Tutorial to NeuroSim: A Versatile Benchmark Framework for AI Hardware

Prof. Shimeng Yu

School of Electrical and Computer Engineering,

Georgia Institute of Technology

Email: shimeng.yu@ece.gatech.edu

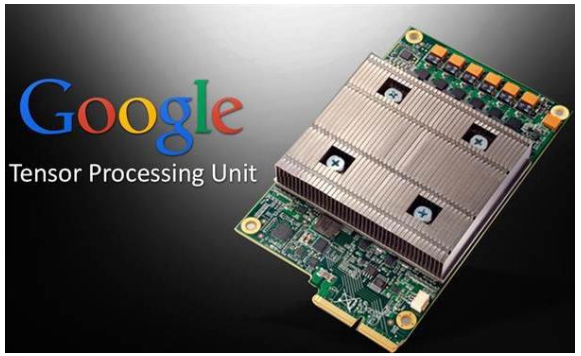
Web: https://shimeng.ece.gatech.edu/

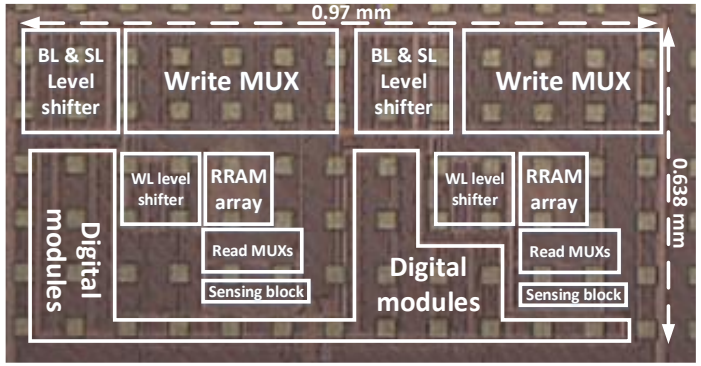
Outline

- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

Hardware Accelerators for AI/ML

- GPU still dominates the training in cloud, FPGA is good for inference for fast prototyping
- TPU (or similar digital ASIC) is ramping up in cloud as well as edge





GPU

Conventional computing platforms

~ 0.1 TOPS/W

Floating-point

TPU

Digital CMOS ASICs

~ 1-10 TOPS/W

Fixed-point

Compute-in-memory (CIM)

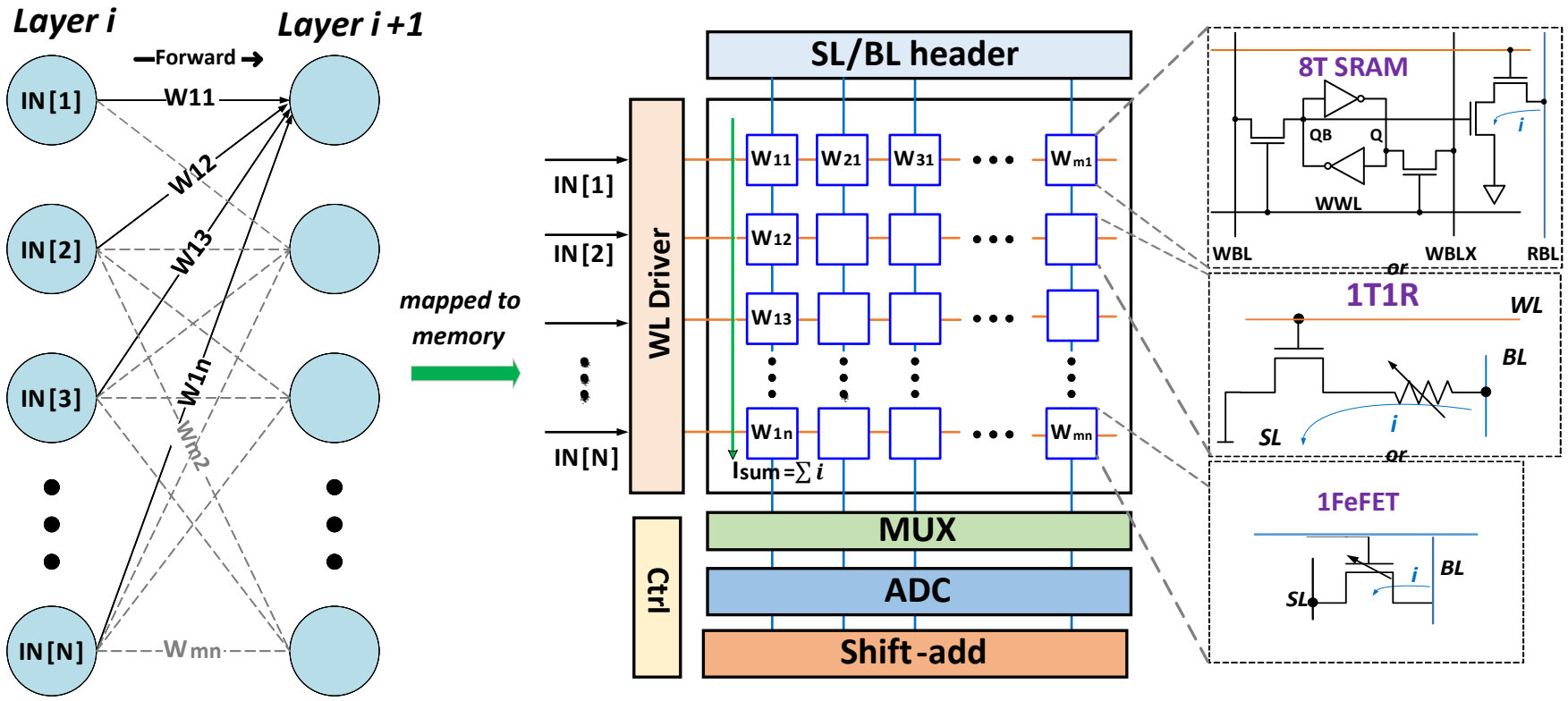
Analog CMOS (or eNVMs)

~ 10-100 TOPS/W

Low-precision→ accuracy?

- To further improve energy efficiency (TOPS/W), analog CIM (possibly with eNVMs) is promising especially in the edge inference where the model is pre-trained.
- NeuroSim is suitable for early design exploration of device candidates for CIM

CIM Basics: Mixed-Signal Compute



Multi-bit memory is preferred to serve as synaptic weights

S. Yu, H. Jiang, S. Huang, X. Peng, A. Lu, "Compute-in-memory chips for deep learning: recent trends and prospects", **IEEE Circuits and Systems Magazine**, vol. 21, no. 3, pp. 31-56, 2021, *invited review*

Key Metrics for Emerging Memories

	Mainstream Charged based Memories				Emerging Non-volatile Memories					
	SRAM	DRAM	FLASH		PCM	RRAM	STT-MRAM	SOT-MRAM	FeRAM	FeFET
			NOR	NAND						
Cell area	>150F^2	6F^2	10F^2	<4F^2 (3D)	4~50F^2	4~50F^2	6~50F^2	12~100F^2	6~50F^2	6~50F^2
Multi-bit	1	1	2	3-4	2-3	2-3	1	1	1	2-3
Voltage	<1V	<1V	>10V	>10V	<3V	<3V	<1V	<1V	<2V	<3V
Read time	~1ns	~10ns	~50ns	~10µs	<10ns	<10ns	<10ns	~1ns	<100ns	<50ns
Write time	~1ns	~10ns	10µs-1ms	100µs-1ms	~50ns	<100ns	<20ns	<3ns	<100ns	<100ns
Retention	N/A	~64ms	>10y	>10y	>10y	>10y	>1y	>1y	>10y	>1y
Endurance	>1E16	>1E16	~1E5	1E3~1E4	1E6~1E9	1E3~1E9	1E6~1E14	~1E12	1E9~1E12	1E6~1E9
Write Energy (J/bit)	~fJ	~10fJ	100pJ	~10fJ	~10pJ	~pJ	~pJ	~pJ	~100fJ	~fJ

F: feature size of the lithography.

The energy estimation is on the cell-level (not on the array-level).

PCM /RRAM/FeFET can potentially achieve less than 4F^2 through 3D integration.

The numbers of this table are representative (not the best or the worst cases).

Question: which eNVM is good for CIM?

S. Yu, **Semiconductor Memory Devices and Circuits**, Publisher: CRC Press/Taylor & Francis, 2022. [Link]

NeuroSim: Open-Source Simulator for AI Hardware

Product Solutions Open Source Pricing Search or jump to... / Sign in Sign up

https://github.com/neurosim

NeuroSim
neurosim

Overview Repositories 11 Projects Packages Stars 2

➢ Pre-RTL simulator for early-stage research

Pinned

MLP_NeuroSim_V3.0 Public
Benchmark framework of synaptic device technologies for a simple neural network
C++ 141 72

DNN_NeuroSim_V2.1 Public
Benchmark framework of compute-in-memory based accelerators for deep neural network (on-chip training chip focused)
C++ 63 36

3D_NeuroSim_V1.0 Public
Benchmark framework of 3D integrated CIM accelerators for popular DNN inference, support both monolithic and heterogeneous 3D integration
C++ 15 7

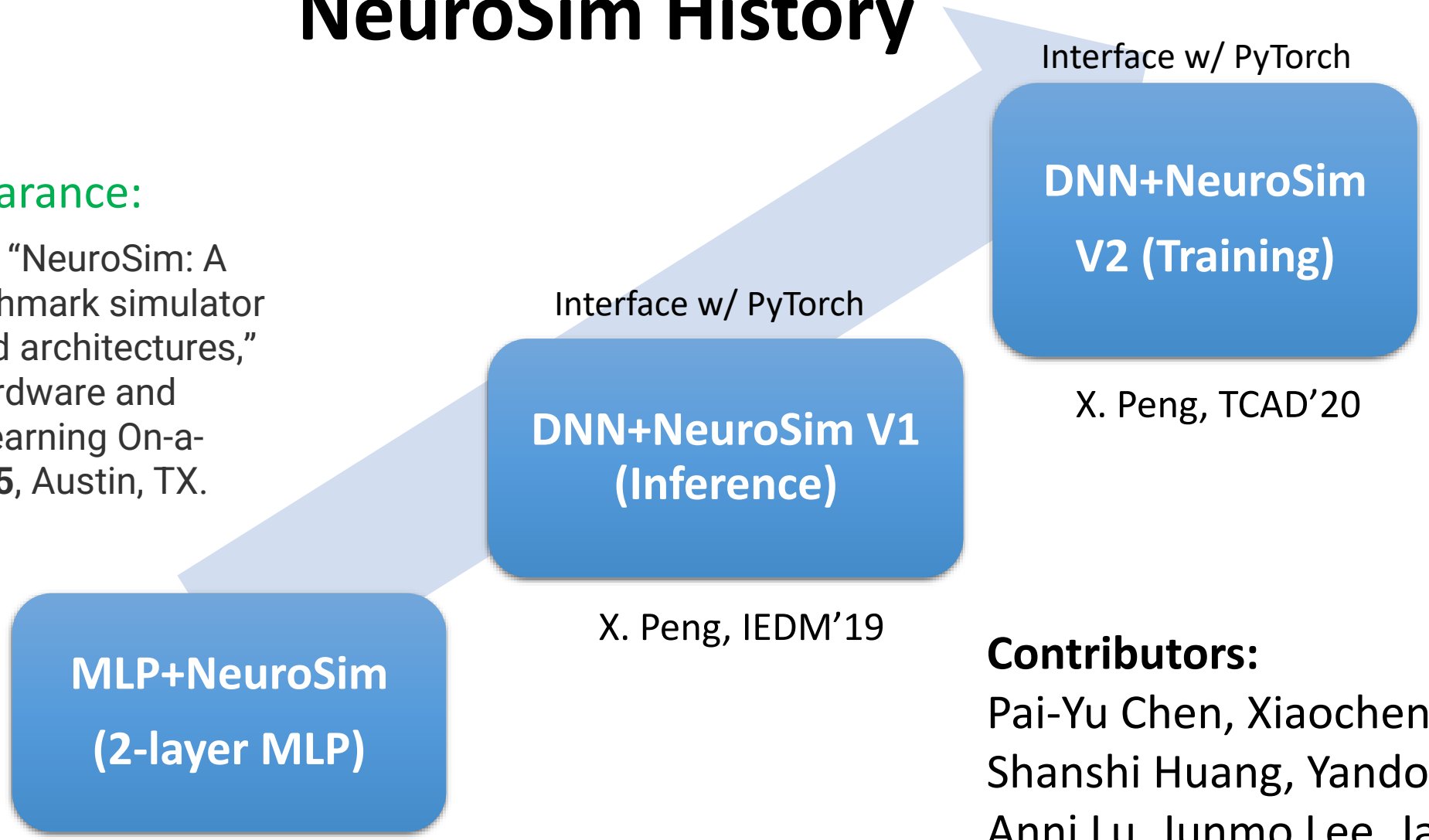
DNN_NeuroSim_V1.4 Public
Benchmark framework of compute-in-memory based accelerators for deep neural network (inference engine focused)
C++ 13

- 500+ citations
- >100 users including industry researchers from SK Hynix, Samsung, TSMC, and Intel
- IEEE Transactions on Computer-Aided Design (TCAD) Donald O. Pederson Best Paper Award 2023
- European Design and Automation Association (EDAA) Outstanding Dissertation Award 2023

NeuroSim History

The first appearance:

P.-Y. Chen, **S. Yu**, "NeuroSim: A circuit-level benchmark simulator for neuro-inspired architectures," Workshop on Hardware and Algorithms for Learning On-a-chip **(HALO) 2015**, Austin, TX.

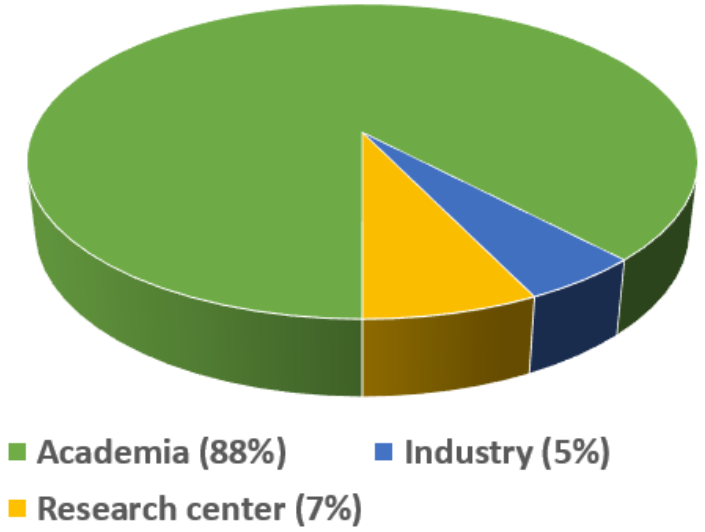


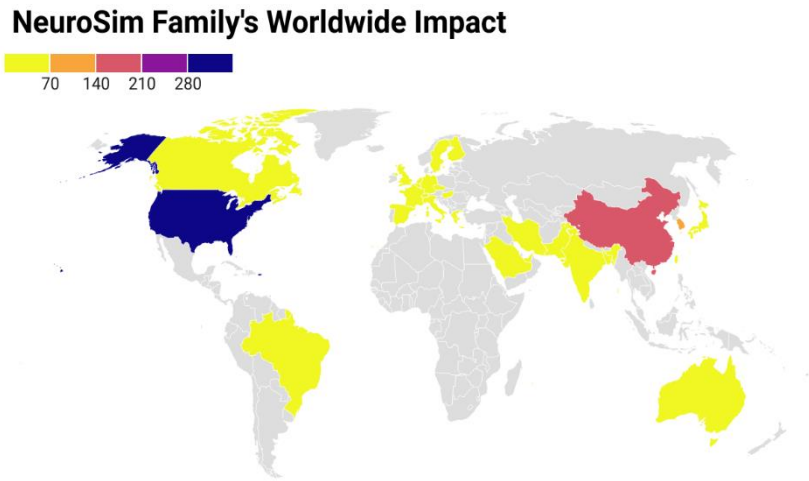
Contributors:

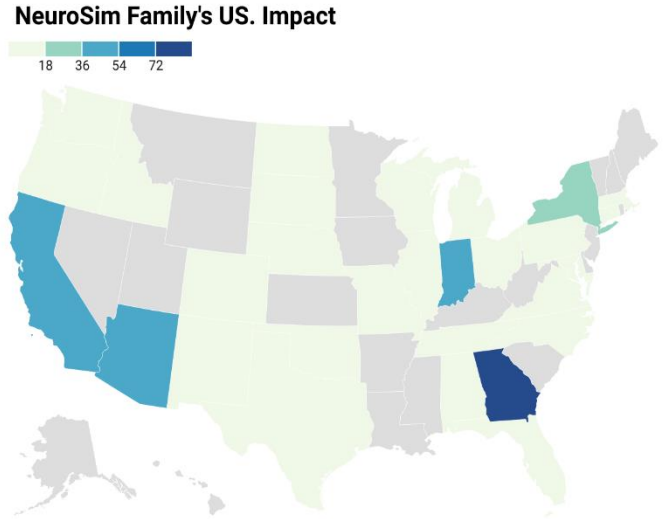
Pai-Yu Chen, Xiaochen Peng
Shanshi Huang, Yandong Luo
Anni Lu, Junmo Lee, James Read, etc...

Impact of NeuroSim Family

Resulted in publications 400+







intel

tsmc

SAMSUNG

SK hynix

Western Digital

imec

IBM

ERICSSON

NOVATEK

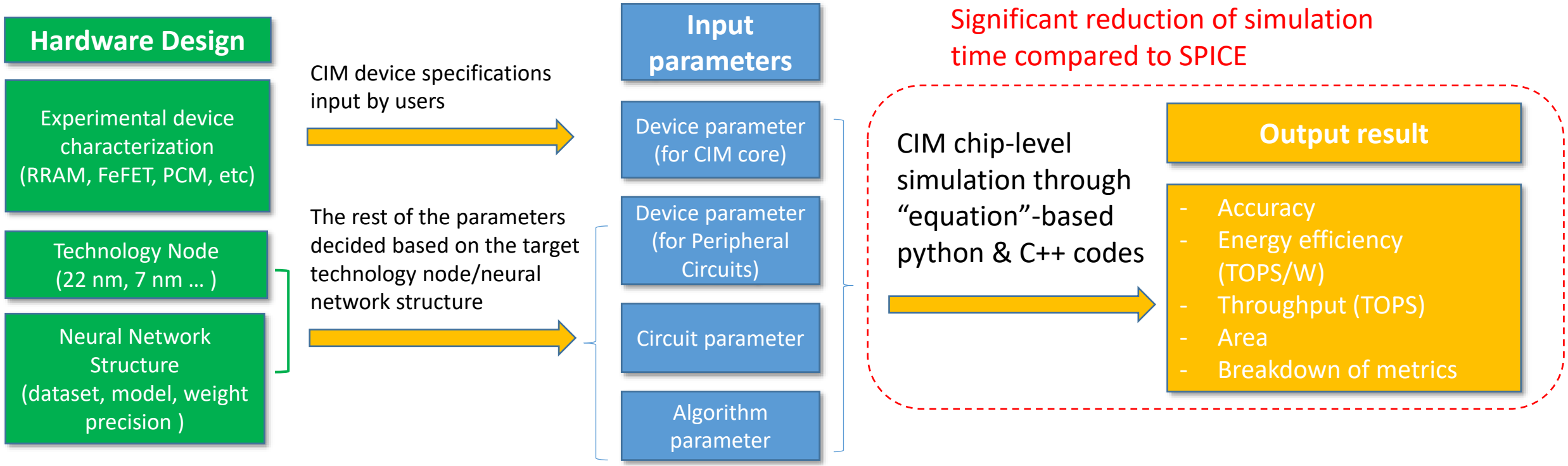
LABS hp

TSRI

NARLabs National Applied Research Laboratories
Taiwan Semiconductor Research Institute

DNN+NeuroSim Methodologies (Python & C++)

- End-to-end frameworks for: [1*]neural network inference engine [1], [2*]online neural network training system [2], [3*]3D-CIM design [3]
- Input parameters: ***devices → circuits → architecture hierarchy and data-flow → algorithms***
- Output results: ***inference/training accuracy, TOPS, TOPS/W and TOPS/mm²***



[1] X. Peng, et al., IEDM'2019; [2] X. Peng, et al., TCAD'2020; [3] X. Peng, et al., IEDM'2020.

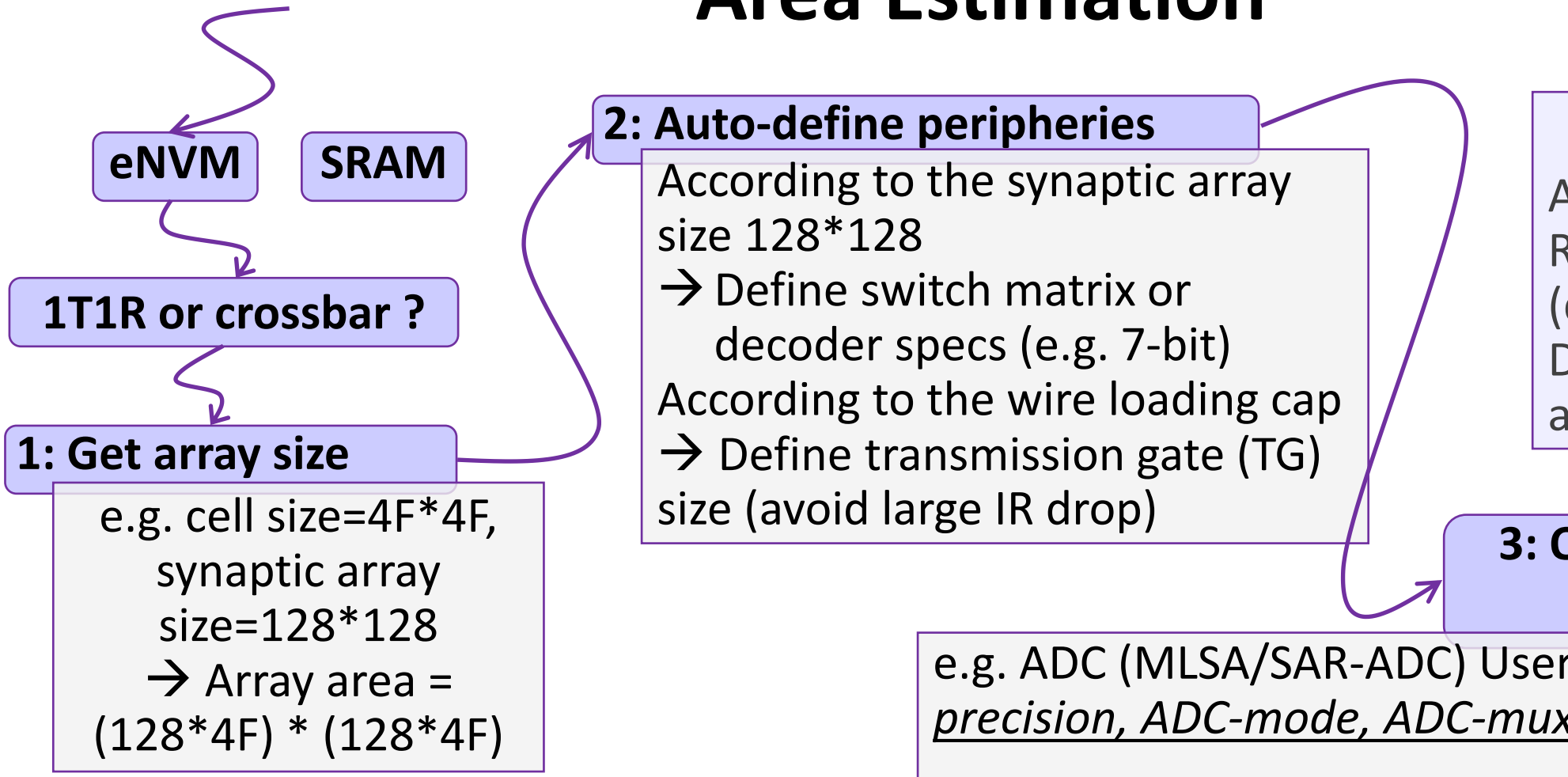
DNN+NeuroSim Methodologies (Python & C++)

Key Features

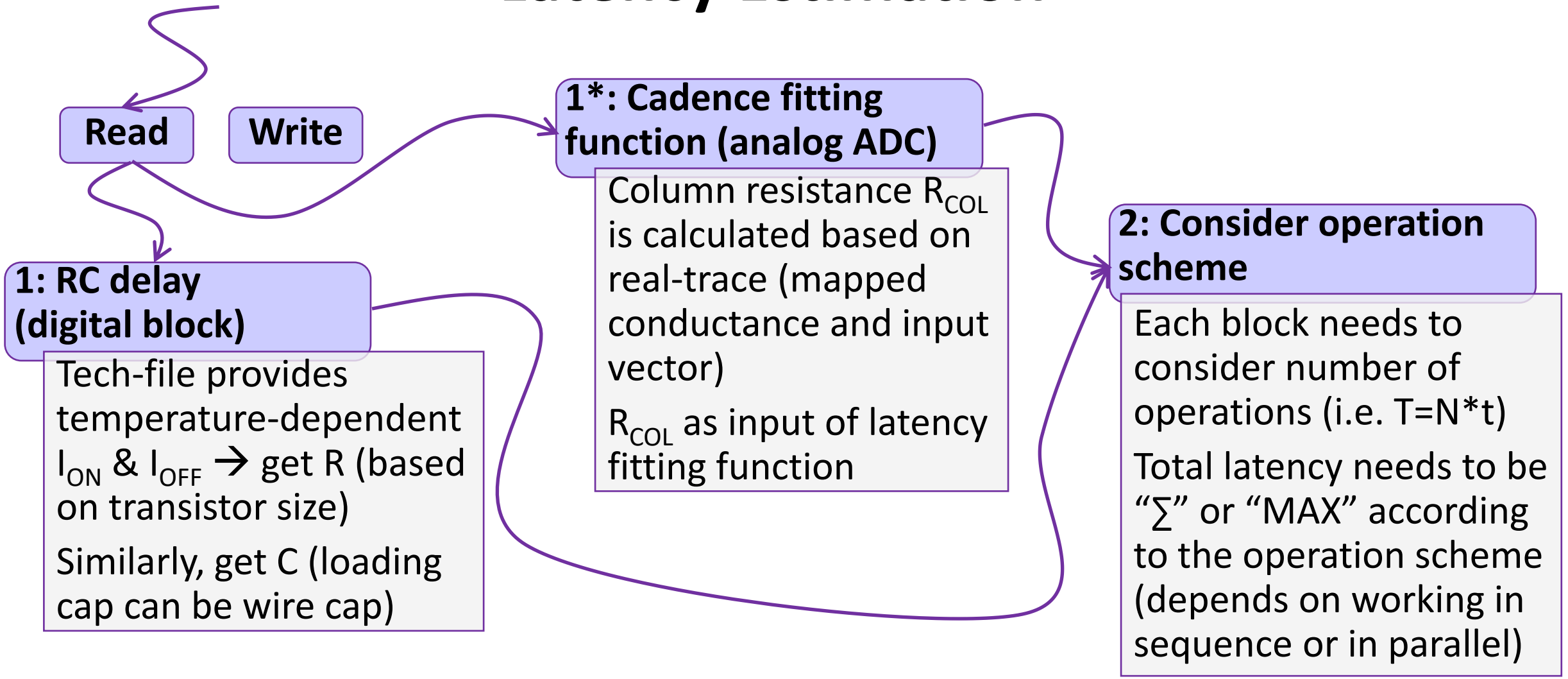
- **Hardware-aware quantization [4]** for weight, activation, gradient, error, as well as partial sum quantization based on ADC precision
- Built-in optimized **weight/activation mapping flow [5] for** maximum memory utilization
- Support **various large-scale network models/datasets (ResNet, VGG / CIFAR-10, 100 and ImageNet)**
- **Compact PPA estimation model exists** for every digital/analog circuit component within a chip, e.g. **ADC, level-shifter, D-flipflop, adder tree, etc.**
- PPA estimation model is **calibrated with SPICE simulation** at module-level
- **Massive options** (technology node from 130nm to 1nm, memory device type, R_{ON}, ADC option…)
- **Interconnect modules (e.g. H-tree)** estimated with parasitic RC delay and power
- Off-chip memory access overhead included (e.g. **LPDDR access** energy per bit)
- Non-ideal device/circuit effects-constrained accuracy estimation (e.g. device **variations/noises**, ADC **quantization loss**/ADC **offset**, etc.).
- Extendibility to variants of CIM hardware and various algorithms

[4] S. Wu, et al. ICLR'2018 [5] X. Peng, et al., ISCAS'2019;

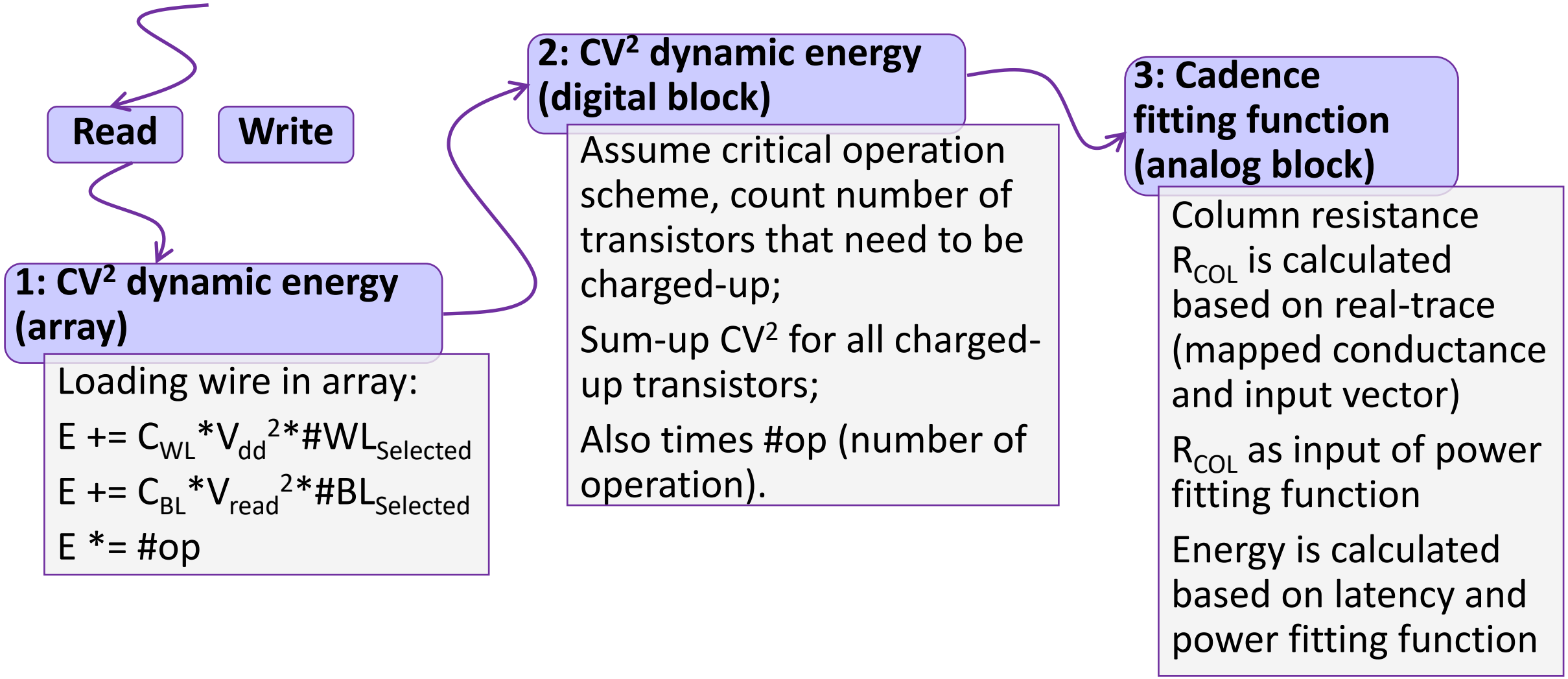
Area Estimation



Latency Estimation



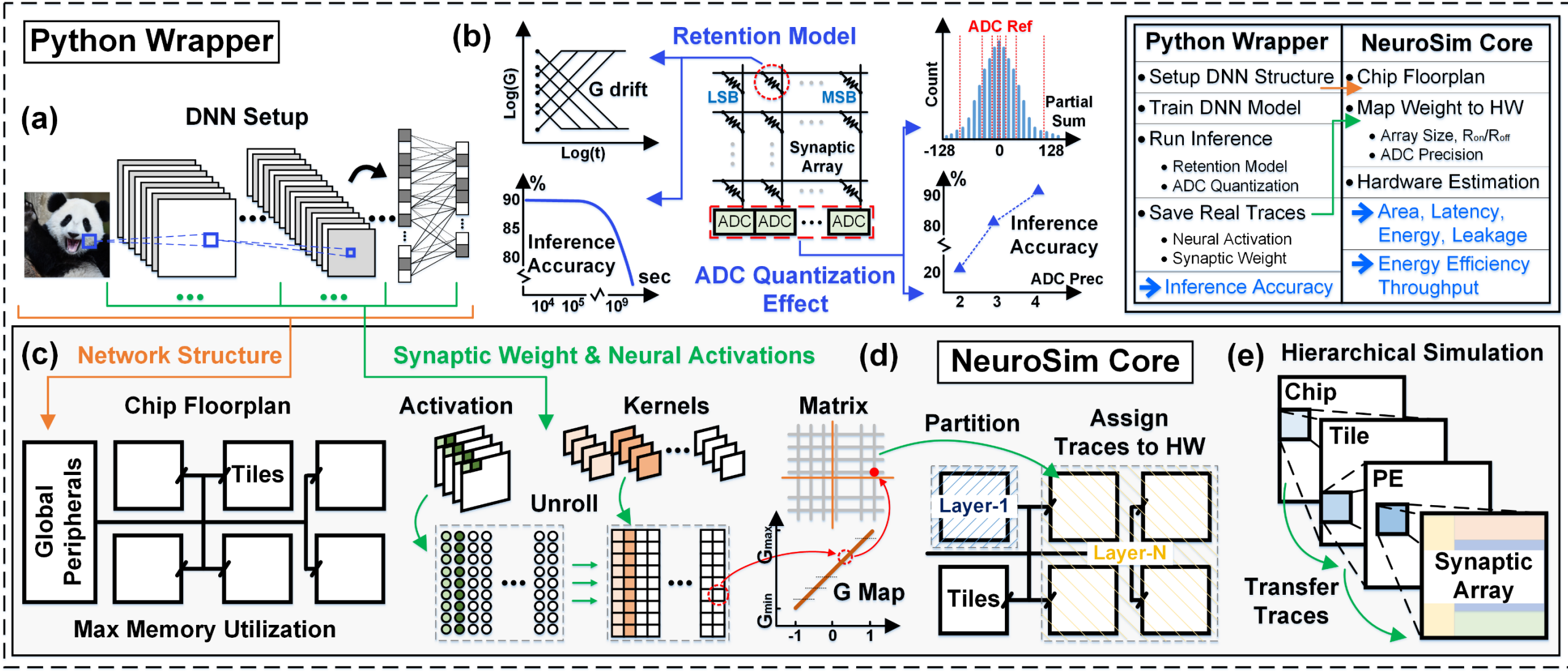
Dynamic Energy Estimation



Outline

- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

DNN+NeuroSim V1 for Inference Engine



X. Peng, et al., IEDM'2019.

DNN+NeuroSim V1 for Inference Engine

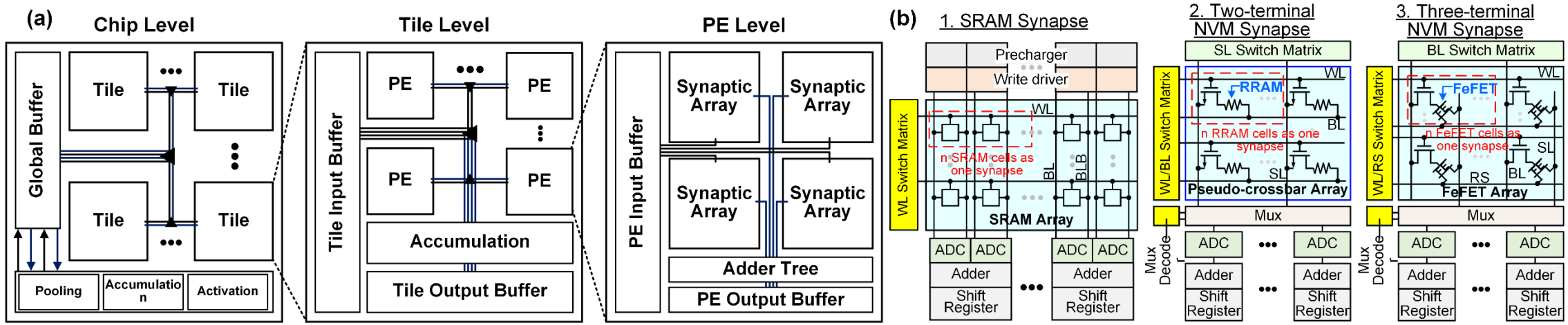


Fig.1 Architecture hierarchy of CIM accelerator (defined in DNN+NeuroSim V1.0).

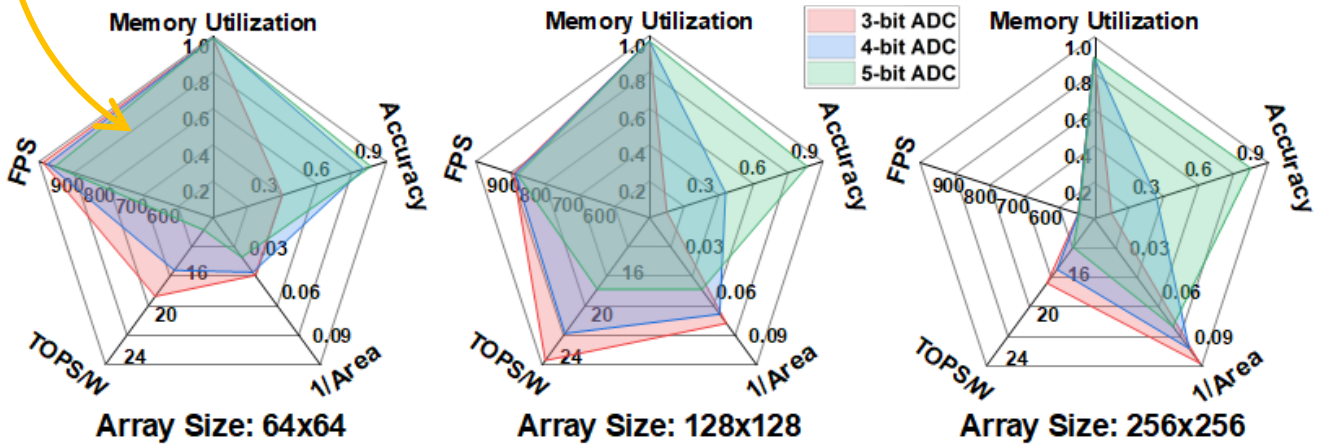


Fig. 2 System-level benchmark (VGG-8) with various design options (4-bit/cell).

Sweep array size & cell precision & ADC precision on CIFAR-10 accuracy

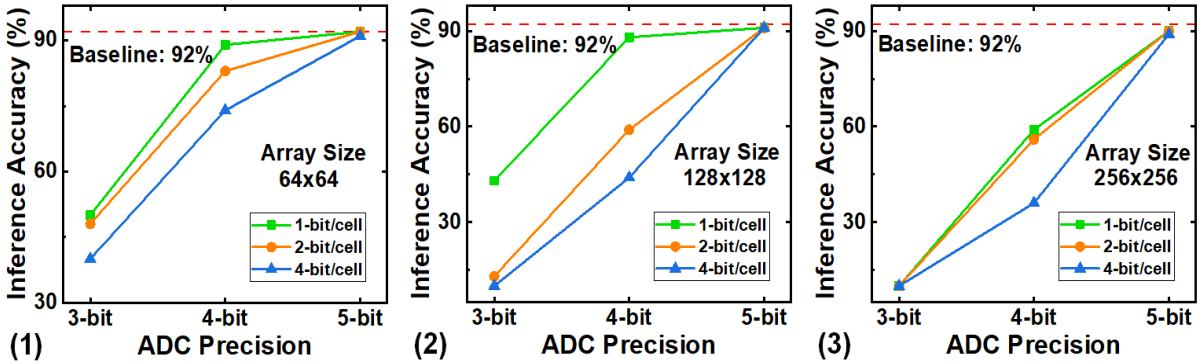
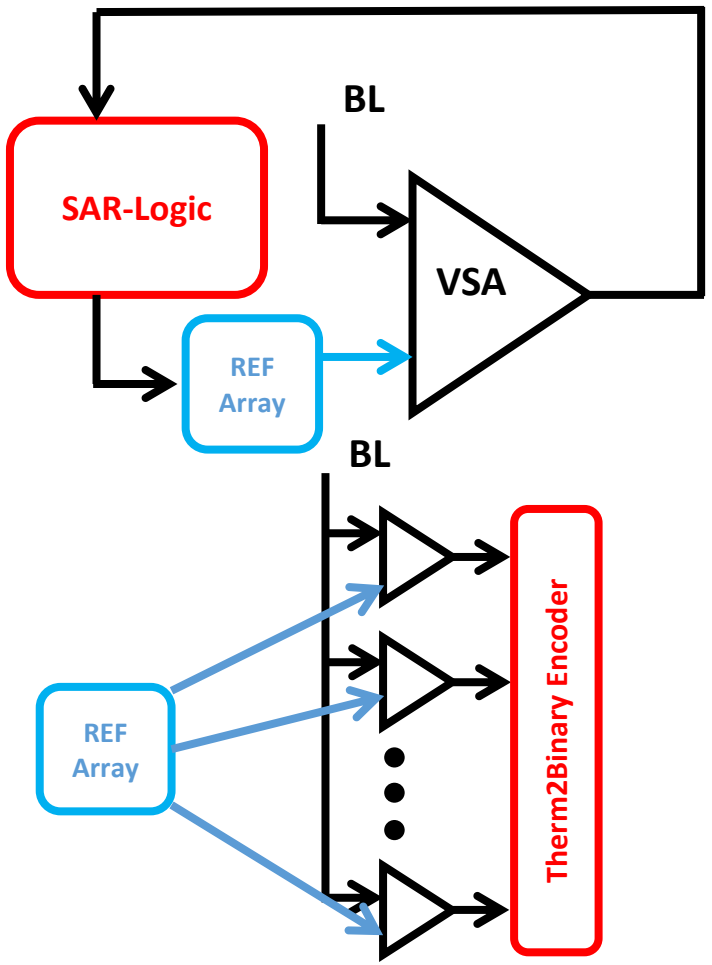
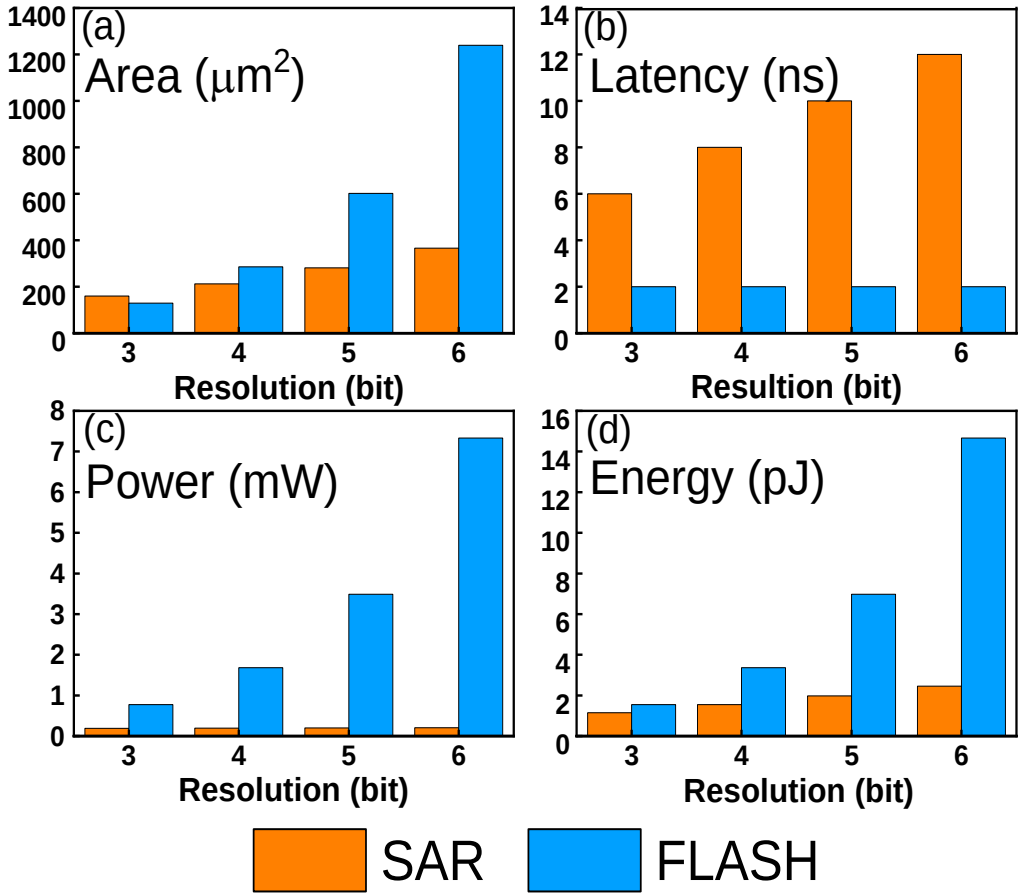


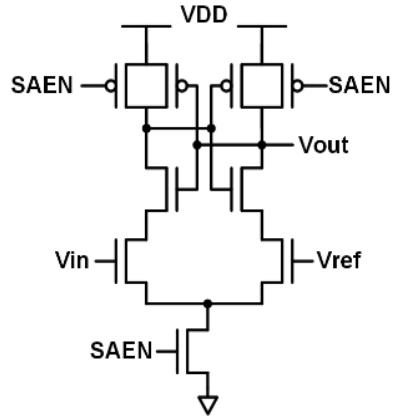
Fig.3 Inference accuracy with different ADC quantization: sweep subarray size and cell precision.

ADC Design Considerations

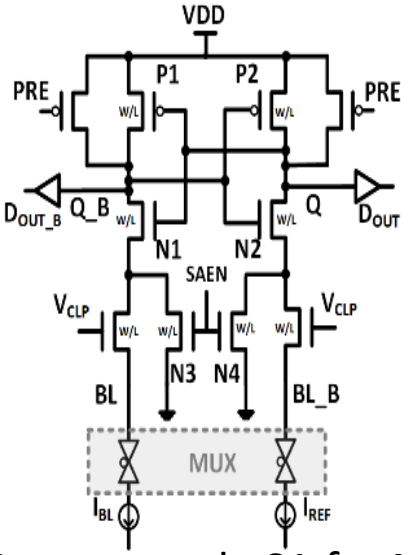


TSMC 40nm PDK





Voltage mode SA
for Flash ADC and SAR ADC



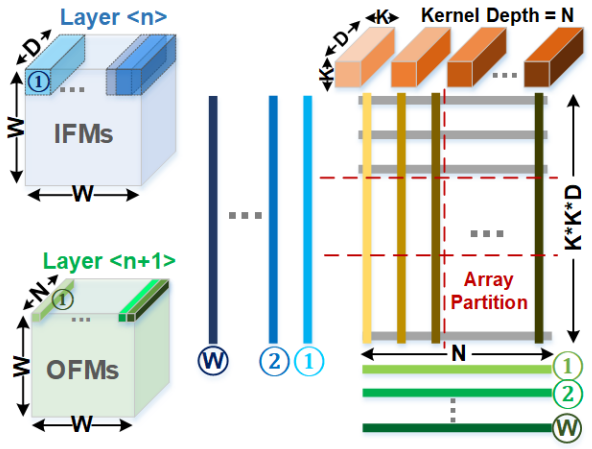
Current mode SA for MLSA

S. Yu. et al. CICC 2020

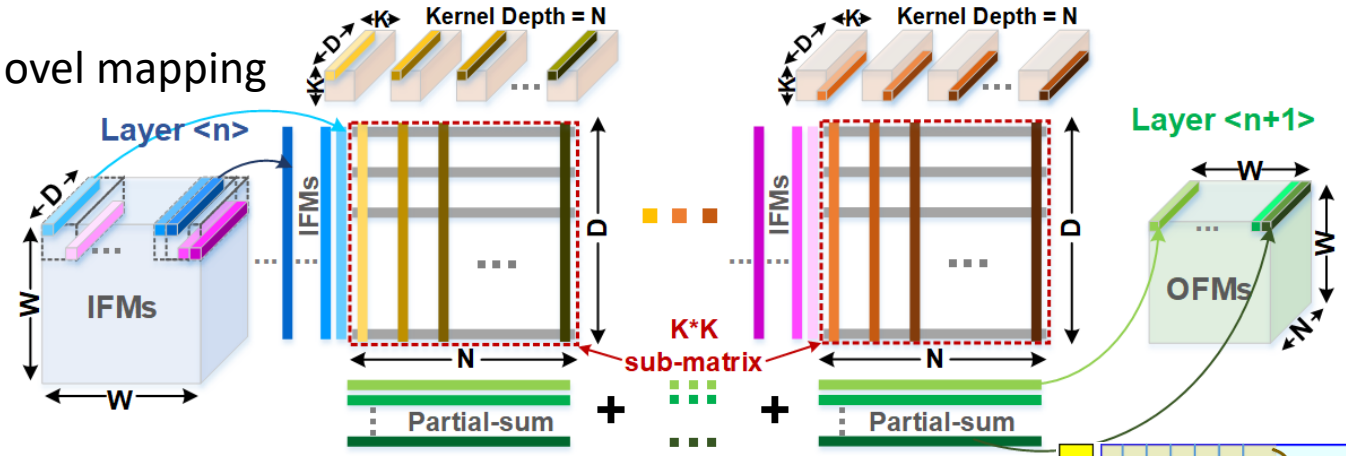
For partial sum precision larger than 4bit, SAR ADC provides better tradeoffs than Flash ADC

Weight Mapping and Duplication for Pipelining

Conventional mapping

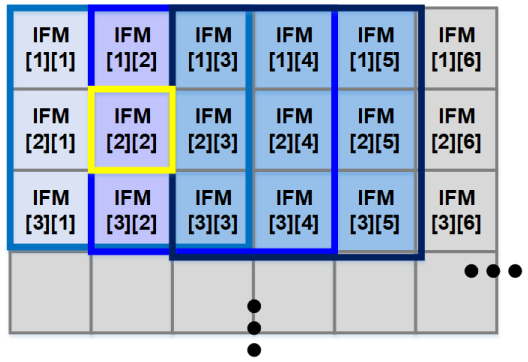


Novel mapping

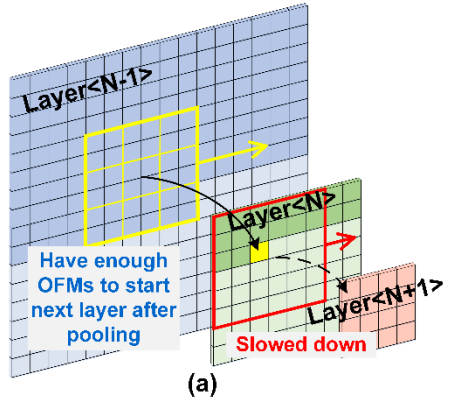


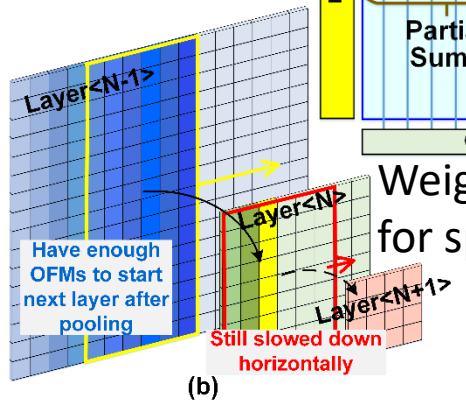
- IFM size = W*W*D, Kernel size = K*K*D*N, OFM size = W*W*N
- Number of sub-matrix = K*K

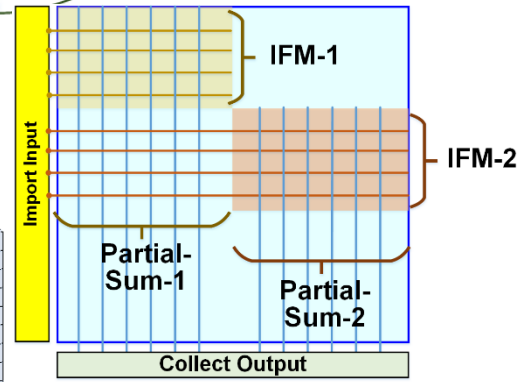
Input reuse in novel mapping



X. Peng TCAS-I 2020



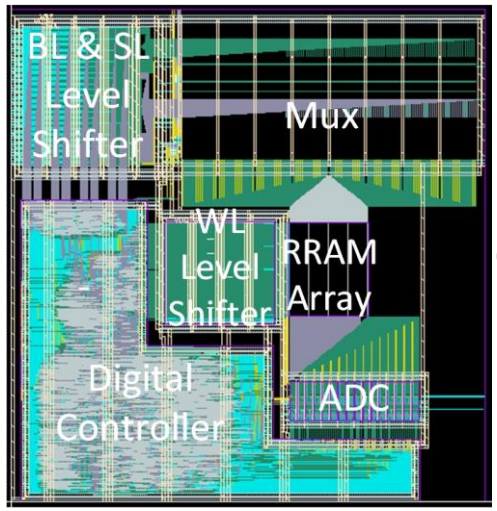




Weight duplication for speed up

NeuroSim Validation with Real Chip Implementation

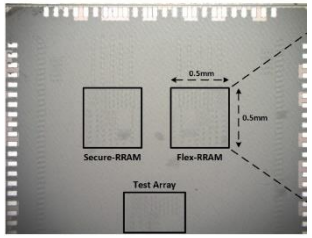
CIM RRAM at TSMC 40nm (W. Li, et al. CICC'2021)



module		area (um^2)		latency	energy (pJ)	
		NeuroSim	Real chip		NeuroSim	Real chip
shift-add	DFF	719	681	1 cycle = 10ns	1.07 × γ = 0.67	0.68
	adder	662	663		1.36 × δ = 0.11	0.10
	inverter	2,133 INV = 1,336	1,334		1.99 × ϵ = 0.18	0.19
accum	DFF	4,968	4,706	1 cycle = 10ns	7.42 × γ = 4.67	4.67
	adder	3,089	3,291		6.28 × δ = 0.50	0.50
	inverter	10,869 INV = 6,808	6,797		10.15 × ϵ = 0.91	0.93
control	DFF	7,743	7,334		11.56 × γ = 7.28	7.28
	inverter	10,485 INV = 6,569	6,558		9.79 × ϵ = 0.88	0.91
total		31,893	31,386		15.20	15.26

Adjustment factors introduced to calibrate:

- transistor sizing
- wiring area
- gate switching activity
- post-layout performance drop



Energy for whole array	NeuroSim	Real chip
pre-layout	3,171.29pJ	10.4TOPS/W → 3,150.8pJ
post-layout	3,171.29pJ×ζ	8.48TOPS/W → 3,864.2pJ

After calibration, the prediction with post-layout simulation error rate is less than 2%.

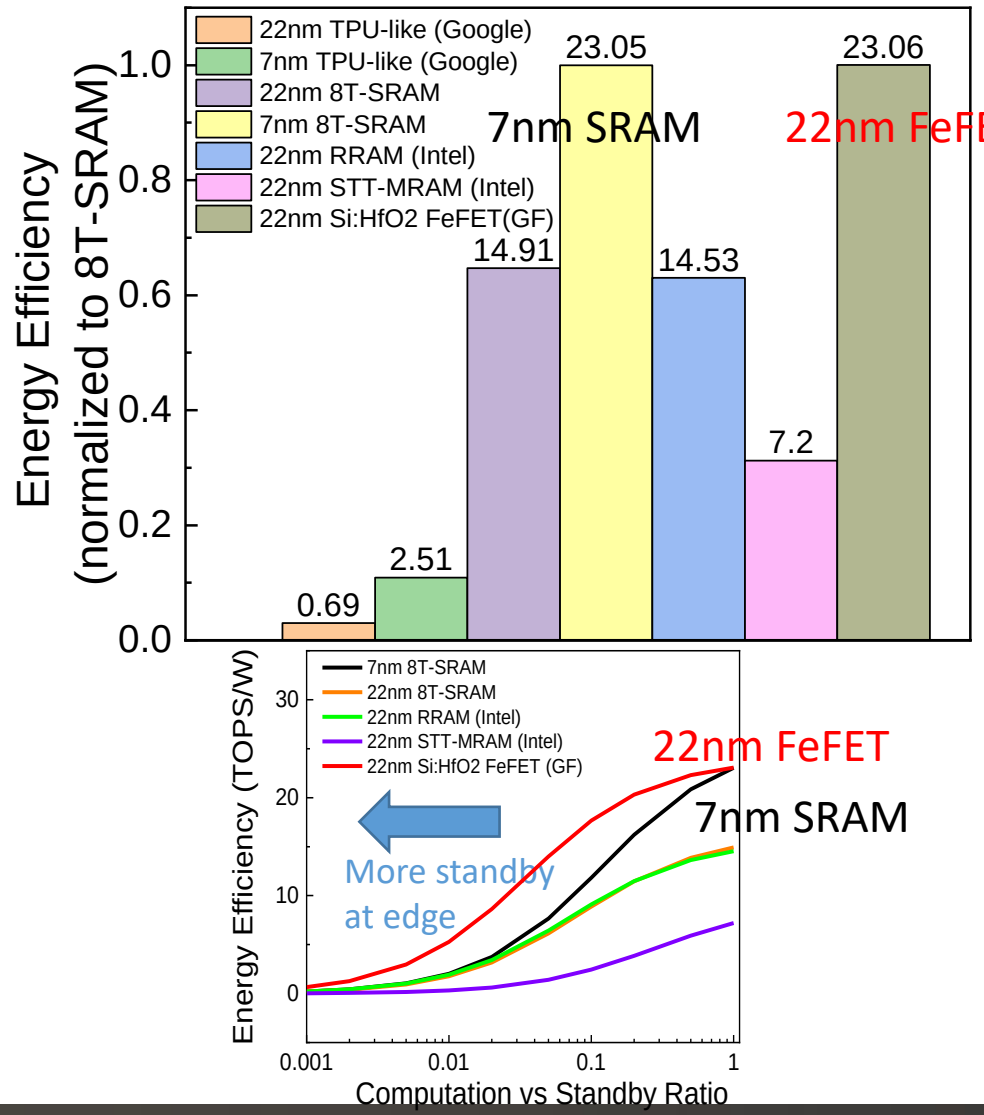
A. Lu, et al. AICAS 2021

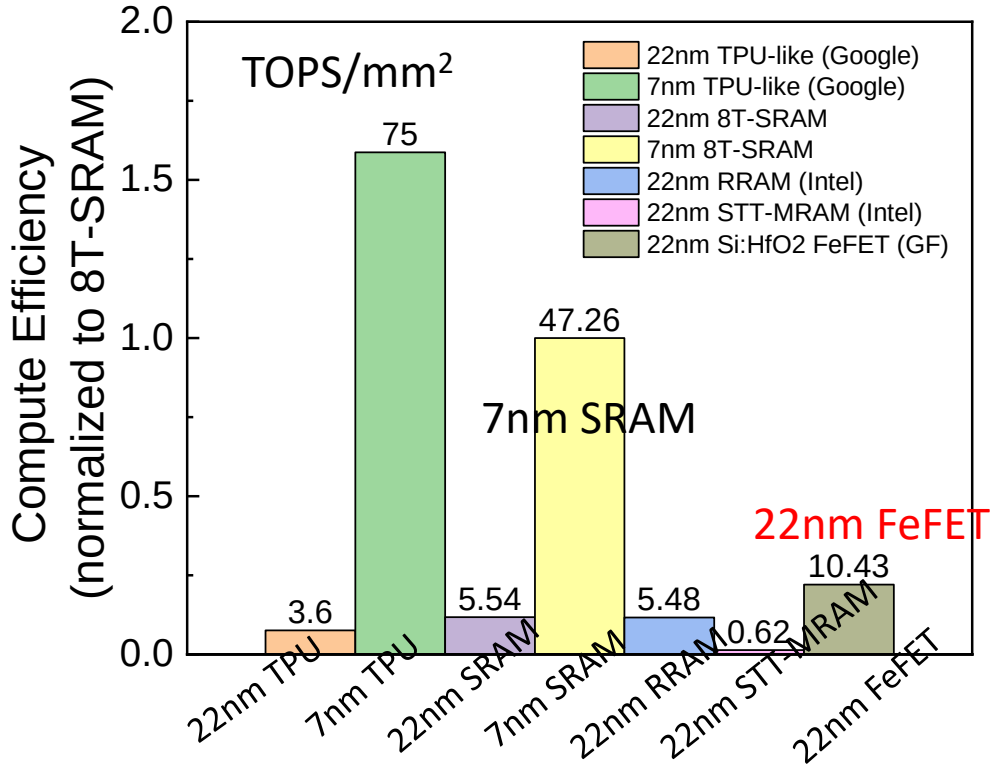
DNN+NeuroSim V1 Example Results for Benchmarking

VGG-8 (8-bit activation; 8-bit weight) on CIFAR10, evaluated with DNN+NeuroSim V1.3							
Technology node (LP)	**7nm**		**22nm**				
Device	**8T-SRAM**	**TPU-like (Google)**	**8T-SRAM**	**RRAM (Intel)**	**STT-MRAM (Intel)**	**Si:HfO2 FeFET (GF)**	**TPU-like (Google)**
Flash-ADC precision	4-bit	8-bit digital	4-bit	4-bit	4-bit	4-bit	8-bit digital
Memory Cell Precision	1-bit	MAC	1-bit	1-bit	1-bit	1-bit	MAC
Ron (Ω)	\	\	\	6k	1.41k	240k	\
Cell area (F^2)	1080	\	280	60	100	40	\
On/Off Ratio	\	\	\	17	2.8	100	\
Area (mm^2)	12.52	15.71	61.92	73.58	57.96	70.34	107.05
L-by-L Leakage power (mW)	2.71	75.96	1.73	1.83	1.08	1.08	5.27
Energy Efficiency (TOPS/W)	23.05	2.51	14.91	14.53	7.20	23.06	0.69
Compute Efficiency (GOPS/mm^2)	47.26	75.00	5.54	5.48	0.62	10.43	3.60

- Memory R_{ON} is the critical factor to determine energy efficiency, desirable target: 100k ~ 1MΩ
- Compute-in-SRAM is a viable solution for high-performance applications, with outstanding energy efficiency and compute density, thanks to the logic scaling.

DNN+NeuroSim V1 Reveals Key Factors in CIM

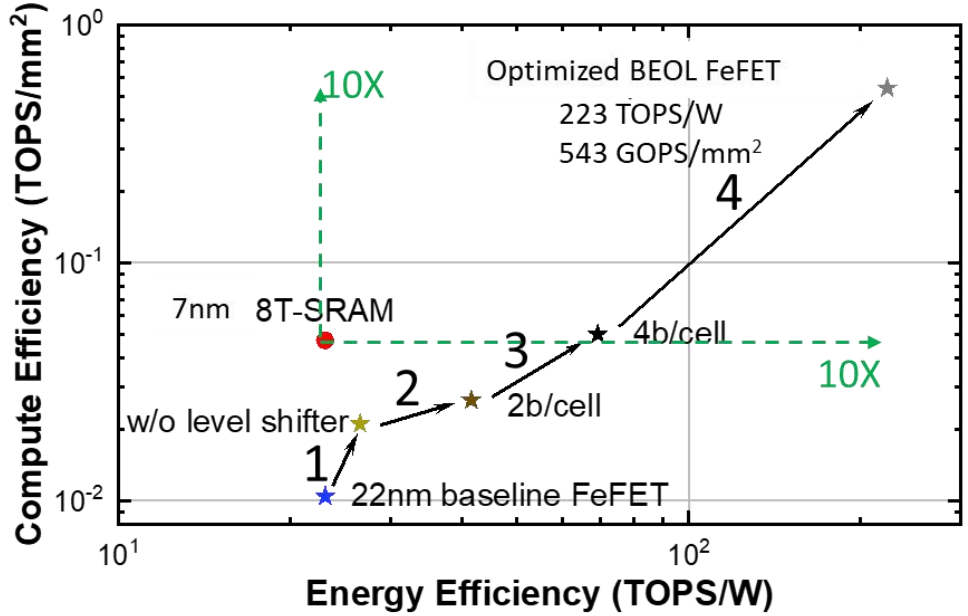


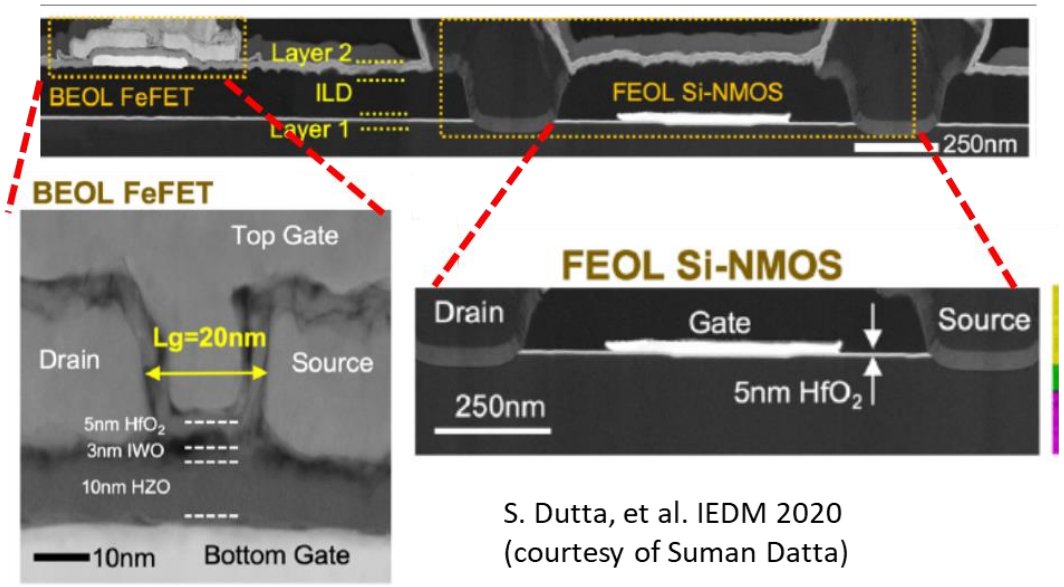


- 7nm SRAM CIM TOPS/W is high, but suffers from leakage when the standby is frequent at edge.
- eNVM is the best option for edge device!
- 22nm FeFET CIM shows good TOPS/W, but not competitive TOPS/mm^2, how to improve?

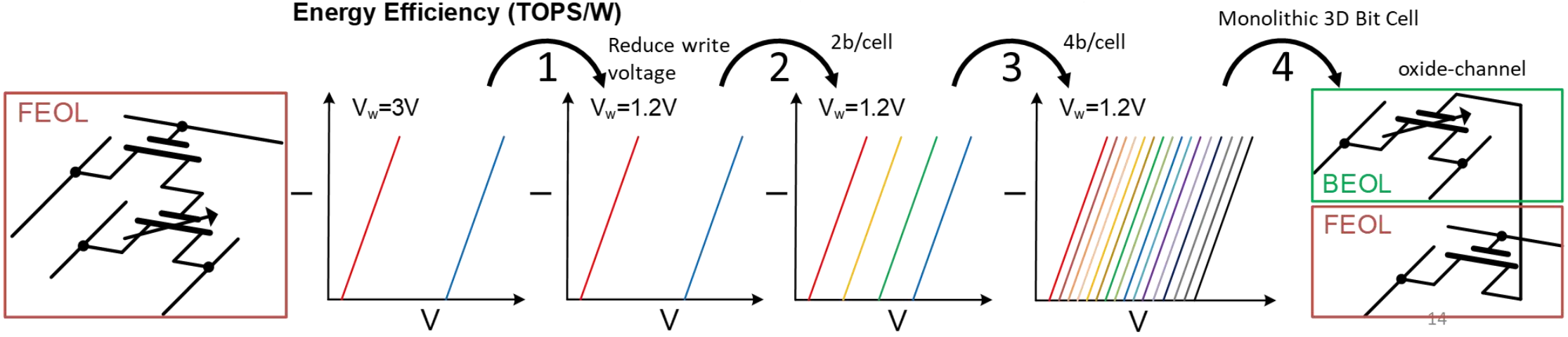
Roadmap for Improving FeFET CIM

End goal: 10X improvement in TOPS/W and GOPS/mm^2 over 7nm SRAM





S. Dutta, et al. IEDM 2020
(courtesy of Suman Datta)

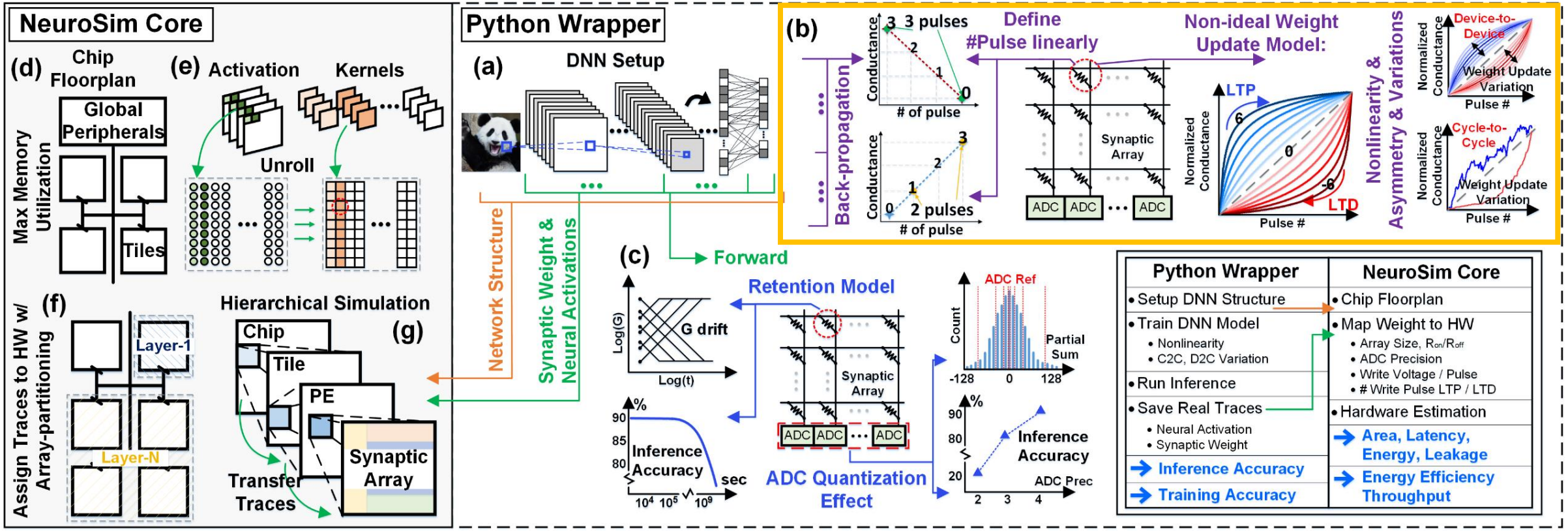


Outline

- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

DNN+NeuroSim V2 for Online Training

- Extends online training hardware and software analysis
- Non-ideal weight-update: asymmetry and non-linearity, device & cycle variation



X. Peng, et al., TCAD'2020.

DNN+NeuroSim V2 Dataflow and Training HW Support

- Transposable synaptic array to support feed-forward & back-propagation (error calculation)
- Add weight gradient calculation (by SRAM-CIM), need frequent data reload (from DRAM)

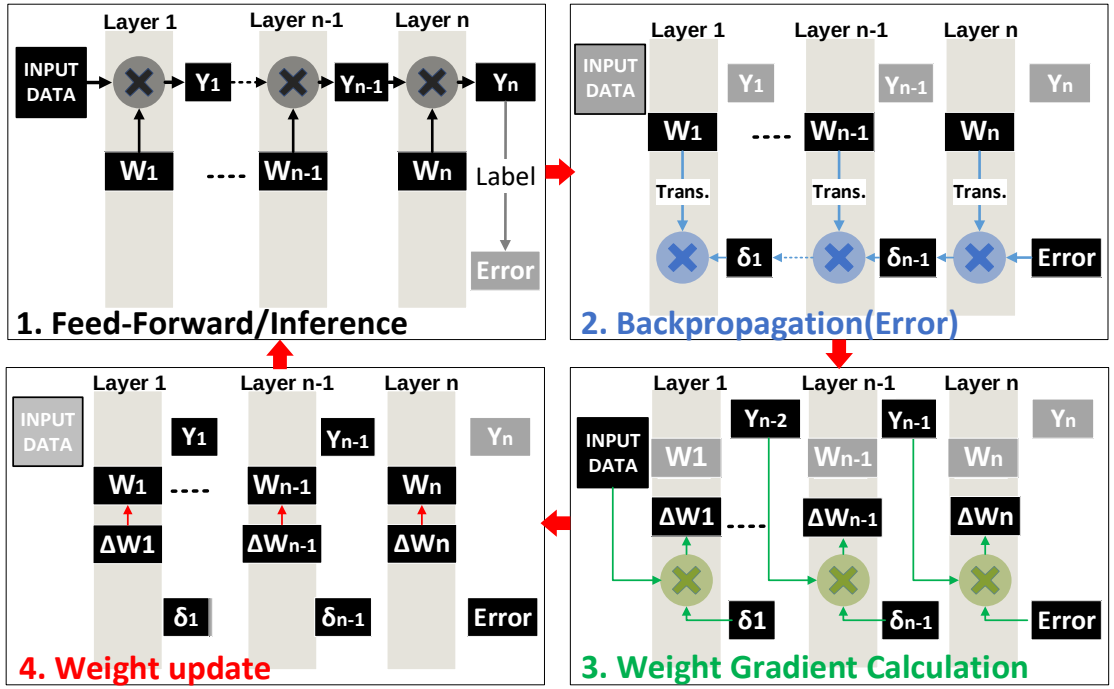


Fig. 1 DNN training data flow.

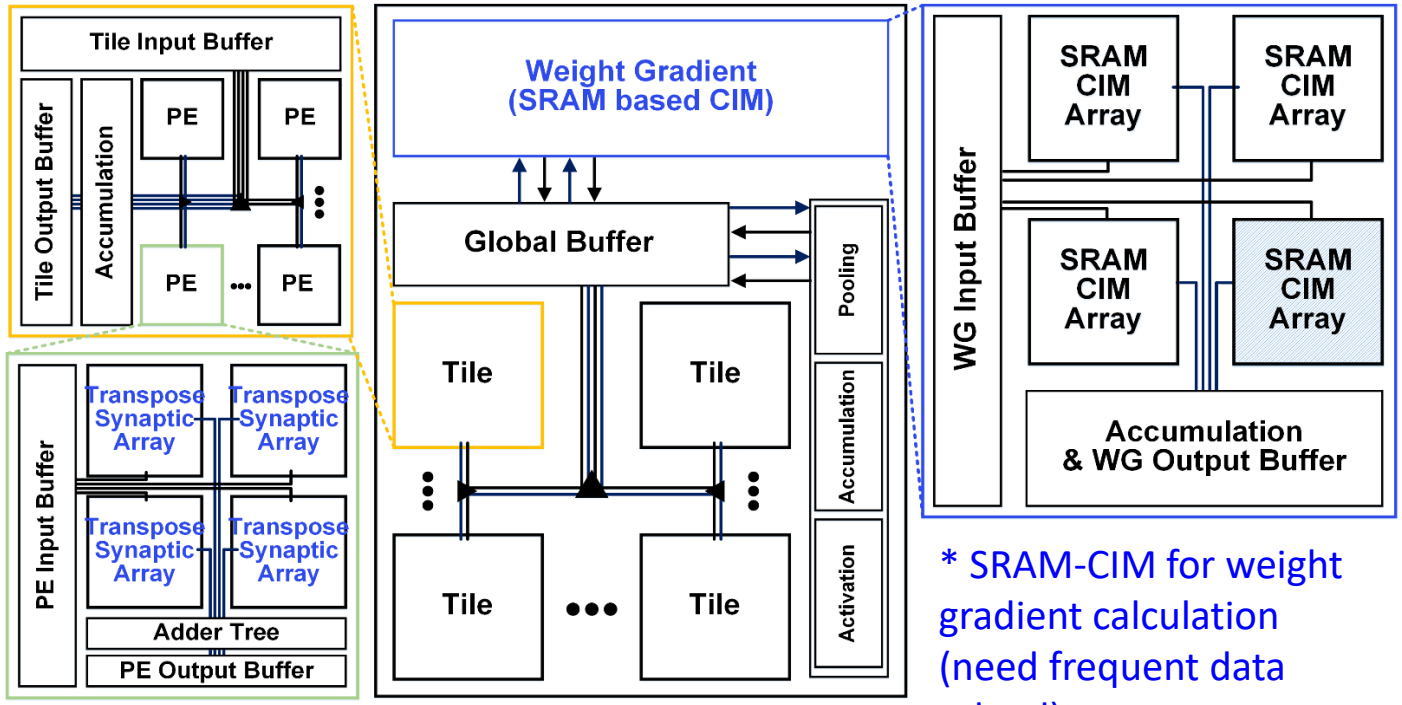
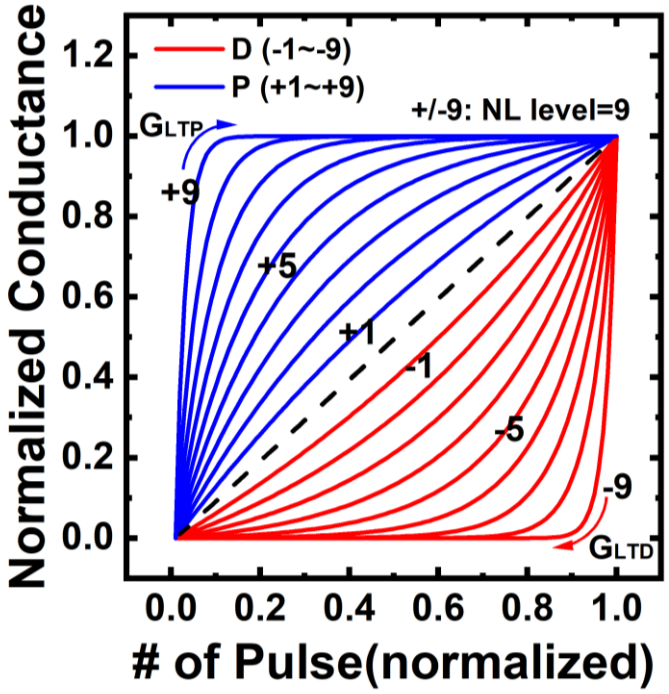


Fig. 2 CIM architecture supports online training.

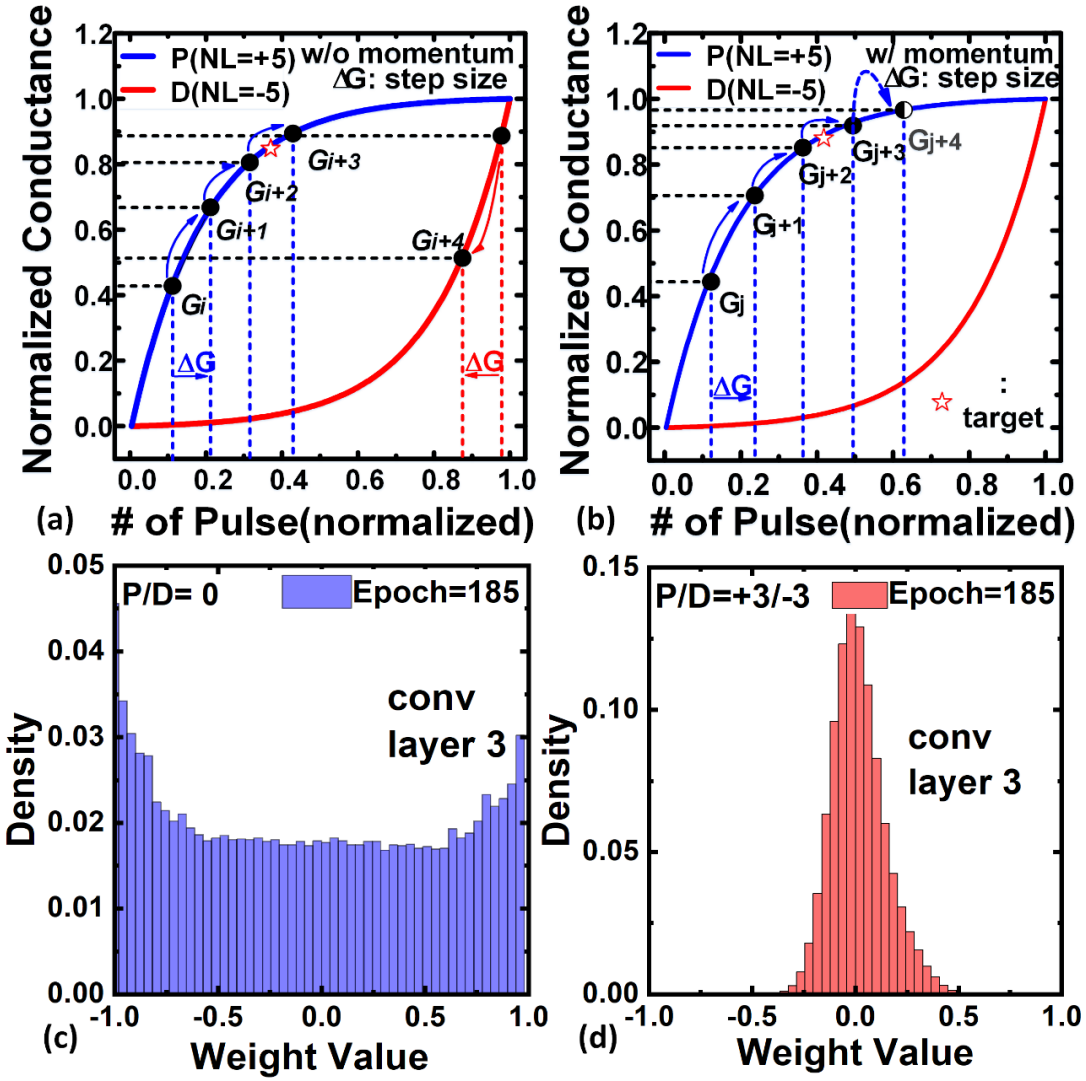
* SRAM-CIM for weight gradient calculation (need frequent data reload)

For batch size=200, VGG-8 creates 2.6 GB, VGG-16 creates 30.4 GB, ResNet-18 creates 3.1 GB intermediate data
→ Too large intermediate data for on-chip cache, thus DRAM access dominates the energy consumption!!

Asymmetry and Nonlinearity in Analog Synapse



- Asymmetry makes devices statistically easier to converge to the middle range of the conductance than approaching Gmax or Gmin.
- Frequent sign flipping causes a large unwanted conductance change towards the middle range.



Online Training: Nonlinearity & Asymmetry

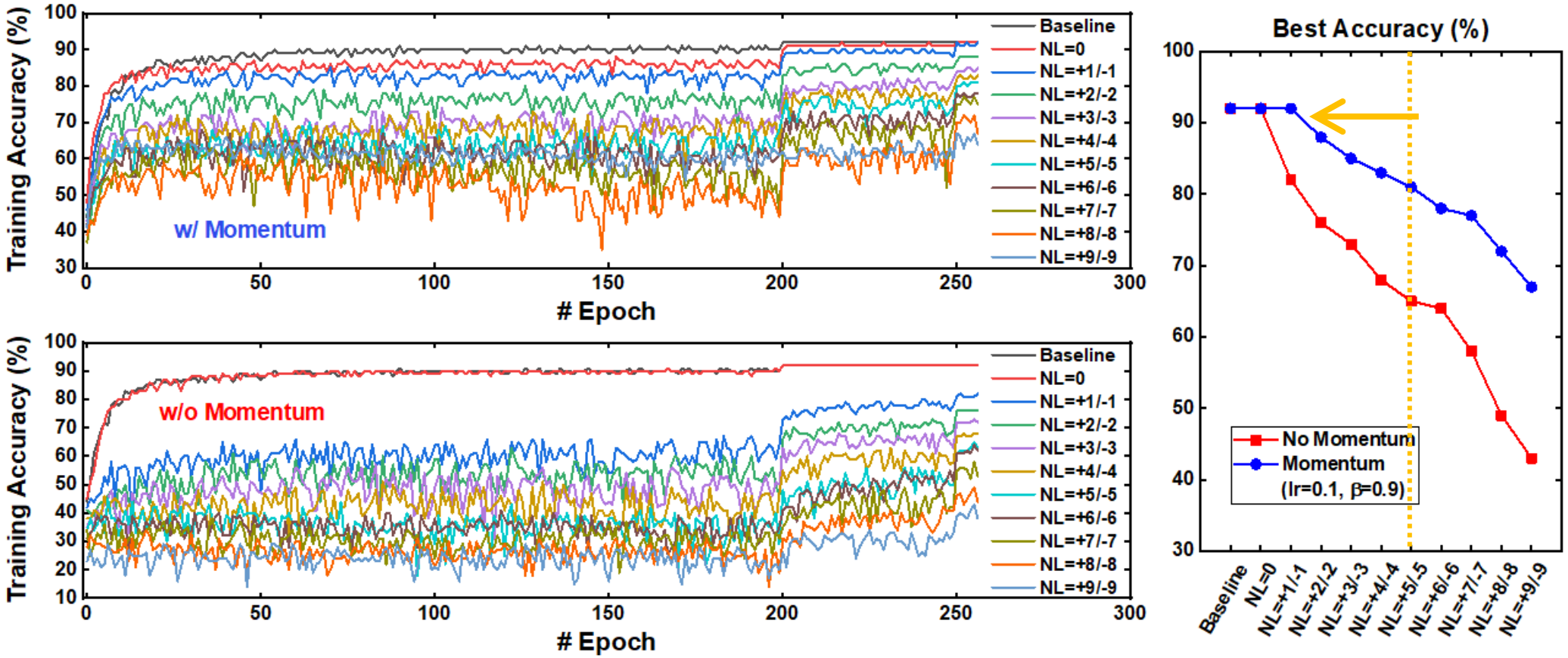


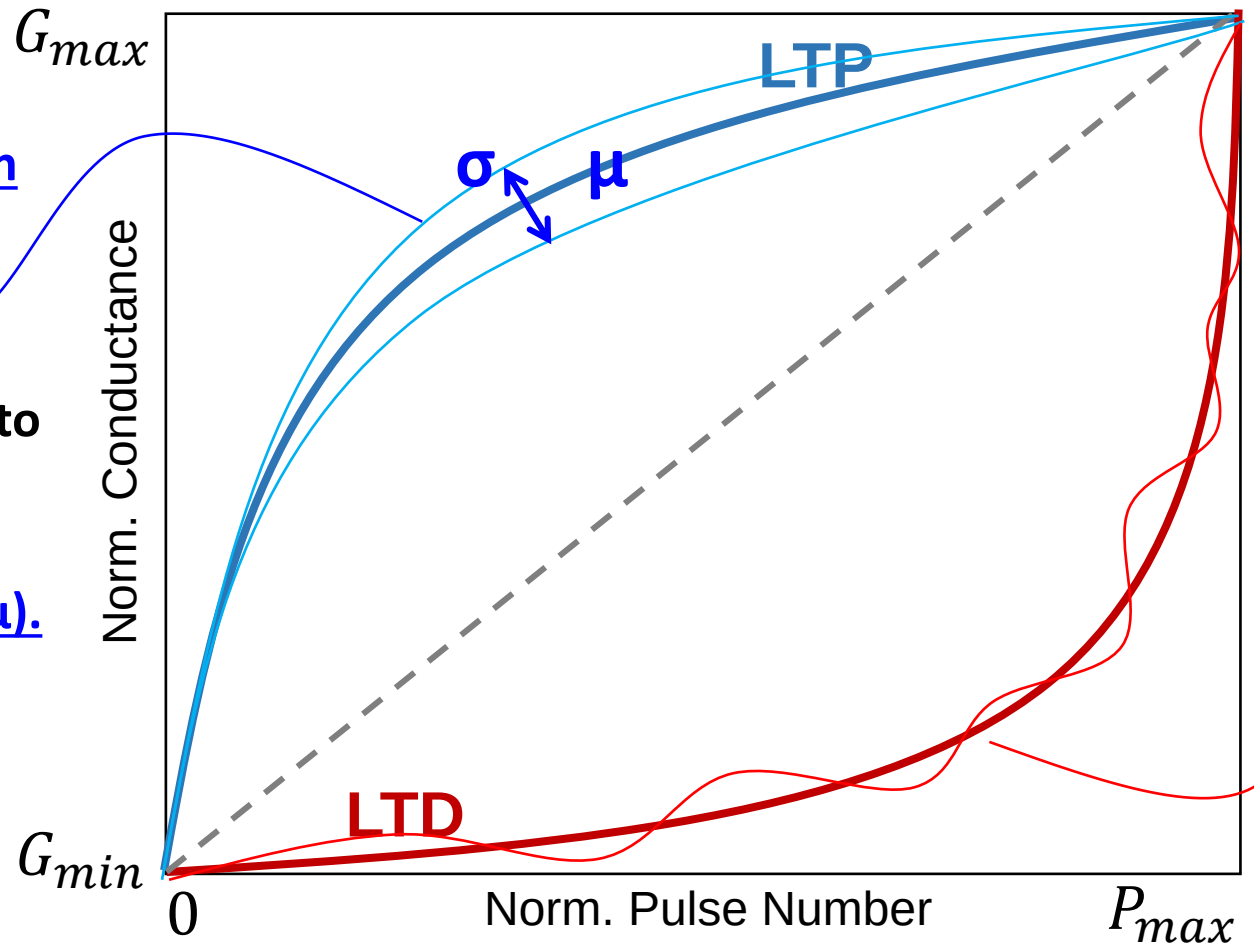
Fig. 1 Analysis of nonlinearity and asymmetry (8-bit VGG-8 for CIFAR-10), w/ and w/o momentum optimization. Results show at +5/-5, the accuracy is still >80% (w/ momentum optimization).

Mommentum optimization: $\Delta W(t) = \beta \Delta W(t-1) + (1-\beta) \cdot (-\frac{\partial L}{\partial W})$

Device & Cycle Variation Behavior Model

Device-to-device variation introduce different nonlinearity (NL).

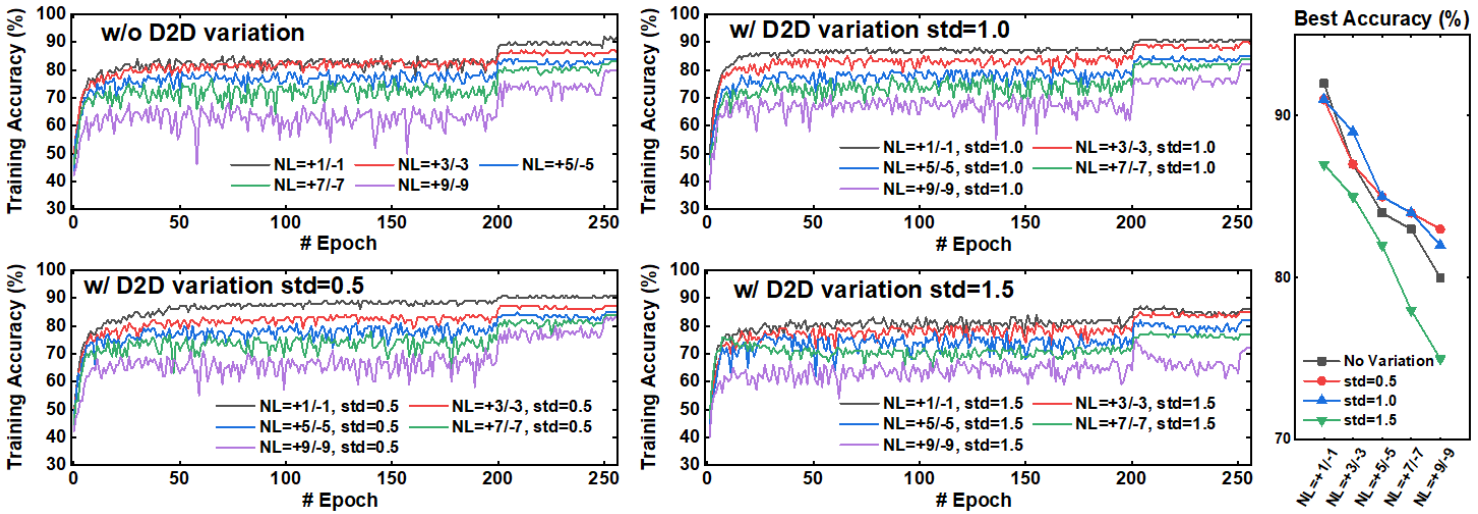
Randomly generates NLs to different synapse with a **standard deviation (σ)** respect to the **mean NL (μ).**



Cycle-to-cycle variation introduce variation in conductance change at every weight-update.

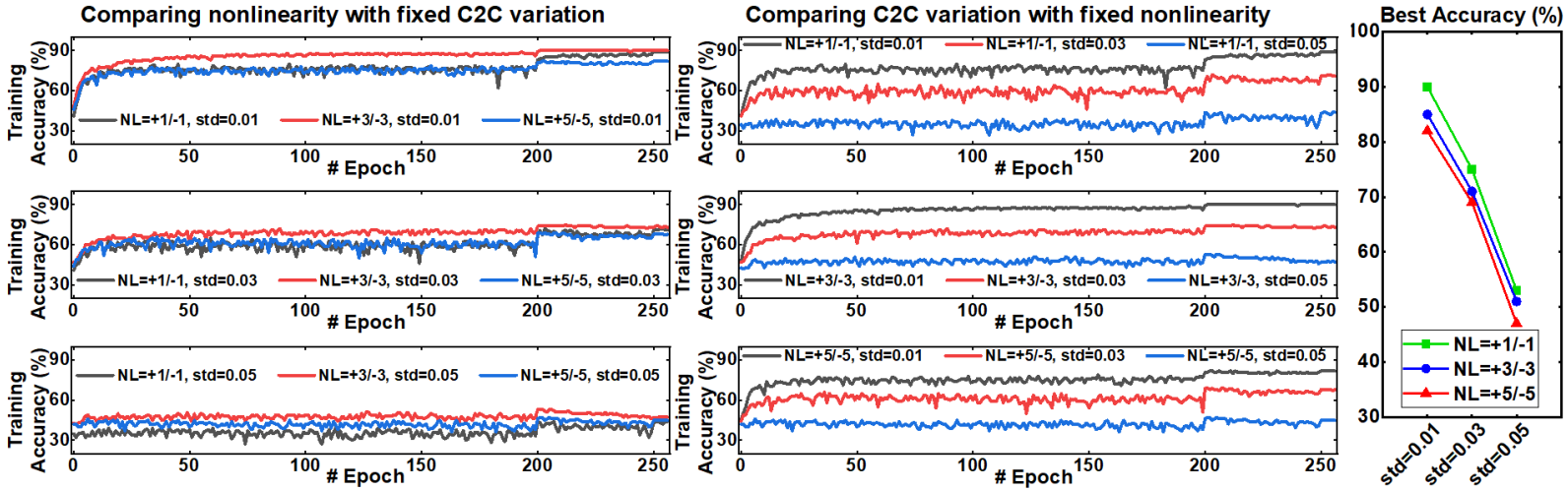
Express C2C variation **standard deviation (σ)** in terms of percentage of entire weight range.

Online Training: Device & Cycle Variation

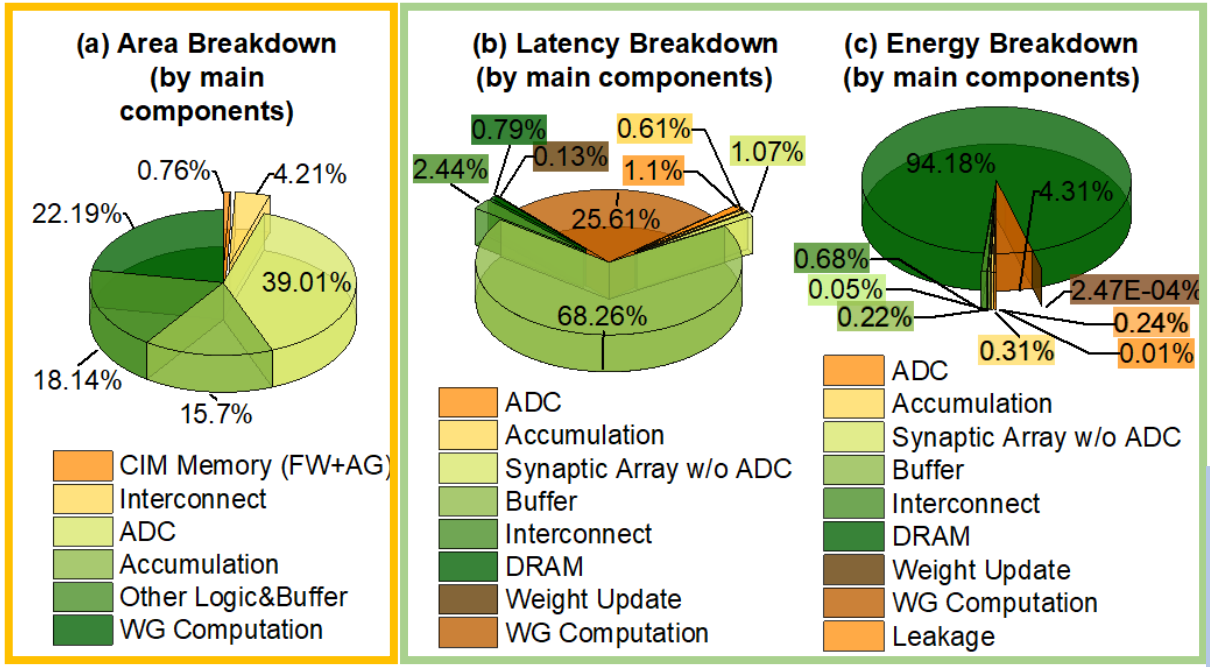


Device-to-device variation could be tolerated to certain degree, but accuracy will drop gradually with increasing variation.

Cycle-to-cycle variation could be tolerated to certain degree, but accuracy will drop quickly beyond the threshold.



Online Training: System Performance



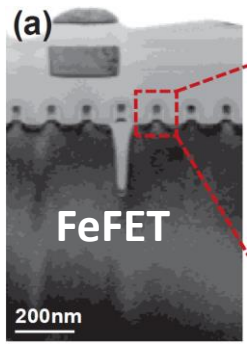
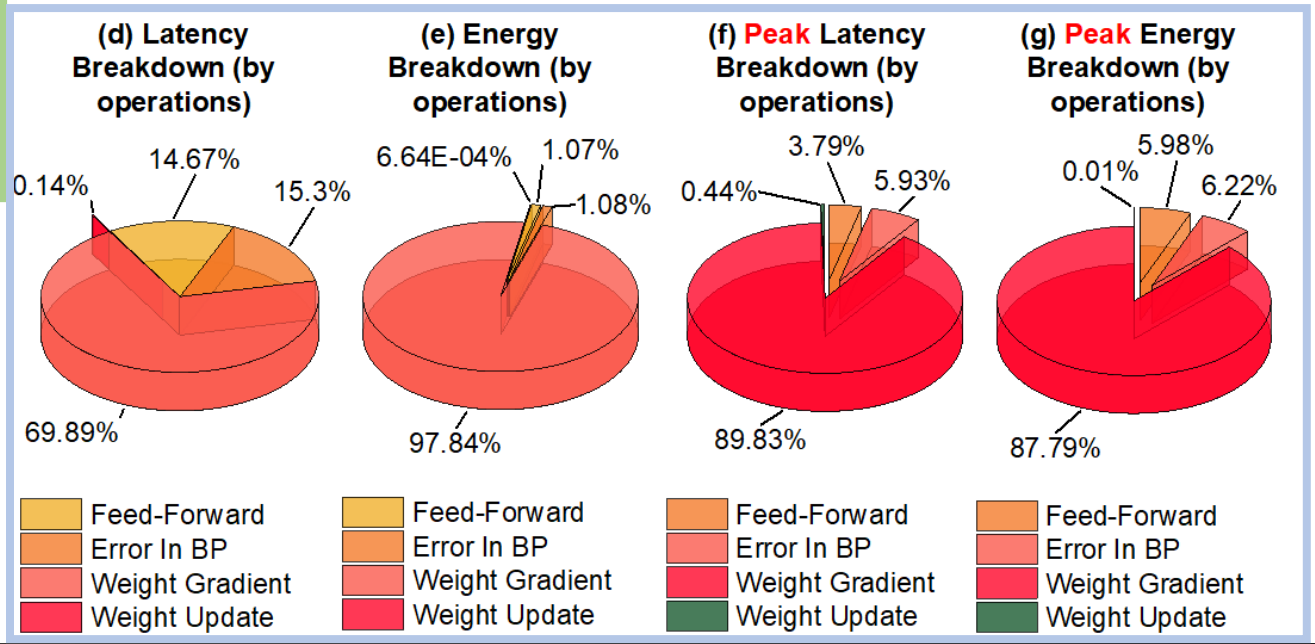
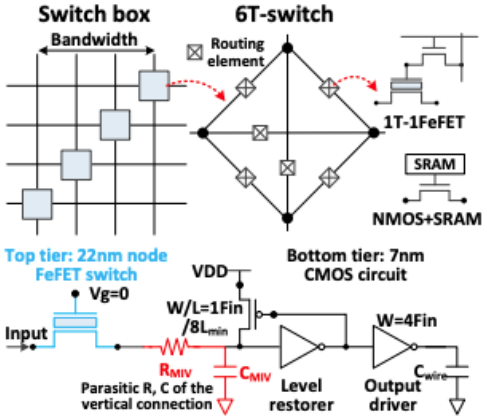


Fig. The data shows the 100th epoch of FeFET-based CIM online training accelerator. (a) area breakdown by main components; (b) latency and (c) energy breakdown by main components; (d) latency and (e) energy breakdown by operations; (f) peak latency and (g) peak energy breakdown by operations.

- *ADC (6-bit) & weight gradient units (SRAM-based)* dominant in total area
- *Buffer latency & DRAM energy* is the bottleneck of performance
- Weight gradient computation is the bottleneck in the entire learning *(require frequent DRAM access)*

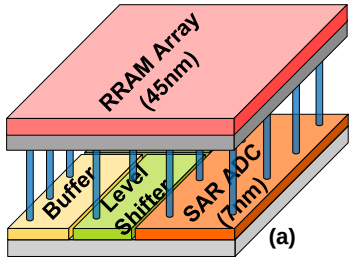


NeuroSim Extension



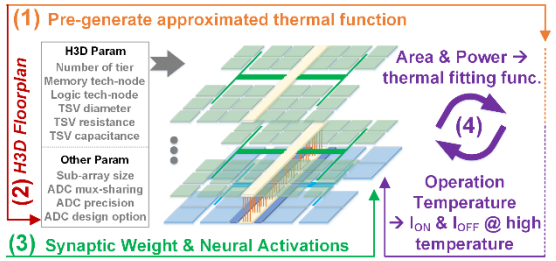
One custom chip supports various DNN models that may have larger on-chip memory capacity than the chip could hold

Reconfigurable NeuroSim *[p1]*



Monolithic 3D partition between memory at legacy node with BEOL oxide transistors and peripheral logic at advanced 7nm node

Monolithic 3D NeuroSim *[p2]*

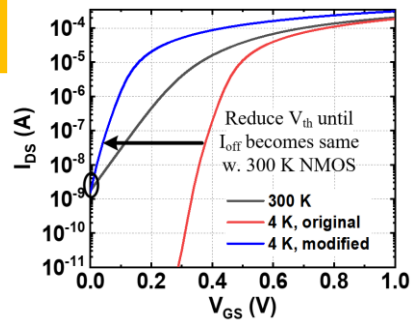


3D stacking with multi-tier memory with logic-tier by hybrid bonding and micro-TSV. Thermal modeling supported

Heterogeneous 3D NeuroSim *[p3]*

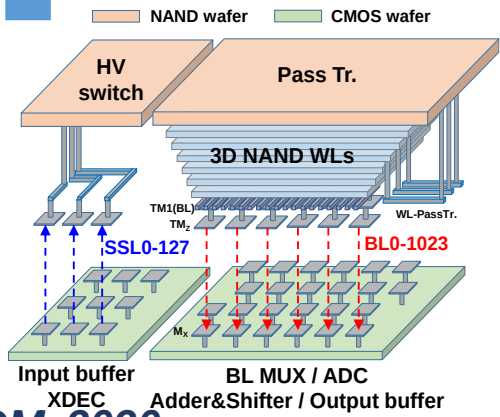
77K and 4K cryogenic transistor technology files calibrated for data-center computing and quantum peripheral control

Cryogenic NeuroSim *[p4]*



3D NAND based architecture for GB-level model for language, graph, genome, and recommendation system.

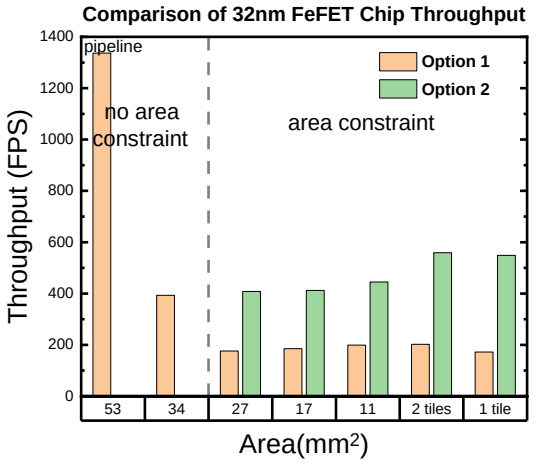
3D NAND NeuroSim *[p5]*

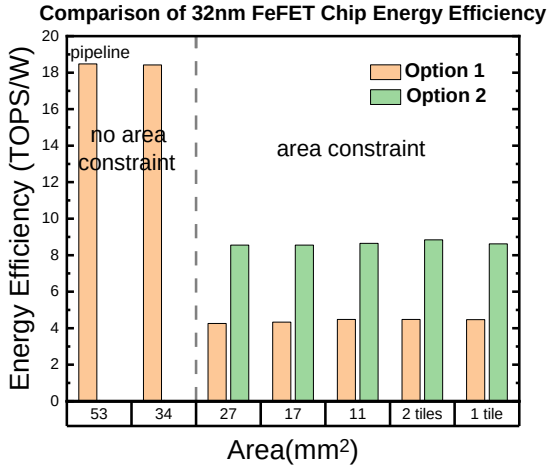


[p1] A. Lu et al., DATE 2020, Y. Luo et al. IEDM 2021; [p2] X. Peng et al., IEDM, 2020
[p3] X. Peng et al., TED, 2021; [p4] P. Wang el al., ISCAS, 2021; [p5] W. Shim et al., EDL 2021

Reconfigurable CIM Design

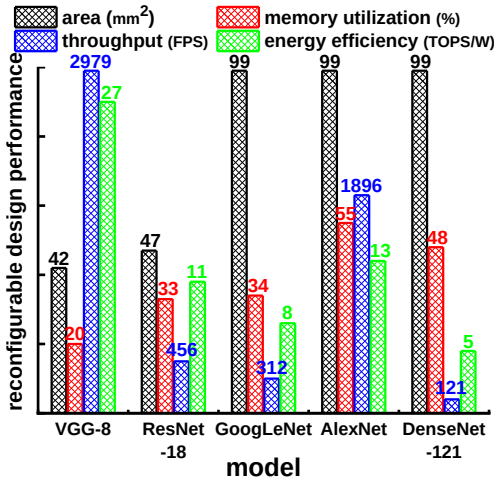
If chip area is constrained, off-chip weight reloading has two options:
Option 1: reload weights (with sequential processing)
Option 2: reload inputs (with batch processing)

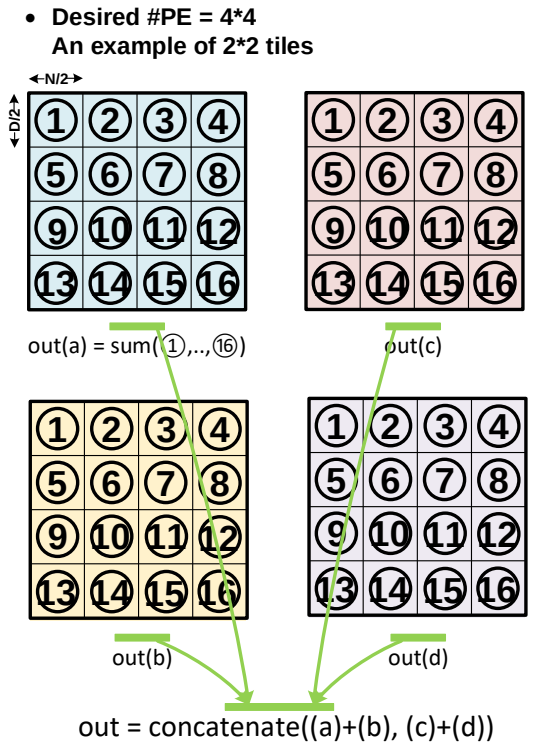




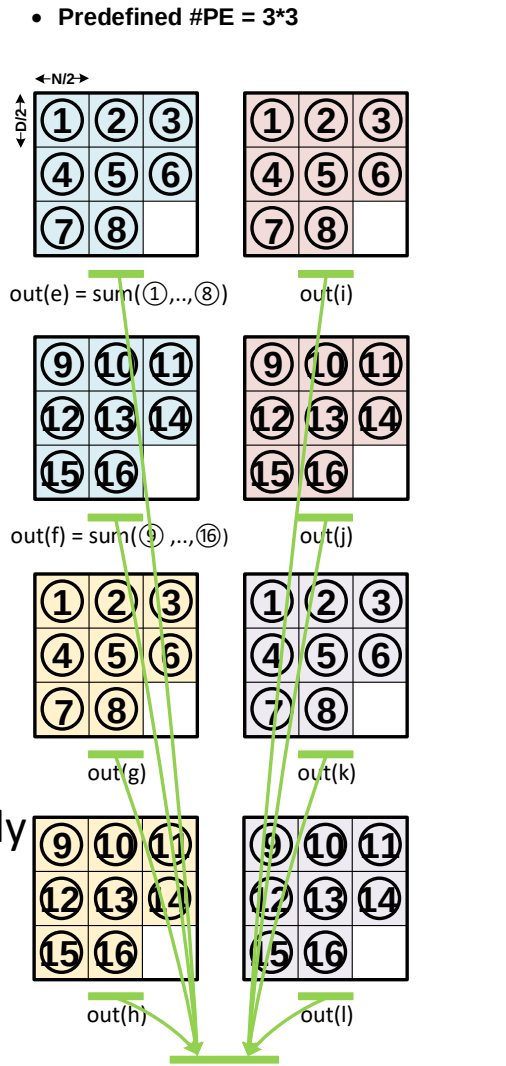
Go off-chip significantly lowers the throughput and energy efficiency ☹

A. Lu et al., T-VLSI, 2021 & DATE 2020

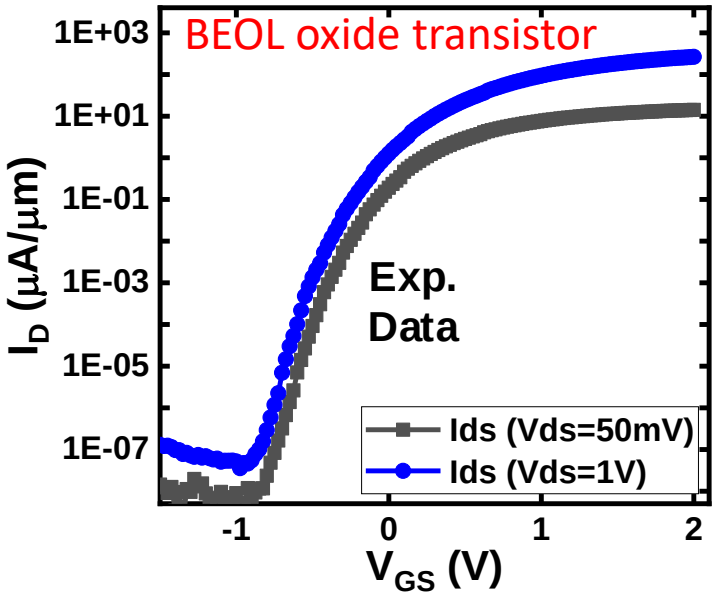




Partial sum: vertically accumulated and horizontal concatenated



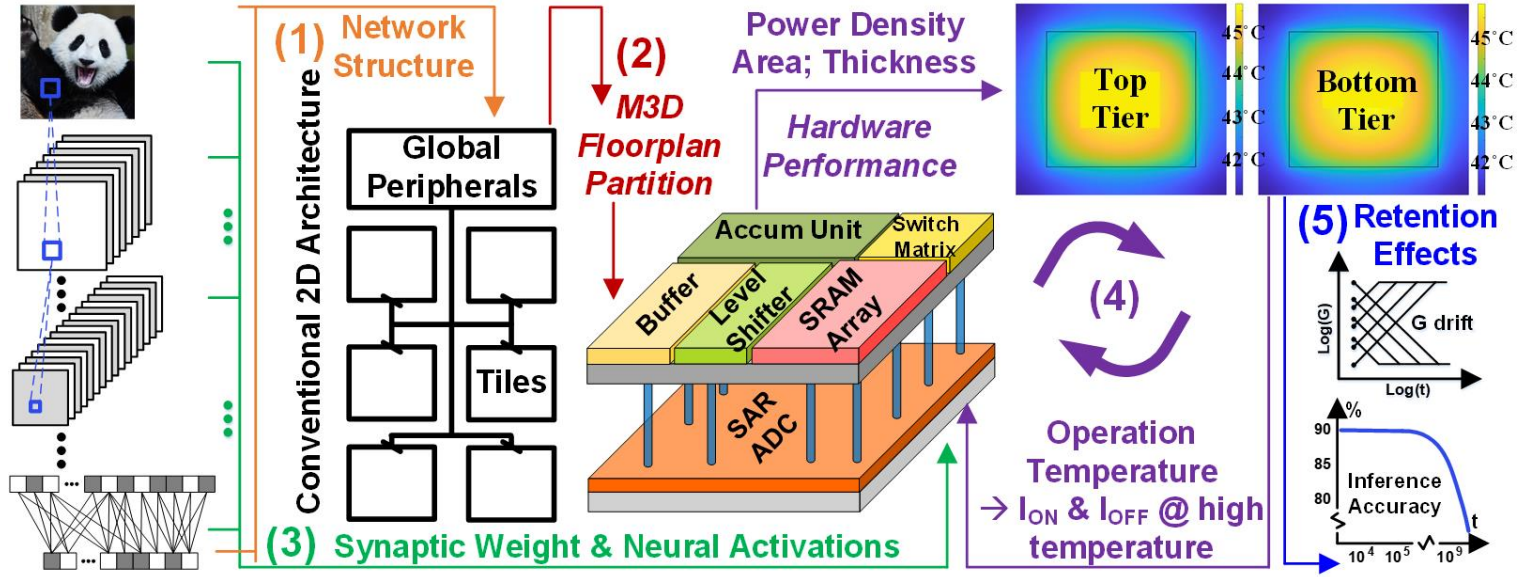
Monolithic 3D



Partition the design into M3D:

- **Si FEOL for logic and ADC modules at 7nm**
- **Oxide BEOL for memory and its periphery at 45nm**

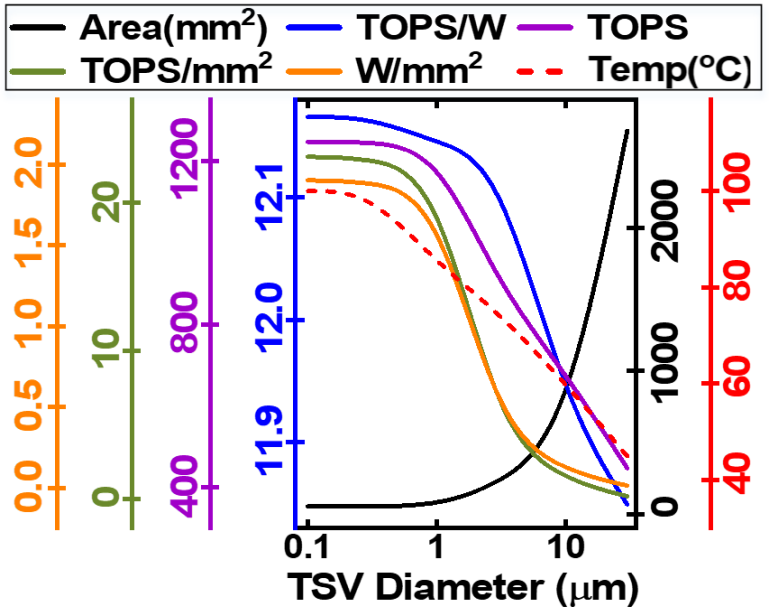
X. Peng et al., IEDM, 2020

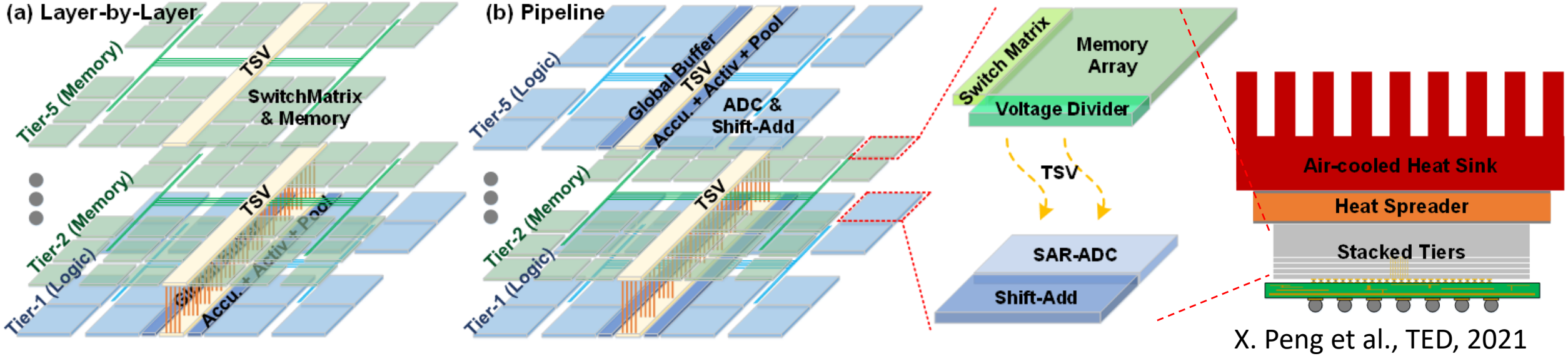


	Integration	2D			3D			
Architecture Specs	[LSTP] Tech node (nm)	45 (a)	7	7 (b)	45Si+45 (c)	45Ox+45 (d)	45Ox+7 (e)	7Si+7 (f)
	Device	RRAM		SRAM	RRAM			SRAM
	ADC precision	5-bit		4-bit	5-bit			4-bit
	Cell Precision	2-bit		1-bit	2-bit			1-bit
	Ron (Ω)	6 K		-	6 K			-
	On/Off Ratio	150		-	150			-
	numColMuxed	8			1			
Performance	Area (mm^2)	163.05	3.61	8.36	269.27	269.27	14.50	13.06
	Energy Efficiency (TOPS/W)	8.58	77.98	30.30	14.46	15.59	81.34	36.48
	Throughput (TOPS)	0.56	1.97	1.95	2.43	2.54	5.50	5.56
	Power Density (W/mm^2)	4.00E-04	7.00E-03	7.72E-03	6.23E-04	6.05E-04	4.66E-03	1.17E-02
	Via Density ($\#/mm^2$)	-	-	-	3.85E+03	3.85E+03	1.07E+05	1.56E+05

Heterogeneous 3D

- TSV and hybrid bonding for 5-tiers
- Two Scheme: “Layer-by-Layer” vs. “Pipeline”
- SRAM: both logic and memory tier @ 7nm
- RRAM: 2-bit/cell, RON is 6kΩ (on/off=150), memory tier @ 22nm, logic tier @ 7nm
- TSV diameter’s sweet spot: 1~3um

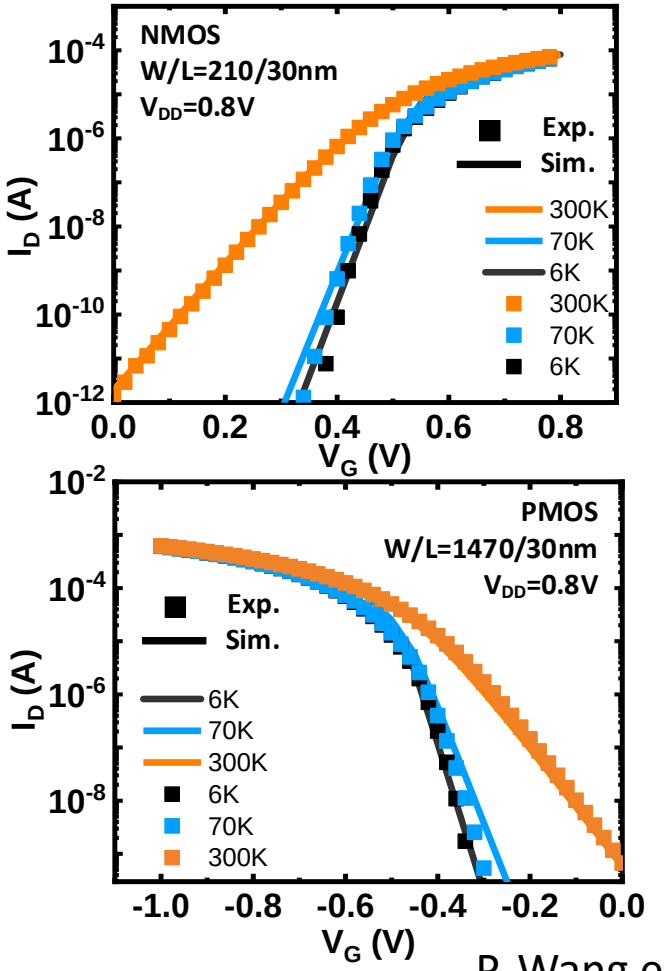




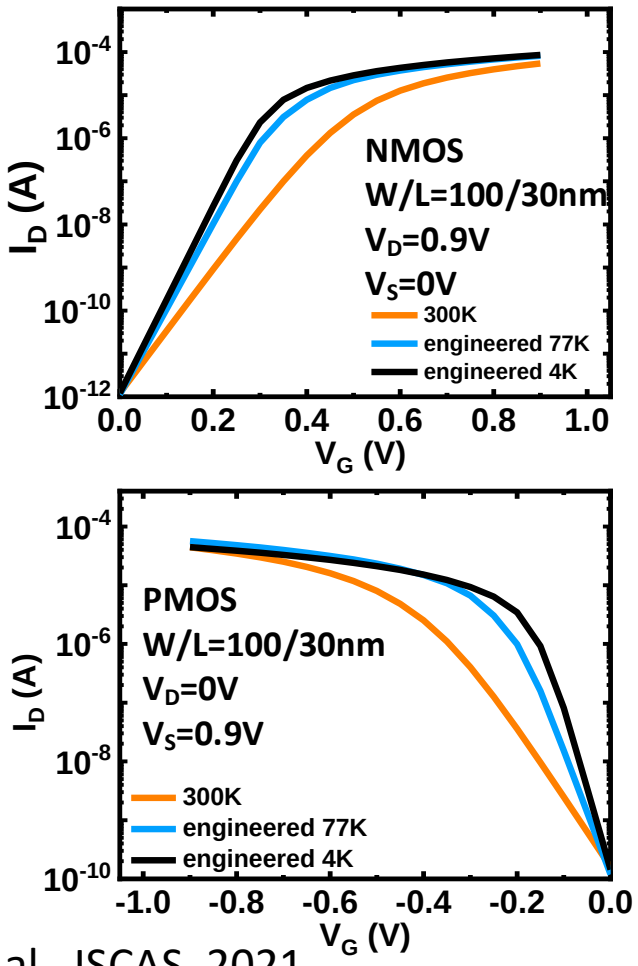
X. Peng et al., TED, 2021

Cryogenic NeuroSim

Exp Data

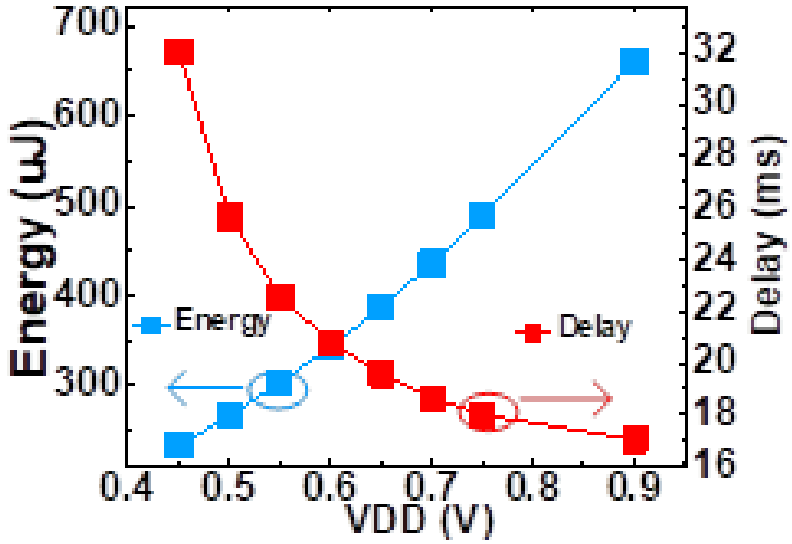


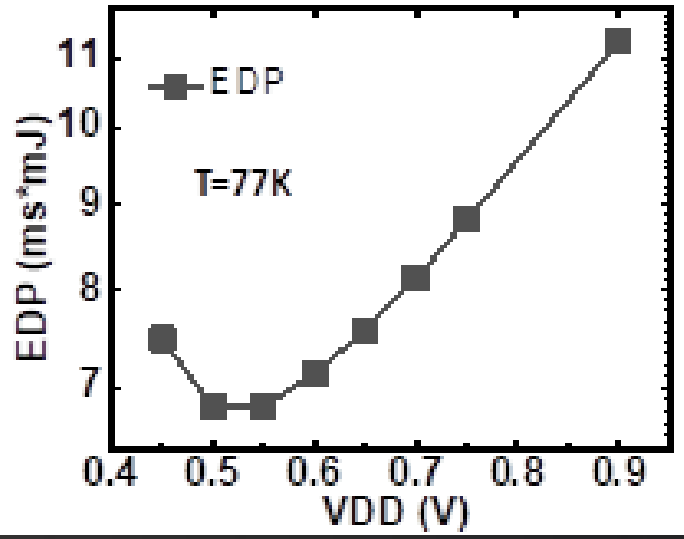
Vth engineering



P. Wang el al., ISCAS, 2021

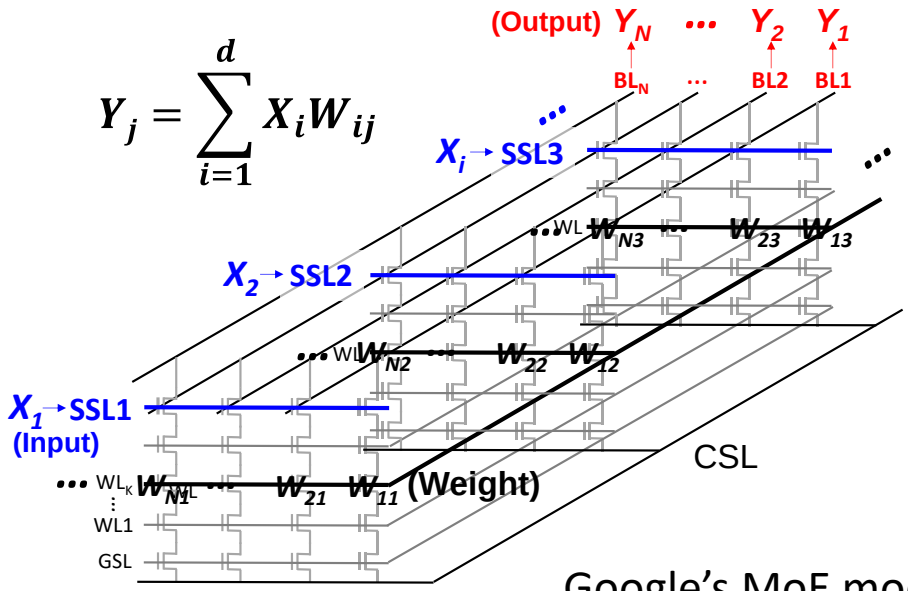
SRAM based CIM accelerator



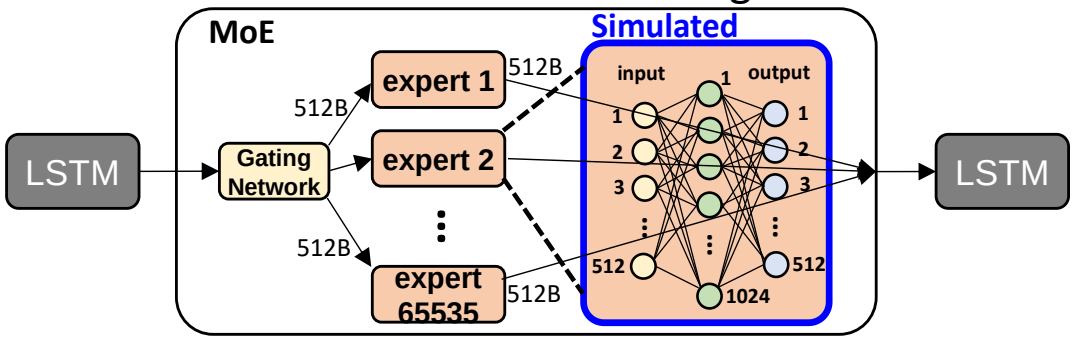


3D NAND based GB model

$$Y_j = \sum_{i=1}^{d} X_i W_{ij}$$

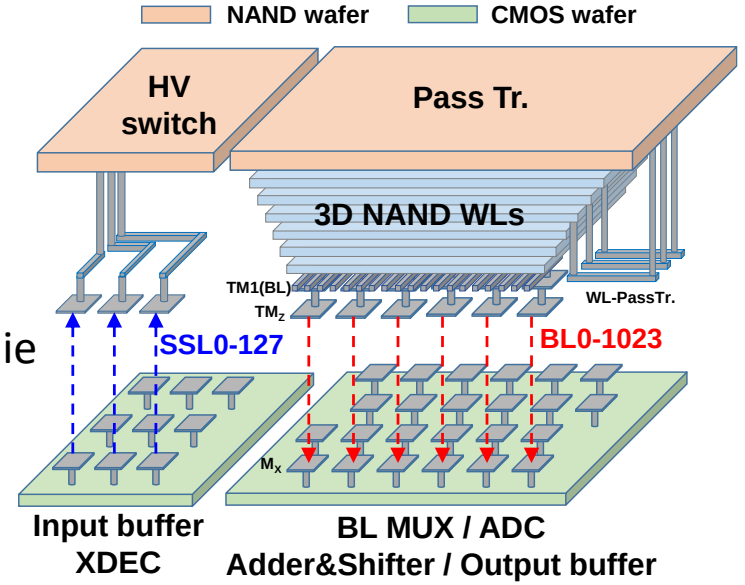


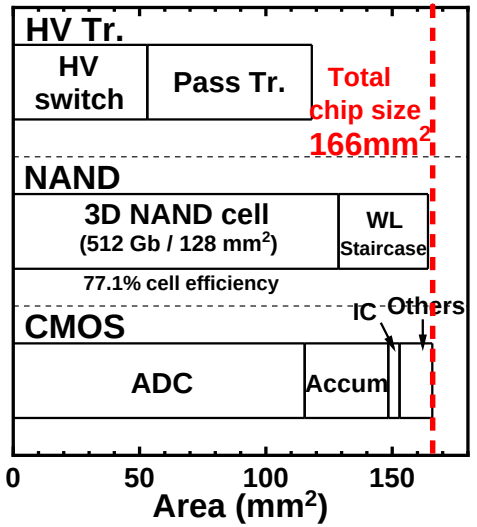
Google's MoE model

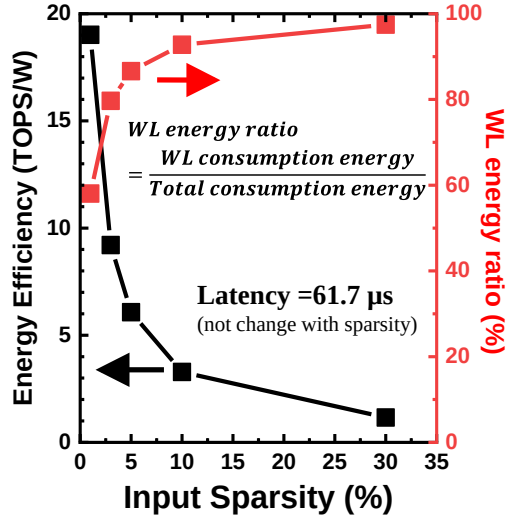


W. Shim et al., EDL 2021

Hybrid bonding with 7nm logic die







Outline

- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

NeuroSim Extension Roadmap

Projection of CIM hardware towards advanced technology nodes

- Technology update to 1 nm node *[2023 July]*
- NeuroSim +Timeloop *[TBD]*
- NeuroSim +GEM5 *[TBD]*
- Chiplet based NeuroSim *[TBD]*

Logic technology is progressing towards angstrom era
Will extend the support from 7nm to beyond 1nm node
Key features to be captured in NeuroSim technology libraries:

- Fin depopulation and standard cell height reduction
- DTCO and backside power delivery
- Transition from FinFET to GAA stacked nanosheet
- Further support 3D CFET (NMOS on top of PMOS)

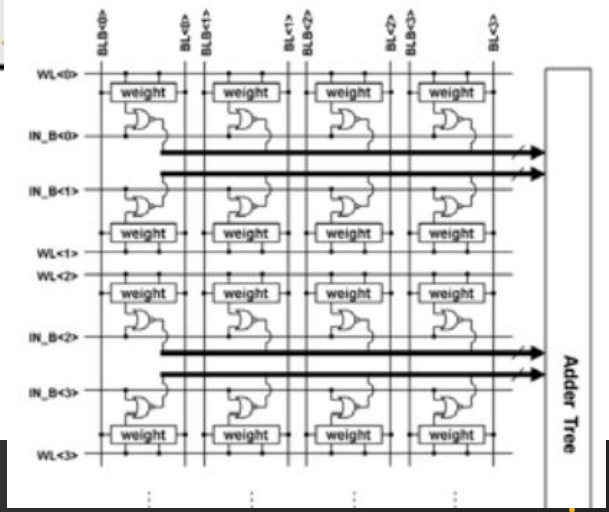
With continued scaling, NeuroSim could support all Digital-based CIM (Digital CIM, DCIM) (as proposed by TSMC)


ISSCC 2021 / SESSION 16 / COMPUTATION IN MEMORY

16.4 An 89TOPS/W and 16.3TOPS/mm² All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications

Yu-Der Chih, Po-Hao Lee, Hidehiro Fujiwara, Yi-Chun Shih, Chia-Fu Lee, Rawan Naous, Yu-Lin Chen, Chieh-Pu Lo, Cheng-Han Lu, Haruki Mori, Wei-Chang Zhao, Dar Sun, Mahmut E. Sinangil, Yen-Huei Chen, Tan-Li Chou, Kerem Akarvardar, Hung-Jen Liao, Yih Wang, Meng-Fan Chang, Tsung-Yung Jonathan Chang

TSMC, Hsinchu, Taiwan

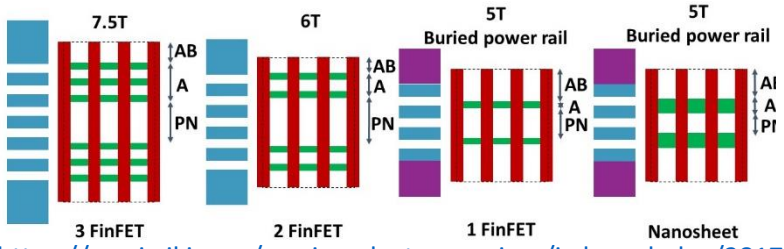


Technology Extension to 1nm node

Standard Cell Dimension Trend

	Technology Node (nm)						
	14	10	7	5	3	2	1
# of fins per standard cell (NMOS + PMOS)	4+4	3+3	2+2	2+2	2+2	1+1	1+1
Cell height (nm)	576	330	240	180	144	114	80
Contacted poly pitch (nm)	78	64	57	51	48	45	40
PN separation length (nm)	136	100	83	64	45	40	15
Fin pitch (nm)	48	36	30	28	24	26	24
Barrier thickness for M0-M2 (nm)	2.5	2.5	2.5	2.0	1.5	0.5	0.5

Standard cell height scaling limited by PN separation



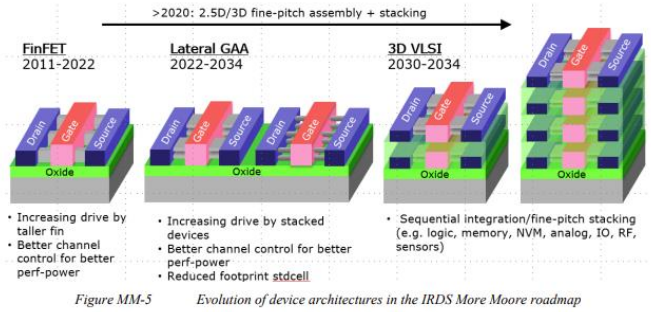


Figure MM-5 Evolution of device architectures in the IRDS More Moore roadmap

https://semiwiki.com/semiconductor-services/ic-knowledge/281778-iedm-2019-imec/

https://irds.ieee.org/images/files/pdf/2021/2021IRDS_MM.pdf

- The standard cell assumptions in NeuroSim capture the actual area scaling trend in industry.

- Gate pitch (CPP) scaling has significantly slowed down in recent nodes, being limited by lithography/device performance requirement (short channel effect).

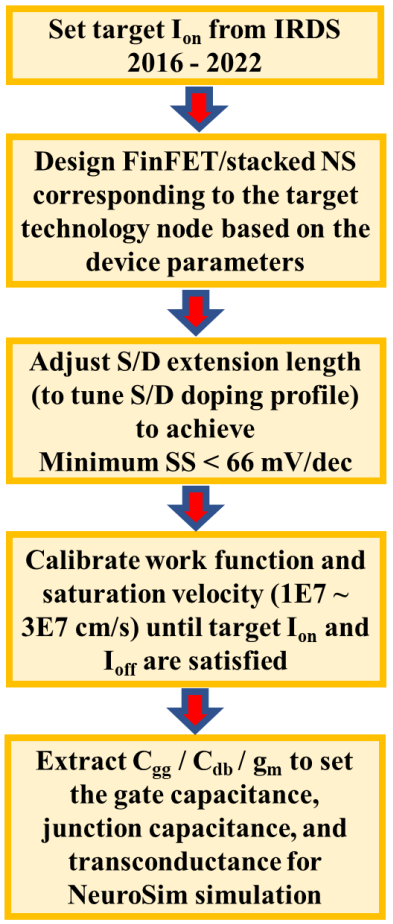
- Cell height scaling is driven by fin depopulation, PN separation scaling in recent nodes.

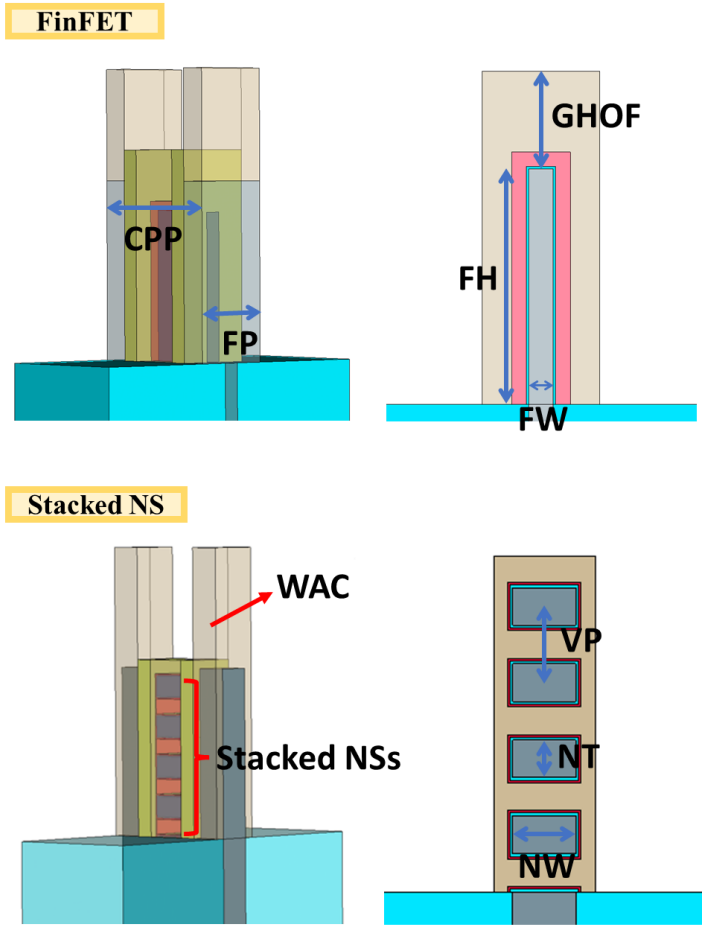
- Moore's law is extended by Backside Power Rail/3D integration techniques beyond 1 nm node.

Technology Extension to 1nm node

TCAD Simulation Assumptions

	Technology Node (nm)						
	14	10	7	5	3	2	1
Fin height [FH] (nm)	42	45	50	50	48	-	-
Fin width [FW] (nm)	8	8	7	6	5	-	-
NS thickness [NT] (nm)	-	-	-	-	-	6	6
NS width [NW] (nm)	-	-	-	-	-	15	10
NS vertical pitch [VP] (nm)	-	-	-	-	-	14	12
# of stacked NSs	-	-	-	-	-	3.0	4.0
Gate length (nm)	26	22	22	20	18	14	12
Equivalent oxide thickness [EOT] (nm)	0.9	0.8	0.7	0.65	0.6	0.55	0.5
Gate height over fin [GHOF] (nm)	40.0	35.0	30.0	25.0	20.0	10.0 IRDS: 15	5.0 IRDS: 10
Spacer width [SW] (nm)	12.0	8.0	8.0	7.0	6.0	6.0	5.0





Technology Extension to 1nm node

NeuroSim Input Device Parameters (from TCAD simulation)

	Technology Node (nm)						
	14	10	7	5	3	2	1
On current/fin (μA)	54.744	58.725	60.139	61.320	64.788	66.385	59.005
On current density (μA/um)	595.045	599.237	562.048	578.495	641.463	526.868	460.980
Off current/fin (pA)	9.856	12.516	15.752	14.676	16.006	9.242	21.747
g_m/fin (mS)	0.130	0.177	0.191	0.193	0.204	0.248	0.307
Supply voltage (V)	0.800	0.750	0.700	0.700	0.700	0.650	0.600
Gate capacitance (nF/m)	1.128	0.995	0.939	0.772	0.719	0.633	0.523
Junction capacitance (F/m^2)	0.012	0.013	0.014	0.012	0.013	0.009	0.010

- We allow some exceptions in IRDS standard: For 7 nm Current/fin, we decide the values based on our own projection to maintain current/fin improvement with technology scaling
- Effective width = fin height * 2 + fin width

Technology Extension to 1nm node

BEOL Trend

- Exponential increase in resistivity at advanced nodes (sub 20 nm) due to surface/grain boundary scattering.
- Assume copper interconnect and introduce modified **FS-MS model** to estimate the copper resistivity as a function of aspect ratio, metal width, barrier thickness and grain size.
- Set 0.2fF/um for parasitic capacitance across all technology nodes.

$$\rho = \rho_0 \frac{3}{8} C(1-p)\left(\frac{1}{h}+\frac{h}{A}\right)\lambda + \rho_0\left[1-\frac{3}{2}\alpha+3\alpha^2-3\alpha^3\ln\left(1+\frac{1}{\alpha}\right)\right]^{-1} \quad (1)$$ (FS-MS model)

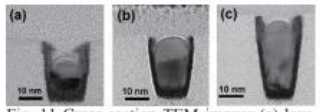


Fig. 11 Cross-section TEM images (a) low, (b) medium, and (c) high AR wires.

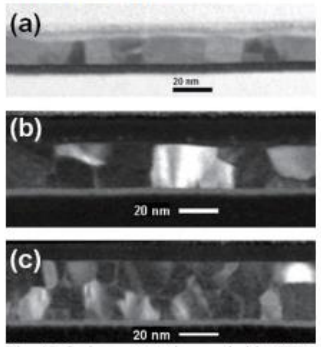


Fig. 12 Grain structure is revealed by TEM of (a) low, (b) medium, and (c) high AR wires as viewed normal to direction of current flow.

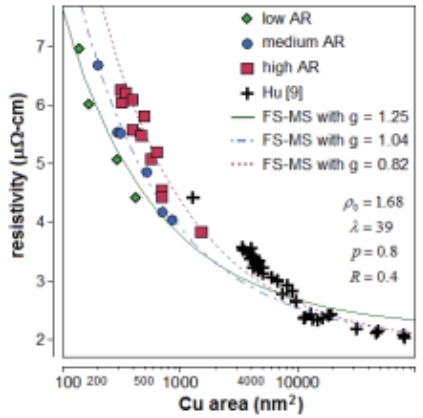


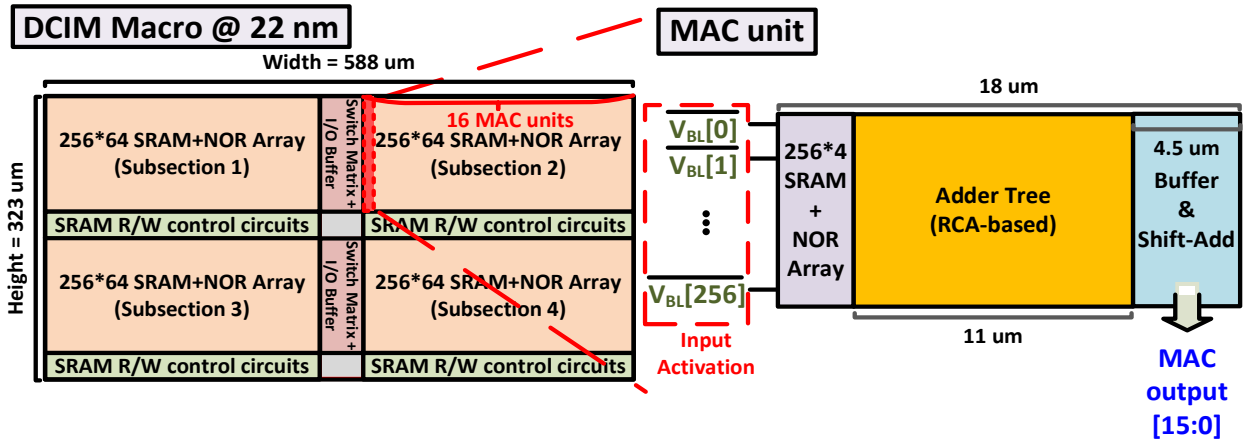
Fig. 13 The FS-MS model is fit to each data set using g values determined from grain size measurements by TEM. The same values of R and p are applied to each data set.

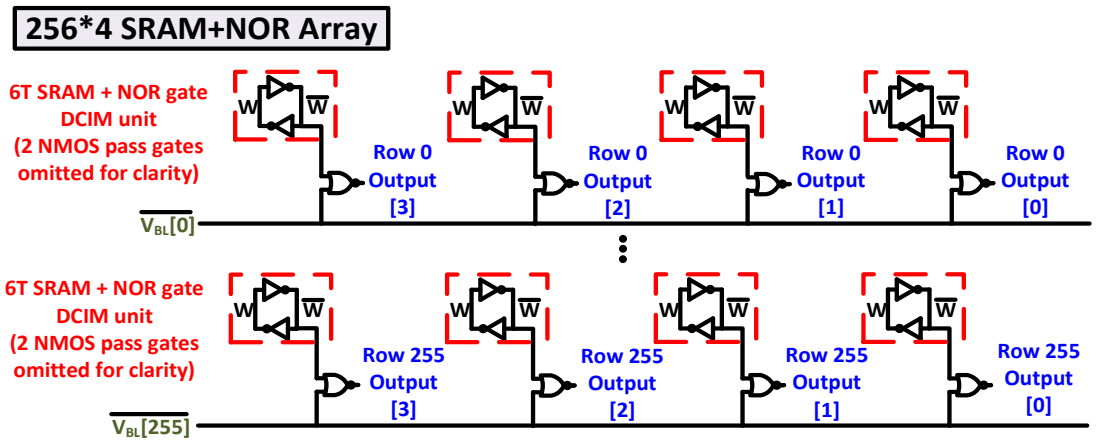
Pyzyna, et al. **Resistivity of copper interconnects beyond the 7 nm node**

Neurosim Metal Width Assumptions

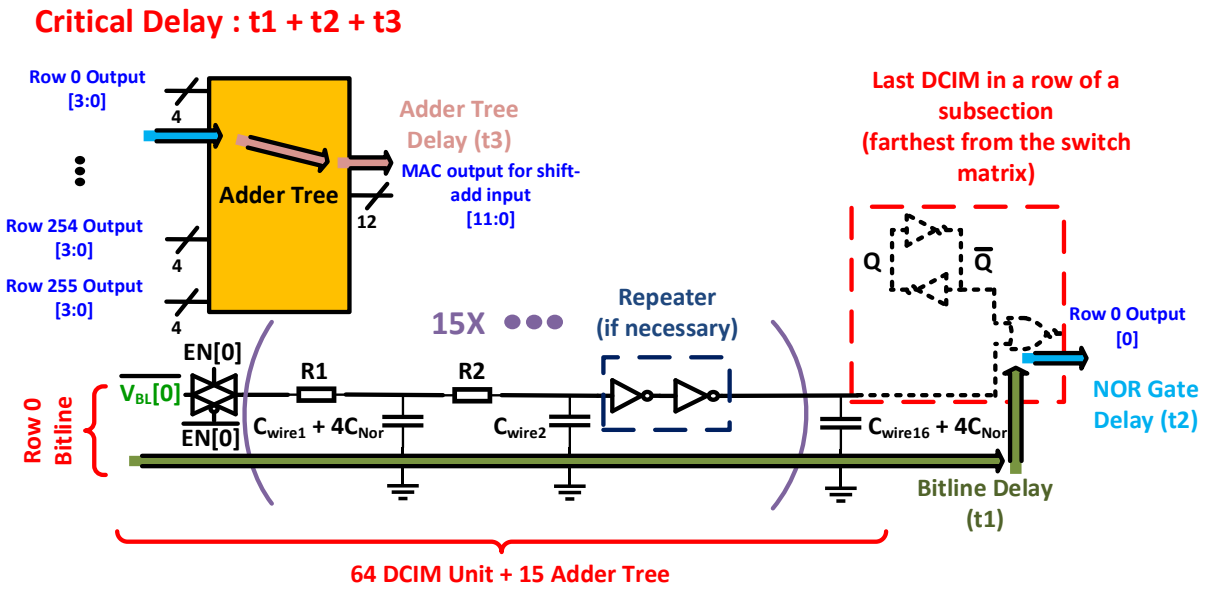
		Technology Node (nm)						
		14	10	7	5	3	2	1
Metal track in M0 (T)		9	7.5	6	6	6	5.7	5
Metal pitch (nm)	M2	64	44	40	36	32	24	16
	M0	64	44	40	30	24	20	16
	M1	78	64	57	34	32	23	20
Barrier thickness for M0-M2 (nm)		2.5	2.5	2.5	2.0	1.5	0.5	0.5

DCIM Scaling Trend

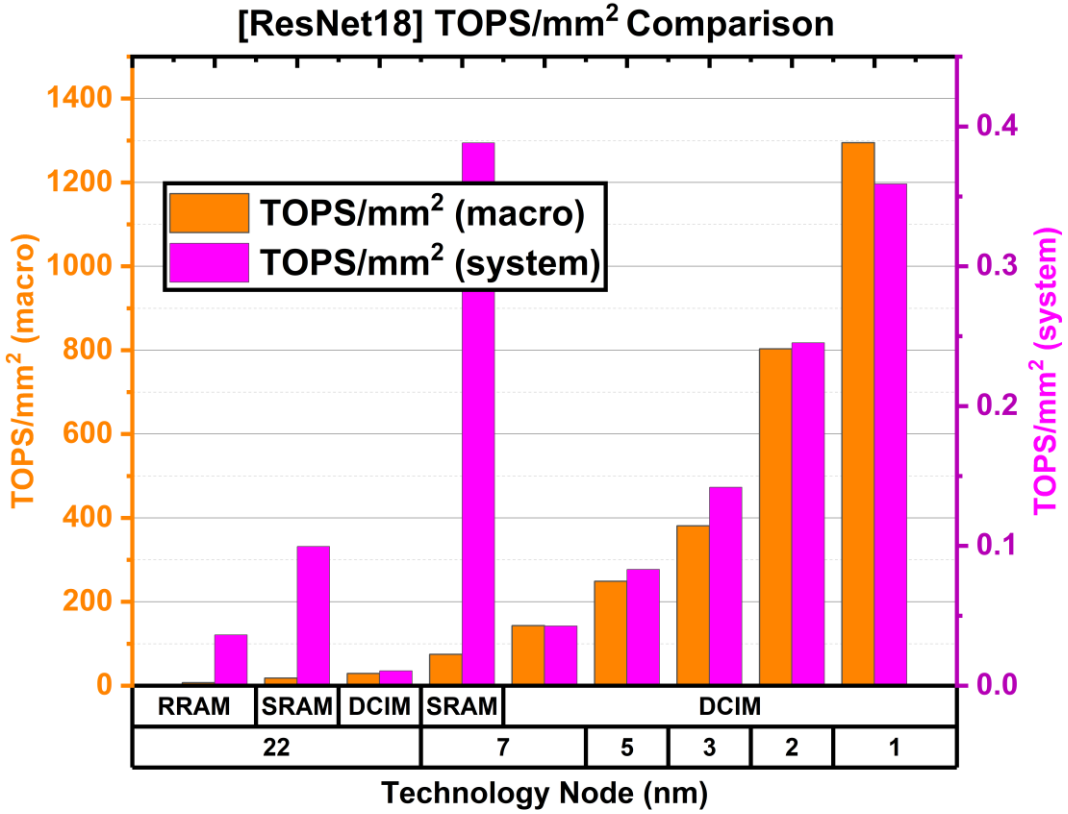


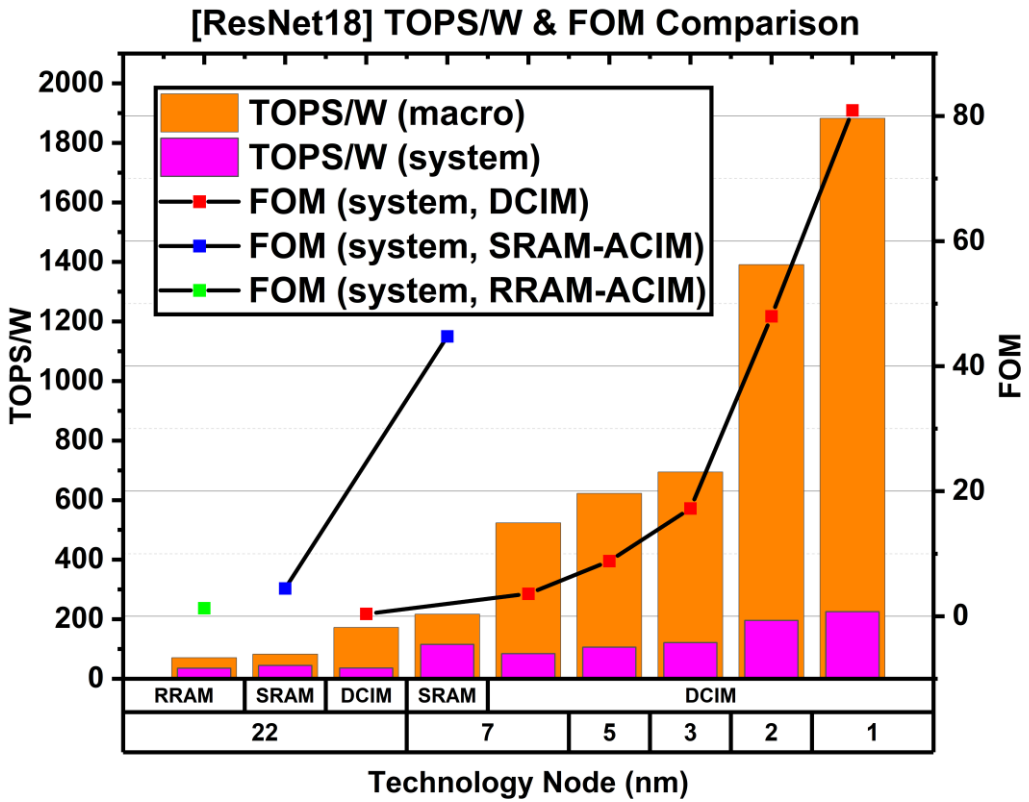


Critical delay in DCIM



DCIM Scaling Trend





- Circuit-level analysis of DCIM down to 1 nm node is enabled by NeuroSim 1.4 framework. **FOM=TOPS/WxTOPS/mm²**
- Macro-level and chip-level DCIM vs ACIM benchmark at state-of-the-art technology node is performed using the updated NeuroSim 1.4 framework.
- Using the nominal chip design parameters for ResNet18-ImageNet workload, 2nm DCIM is found to outperform state-of-the-art 7nm SRAM-based ACIM.

Outline

- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

NeuroSim Extension Roadmap

Projection of CIM hardware towards advanced technology nodes

Technology update to 1 nm node

[2023 July]

Integrate NeuroSim and Timeloop for **TPU** simulation

NeuroSim +Timeloop

[TBD]

NeuroSim +GEM5

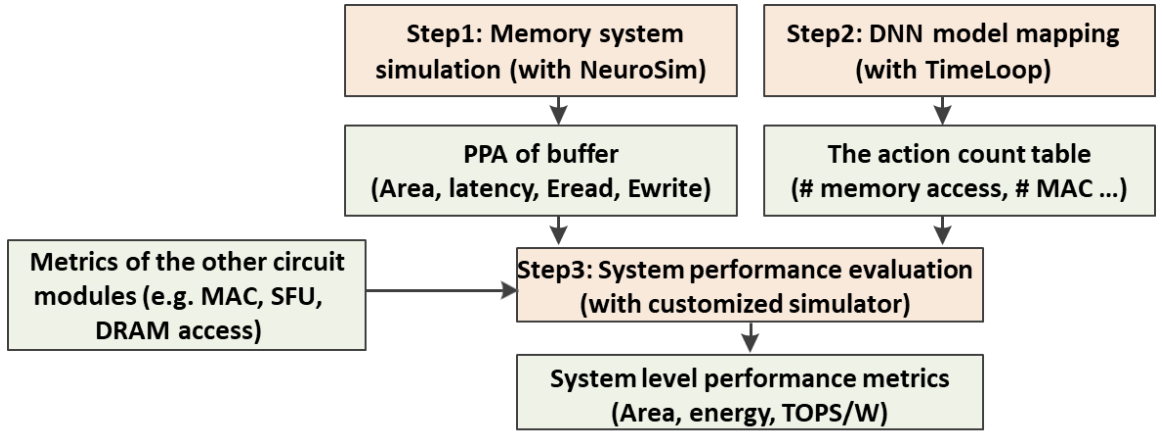
[TBD]

Chiplet based NeuroSim

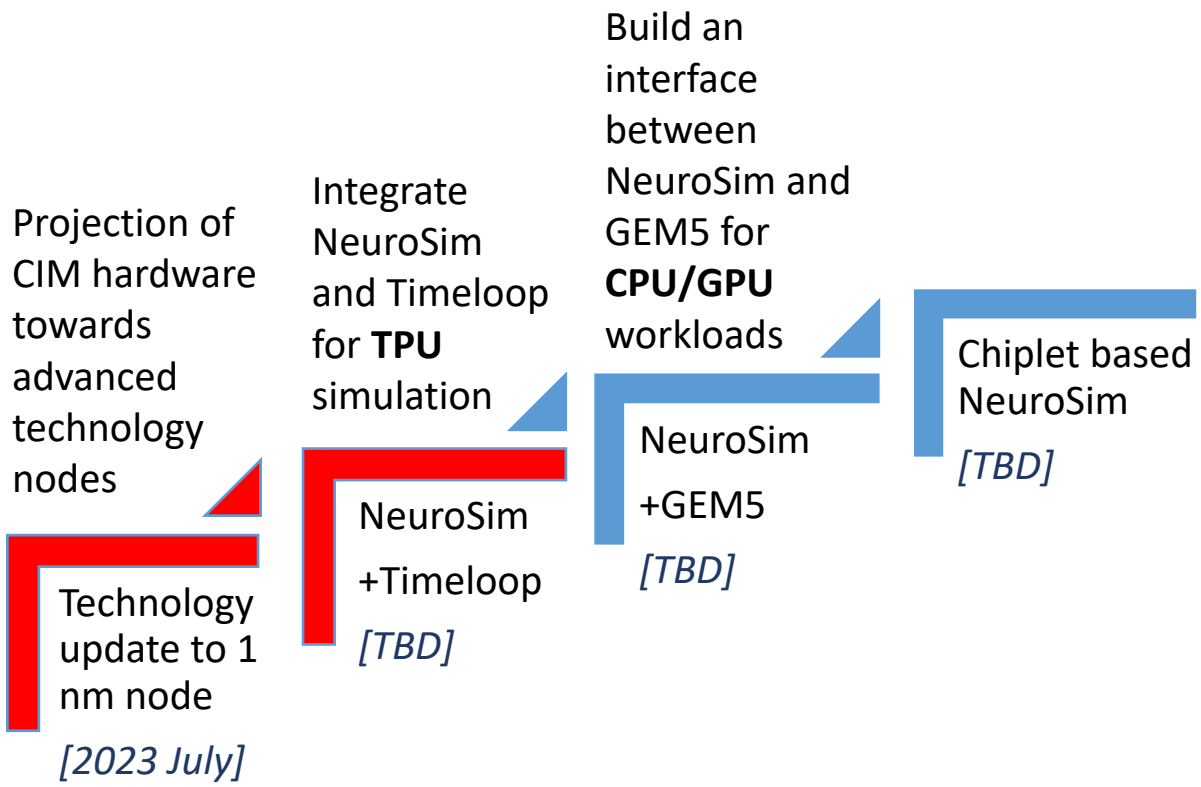
[TBD]

Timeloop: simulator for systolic array-based DNN accelerator (TPU-like workloads)

NeuroSim: provide macro-level memory read/write latency/energy
Integrate NeuroSim and Timeloop to get system-level performance with more diverse memory technologies as TPU global buffer



NeuroSim Extension Roadmap

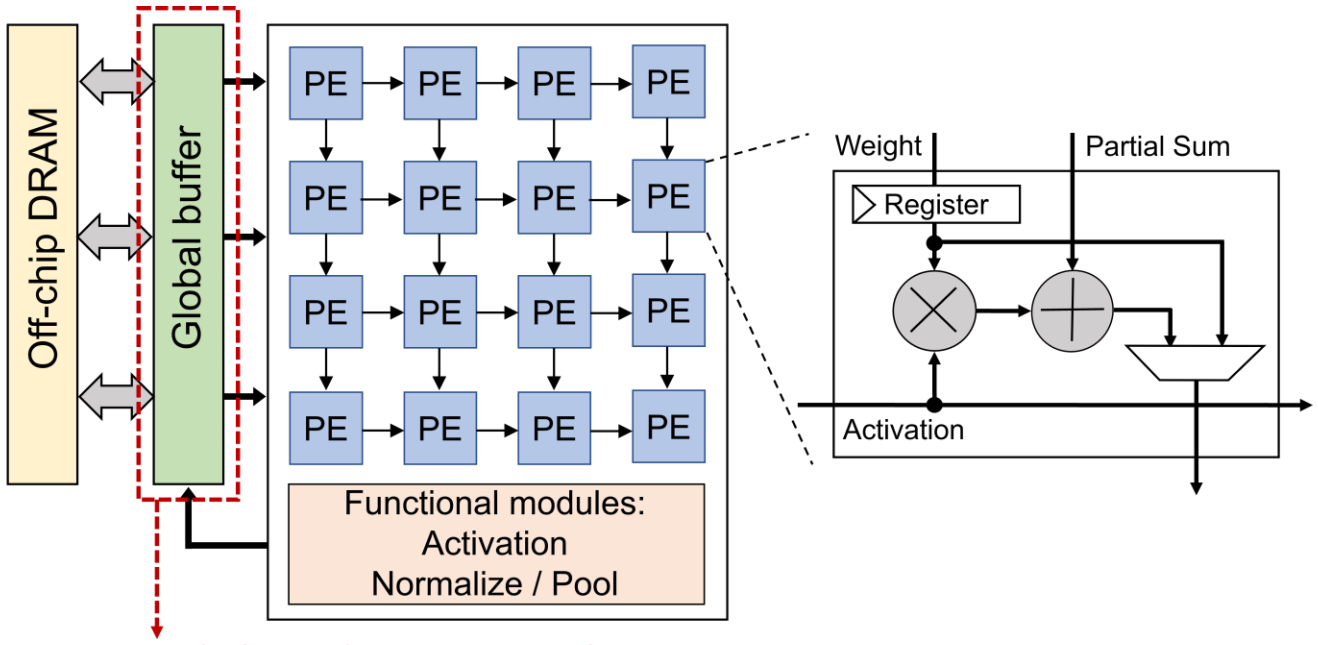


GEM5: a common simulator for computer architecture research (CPU/GPU workloads)

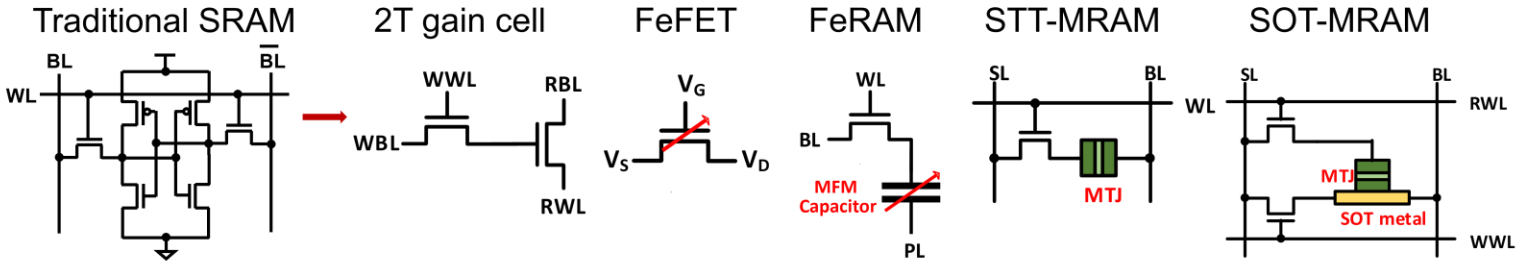
NeuroSim: provide macro-level memory read/write latency/energy (similarly as CACTI and NVSim, but supporting more advanced and emerging technologies)

-> Build an interface between NeuroSim and GEM5 to run the IPC (instructions per cycle) with more diverse and accurate memory model

Emerging memories as TPU global buffer



Emerging high-speed memories to replace SRAM



SRAM suffers from low density and high leakage

Emerging memory criteria to replace SRAM:

- high speed access (<10ns)
- high endurance (>10^{12})

Emerging memory candidates:

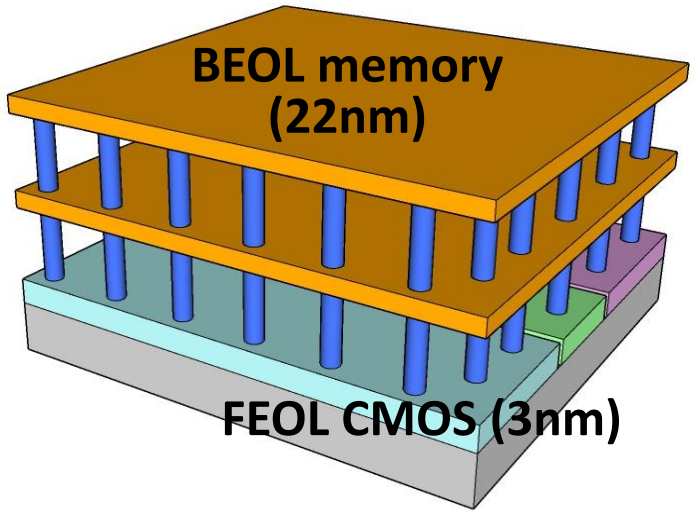
- 2T gain cell
- Ferroelectric memories (e.g., FeFET, FeRAM)
- Magnetic memories (e.g., STT-MRAM, SOT-MRAM)

A. Lu, et al. Nature Review 2023

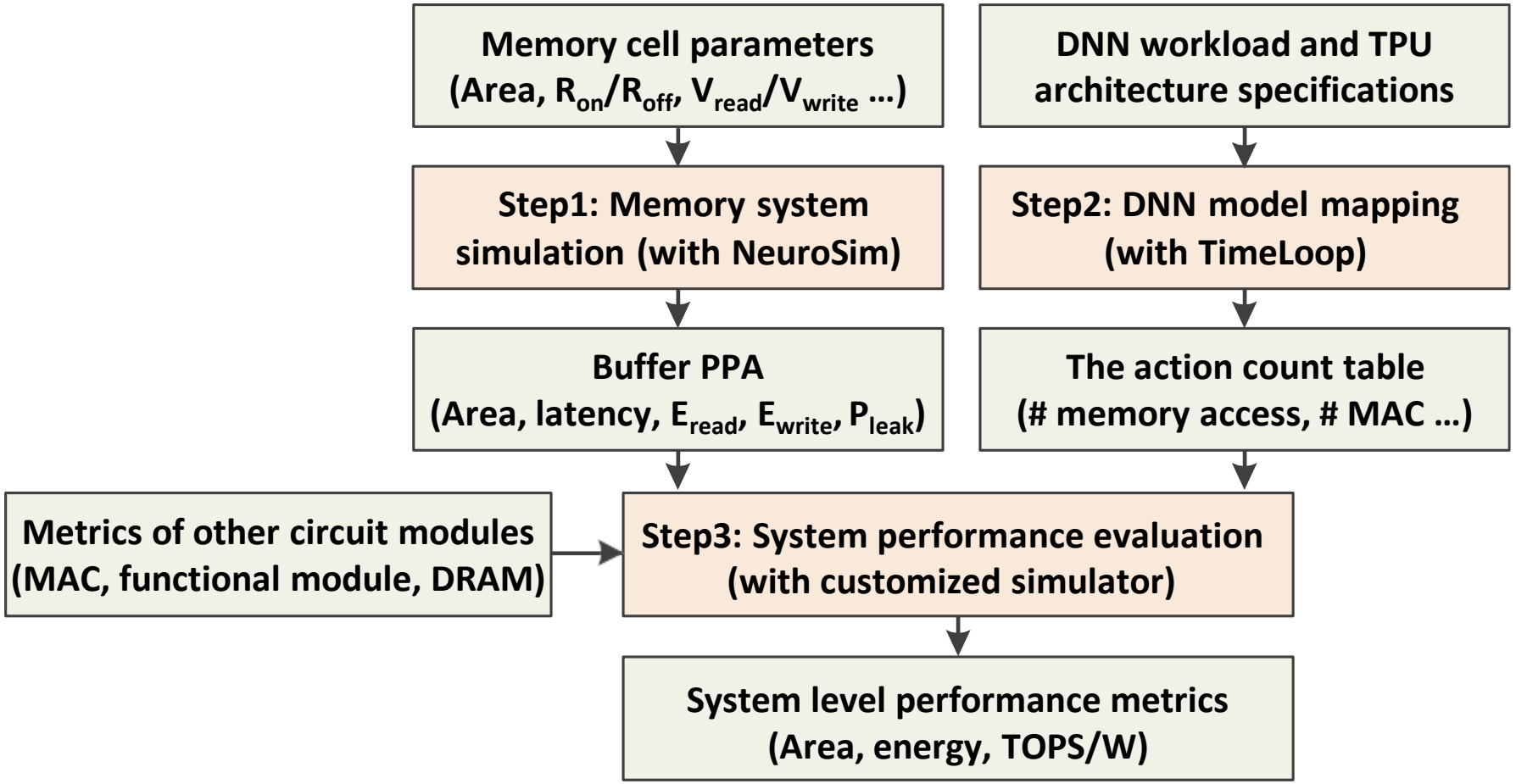
TPU global buffer benchmarking settings

	Edge TPU	Cloud TPU
Technology node	22nm	3nm
Clock frequency	200MHz	700MHz
8-bit MAC PE size	16 × 16	256 × 256
Register file size	512-bit/PE	
Global buffer size	2MB	24MB
DRAM interface	LPDDR4, 1.2pJ/bit	
Global buffer memory device type	22nm SRAM	3nm SRAM
	22nm BEOL 2T gain cell	3D projection: 22nm BEOL 2T gain cell & 3nm FEOL peripherals
	22nm BEOL FeFET	3D projection: 22nm BEOL FeFET & 3nm FEOL peripherals
	22nm STT-MRAM	3nm STT-MRAM projection
	22nm SOT-MRAM	3nm SOT-MRAM projection

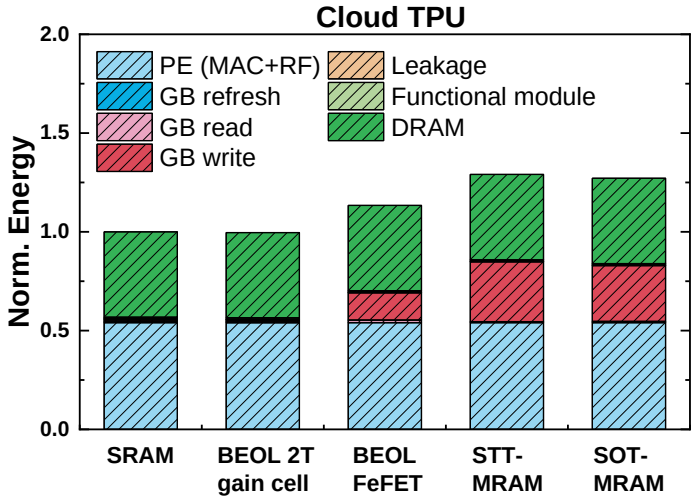
3D projection

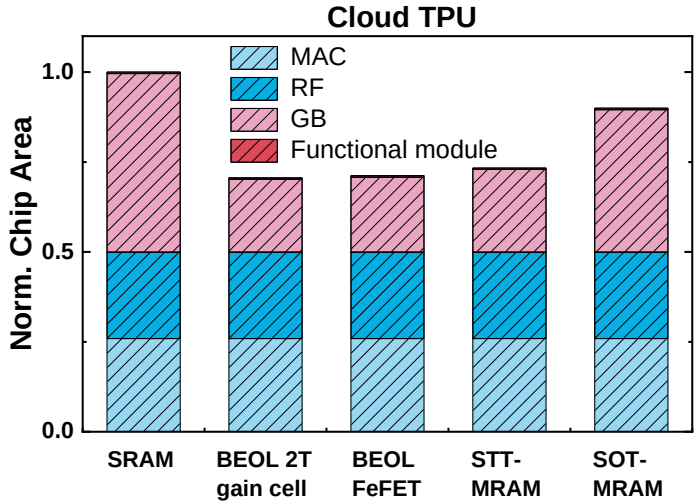


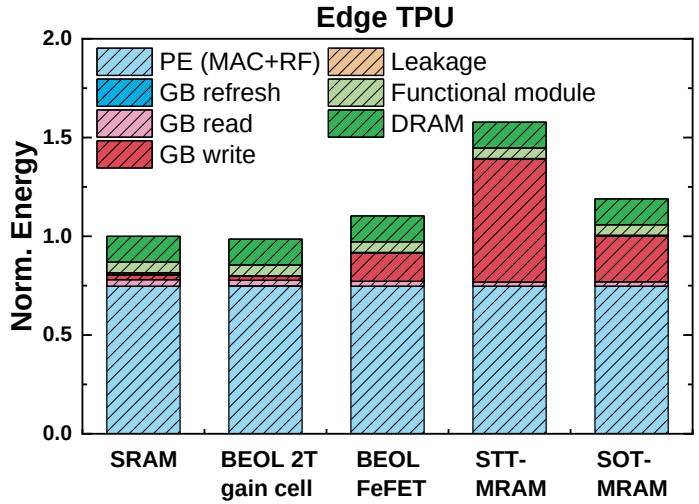
TPU global buffer benchmark flow

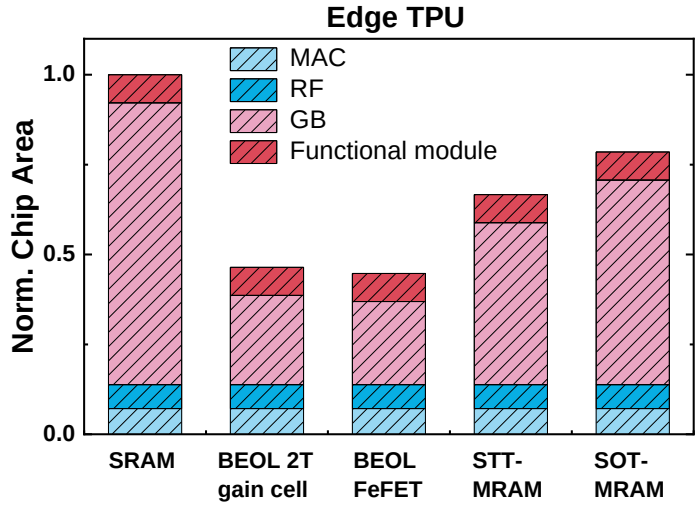


TPU global buffer benchmarking results

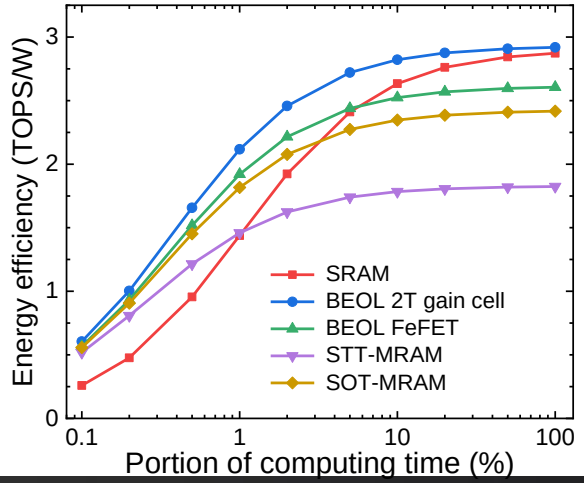








- SRAM is still competitive for high energy efficiency
- High write energy of ferroelectric and magnetic memories should be improved
- Emerging memories with BEOL integration or scaling down with logic technology are promising
- in terms of standby leakage, eNVMs are more attractive when the compute activity is low (<1%) for edge intelligence



NeuroSim Extension Roadmap

Projection of CIM hardware towards advanced technology nodes

Technology update to 1 nm node

[2023 July]

Integrate NeuroSim and Timeloop for **TPU** simulation

NeuroSim +Timeloop

[TBD]

Build an interface between NeuroSim and GEM5 for **CPU/GPU** workloads

NeuroSim +GEM5

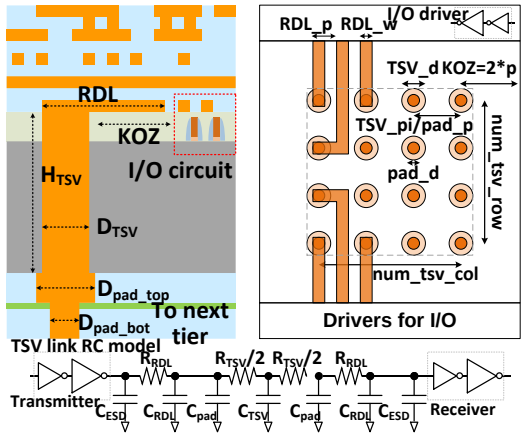
[TBD]

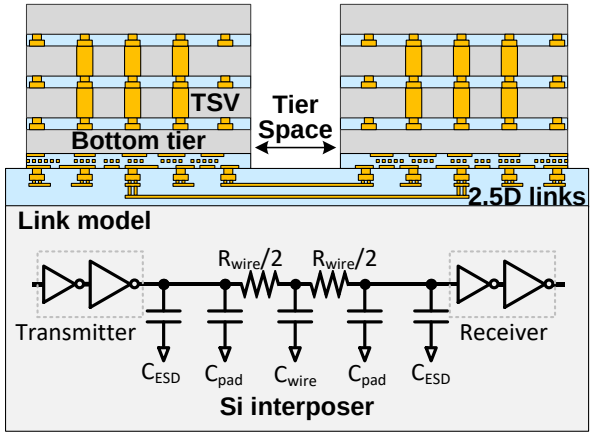
Reconfigurable 2.5D interconnect on interposer

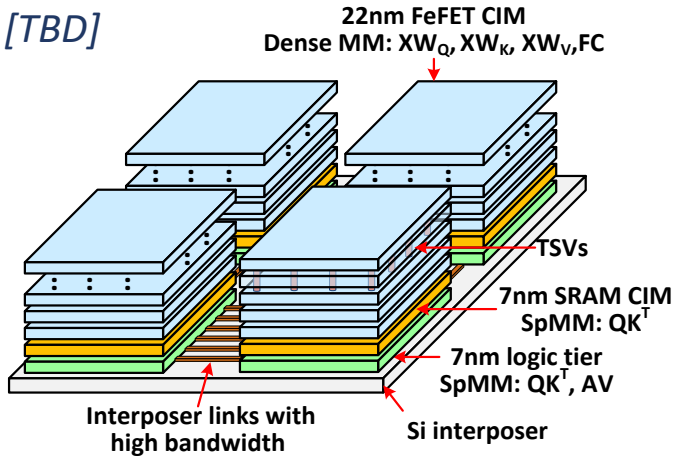
Chiplet based NeuroSim

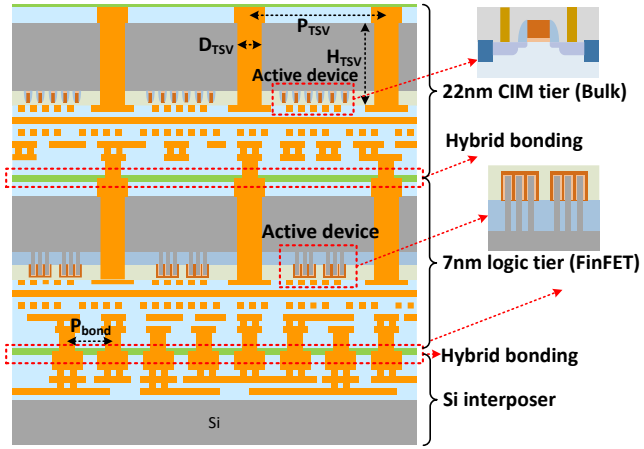
[TBD]

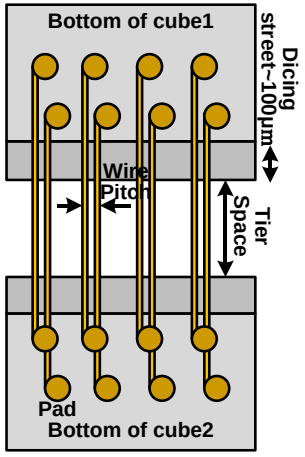
Extension for 2.5D/3D integration, TSV, I/O link, etc.











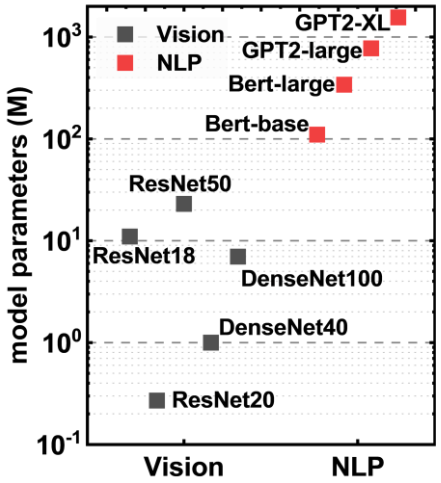
Outline

- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

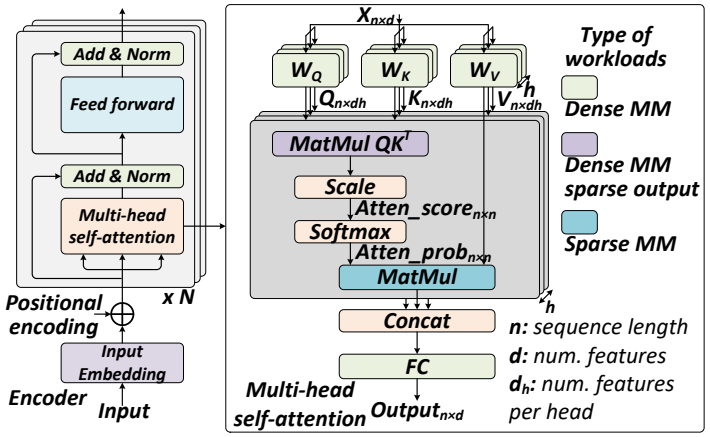
Motivation: large language model (transformer)

- **Large model size:** ~ hundreds GB parameters
- **Various types of matrix multiplication (MM) workloads**
  - Dense MM (e. g. $W_Q X$), sparse input (e.g. AV), dense input & sparse output (e.g. QK^T)

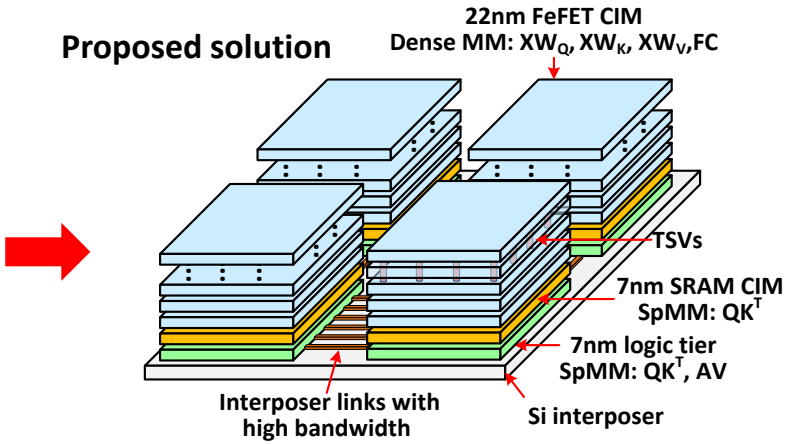
Technology	Digital MAC (7nm)	SRAM CIM (7nm)	2-bit FeFET CIM (22nm)
Energy/8-bit MAC	~1pJ	~0.5pJ	~0.2pJ
SpMM	Yes	No	No
Write endurance	Unlimited	Unlimited	~10^5
Write speed	1 cycle	1 cycle	> 100ns



Large model size
(100MB ~ a few GBs)



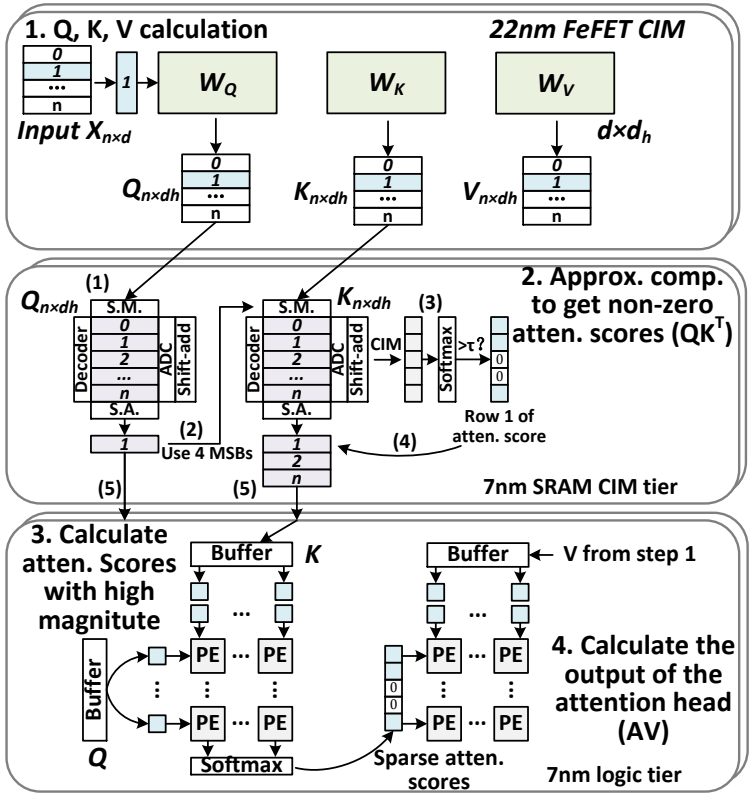
Various types of MM workload
in MHSA module



- 3D Heterogeneous computing platform
- Approximate computing for attention probability

Approximate computing for attention prob.

- Use low precision (4-bit) CIM computing to determine the small attention scores
- Use 8-bit digital MAC for the attention scores with high magnitude



Pseudo code: approximate computing

4-bit approx. computing with CIM

$S_{low} = Q_{4b} \times K_{4b}^T$ (CIM)

$Atten\_prob_{low} = Softmax(S_{low})$

$Mask = where(Atten\_prob_{low} > threshold, 1, 0)$

8-bit digital MAC for large scores

$S_{high} = Q_{8b} \times K_{8b}^T$ for Mask > 0

$S = where(Mask > 0, S_{high}, S_{low})$

do softmax

$S = S/sqrt(d_h)$

$Atten\_prob = Softmax(S)$

$Atten\_prob = max(Atten\_prob, threshold)$

linear combination with V

$Out = SpMM(Atten\_prob, V)$

Fig. An illustration of the approximate computing based MHSA

3D Dataflow for MHSA computing: Q, K

- Mapping: Q, K in the same cube
- IFM delivery: broadcast to each heads
  - By group (input channel)
- Q, K: send to the bottom SRAM CIM for storage
  - Q, K of the same head will be in the same tile
  - Send to the other cube if it can not store all the heads (e.g. 4 heads in this case)

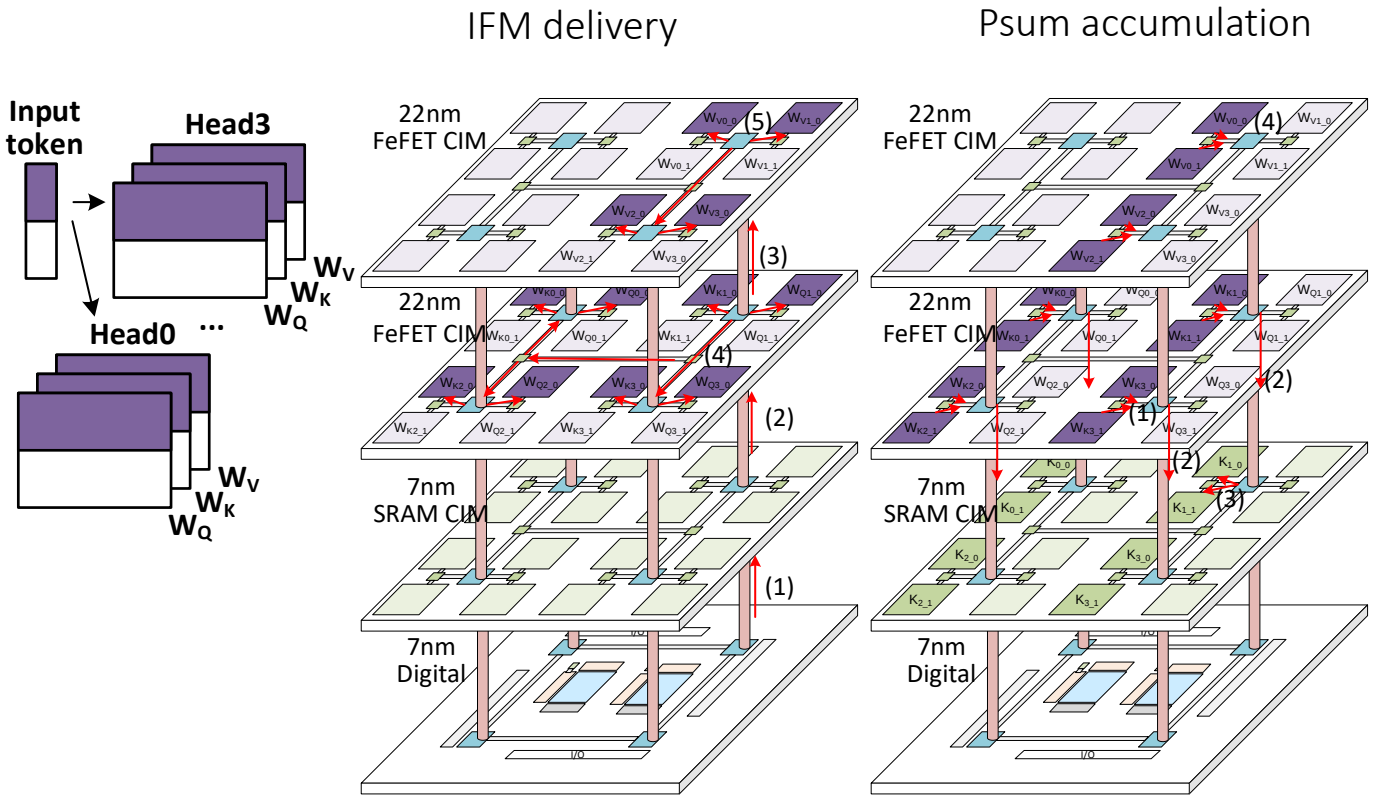


Fig. the dataflow for IFM delivery (left) and partial sum accumulation (right) for Q, K multiplication

3D Dataflow for MHSA computing: QK^T

- **SRAM CIM for approximate computing**
  - Send the mask of attention prob of the digital MAC
- **8-bit MAC for attention prob of high magnitude**
  - Output stationary dataflow

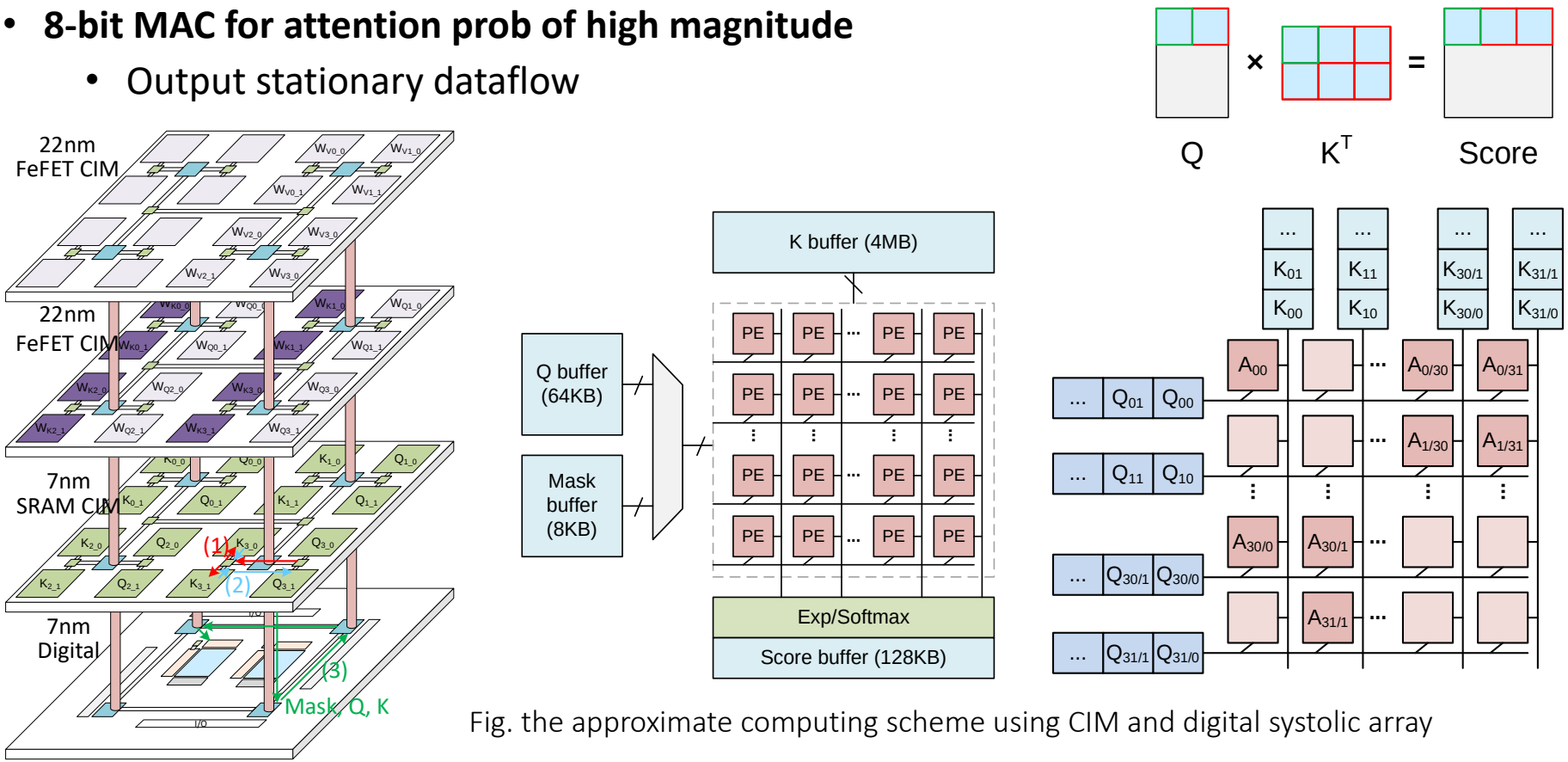
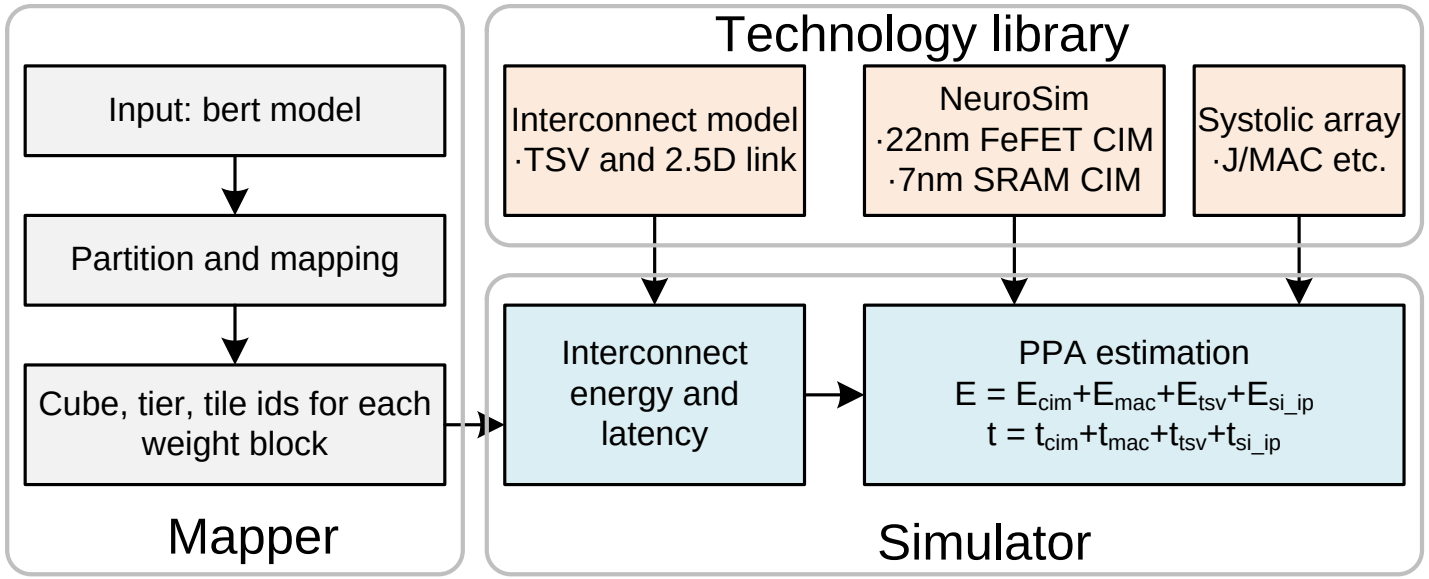


Fig. the approximate computing scheme using CIM and digital systolic array

2.5D/3D Package-Level Benchmark Framework



Design Parameters

TABLE 1. The configurations of the proposed design and baselines	
Design name	Configuration of a cube (16 tiers)
H3D_Hybrid (Optimal)	7nm systolic + 7nm SRAM CIM + 22nm FeFET CIM × 14
H3D_MAC	7nm systolic + 7nm SRAM buffer + 22nm FeFET buffer× 14
H3D_FeFET	22nm 1-bit FeFET CIM × 2 + 22nm 2-bit FeFET CIM × 14
H3D_SRAM	7nm SRAM CIM× 16

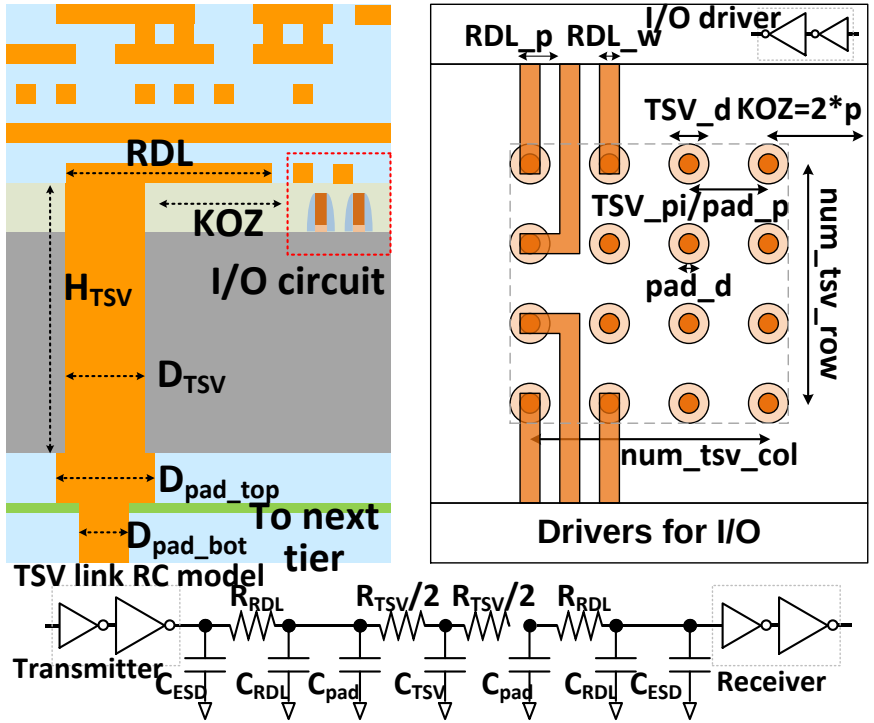
TABLE 2. The device and circuit parameters in the simulation						
Device	Tech. node	Cell bit	Cell size	ADC-bit	Parallel read	Write
8T SRAM	F=7nm	1-bit	$1080F^2$	3-bit	8 rows	0.8V/1cycle
FeFET	F=22nm	1 or 2-bit	$54F^2$	3-bit	8 rows	3V/50ns (1-bit)

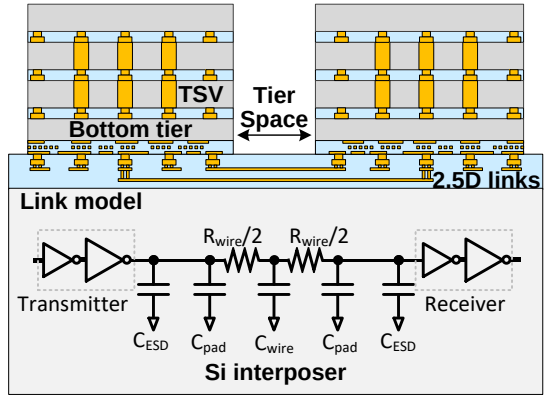
TABLE 3. The interconnect parameters for simulation						
Name	Pitch	Diameter	Height	C_{TSV}	Band width	Energy
TSV	20µm	10µm	100µm	~360fF	256-bit	0.35pJ/bit

Name	Length	Pitch	Width	Height	C_{wire}	Band width	Energy
Si interposer	500µm	10µm	2µm	2µm	~73fF	256-bit	0.17pJ/bit

TABLE 4. The projected performance of a 7nm systolic array (32×32)				
Name	MAC energy	Throughput (dense)	Throughput (sparse)	Area
Sparse engine	0.34pJ/MAC	1024 MACs/cycle	675 MAC/cycle	$0.89mm^2$

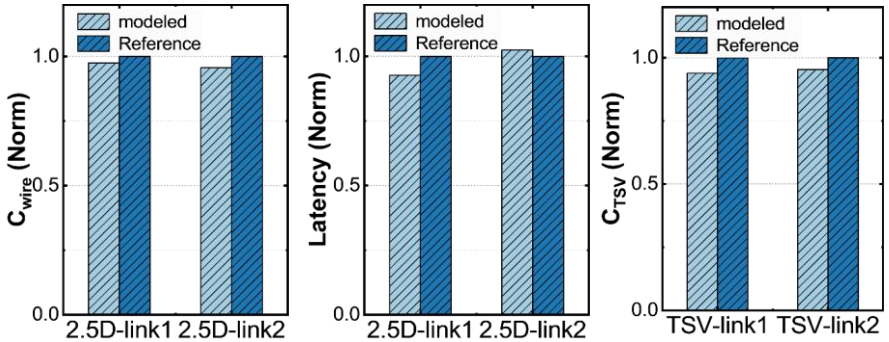
Interconnect Modeling and Validation



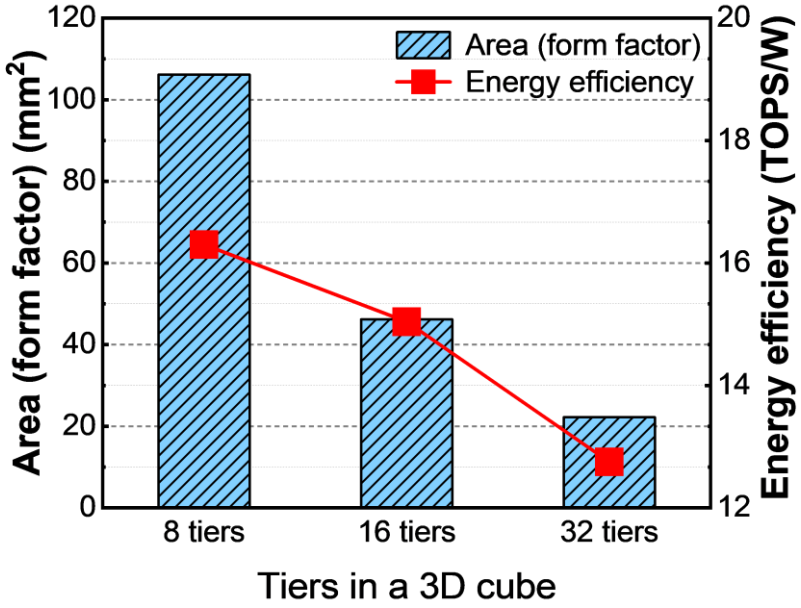


RC parasitics modeled for TSV, given the height/diameter and distance of keep-out-zone

2.5D links modeled as RC network



Impact of 2.5D/3D Technology Parameters



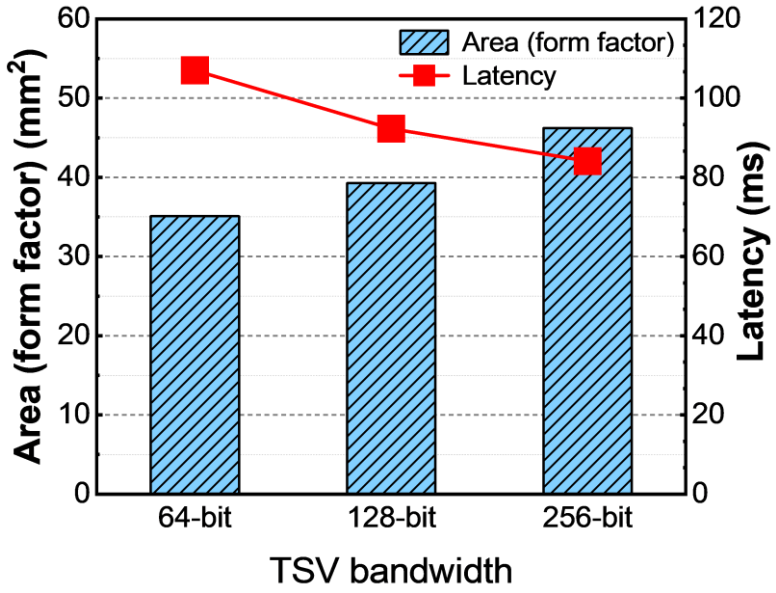


Fig. (a) The interposer area (form factor) and energy efficiency with different number of tiers stacked in a 3D cube. Energy efficiency reduces in more stacked tiers due to parasitics; (b) The interposer area and latency with different TSV bandwidth. The latency is reduced by 21.5% with 256-bit TSV bandwidth.

Results: approximate computing accuracy

- 4-bit weight and activation is needed for QK multiplication
- Negligible accuracy loss using Bert-base (~110M parameters) or GPT-2 (~1.7G parameters)

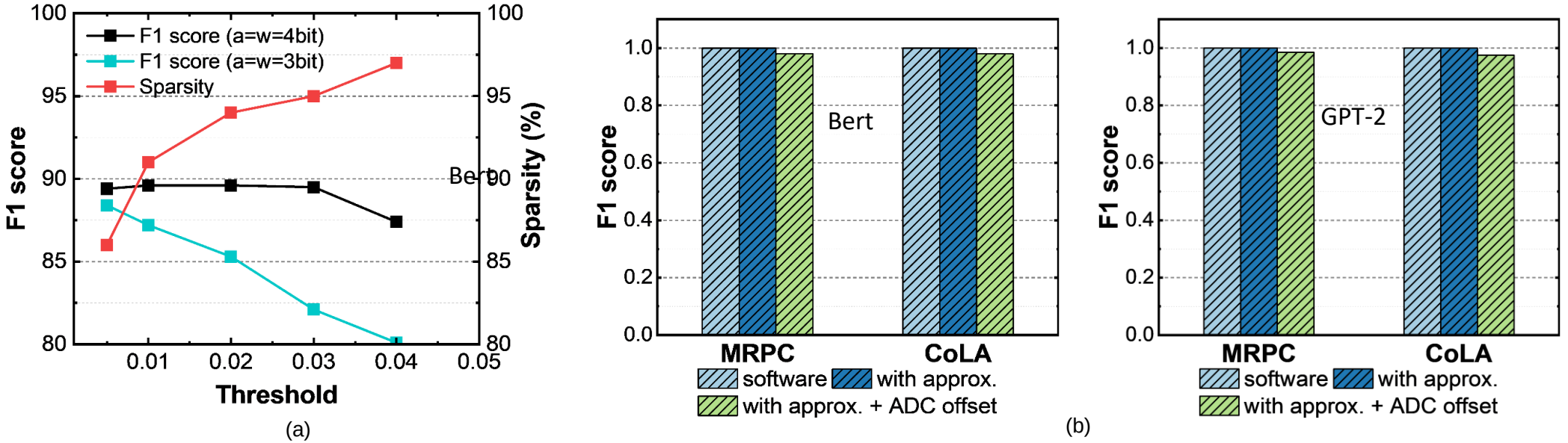
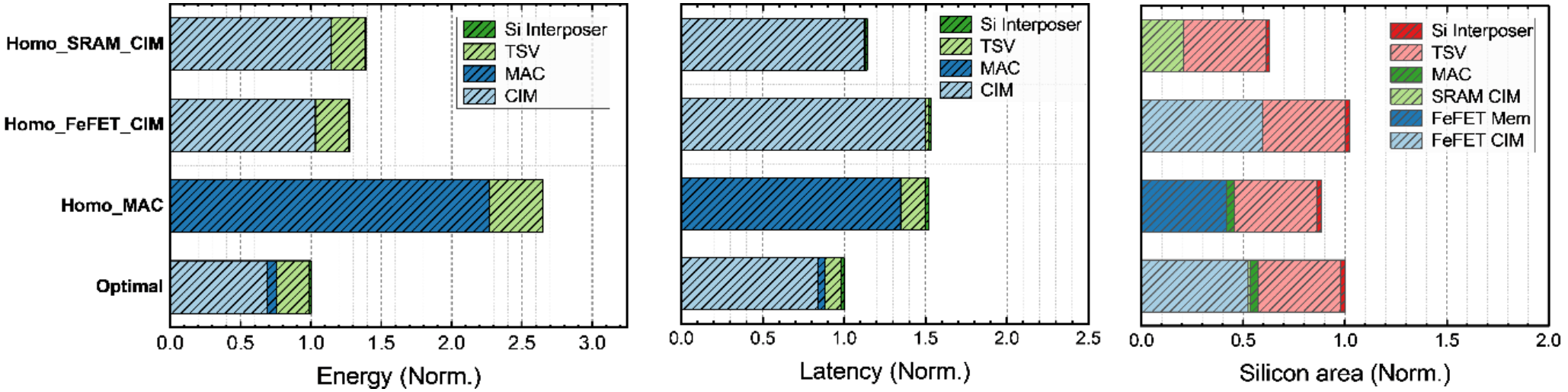


Fig. (a) The F1 score and sparsity of attention score with different threshold values to zero out small attention scores.(b) The accuracy w. and w/o. hardware effects of SRAM CIM array when calculating the attention scores for Bert-base and GPT-2

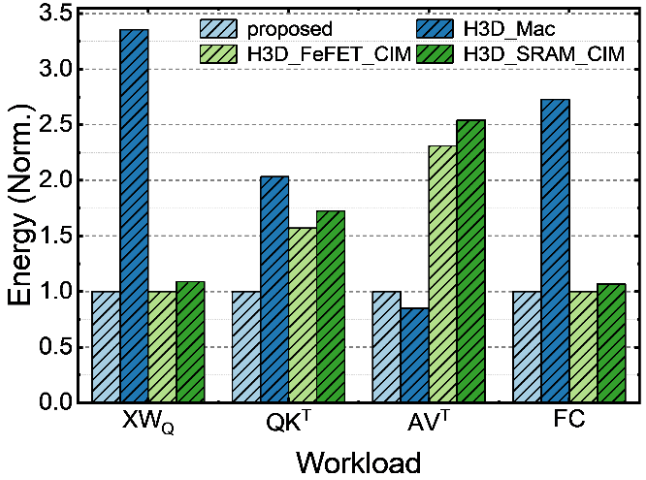
Microsoft Research Paraphrase Corpus (MRPC)
Corpus of Linguistic Acceptability (CoLA)

Results: Hardware PPA Breakdown

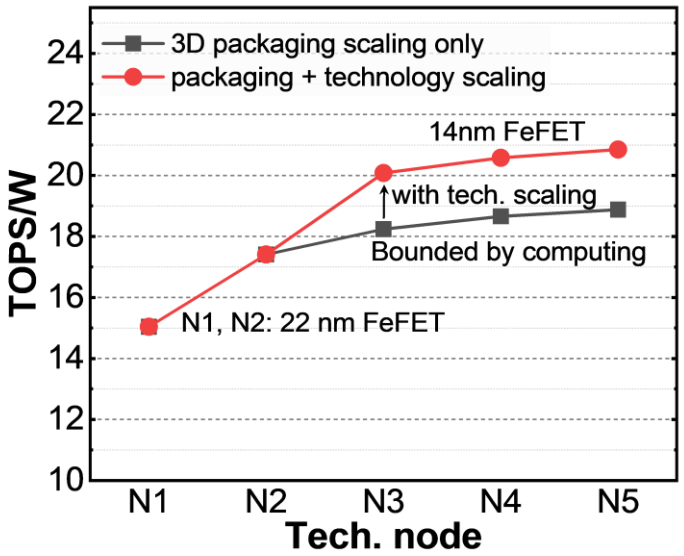


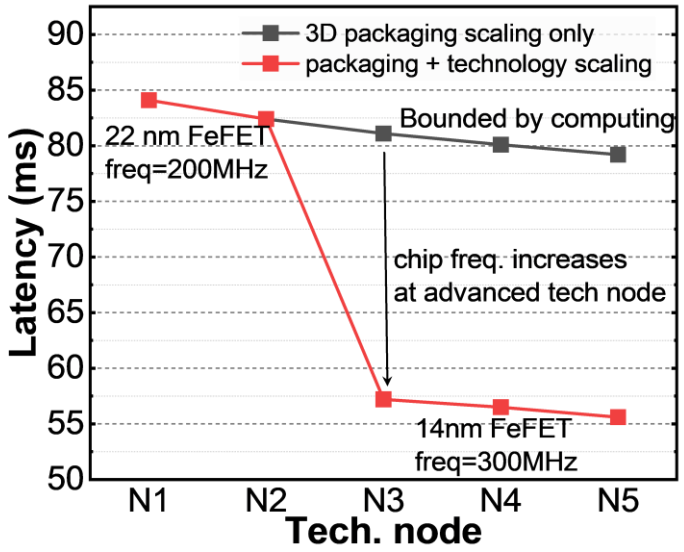
A layer-by-layer computing scheme is adopted to achieve low power consumption for edge device. With such as low total power (~30mW), the temperature rise is small.

Energy breakdown for different workloads



Technology/Packaging Scaling Perspective





	N1	N2	N3	N4	N5
TSV pitch (μm)	20	14	9	6	4.5
Si-interposer Pitch (μm)	10	7	4.5	3	2
FeFET Tech. node (nm)	22	22	14	14	14
FeFET chip freq. (MHz)	200	200	300	300	300

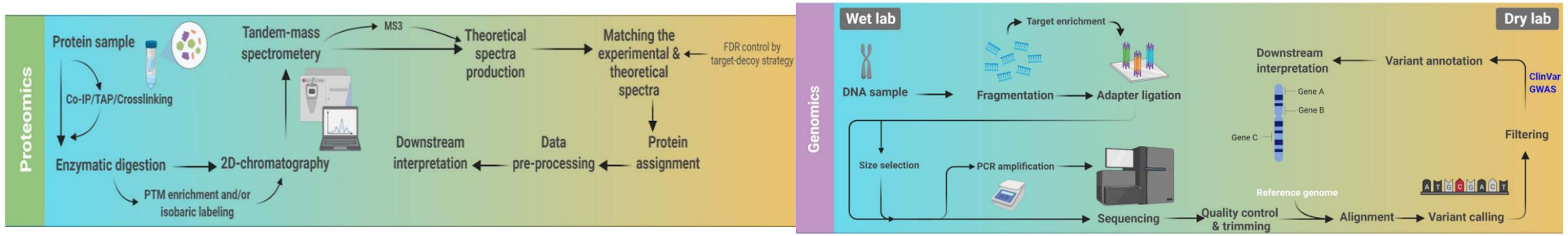
- Two scaling schemes are considered, one with packaging scaling only while the other with both packaging and transistor technology scaling.
- It is observed that both the energy efficiency and the latency could be bounded by computing if there is only packaging technology scaling.
- Therefore, it is important to synchronize the scaling of the packaging technology and transistor technology.

Outline

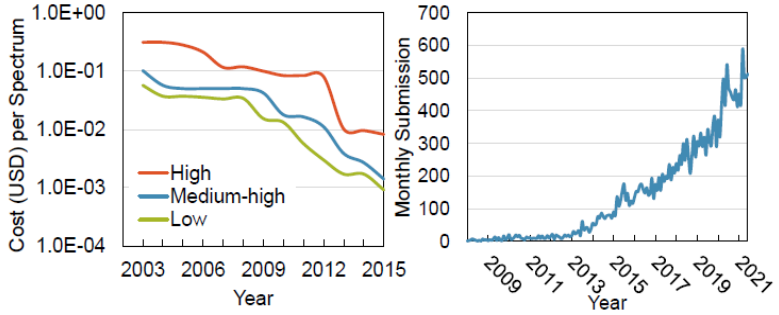
- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

Bioinformatics Challenges

- **Motivation**:
  - Mass spectrometry (MS) and genome analysis are two foundational workloads for proteomics, drug discovery, and genetic diagnosis



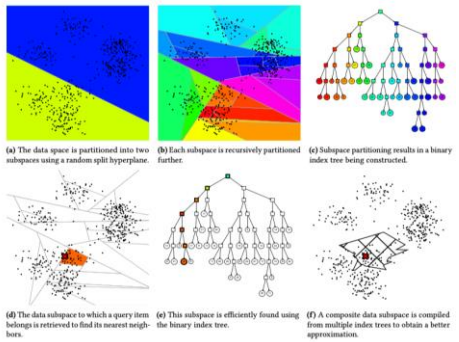
- **Challenges**:
  - MS and genome data are booming, e.g. Over 521 TB MS data in UCSD MassIVE database (as of 05/27/2022)
  - MS and genome analysis are memory-intensive workloads that take long time (a few hours to days)



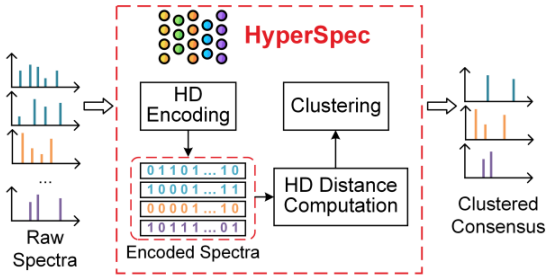
1. Karimi, M. R., Karimi, A. H., Abolmaali, S., Sadeghi, M., & Schmitz, U. (2022). Prospects and challenges of cancer systems medicine: From genes to disease networks. Briefings in Bioinformatics, 23(1), bbab343.

Motivation & Goals

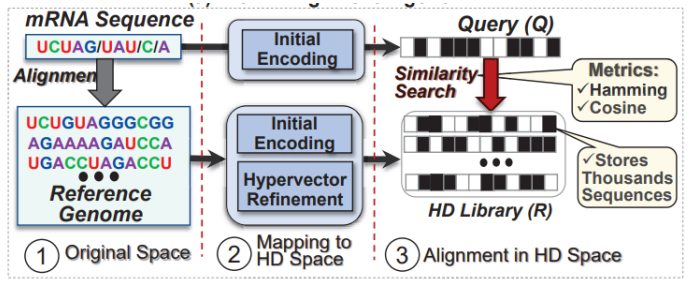
- **State of the art**:
  - Mass spectrometry: Ann-Solo[1] and HOMS-TC[2] for peptide identification and HyperSpec[3] for clustering
  - Genome analysis: BioHD[4] for pattern matching



Ann-Solo



HyperSpec

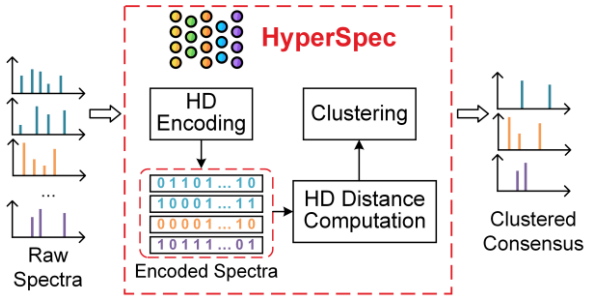


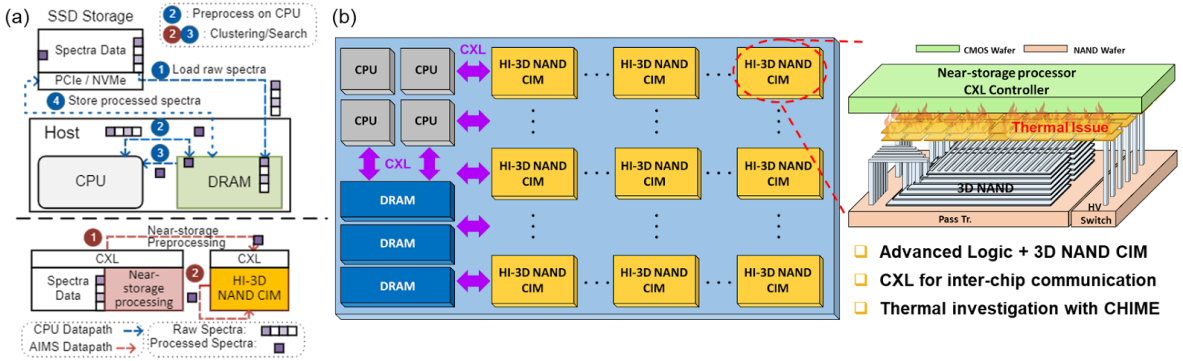
BioHD

- Metrics:
  - Algorithm: Accuracy and identification rate for identification/matching
  - Hardware: Power, performance, area (PPA) for proposed designs

1. W. Bittremieux. et al. Extremely fast and accurate open modification spectral library searching of high-resolution mass spectra using feature hashing and graphics processing units. Journal of Proteome Research 18, 3792–3799 (2019)
2. Kang, Jaeyoung, et al. "Accelerating open modification spectral library searching on tensor core in high-dimensional space." Bioinformatics 39.7 (2023): btad404.
3. Xu, Weihong, et al. "HyperSpec: Ultrafast Mass Spectra Clustering in Hyperdimensional Space." Journal of Proteome Research (2023).
4. Zou, Zhuowen, et al. "Biohd: an efficient genome sequence search platform using hyperdimensional memorization." ISCA, 2022.

Motivation & Goals

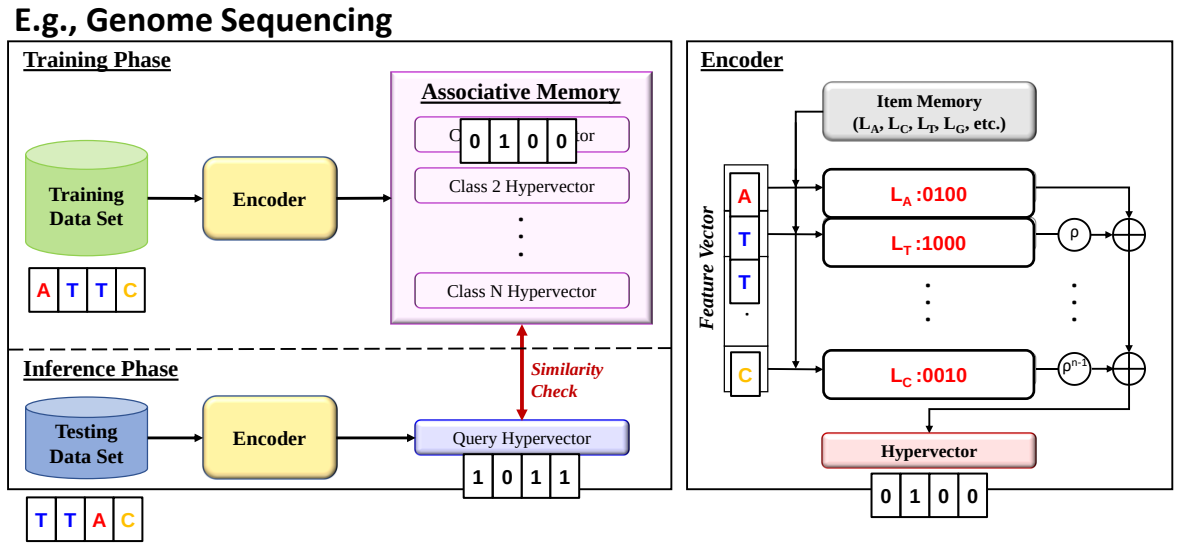




- **Proposed solution:**
  - Algorithm: Use hyperdimensional computing (HDC) to present/process bio workloads
  - Hardware: Develop AIMS architecture based on 3D NAND Flash incorporated with 2.5D/3D heterogeneous integration to reduce the latency and improve the energy efficiency of current Mass Spectrometry (MS) analysis on CPU/GPU by **10-100x**
- **Success criteria:**
  - Demonstrate accuracy of HDC-aided bio algorithms on CPU/GPU
  - Present 3D NAND-based accelerator or memory system
  - Evaluate the hardware feasibility using electrothermal simulation. Compare the PPA metrics with SOA GPU or in-memory solutions

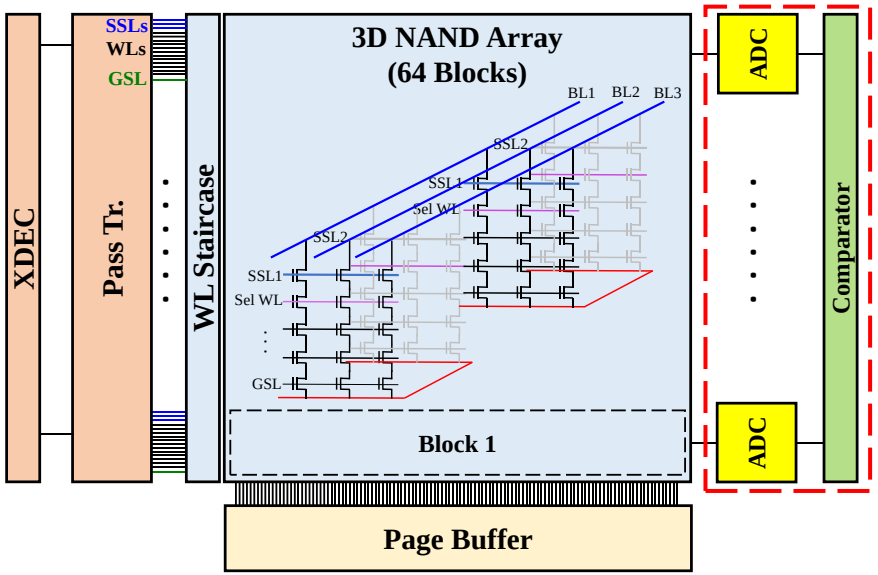
Hyperdimensional Computing (HDC)

- Hyperdimensional Computing consists of two phases: **training(store) phase** and **inference phase**.



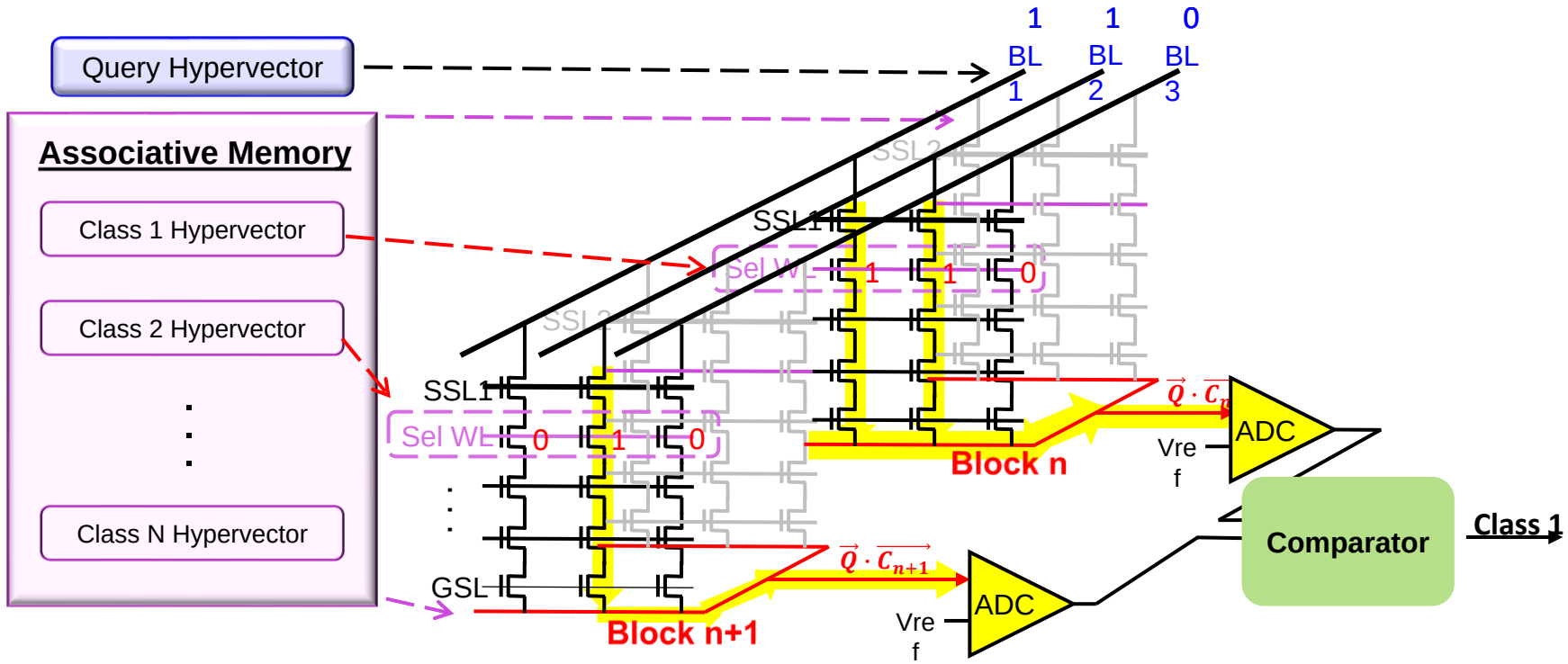
Proposed 3D NAND HDC Engine

- The peripheral circuits are adapted from common 3D NAND periphery and adding additional ADC and comparator for hyperdimensional computing



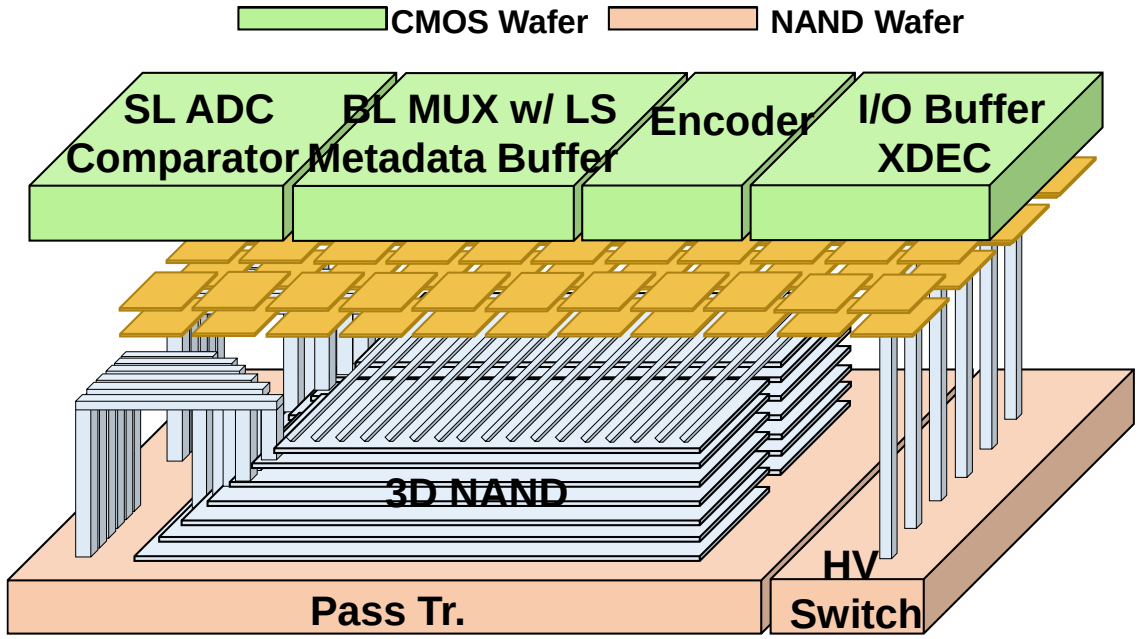
3D NAND Associative Memory

- The query hypervector is sent in as BL voltage and times the conductance representing the corresponding class hypervector. The summed current is the dot product used for similarity check.



Heterogeneous Integration

- Two heterogeneous integration techniques for 3D NAND, Cu-Cu hybrid bonding and CMOS under array (CuA), are incorporated to achieve a compact form factor.



Simulation Parameters

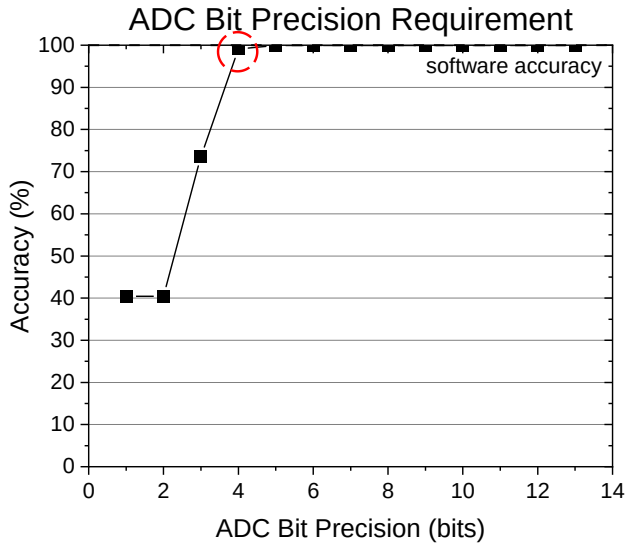
- Dataset
  - Training dataset [1]:10,575 microorganism genome sequences with an average length of 3 million nitrogen bases per sequence.
  - Testing dataset: Artificially generate mutated sequences with the mutation rate of 10^{-3} per generation for classification.
- 3D NAND Parameters from industry [2].
- Digital circuits are synthesized using ASAP 7 [3].
- 3D NAND Peripherals are extracted from NeuroSim [4].

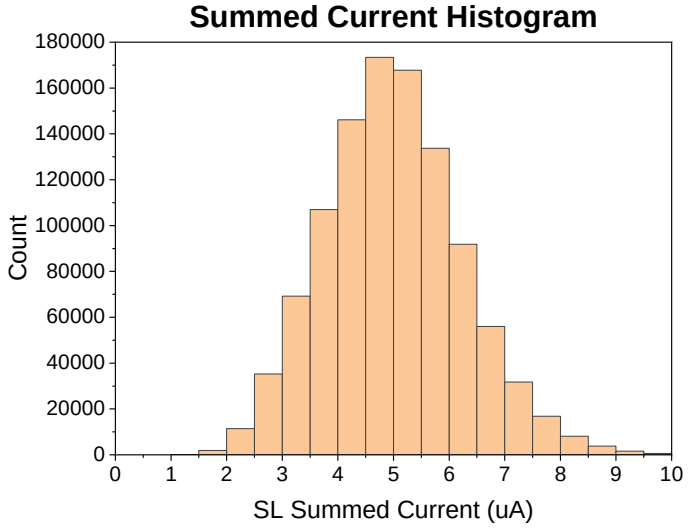
	Parameters	Values
Advanced CMOS Tier	Technology	7nm FinFET Process
	VDD1	0.7 V
	ADC Type	4-bit Flash CSA
	Encoder Dimensionality	10000
3D NAND Physical Parameter	Device Dimensions	N/A
	SSL Pitch	220 nm
	BL Pitch	100 nm
	No. of WL/SSL/BL	32/15/10000
	No. of Block	64
	Array Size	960 SSL × 10000 BL
	WL Staircase pitch	500 nm
3D NAND Electrical Parameter	WL Read Voltage	1 V/4.5 V (V_select/V_pass)
	SSL Read Voltage	4.5 V(activated)
	BL Read Voltage	0.2 V
	I_{on}/I_{off}	2 nA/1 pA
CMOS Under Array	Technology	130nm Process
	VDD2	1.3 V

[1] "WoL: Reference Phylogeny for Microbes" https://biocore.github.io/wol/
[2] Hang-Ting Lue et al. 2019. Optimal Design Methods to Transform 3D NAND Flash into a High-Density, High-Bandwidth and Low-Power Nonvolatile Computing in Memory (nvCIM) Accelerator for Deep-Learning Neural Networks (DNN). In IEEE International Electron Devices Meeting (IEDM). 38.1.1–38.1.4.
[3] L. T. Clark et al.,"ASAP7: A 7-nm FinFET predictive process design kit," Microelectronics Journal, vol. 53, pp. 105-115, July 2016.
[4] Xiaochen Peng et al. 2019. DNN+NeuroSim: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators with Versatile Device Technologies. In 2019 IEEE International Electron Devices Meeting (IEDM). 32.5.1–32.5.4.

ADC Precision Requirement

- The simulation result indicates that the minimum requirement of ADC bit is 4-bit.
- The summed current distribution of each SL before sending to ADC with ON current of 2nA [1] is also simulated. The range of summed currents is in few μA which is reasonable for sensing.

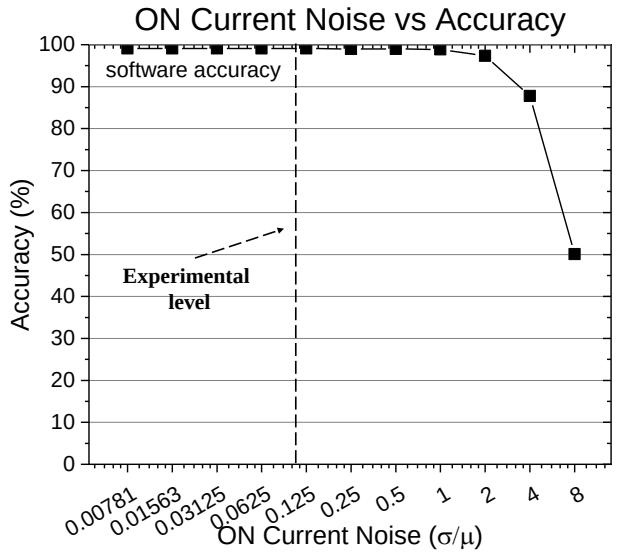


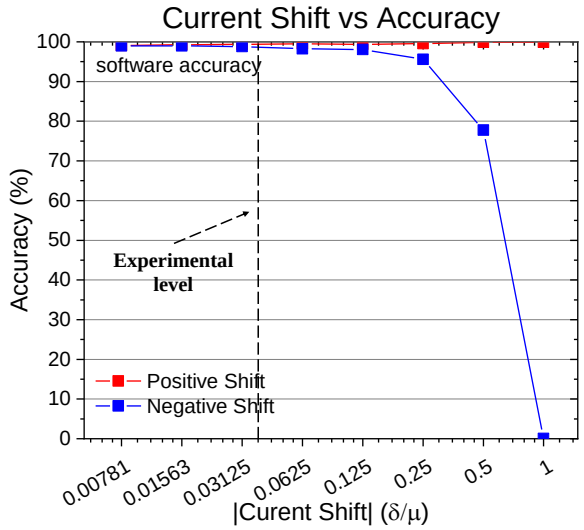


[1] Hang-Ting Lue et al. 2019. Optimal Design Methods to Transform 3D NAND Flash into a High-Density, High-Bandwidth and Low-Power Nonvolatile Computing in Memory (nvCIM) Accelerator for Deep-Learning Neural Networks (DNN). In IEEE International Electron Devices Meeting (IEDM). 38.1.1–38.1.4.

3D NAND Device Non-ideal Effects

- The 3D NAND parameters [1] are ON current = 2nA, OFF current = 1pA, ADC bit precision = 10-bit, VBL = 0.2V and Vpass,r = 4.5V.
- From the silicon data, the current variation (σ/μ) is 0.1 and current shift (δ/μ) is ±0.05 which is tolerable from the simulation results.





[1] Hang-Ting Lue et al. 2019. Optimal Design Methods to Transform 3D NAND Flash into a High-Density, High-Bandwidth and Low-Power Nonvolatile Computing in Memory (nvCIM) Accelerator for Deep-Learning Neural Networks (DNN). In IEEE International Electron Devices Meeting (IEDM). 38.1.1–38.1.4.

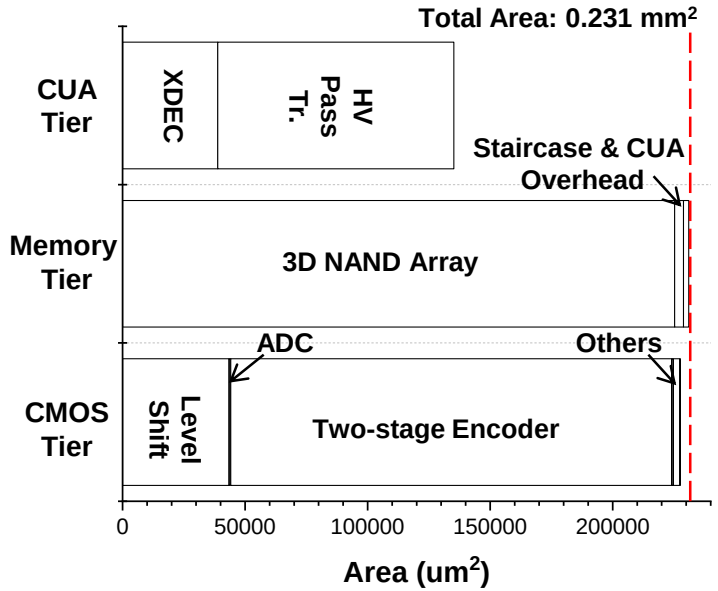
Performance

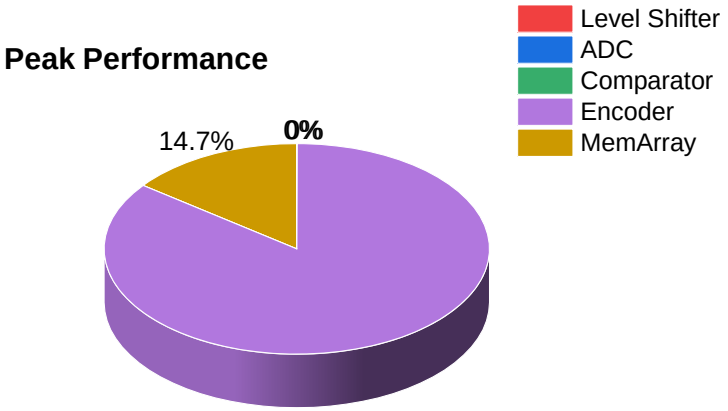
- Peak performance and average performance with different lengths of the genome sequences are calculated.
- Throughput is estimated to produce more than 100 genome sequences classification per second and the average energy consumption for a single genome is 16.86 mJ.
- Comparing with commercial 3D NAND which requires >16 mJ to transfer the class hypervectors, the 3D NAND read energy of 11.7 µJ in the proposed system is >1000x smaller.

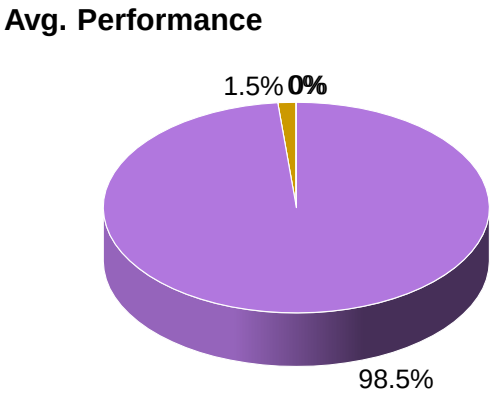
	Seq. Length	CMOS Tier	Memory Tier	Overall
Energy				
Peak Performance	280663	1.568 mJ	11.7 uJ	1.58 mJ
Avg. Performance	3107507	16.85 mJ	11.7 uJ	16.86 mJ
Latency				
Peak Performance	280663	0.7 ms	0.121 ms	0.82 ms
Avg. Performance	3107507	7.7 ms	0.121 ms	7.89 ms

Performance Breakdown

- Area breakdown shows a balanced design between two wafers.
- Latency and energy are dominated by the encoders. Future improvement is needed.







Outline

- Introduction to AI hardware and CIM paradigm
- NeuroSim V1 inference methodologies: area, latency, and energy estimation
- NeuroSim V2 training methodologies: supporting quantization and other non-ideal device effects.
- NeuroSim extension 1: technology updates to 1 nm node and DCIM support
- NeuroSim extension 2: TPU-like architecture benchmark with novel global buffer memory designs
- NeuroSim extension 3: chiplet based integration for ultra-large-scale transformer model
- NeuroSim extension 4: 3D NAND based CIM for hyperdimensional computing
- Demo of running inference engine benchmarking (DNN+NeuroSim V1.4)

Examples of Running DNN+NeuroSim V1.4

- NeuroSim core: consists of multiple .cpp/.h files
  - .cpp files in the Neurosim core can be classified into 4 categories:

  (1) Parameter files: ex) Param.cpp, Technology.cpp

  (2) Circuit module files: ex) Adder.cpp, Buffer.cpp, Comparator.cpp …

  (3) Subarray/ProcessingUnit/Tile/Chip.cpp: Describes the circuit operation in each level of chip hierarchy

  (4) main.cpp: description of the overall chip computation flow, final performance metrics calculation

  - Parameter files should be accessed to set the device/hardware parameters
  - Subarray.cpp, ProcessingUnit.cpp, Tile.cpp, and Chip.cpp include parameter files, circuit module files to estimate the power/area/latency of CIM hardware.

Examples of Running DNN+NeuroSim V1.4

- NeuroSim core: device parameters input in param.cpp
  - Type:
  - SRAM:
  - eNVM:

```
memcelltype = 2;            // 1: cell.memCellType = Type::SRAM
                            // 2: cell.memCellType = Type::RRAM
                            // 3: cell.memCellType = Type::FeFET
cellBit = 2;                        // precision of memory device

/*** parameters for SRAM ***/
// due the scaling, suggested SRAM cell size above 22nm: 160F^2
// SRAM cell size at 14nm: 300F^2
// SRAM cell size at 10nm: 400F^2
// SRAM cell size at 7nm: 600F^2
heightInFeatureSizeSRAM = 10;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 28;        // SRAM Cell width in feature size
widthSRAMCellNMOS = 2;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
minSenseVoltage = 0.1;

/*** parameters for analog synaptic devices ***/
heightInFeatureSize1T1R = 4;        // 1T1R Cell height in feature size
widthInFeatureSize1T1R = 12;         // 1T1R Cell width in feature size
heightInFeatureSizeCrossbar = 2;    // Crossbar Cell height in feature size
widthInFeatureSizeCrossbar = 2;     // Crossbar Cell width in feature size

resistanceOn = 6e3;              // Ron resistance at Vr in the reported measurement data (need to
resistanceOff = 6e3*150;          // Roff resistance at Vr in the reported measurement dat (need t
maxConductance = (double) 1/resistanceOn;
minConductance = (double) 1/resistanceOff;

readVoltage = 0.5;                  // On-chip read voltage for memory cell
readPulseWidth = 10e-9;             // read pulse width in sec
accessVoltage = 1.1;                // Gate voltage for the transistor in 1T1R
resistanceAccess = resistanceOn*IR_DROP_TOLERANCE;          // resistance of access CMOS in 1T1R
writeVoltage = 2;                   // Enable level shifer if writeVoltage > 1.5V
```

Examples of Running DNN+NeuroSim V1.4

- NeuroSim core: V1.4 update - SRAM scaling to 1nm node

```
// 1.4 update : SRAM size update

if (technode>14){
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 10;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 28;        // SRAM Cell width in feature size
}
else if (technode==14){ // Samsung 14 nm
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 10.6;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 30.8;        // SRAM Cell width in feature size
}
else if (technode==10){ // TSMC 10 nm
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 12.8;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 31.25;        // SRAM Cell width in feature size
}
else if (technode==7){ // TSMC IEDM 2016
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 16;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 34.43;        // SRAM Cell width in feature size
```

```
else if (technode==5){ // IRDS
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 19.2;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 43.75;        // SRAM Cell width in feature size
}
else if (technode==3){ // IRDS
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 30;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 68.26;        // SRAM Cell width in feature size
}
else if (technode==2){ // IRDS
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 42;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 120;// 111.42;        // SRAM Cell width in feature size
}
else if (technode==1){ // IRDS
widthSRAMCellNMOS = 1;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
heightInFeatureSizeSRAM = 80;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 144;        // SRAM Cell width in feature size
```

Examples of Running DNN+NeuroSim V1.4

- NeuroSim core: circuit parameters input in param.cpp
  - Technology:

```
// technode: 130     --> wireWidth: 175
// technode: 90      --> wireWidth: 110
// technode: 65      --> wireWidth: 105
// technode: 45      --> wireWidth: 80
// technode: 32      --> wireWidth: 56
// technode: 22      --> wireWidth: 40
// technode: 14      --> wireWidth: 25
// technode: 10, 7   --> wireWidth: 18
technode = 22;                    // Technology
featuresize = 40e-9;              // Wire width for subArray simulation
wireWidth = 40;                   // wireWidth of the cell for Accuracy calculation
```

  - Sub-array size:

```
numRowSubArray = 128;             // # of rows in single subArray
numColSubArray = 128;             // # of columns in single subArray
```

  - ADC type/sharing/resolution:

```
SARADC = false;              // false: MLSA
                             // true: sar ADC
currentMode = true;          // false: MLSA use VSA
                             // true: MLSA use CSA

numColMuxed = 8;                  // How many columns share 1 ADC (for eNVM and FeFET) or parallel SRAM
levelOutput = 32;                 // # of levels of the multilevelSenseAmp output, should be in 2^N forms; e.g. 32 levels --> 5-bit ADC
```

Examples of Running DNN+NeuroSim V1.4

- NeuroSim core: V1.4 update – improved interconnect models & scaling

- Wire width:

```
// 1.4 update: wirewidth
if (wireWidth >= 175) {
    AR = 1.6;
    Rho = 2.01*1e-8;
} else if ((110 <= wireWidth) &&  (wireWidth < 175)) {
    AR = 1.6;
    Rho = 2.20*1e-8;
} else if ((8 <= wireWidth) &&  (wireWidth < 10)){
    AR = 3.00; Rho = 7.87*1e-8;
} else {
    exit(-1); puts("Wire width out of range");
}

Rho = Rho * 1 / (1- ( (2*AR*wireWidth + wireWidth)*barrierthickness / (AR*pow(wireWidth,2) ) ));
```

- Metal0:

```
// 1.4 update: Metal0
if (Metal0 >= 175) {
    AR_Metal0 = 1.6;
    Rho_Metal0 = 2.01*1e-8;
} else if ((110 <= Metal0) &&  (Metal0< 175)) {
    AR_Metal0 = 1.6;
    Rho_Metal0 = 2.20*1e-8;
} else if ((8 <= Metal0) &&  (Metal0< 10)){
    AR_Metal0 = 3.00; Rho_Metal0 = 7.87*1e-8;
} else {
    exit(-1); puts("Wire width out of range");
}

Rho_Metal0 = Rho_Metal0 * 1 / (1- ( (2*AR_Metal0*Metal0 + Metal0)*barrierthickness / (AR_Metal0*pow(Metal0,2) ) ));
```

- Metal1:

```
// 1.4 update: Metal1
if (Metal1 >= 175) {
    AR_Metal1 = 1.6;
    Rho_Metal1 = 2.01*1e-8;
} else if ((110 <= Metal1) &&  (Metal1 < 175)) {
    AR_Metal1 = 1.6;
    Rho_Metal1 = 2.20*1e-8;
} else if ((8 <= Metal1) &&  (Metal1 < 10)){
    AR_Metal1 = 3.00; Rho_Metal1 = 7.87*1e-8;
} else {
    exit(-1); puts("Wire width out of range");
}

Rho_Metal1 = Rho_Metal1 * 1 / (1- ( (2*AR_Metal1*Metal1 + Metal1)*barrierthickness / (AR_Metal1*pow(Metal1,2) ) ));

Metal0_unitwireresis =  Rho_Metal0 / ( Metal0*1e-9 * Metal0*1e-9 * AR_Metal0 );
Metal1_unitwireresis =  Rho_Metal1 / ( Metal1*1e-9 * Metal1*1e-9 * AR_Metal1 );
```

Examples of Running DNN+NeuroSim V1.4

- NeuroSim core: technology parameters in Technology.cpp

-> contains comprehensive device specifications for logic transistors down to 1 nm node.

technology parameters

```
if (transistorType == conventional) {
        if (featureSizeInNano == 130) {
                if (deviceRoadmap == HP) {
                        /* PTM model: 130nm_HP.pm, from http://ptm.asu.edu/ */
                        vdd = 1.3;
                        vth = 128.4855e-3;
                        phyGateLength = 7.5e-8;
                        capIdealGate = 6.058401e-10;
                        capFringe = 6.119807e-10;
                        effectiveResistanceMultiplier = 1.54;    /* from CACTI */
                        current_gmNmos=3.94E+02;
                        current_gmPmos=2.61E+02;
                        currentOnNmos[0]    = 0.93e3;
```

```
else if (featureSizeInNano == 1) {
    // 1.4 update: IRDS 2022
    vdd = 0.6;
    vth = vth_list[6];
    PitchFin= 24e-9;
    phyGateLength = 1.2e-8;

    // 1.4 update: IRDS 2022 - GAA specfic parameters
    max_fin_per_GAA=1;
    max_sheet_num=4;
    thickness_sheet=6*1e-9;
    width_sheet=10*1e-9;

    widthFin= width_sheet; // for drain height calculation
    effective_width=(thickness_sheet+width_sheet)*2;

    capIdealGate = caplist[6] * 1E-18 /  (effective_width*max_sheet_num) ;
    cap_draintotal = cap_draintotallist[6]/ (effective_width);
    capFringe = 0;

    effectiveResistanceMultiplier = eff_res_mul[6]; /* from CACTI */
    current_gmNmos= gm[6];
    current_gmPmos= gm[6];
    gm_oncurrent = gm[6];  // gm at on current
```

Examples of Running DNN+NeuroSim V1.4

- Python wrapper that interfaces with PyTorch
  - inference.py: accuracy estimation w/ hardware effects
  - Input non-ideality effects of characterized devices + algorithmic parameters (dataset, weight/bit resolution)

Model/dataset/mode

Resolution

Hardware properties

Nonideal effects (variation, retention)

```
parser.add_argument('--dataset', default='cifar10', help='cifar10|cifar100|imagenet')
parser.add_argument('--model', default='VGG8', help='VGG8|DenseNet40|ResNet18')
parser.add_argument('--mode', default='WAGE', help='WAGE|FP')

parser.add_argument('--wl_weight', type=int, default=8)
parser.add_argument('--wl_grad', type=int, default=8)
parser.add_argument('--wl_activate', type=int, default=8)
parser.add_argument('--wl_error', type=int, default=8)
# Hardware Properties
# if do not consider hardware effects, set inference=0
parser.add_argument('--inference', type=int, default=0, help='run hardware inference simulation')
parser.add_argument('--subArray', type=int, default=128, help='size of subArray (e.g. 128*128)')
parser.add_argument('--parallelRead', type=int, default=128, help='number of rows read in parallel (<= subArray e.g. 32)')
parser.add_argument('--ADCprecision', type=int, default=5, help='ADC precision (e.g. 5-bit)')
parser.add_argument('--cellBit', type=int, default=1, help='cell precision (e.g. 4-bit/cell)')
parser.add_argument('--onoffratio', type=float, default=10, help='device on/off ratio (e.g. Gmax/Gmin = 3)')
# if do not run the device retention / conductance variation effects, set vari=0, v=0
parser.add_argument('--vari', type=float, default=0., help='conductance variation (e.g. 0.1 standard deviation to generate random variation)')
parser.add_argument('--t', type=float, default=0, help='retention time')
parser.add_argument('--v', type=float, default=0, help='drift coefficient')
parser.add_argument('--detect', type=int, default=0, help='if 1, fixed-direction drift, if 0, random drift')
parser.add_argument('--target', type=float, default=0, help='drift target for fixed-direction drift, range 0-1')
current_time = datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
```

V1.4 update: user defines number of rows read in parallel in the sub-arrays

Examples of Running DNN+NeuroSim V1.4

- Circuit Modules in NeuroSim

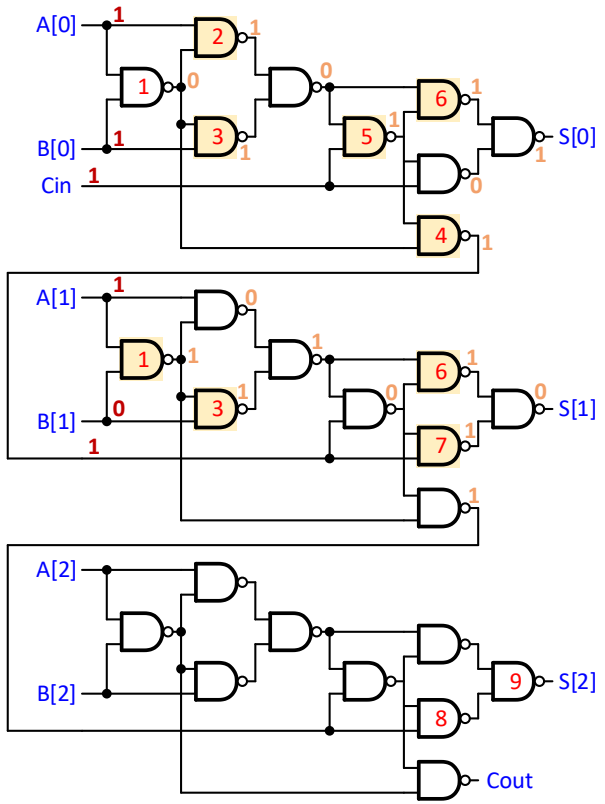
master ▾ DNN_NeuroSim_V1.3 / Inference_pytorch / NeuroSIM / Go to file Add file ▾ ···

neurosim Update Technology.cpp 168523b on Nov 6, 2022 History

..

Adder.cpp	add hw effect for resnet	2 years ago
Adder.h	add hw effect for resnet	2 years ago
AdderTree.cpp	add hw effect for resnet	2 years ago
AdderTree.h	add hw effect for resnet	2 years ago
BitShifter.cpp	add hw effect for resnet	2 years ago
BitShifter.h	add hw effect for resnet	2 years ago
Buffer.cpp	Update Buffer.cpp	5 months ago
Buffer.h	add hw effect for resnet	2 years ago
Bus.cpp	Update Bus.cpp	2 years ago
Bus.h	add hw effect for resnet	2 years ago
Chip.cpp	Update Chip.cpp	3 months ago
Chip.h	add hw effect for resnet	2 years ago
Comparator.cpp	add hw effect for resnet	2 years ago
Comparator.h	add hw effect for resnet	2 years ago
CurrentSenseAmp.cpp	add hw effect for resnet	2 years ago
CurrentSenseAmp.h	add hw effect for resnet	2 years ago
DFF.cpp	add hw effect for resnet	2 years ago

Ex: 3-bit ripple carry adder



-> Extendible to N-bit ripple carry adder

Examples of Running DNN+NeuroSim V1.4

- Circuit Modules in NeuroSim

```
void Adder::Initialize(int _numBit, int _numAdder, double _clkFreq){
        if (initialized)
                cout << "[Adder] Warning: Already initialized!" << endl;

        numBit = _numBit;
        numAdder = _numAdder;
        clkFreq = _clkFreq;

        widthNandN = 2 * MIN_NMOS_SIZE * tech.featureSize;
        widthNandP = tech.pnSizeRatio * MIN_NMOS_SIZE * tech.featureSize;

        EnlargeSize(&widthNandN, &widthNandP, tech.featureSize*MAX_TRANSISTOR_HEIGHT, tech);

        initialized = true;
}
```

Basic setting of the circuit is initialized at the beginning of the simulation based on the input parameters

-> transistor sizing/adder bit etc. are determined based on the input arguments

Examples of Running DNN+NeuroSim V1.4

- Circuit Modules in NeuroSim

```
void Adder::Initialize(int _numBit, int _numAdder, double _clkFreq){
        if (initialized)
                cout << "[Adder] Warning: Already initialized!" << endl;

        numBit = _numBit;
        numAdder = _numAdder;
        clkFreq = _clkFreq;

        widthNandN = 2 * MIN_NMOS_SIZE * tech.featureSize;
        widthNandP = tech.pnSizeRatio * MIN_NMOS_SIZE * tech.featureSize;

        EnlargeSize(&widthNandN, &widthNandP, tech.featureSize*MAX_TRANSISTOR_HEIGHT, tech);

        initialized = true;
}
```

```
void Adder::CalculateArea(double _newHeight, double _newWidth, AreaModify _option) {…
}

void Adder::CalculateLatency(double _rampInput, double _capLoad, double numRead){…
}

void Adder::CalculatePower(double numRead, int numAdderPerOperation) {…
}
```

Basic setting of the circuit is initialized at the beginning of the simulation based on the input parameters

-> transistor sizing/adder bit etc. are determined based on the input arguments

Each circuit module include area, latency and power calculations.

V1.4 update: calculations support GAA special layout

Examples of Running DNN+NeuroSim V1.4

- Circuit Modules in NeuroSim

```
void Adder::CalculateArea(double _newHeight, double _newWidth, AreaModify _option) {
    if (!initialized) {
        cout << "[Adder] Error: Require initialization first!" << endl;
    } else {
        double hNand, wNand;

        // 1.4 update: GAA special layout
        if ((tech.featureSize <= 2e-9) && param->speciallayout) {CalculateGateArea(NAND, 2, MIN_NMOS_SIZE * tech.featureSize,
                                                                                   MIN_NMOS_SIZE * tech.featureSize, tech.featureSize * MAX_TRANSISTOR_HEIGHT,
                                                                                   tech, &hNand, &wNand);}
        else {CalculateGateArea(NAND, 2, widthNandN, widthNandP, tech.featureSize * MAX_TRANSISTOR_HEIGHT, tech, &hNand, &wNand);}

        area = 0;
        height = 0;
        width = 0;
        if (_newHeight && _option==NONE) {   // Adder in multiple columns given the total height
            hAdder = hNand;
            wAdder = wNand * 9 * numBit;

            if (hAdder > _newHeight) {
                cout << "[Adder] Error: A single adder height is even larger than the assigned height ! " << endl;
            } else {
                height = _newHeight;
                width = wAdder * hAdder * numAdder / _newHeight;
            }
        } else if (_newWidth && _option==NONE) { // Adder in multiple rows given the total width
            hAdder = hNand * numBit;
            wAdder = wNand * 9;
```

V1.4: Area for GAA special layout

Area estimation code

Uses the typical CMOS layout rule for estimating the area and special rule for GAA.

-> Given the number of NMOS/PMOS & the maximum possible width/fin number of NMOS/PMOS of standard cell, the area is calculated.

-> Folding in the CPP direction is applied to support various circuit topologies considering the area efficiency.

Examples of Running DNN+NeuroSim V1.4

- Circuit Modules in NeuroSim

```
void Adder::CalculateArea(double _newHeight, double _newWidth, AreaModify _option) {
    if (!initialized) {
        cout << "[Adder] Error: Require initialization first!" << endl;
    } else {
        double hNand, wNand;

        // 1.4 update: GAA special layout
        if ((tech.featureSize <= 2e-9) && param->speciallayout) {CalculateGateArea(NAND, 2, MIN_NMOS_SIZE * tech.featureSize,
                                                                                   MIN_NMOS_SIZE * tech.featureSize, tech.featureSize * MAX_TRANSISTOR_HEIGHT,
                                                                                   tech, &hNand, &wNand);}

        else {CalculateGateArea(NAND, 2, widthNandN, widthNandP, tech.featureSize * MAX_TRANSISTOR_HEIGHT, tech, &hNand, &wNand);}

        area = 0;
        height = 0;
        width = 0;
        if (_newHeight && _option==NONE) {   // Adder in multiple columns given the total height
            hAdder = hNand;
            wAdder = wNand * 9 * numBit;

            if (hAdder > _newHeight) {
                cout << "[Adder] Error: A single adder height is even larger than the assigned height ! " << endl;
            } else {
                height = _newHeight;
                width = wAdder * hAdder * numAdder / _newHeight;
            }
        } else if (_newWidth && _option==NONE) { // Adder in multiple rows given the total width
            hAdder = hNand * numBit;
            wAdder = wNand * 9;

            if (wAdder > _newWidth) {
                cout << "[Adder] Error: A single adder width is even larger than the assigned width ! " << endl;
            } else {
                width = _newWidth;
                height = wAdder * hAdder * numAdder / _newWidth;
```

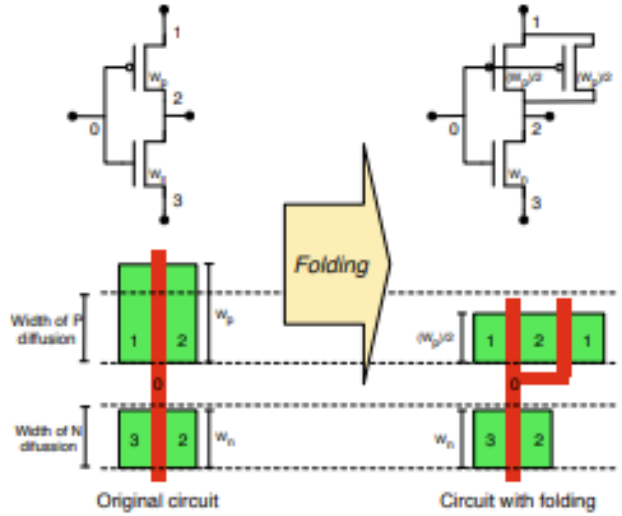


Fig. 2. Example of transistor folding. [9]

Gustavo, et al. Toward Better Layout Design in ASTRAN CAD Tool by Using an Efficient Transistor Folding

Examples of Running DNN+NeuroSim V1.4

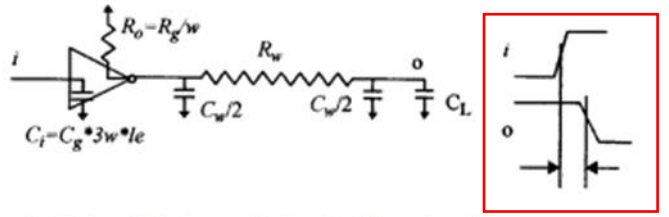
- Circuit Modules in NeuroSim

```
void Adder::CalculateLatency(double _rampInput, double _capLoad, double numRead){
    if (!initialized) {
        cout << "[Adder] Error: Require initialization first!" << endl;
    } else {
        readLatency = 0;
        rampInput = _rampInput;
        capLoad = _capLoad;
        double tr;      /* time constant */
        double gm;      /* transconductance */
        double beta;    /* for horowitz calculation */
        double resPullUp, resPullDown;
        double readLatencyIntermediate = 0;
        double ramp[10];

        ramp[0] = rampInput;

        // 1.4 update: GAA special layout
        // Calibration data pattern is A=1111111..., B=1000000... and Cin=1
        // 1st
        resPullDown = CalculateOnResistance(widthNandN, NMOS, inputParameter.temperature, tech) * 2;
        if ((tech.featureSize <= 2e-9) && param->speciallayout)
        {resPullDown = resPullDown *3.0/2.0;}
        tr = resPullDown * (capNandOutput + capNandInput * 3);
        gm = CalculateTransconductance(widthNandN, NMOS, tech);
        beta = 1 / (resPullDown * gm);
        readLatency += horowitz(tr, beta, ramp[0], &ramp[1]);

        // 2nd
        resPullUp = CalculateOnResistance(widthNandP, PMOS, inputParameter.temperature, tech);
        if ((tech.featureSize <= 2e-9) && param->speciallayout)
        {resPullUp= resPullUp *3.0/2.0;}
        tr = resPullUp * (capNandOutput + capNandInput * 2);
        gm = CalculateTransconductance(widthNandP, PMOS, tech);
        beta = 1 / (resPullUp * gm);
        readLatency += horowitz(tr, beta, ramp[1], &ramp[2]);
```

V1.4: Latency for GAA special layout

Latency estimation code

Horowitz delay estimation model is adopted for latency estimation.
-> Given the input ramp rate/Elmore delay, the propagation delay/output ramp rate is calculated.



Input/output pulse has a certain ramp rate

Fig. 4. Delay of a logic gate driving a load through an RC line.

$$D = \underbrace{R_o \cdot (C_L + C_w) + p}_{\text{gate delay}} + \underbrace{R_w \cdot \left(\frac{C_w}{2} + C_L\right)}_{\text{wire delay}} \quad (1)$$

Bharadwaj S, et al. Speed and Power Scaling of SRAM's

Examples of Running DNN+NeuroSim V1.4

- Circuit Modules in NeuroSim

```
void Adder::CalculateLatency(double _rampInput, double _capLoad, double numRead){
    if (!initialized) {
        cout << "[Adder] Error: Require initialization first!" << endl;
    } else {
        readLatency = 0;
        rampInput = _rampInput;
        capLoad = _capLoad;
        double tr;      /* time constant */
        double gm;      /* transconductance */
        double beta;    /* for horowitz calculation */
        double resPullUp, resPullDown;
        double readLatencyIntermediate = 0;
        double ramp[10];

        ramp[0] = rampInput;

        // 1.4 update: GAA special layout
        // Calibration data pattern is A=1111111..., B=1000000... and Cin=1
        // 1st
        resPullDown = CalculateOnResistance(widthNandN, NMOS, inputParameter.temperature, tech) * 2;
        if ((tech.featureSize <= 2e-9) && param->speciallayout)
        {resPullDown = resPullDown *3.0/2.0;}
        tr = resPullDown * (capNandOutput + capNandInput * 3);
        gm = CalculateTransconductance(widthNandN, NMOS, tech);
        beta = 1 / (resPullDown * gm);
        readLatency += horowitz(tr, beta, ramp[0], &ramp[1]);

        // 2nd
        resPullUp = CalculateOnResistance(widthNandP, PMOS, inputParameter.temperature, tech);
        if ((tech.featureSize <= 2e-9) && param->speciallayout)
        {resPullUp= resPullUp *3.0/2.0;}
        tr = resPullUp * (capNandOutput + capNandInput * 2);
        gm = CalculateTransconductance(widthNandP, PMOS, tech);
        beta = 1 / (resPullUp * gm);
        readLatency += horowitz(tr, beta, ramp[1], &ramp[2]);
```

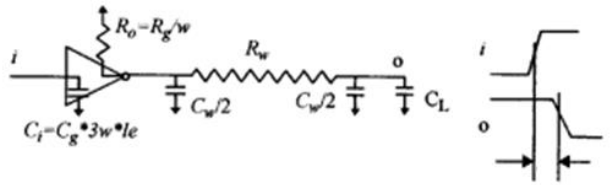


Fig. 4. Delay of a logic gate driving a load through an RC line.

Bharadwaj S, et al. Speed and Power Scaling of SRAM's

1. Circuit RC parasitic modeling

$$D = \underbrace{R_o \cdot (C_L + C_w) + p}_{\text{gate delay}} + \underbrace{R_w \cdot \left(\frac{C_w}{2} + C_L\right)}_{\text{wire delay}} \quad (1)$$

Bharadwaj S, et al. Speed and Power Scaling of SRAM's

2. Elmore delay (D) calculation

$$\text{Latency} = \tau_f \sqrt{\ln(v_s)^2 + \frac{2}{\text{rampInput} \times \tau_f} \beta(1 - v_s)} \quad (3)$$

$$\text{rampOutput} = (1 - v_s)/\text{Latency}$$

Pai-Yu Chen, et al. NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning

3. Propagation delay/output ramp rate estimation through Horowitz delay model

Examples of Running DNN+NeuroSim V1.4

- Circuit Modules in NeuroSim

```
void Adder::CalculatePower(double numRead, int numAdderPerOperation) {
    if (!initialized) {
        cout << "[Adder] Error: Require initialization first!" << endl;
    } else {
        leakage = 0;
        readDynamicEnergy = 0;

        // 1.4 update - updated
        /* Leakage power */
        if ((tech.featureSize == 2e-9) && param->speciallayout) {
            /* Leakage power */
        leakage += CalculateGateLeakage(NAND, 2, MIN_NMOS_SIZE * tech.featureSize, MIN_NMOS_SIZE * tech.featureSize,
                                        inputParameter.temperature, tech) * tech.vdd * 9 * numBit * numAdder * 2/3; }
        else if ((tech.featureSize == 1e-9) && param->speciallayout) {
            /* Leakage power */
        leakage += CalculateGateLeakage(NAND, 2, MIN_NMOS_SIZE * tech.featureSize, MIN_NMOS_SIZE * tech.featureSize,
                                        inputParameter.temperature, tech) * tech.vdd * 9 * numBit * numAdder * 2/3;}
        else {
            /* Leakage power */
        leakage += CalculateGateLeakage(NAND, 2, widthNandN, widthNandP,
                                        inputParameter.temperature, tech) * tech.vdd * 9 * numBit * numAdder;
        }

        /* Read Dynamic energy */
        // Calibration data pattern of critical path is A=1111111..., B=1000000... and Cin=1
        // Only count 0 to 1 transition for energy
        // First stage
        readDynamicEnergy += (capNandInput * 6) * tech.vdd * tech.vdd;    // Input of 1 and 2 and Cin
        readDynamicEnergy += (capNandOutput * 2) * tech.vdd * tech.vdd;  // Output of S[0] and 5
        // Second and later stages
        readDynamicEnergy += (capNandInput * 7) * tech.vdd * tech.vdd * (numBit-1);
        readDynamicEnergy += (capNandOutput * 3) * tech.vdd * tech.vdd * (numBit-1);

        // Hidden transition
        // First stage
        readDynamicEnergy += (capNandOutput + capNandInput) * tech.vdd * tech.vdd * 2;  // #2 and #3
        readDynamicEnergy += (capNandOutput + capNandInput * 2) * tech.vdd * tech.vdd;  // #4
```

V1.4: Power for GAA special layout

Power estimation code

1. Dynamic Energy
-> $C_{node}V_{dd}^2$ * (switching factor) per each cycle is summed up <u>for all nodes.</u>
-> switching factor considers the average switching event during a cycle.

2. Leakage Energy
-> I_{off} * V_{dd} * (averaging factor) * (Idle time) is summed up <u>for all gates.</u>
-> the averaging factor considers the average leakage energy of the transistors in a gate. It is dependent on the type of gate/number of inputs.

Examples of Running DNN+NeuroSim V1.4

- Optional: self-defined network models
  - Create your own file in “models” folder: Replace nn.Conv2d / nn.Linear as QConv2d / QLinear

```
self.conv1 = make_layers([('C', 3, num_planes, 3, 'same', 1)], args, logger)
```

  - make_layers function (just call it, no need to change)

```
def make_layers(cfg, args, logger):
    global name
    layers = []
    for i, v in enumerate(cfg):
        if v[0] == 'M':
            layers += [nn.MaxPool2d(kernel_size=v[1], stride=v[2])]
        if v[0] == 'C':
            in_channels = v[1]
            out_channels = v[2]
            if v[4] == 'same':
                padding = v[3]//2
            else:
                padding = 0
            if args.mode == "WAGE":
                conv2d = QConv2d(in_channels, out_channels, kernel_size=v[3], stride=v[5], padding=padding,
                                 logger=logger,wl_input = args.wl_activate,wl_activate=args.wl_activate,
                                 wl_error=args.wl_error,wl_weight= args.wl_weight,inference=args.inference,onoffratio=args.onoffratio,cellBit=args.cellBit,
                                 subArray=args.subArray,ADCprecision=args.ADCprecision,vari=args.vari,t=args.t,v=args.v,detect=args.detect,target=args.target,
                                 name = 'Conv'+str(name)+'_' )
            elif args.mode == "FP":
                conv2d = nn.Conv2d(in_channels, out_channels, kernel_size=v[3], stride=v[5], padding=padding, bias=False)
            name += 1
            batchnorm = nn.BatchNorm2d(out_channels)
            non_linearity_activation =  nn.ReLU()
            layers += [conv2d, batchnorm, non_linearity_activation]
            in_channels = out_channels
        if v[0] == 'L':
            if args.mode == "WAGE":
                linear = QLinear(in_features=v[1], out_features=v[2],
                                 logger=logger, wl_input = args.wl_activate,wl_activate=args.wl_activate,wl_error=args.wl_error,
                                 wl_weight=args.wl_weight,inference=args.inference,onoffratio=args.onoffratio,cellBit=args.cellBit,
                                 subArray=args.subArray,ADCprecision=args.ADCprecision,vari=args.vari,t=args.t,v=args.v,detect=args.detect,target=args.target,
                                 name='FC'+str(i)+'_' )
            elif args.mode == "FP":
                linear = nn.Linear(in_features=v[1], out_features=v[2], bias=False)
            layers += [linear]
    return nn.Sequential(*layers)
```

Examples of Running DNN+NeuroSim V1.4

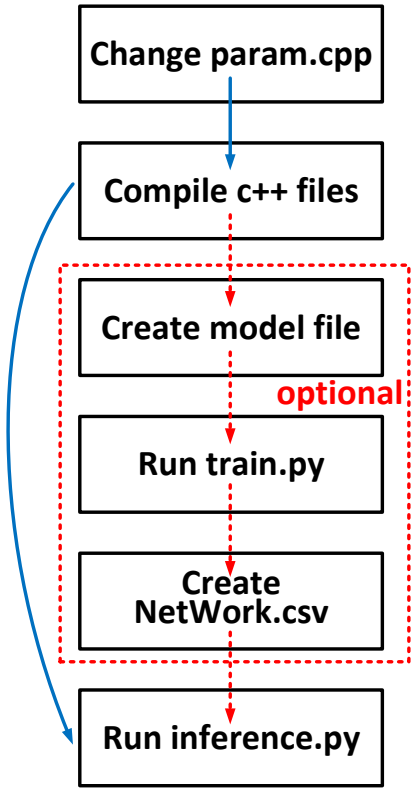
- Optional: self-defined network models
  - Create your own NetWork.csv

Example: VGG-8

	IFM Length	IFM Width	IFM Channel Depth	Kernel Length	Kernel Width	Kernel Depth	Followed by pooling or not?
Layer 1	32	32	3	3	3	128	0
Layer 2	32	32	128	3	3	128	1
Layer 3	16	16	128	3	3	256	0
Layer 4	16	16	256	3	3	256	1
Layer 5	8	8	256	3	3	512	0
Layer 6	8	8	512	3	3	512	1
Layer 7	1	1	8192	1	1	1024	0
Layer 8	1	1	1024	1	1	10	0

Examples of Running DNN+NeuroSim V1.4

- Workflow:



- Output example:
  - Accuracy
  - Floorplan

```
Test set: Average loss: 1.5865, Accuracy: 9007/10000 (90%)
---------- FloorPlan ----------

Tile and PE size are optimized to maximize memory utilization ( = memory mapped by synapse / total memory on chip)

Desired Conventional Mapped Tile Storage Size: 1024x1024
Desired Conventional PE Storage Size: 512x512
Desired Novel Mapped Tile Storage Size: 9x512x512
User-defined SubArray Size: 128x128

---------- # of tile used for each layer ----------
layer1: 1
layer2: 1
...
---------- Speed-up of each layer ----------
layer1: 16
layer2: 4
...
---------- Utilization of each layer ----------
layer1: 0.210938
layer2: 1
...
Memory Utilization of Whole Chip: 96.8584 %

---------- FloorPlan Done ----------
```

Examples of Running DNN+NeuroSim V1.4

- Output example:
  - each layer performance breakdown

```
----------------------------------- Hardware Performance -------------------------------------
-------------------- Estimation of Layer 1 ----------------------
layer1's readLatency is: 377982ns
layer1's readDynamicEnergy is: 1.89268e+06pJ
layer1's leakagePower is: 8.47407uW
layer1's leakageEnergy is: 147340pJ
layer1's buffer latency is: 337150ns
layer1's buffer readDynamicEnergy is: 22681.5pJ
layer1's ic latency is: 24831ns
layer1's ic readDynamicEnergy is: 487920pJ

************************ Breakdown of Latency and Dynamic Energy ************************

---------- ADC (or S/As and precharger for SRAM) readLatency is : 7385.06ns
---------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readLatency is : 7385.06ns
---------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readLatency is : 363212ns
---------- ADC (or S/As and precharger for SRAM) readDynamicEnergy is : 1.03834e+06pJ
---------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readDynamicEnergy is : 155498pJ
---------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readDynamicEnergy is : 698849pJ

************************ Breakdown of Latency and Dynamic Energy ************************
```

Latency & energy of:

- Whole layer
- Buffer
- Interconnection
- ADC
- Accumulation
- Other

Leakage

Examples of Running DNN+NeuroSim V1.4

- Output examples:
  - System-level summary

```
---------- Summary ----------

ChipArea : 4.5554e+07um^2
Chip total CIM array : 1.5408e+06um^2
Total IC Area on chip (Global and Tile/PE local): 6.17954e+06um^2
Total ADC (or S/As and precharger for SRAM) Area on chip : 1.46062e+07um^2
Total Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) on chip : 3.13322e+06um^2
Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, pooling and activation units) : 2.00943e+07um^2

Chip clock period is: 2.05141ns
Chip layer-by-layer readLatency (per image) is: 1.45613e+06ns
Chip total readDynamicEnergy is: 4.03921e+07pJ
Chip total leakage Energy is: 1.19011e+06pJ
Chip total leakage Power is: 628.781uW
Chip buffer readLatency is: 1.00082e+06ns
Chip buffer readDynamicEnergy is: 412969pJ
Chip ic readLatency is: 141884ns
Chip ic readDynamicEnergy is: 8.94262e+06pJ

********** Breakdown of Latency and Dynamic Energy **********

---------- ADC (or S/As and precharger for SRAM) readLatency is : 63462.3ns
---------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readLatency is : 246001ns
---------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readLatency is : 1.14667e+06ns
---------- ADC (or S/As and precharger for SRAM) readDynamicEnergy is : 2.08642e+07pJ
---------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readDynamicEnergy is : 5.16853e+06pJ
---------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readDynamicEnergy is : 1.43594e+07pJ

********** Breakdown of Latency and Dynamic Energy **********


---------- Performance ----------
Energy Efficiency TOPS/W (Layer-by-Layer Process): 24.282
Throughput TOPS (Layer-by-Layer Process): 0.845963
Throughput FPS (Layer-by-Layer Process): 686.75
Compute efficiency TOPS/mm^2 (Layer-by-Layer Process): 0.0185705
---------- Hardware Performance Done ----------

---------- Simulation Performance ----------
Total Run-time of NeuroSim: 151 seconds
---------- Simulation Performance ----------
```

Area

Latency & energy of:
- Whole chip
- Buffer
- Interconnection
- ADC
- Accumulation
- Other peripheries

Leakage

- Energy efficiency
- Throughput
- Compute efficiency

References

- J. Lee, A. Lu, W. Li, **S. Yu** , "NeuroSim V1.4: Extending technology support for digital compute-in-memory towards 1nm node", **IEEE TCAS-I**, accepted.
- A. Lu, J. Lee, T.-H. Kim, M. Karim, R. Park, H. Simka, **S. Yu**, "High-speed emerging memories for AI hardware accelerators," **Nature Reviews Electrical Engineering**, accepted.
- Y. Luo, **S. Yu**, "H3D-Transformer: A heterogeneous 3D (H3D) computing platform for transformer model acceleration on edge devices," **ACM TODAES**, accepted.
- P.-K. Hsu, V. Garg, A. Lu, **S. Yu**, "A heterogeneous platform for 3D NAND-based in-memory hyperdimensional computing engine for genome sequencing applications," **IEEE TCAS-I**, accepted.
- A. Lu, X. Peng, W. Li, H. Jiang, **S. Yu**, "NeuroSim simulator for compute-in-memory hardware accelerator: validation and benchmark," **Frontiers in Artificial Intelligence**, vol. 4, 659060, 2021.
- X. Peng, S. Huang, H. Jiang, A. Lu, **S. Yu**, "DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," **IEEE Trans. CAD**, vol. 40, no. 11, pp. 2306-2319, 2021.
- X. Peng, W. Chakraborty, A. Kaul, W. Shim, M. S. Bakir, S. Datta, **S. Yu**, "Benchmarking monolithic 3D integration for compute-in-memory accelerators: overcoming ADC bottlenecks and maintaining scalability to 7nm or beyond," IEEE International Electron Devices Meeting **(IEDM) 2020.**
- X. Peng, S. Huang, Y. Luo, X. Sun, **S. Yu**, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," IEEE International Electron Devices Meeting **(IEDM) 2019.**

Acknowledgement

CAREER award

(ASCENT, PRSIM, and CHIMES centers)

E2CDA program