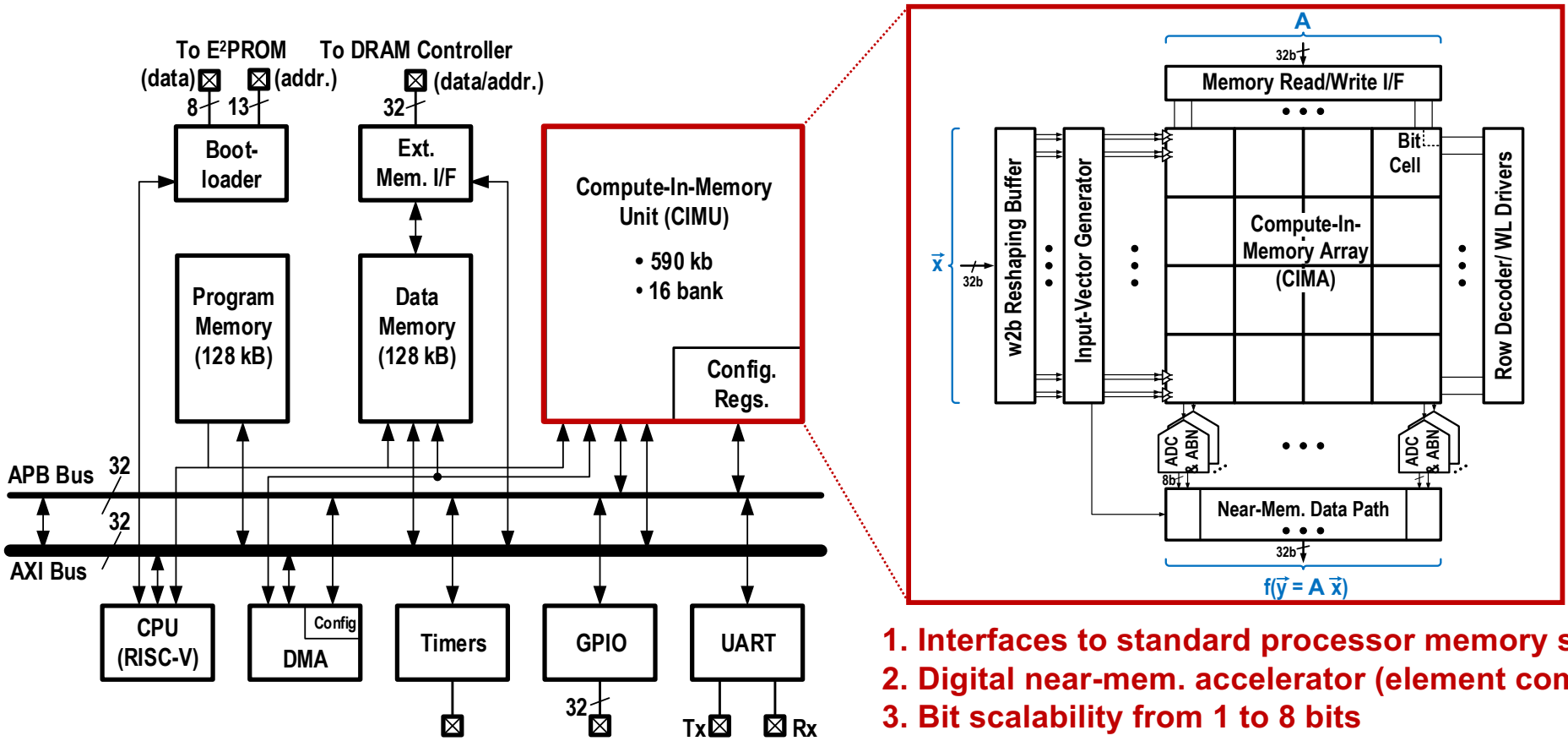


	Moons, ISSCC'17	Bang, ISSCC'17	Ando, VLSI'17	Bankman, ISSCC'18	Valavi, VLSI'18
Technology	28nm	40nm	65nm	28nm	65nm
Area (mm^2)	1.87	7.1	12	6	17.6
Operating VDD	1	0.63-0.9	0.55-1	0.8/0.8 (0.6/0.5)	0.94/0.68/1.2
Bit precision	4-16b	6-32b	1b	1b	1b
on-chip Mem.	128kB	270kB	100kB	328kB	295kB
Throughput (GOPS)	400	108	1264	400 (60)	18,876
TOPS/W	10	0.384	6	532 (772)	866

- 10-layer CNN demos for MNIST/CIFAR-10/SVHN at energies of 0.8/3.55/3.55 µJ/image
- Equivalent performance to software implementation

[H. Valavi, *VLSI'18*]

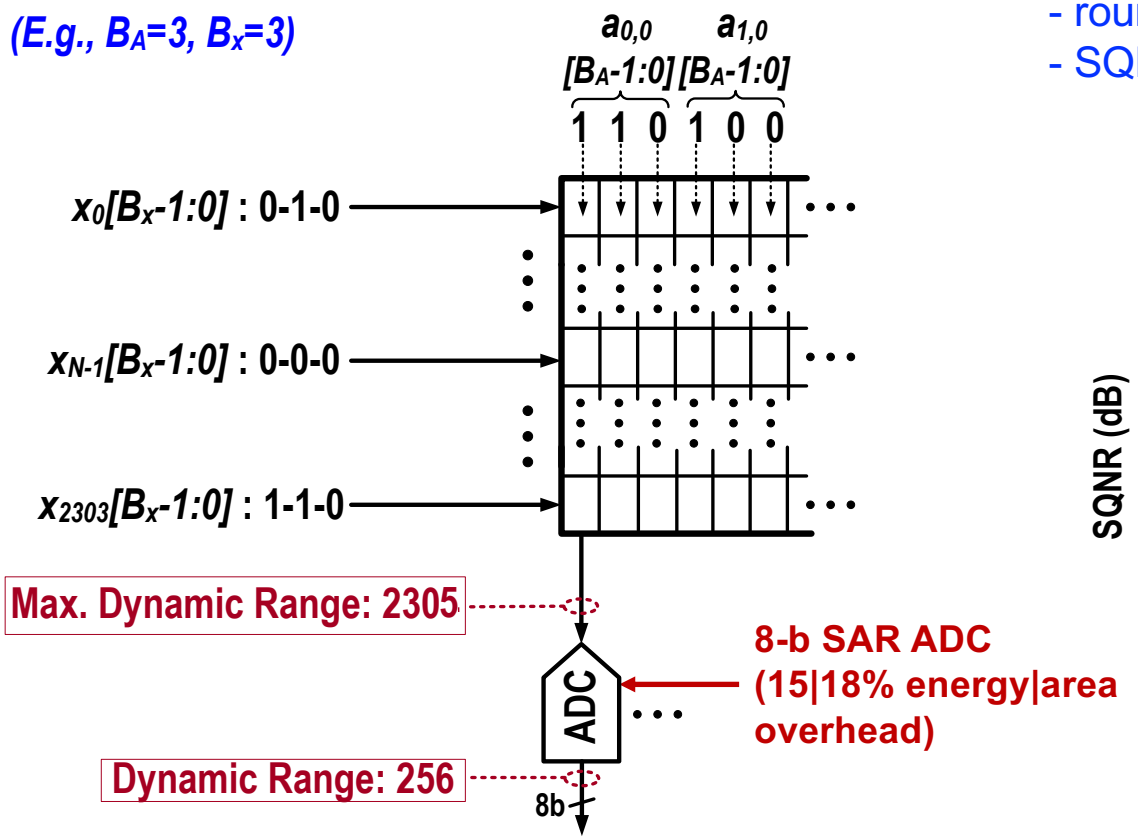
Programmable heterogeneous IMC processor



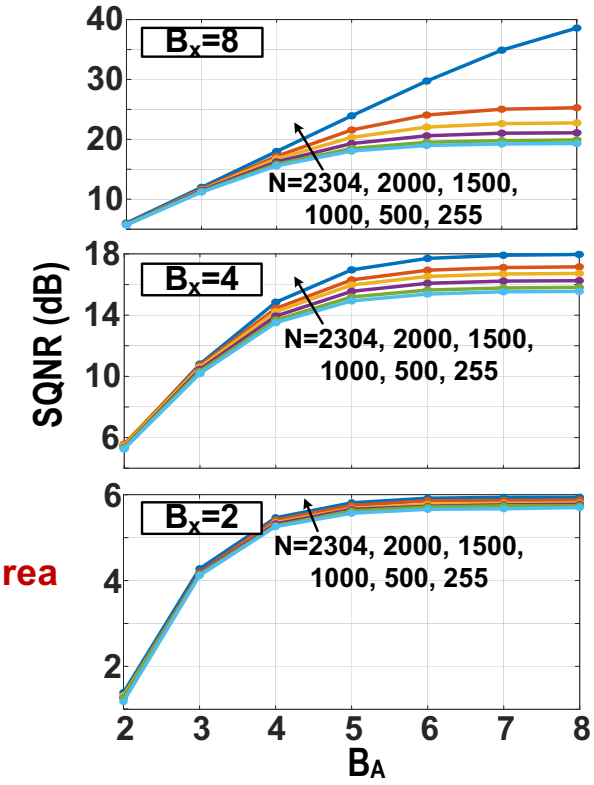
1. Interfaces to standard processor memory space
2. Digital near-mem. accelerator (element compute)
3. Bit scalability from 1 to 8 bits

[H. Jia, *arXiv*:1811.04047] [H. Jia, HotChips'19]

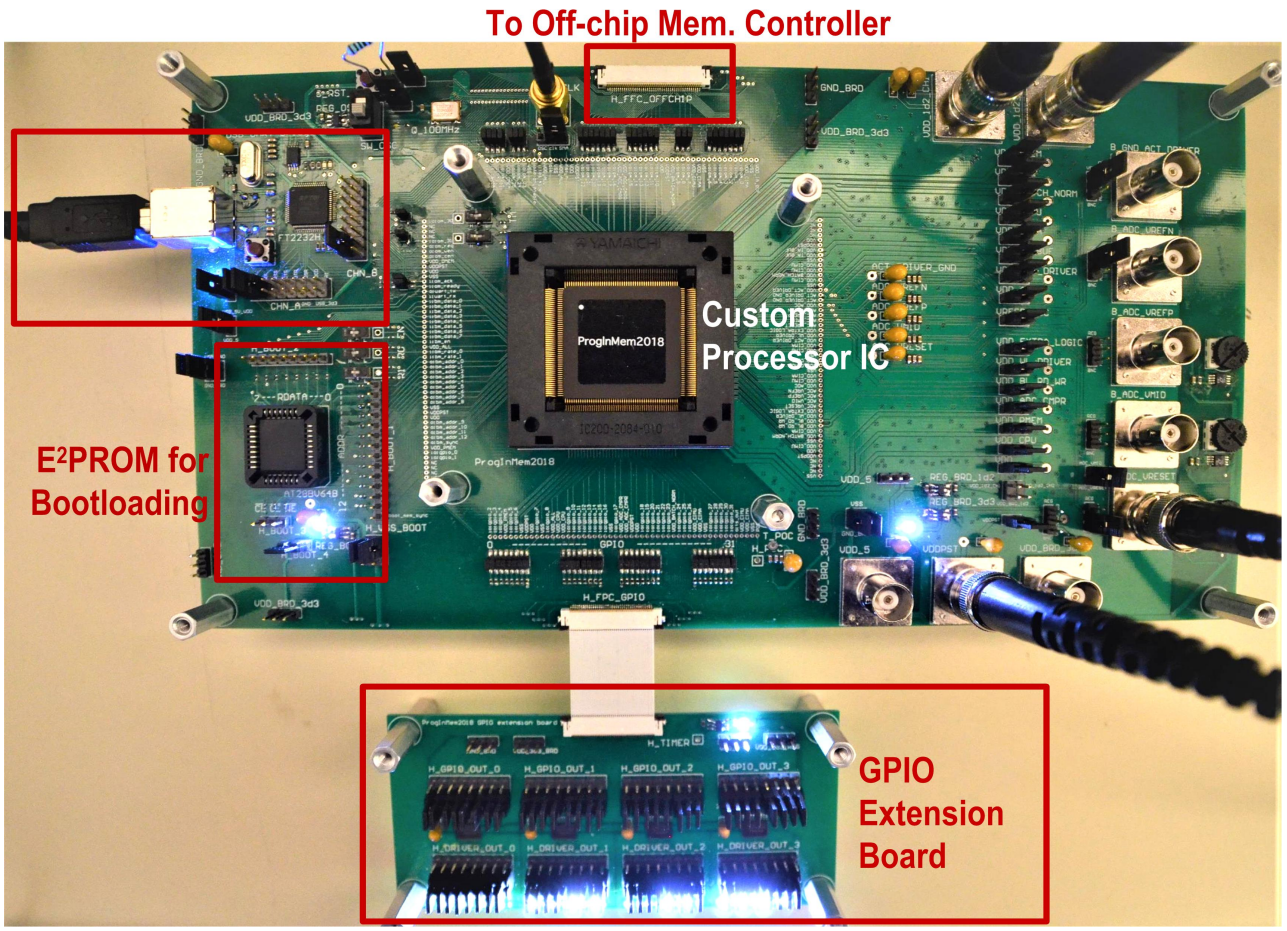
Bit-Parallel/Bit-Serial (BP/BS) Multi-bit IMC



- **SQNR different that standard INT compute**
  - rounding effects are well modeled
  - SQNR is high at precisions of interest



Development board



Software libraries

1. Deep-learning Training Libraries (Keras)

Standard Keras libs:

```
Dense(units, ...)
Conv2D(filters, kernel_size, ...)
...
```

Custom libs:
(INT/CHIP quant.)

```
QuantizedDense(units, nb_input=4, nb_weight=4,
        chip_quant=False, ...)
QuantizedConv2D(filters, kernel_size, nb_input=4,
        nb_weight=4, chip_quant=False, ...)
...
```

```
QuantizedDense(units, nb_input=4, nb_weight=4,
        chip_quant=True, ...)
QuantizedConv2D(filters, kernel_size, nb_input=4,
        nb_weight=4, chip_quant=True, ...)
...
```

2. Deep-learning Inference Libraries (Python, MATLAB, C)

High-level network build (Python):

```
chip_mode = True
outputs = QuantizedConv2D(inputs,
        weights, biases, layer_params)
outputs = BatchNormalization(inputs,
        layer_params)
...
```

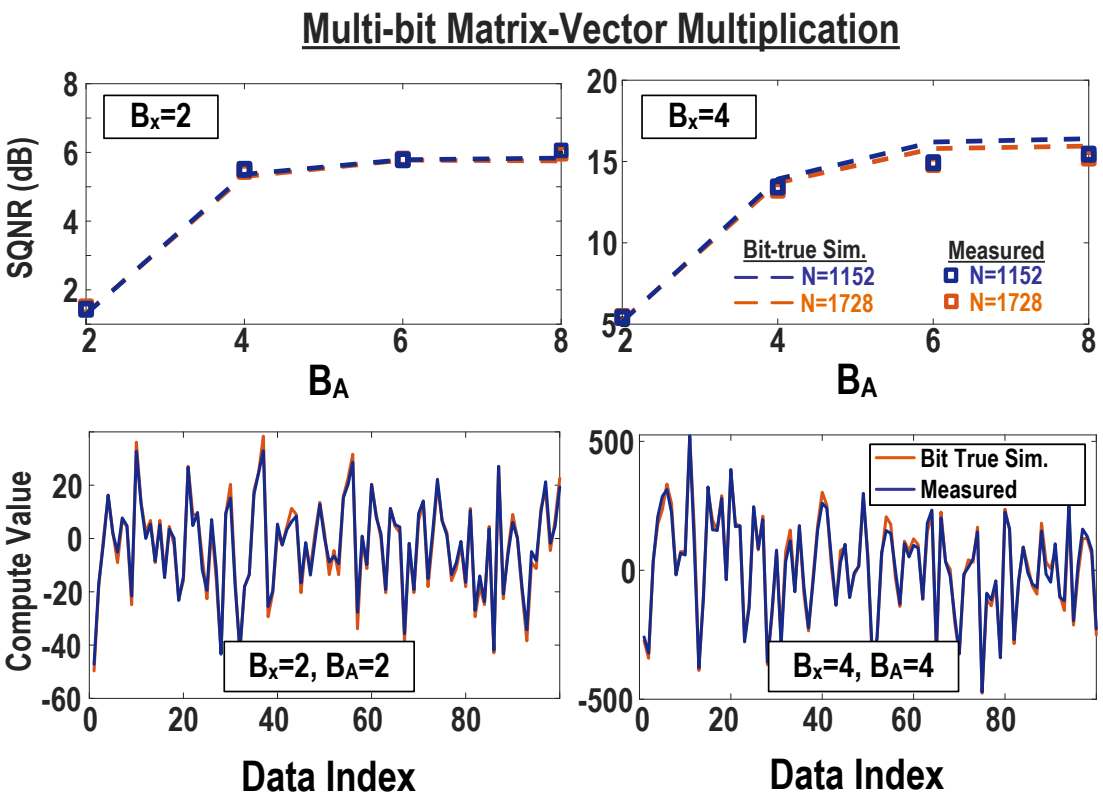
Function calls to chip (Python):

```
chip.load_config(num_tiles, nb_input=4,
        nb_weight=4)
chip.load_weights(weights2load)
chip.load_image(image2load)
outputs = chip.image_filter()
```

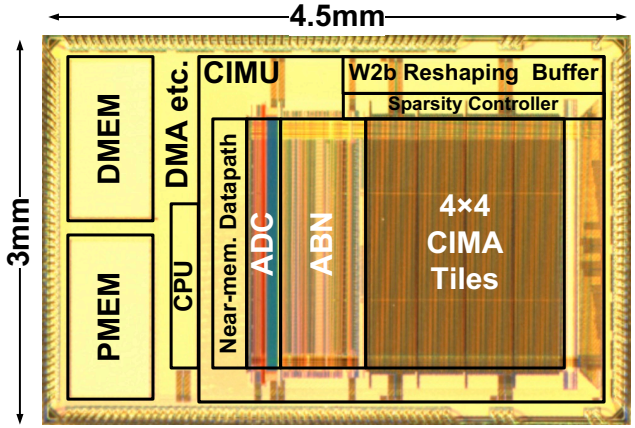
Embedded C:

```
chip_command = get_uart_word();
chip_config();
load_weights(); load_image();
image_filter(chip_command);
read_dotprod_result(image_filter_command);
```

Demonstrations



[H. Jia, *arXiv*:1811.04047]



Neural-Network Demonstrations		
	Network A (4/4-b activations/weights)	**Network B** (1/1-b activations/weights)
Accuracy of chip (vs. ideal)	92.4% (vs. 92.7%)	89.3% (vs. 89.8%)
Energy/10-way Class.[1]	105.2 µJ	5.31 µJ
Throughput[1]	23 images/sec.	176 images/sec.
Neural Network Topology	L1: 128 CONV3 – Batch norm L2: 128 CONV3 – POOL – Batch norm. L3: 256 CONV3 – Batch. norm L4: 256 CONV3 – POOL – Batch norm. L5: 256 CONV3 – Batch norm. L6: 256 CONV3 – POOL – Batch norm. L7-8: 1024 FC – Batch norm. L9: 10 FC – Batch norm.	L1: 128 CONV3 – Batch Norm. L2: 128 CONV3 – POOL – Batch Norm. L3: 256 CONV3 – Batch Norm. L4: 256 CONV3 – POOL – Batch Norm. L5: 256 CONV3 – Batch Norm. L6: 256 CONV3 – POOL – Batch Norm. L7-8: 1024 FC – Batch norm. L9: 10 FC – Batch norm.

Conclusions & summary

Matrix-vector multiplies (MVMs) are a little different than other computations

⟶ *high-dimensionality operands lead to data movement / memory accessing*

Bit cells make for dense, energy-efficient PE's in spatial array

⟶ *but require analog operation to fit compute, and impose SNR tradeoff*

Must focus on SNR tradeoff to enable
scaling (technology/platform levels) and architectural integration

In-memory computing greatly affects the architectural tradeoffs,
requiring new strategies for mapping applications

Acknowledgements: funding provided by ADI, DARPA, SRC/STARnet