Efficient Machine Learning: Algorithms-Circuits-Devices Co-design

Hai Li, Ph.D.
Duke Center for Evolutionary Intelligence
Duke University

Mondays in Memory (MiM) Webinar Series
February 7, 2022

Duke University

Duke Chapel against a blue sky,
Duke campus, Durham, NC, USA

Life is Powered by Machine Learning

Most desirable modern features/services are power by DNNs

Object Localization

Translation

Speech Recognition

Gesture Interpretation

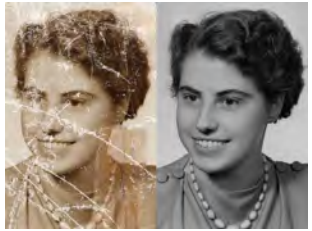
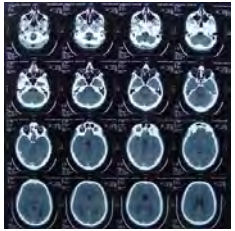


Image Restoration

Gaming AI



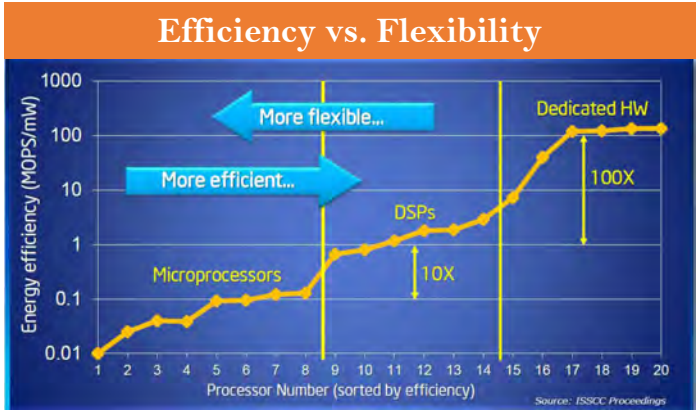
Medical Diagnosis

Malware Detection

Why is Machine Learning Exploding?

Application (Data)	Algorithm (Model)	Hardware (Computing)
		
Big data has resulted in huge training corpus Use this data to intelligently solve previously intractable problems	Theory behind ML and DNN has made leaps and bounds in recent years Can now efficiently deploy solutions to both the cloud and the edge	Transistor scaling allows for massively parallelized tasks Emerging architectures promise huge gains in energy efficiency

DNN Acceleration on Conventional Platforms



Mismatch: Algorithm vs. Hardware

	Algorithm	Hardware
Model/Component Scale	Large	Small/Moderate
Reconfigurability	Easy	Hard
Accuracy vs. Power	Accuracy	Tradeoff
Training Implementation	Easy	Hard
Precision vs. Limited Programmability	Double (high) precision	Low precision (often a few bits)
Connectivity Realization	Easy	Hard

New Architectures and Systems

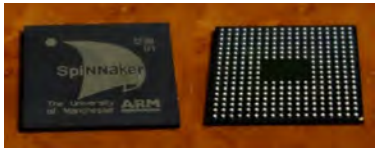
Google Tenser Processing Unit (TPU)

- A custom ASIC specifically for machine learning — and tailored for TensorFlow.
- 10X better-optimized performance/watt for machine learning

Google, https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html

ARM-SpiNNake

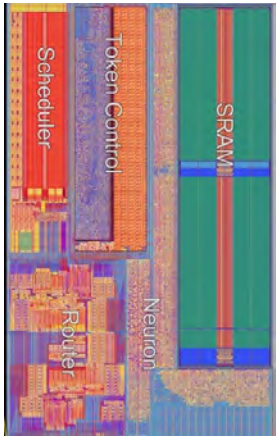
- Human Brain Project
- 18 ARM968 embedded processors
- DDR SDRAM



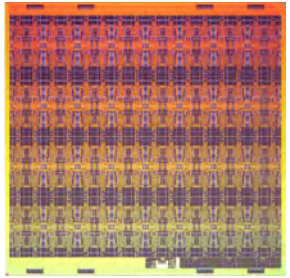
SpiNNaker, http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/

IBM TrueNorth

- 4,096 neurosynaptic cores
- 1 million neurons
- 256 million synapses
- A 65mW real-time neurosynaptic processor



Merolla P A,et al. Science, 2014



Intel Loihi

- 128 neuromorphic cores
- 3 Lakemont x86 cores (Quark)
- 130,000 artificial neurons
- 130 million synapses

Outline

- Introduction
- ReRAM-based Neuromorphic Chip Design
- Cross-Layer Codesign for Efficiency & Reliability
- Conclusion

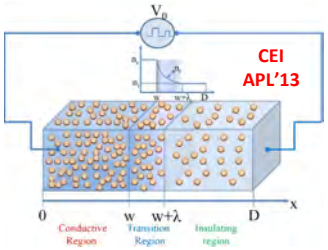
ReRAM-based Neuromorphic Chip

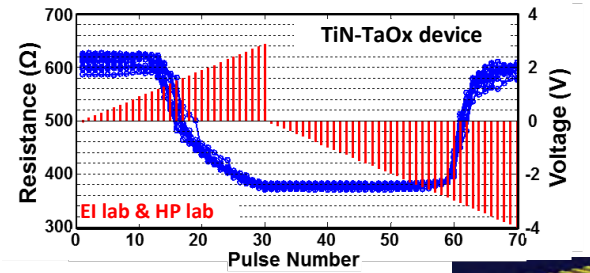
Duke UNIVERSITY

Autumn on Chapel Drive,
Duke campus, Durham, NC, USA

Memristor – Rebirth of Neuromorphic Circuits

Memristor, or Metal-oxide Resistive Random Access Memory a.k.a. ReRAM

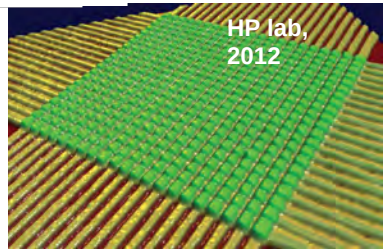




Programmable resistor w/ analog states

Synapse Network

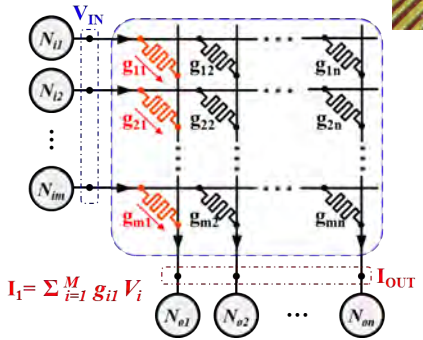
Memristor Crossbar



Natural matrix operation

$$[x_1 \; x_2 \; \dots \; x_m] \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{m1} & g_{m2} & \cdots & g_{mn} \end{bmatrix} = [\, y_1 \;\; y_2 \;\; \cdots \;\; y_n \,]$$

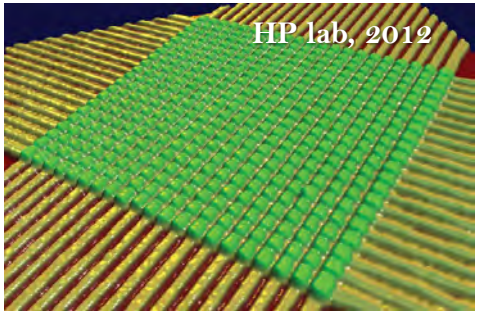
$$y_1 = \Sigma\, x_i \cdot g_{i1}$$

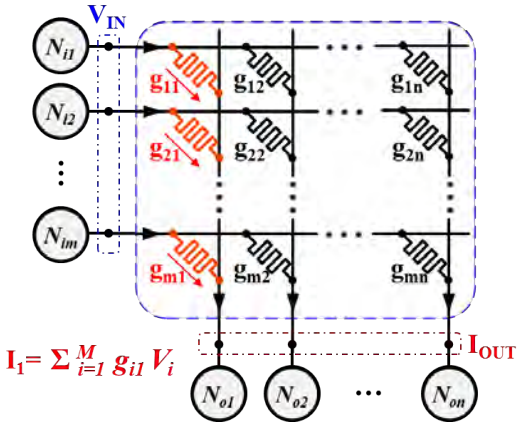


High density

[DAC12, M. Hu et al.]

Memristor (ReRAM) for Computation





Brain-State-in-a-Box (BSB) Recall & Training

$$\Delta A = lr * (X - AX) \otimes X$$

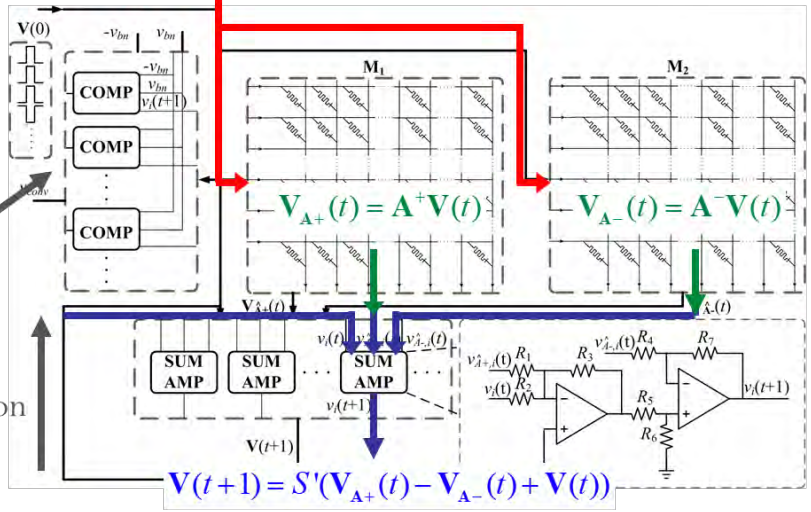
$$A = A + \Delta A$$

$$X(t+1) = S(\alpha \cdot A \cdot X(t) + \lambda \cdot X(t))$$

Input vector V(0)

Comparers detect the converge status.

Next iteration

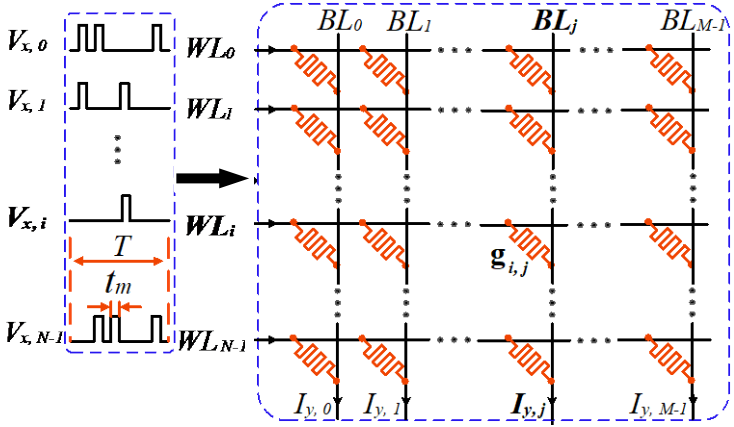


We need two memristor arrays since memristor can only represent positive weights.

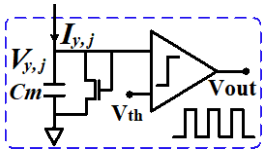
Summing op-amps perform analog voltage signal addition/subtraction

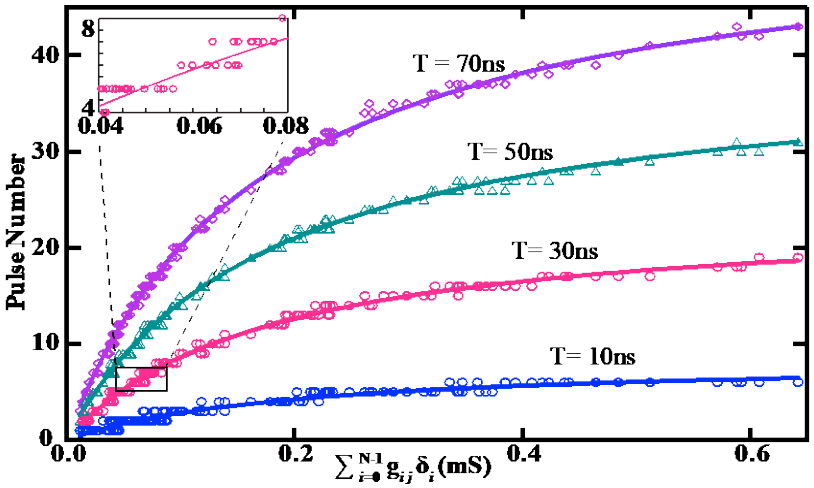
M. Hu, et al., DAC 2012, TNNLS 2014

The Spike-based Neuromorphic Design



Integrate-and-fire Circuit (IFC)





Real Computation Result

$$n_{y,j} = \frac{t_m}{\frac{\alpha}{\sum_{i=0}^{n-1} g_{ij} v_i} + t_0}$$

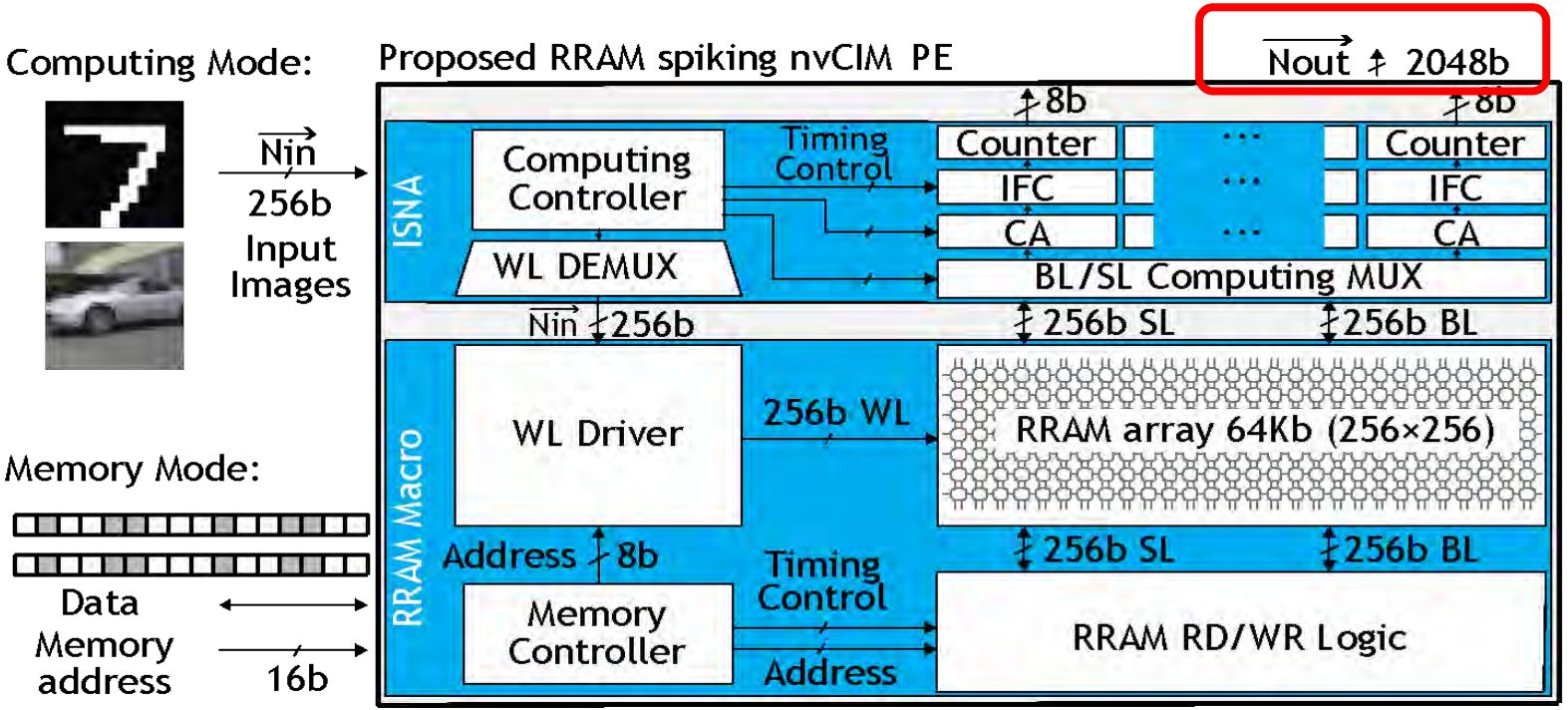
Crossbar IFC

Ideal Computation Result

$$n_{y,j}(t) \propto \int_{\tau=0}^{t} \sum_{i=0}^{N-1} g_{ij}\, V_{x,i}(\tau)\, d\tau$$

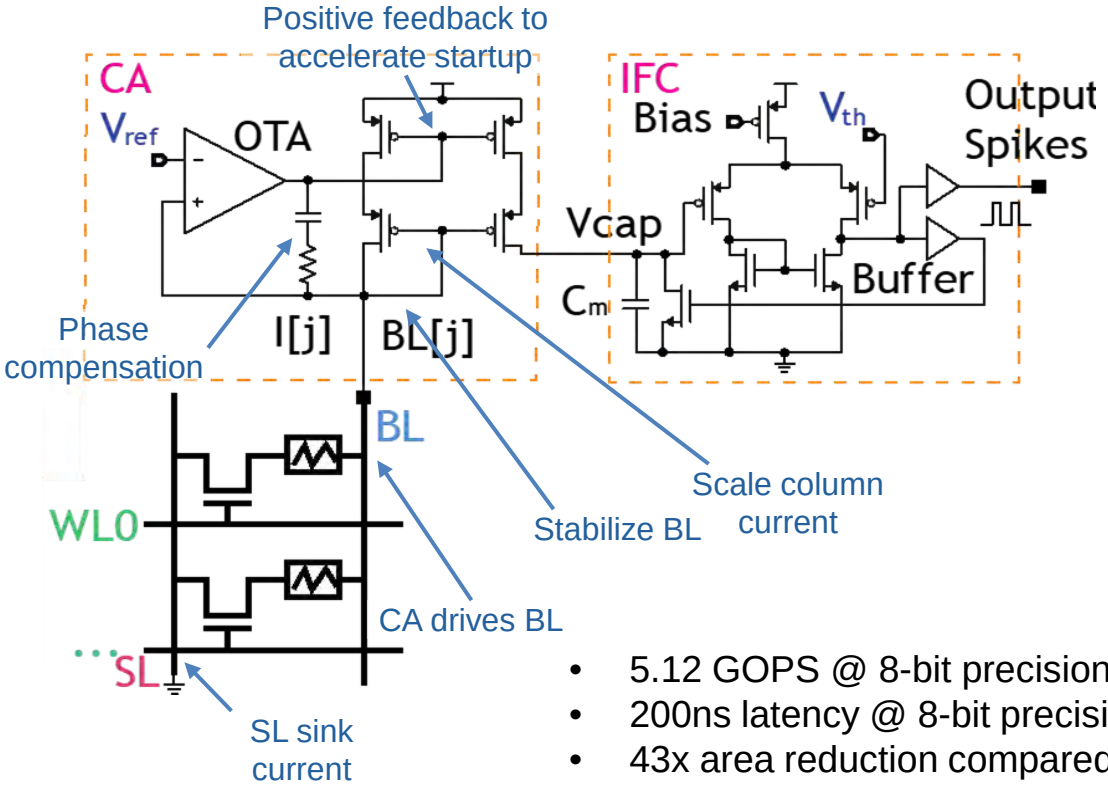
C. Liu, et al., DAC 2015

Processing Engine with In-Situ Neuron Activation (ISNA)



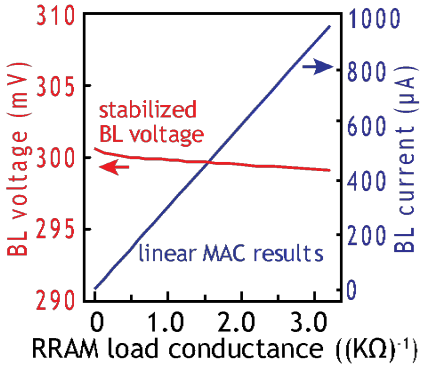
[B. Yan, et al., VLSI Symp. 2019]

Spike Conversion

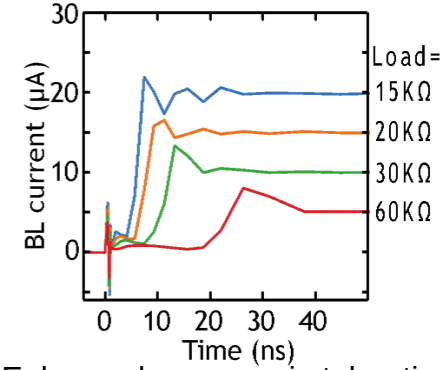


[B. Yan, et al., VLSI Symp. 2019]

- 5.12 GOPS @ 8-bit precision
- 200ns latency @ 8-bit precision
- 43x area reduction compared to ADC design by K. Ohhata (JSSC 2019)



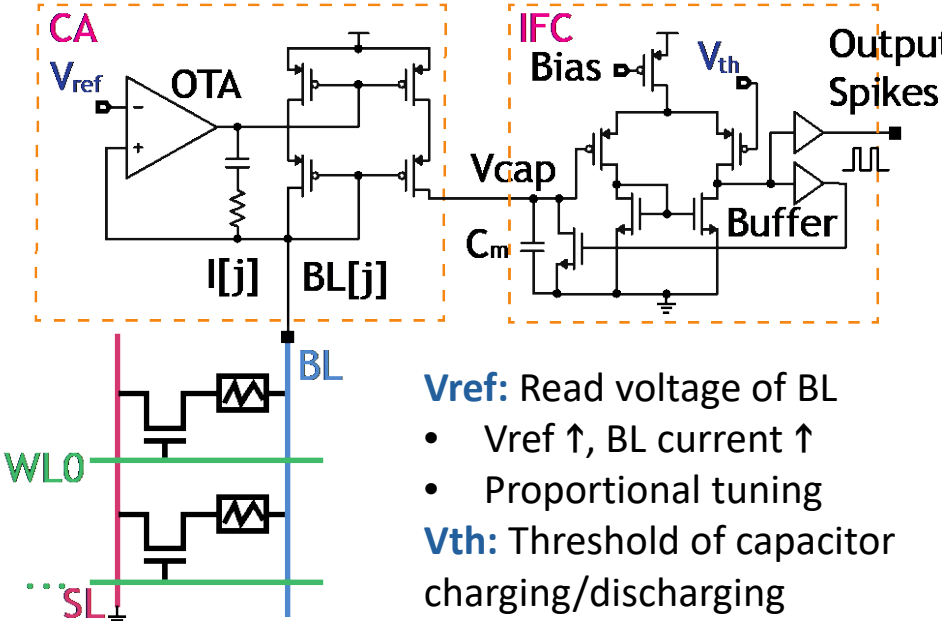
Tradeoff between large input current range and response speed



Enlarge phase margin tolerating capacitor positive feedback

In Situ Nonlinear Activation Function

- How to adjust nonlinear region shape?

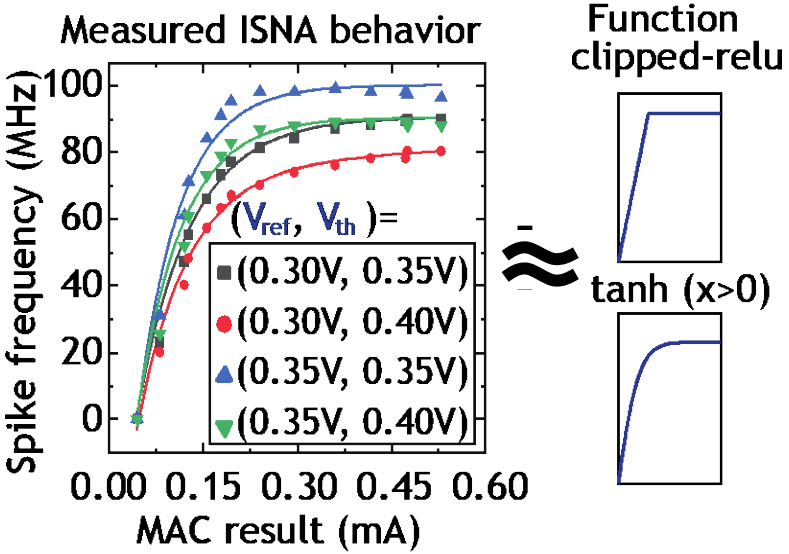


Vref: Read voltage of BL

- Vref ↑, BL current ↑
- Proportional tuning

Vth: Threshold of capacitor charging/discharging

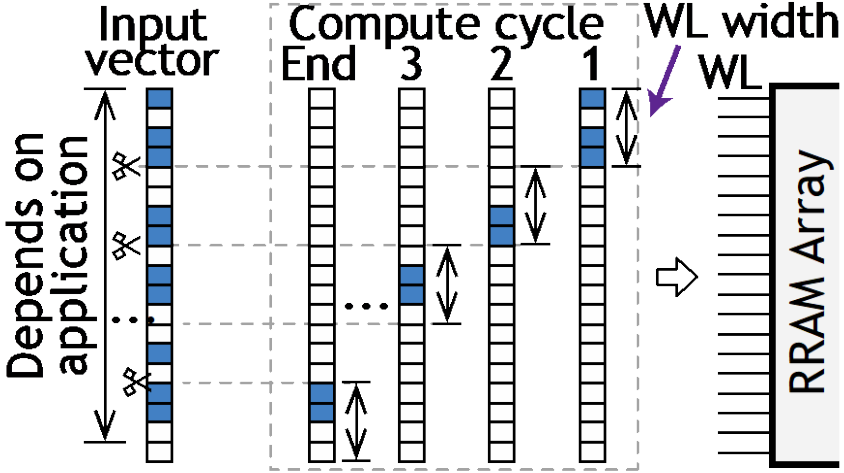
- Vth ↓, Charging/discharging ↑
- Distorted tuning

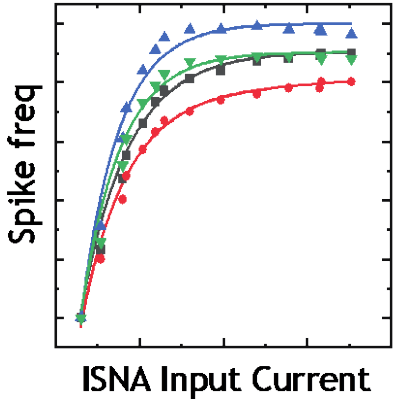


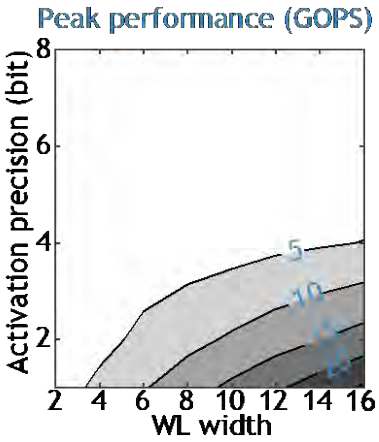
B. Yan, et al., VLSI Symp. 2019

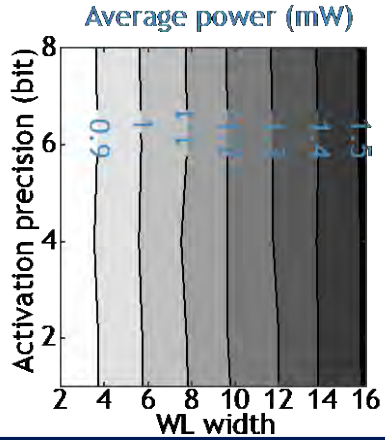
Configurable Activation Precision

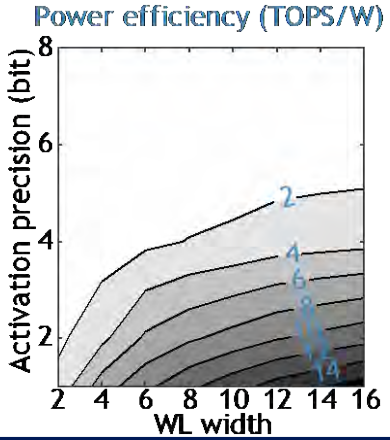
- Tradeoff between high-precision and high-performance
- Useful for system-level fine-grained optimization





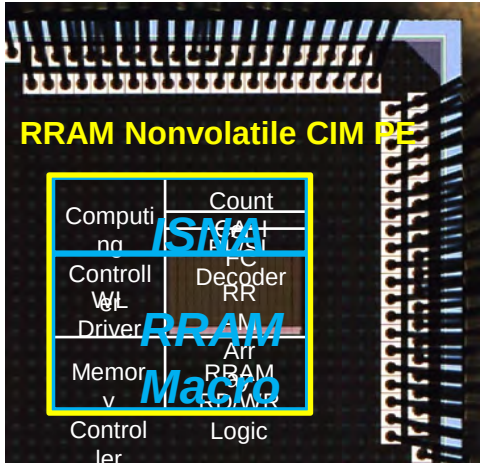




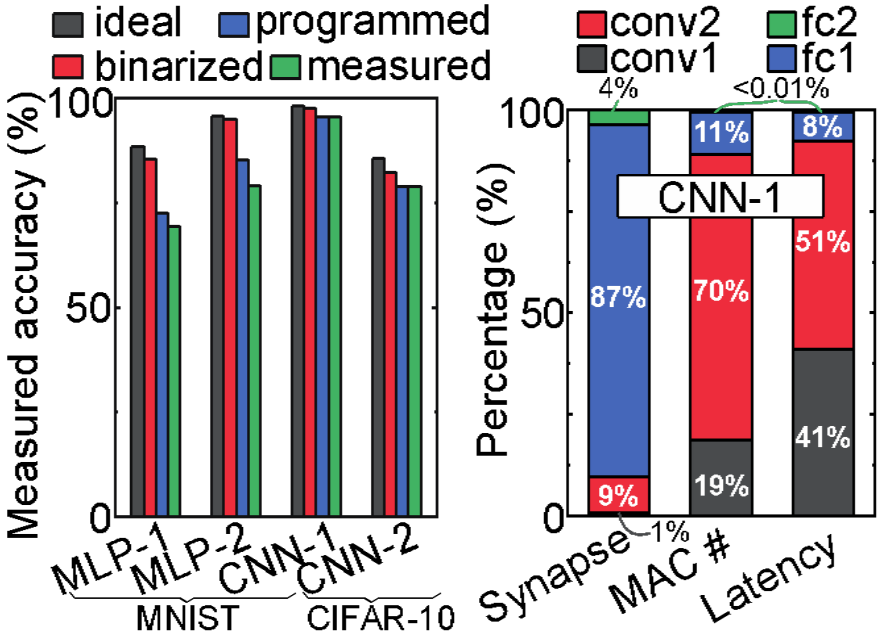


B. Yan, et al., VLSI Symp. 2019

Measured Neural Network Results



Technology	0.15µm CMOS +HfO RRAM
Macro capacity	64K (256×256)
Clock frequency	50MHz
Energy efficiency	0.257pJ/Mac
Average power	1.52 mW

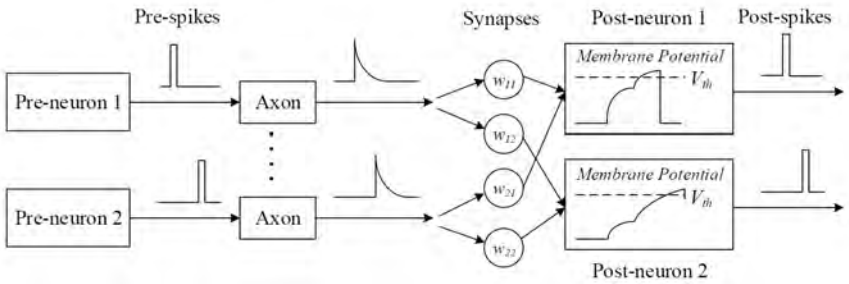


Demo video online: https://bit.ly/AICHIP

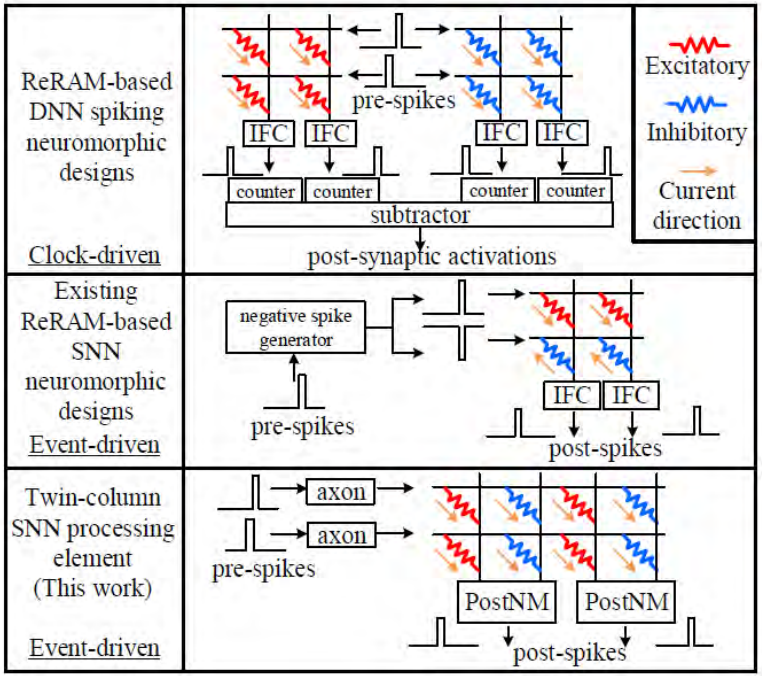
[B. Yan, et al., VLSI Symp. 2019]

Adaptable Threshold Spike-timing Neuromorphic Design

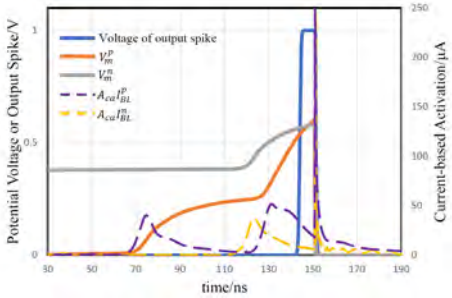
- Neuron dynamics under TTFS scheme

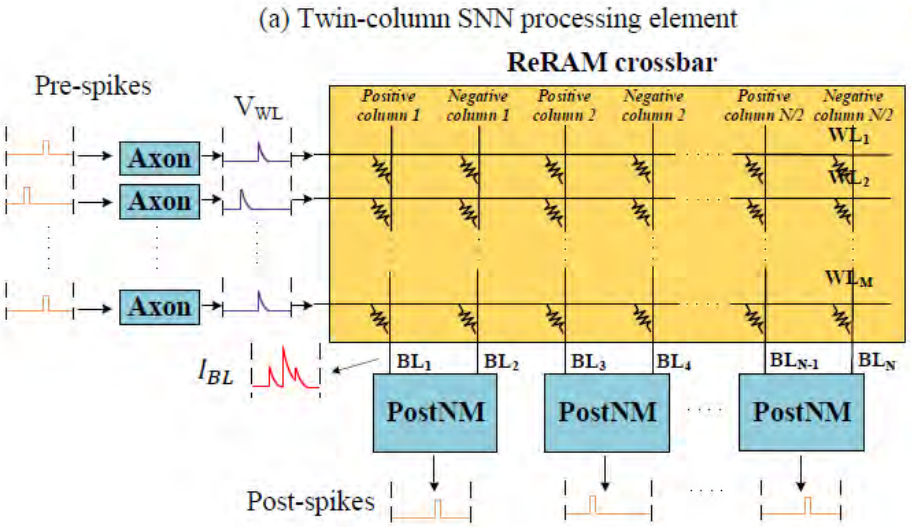


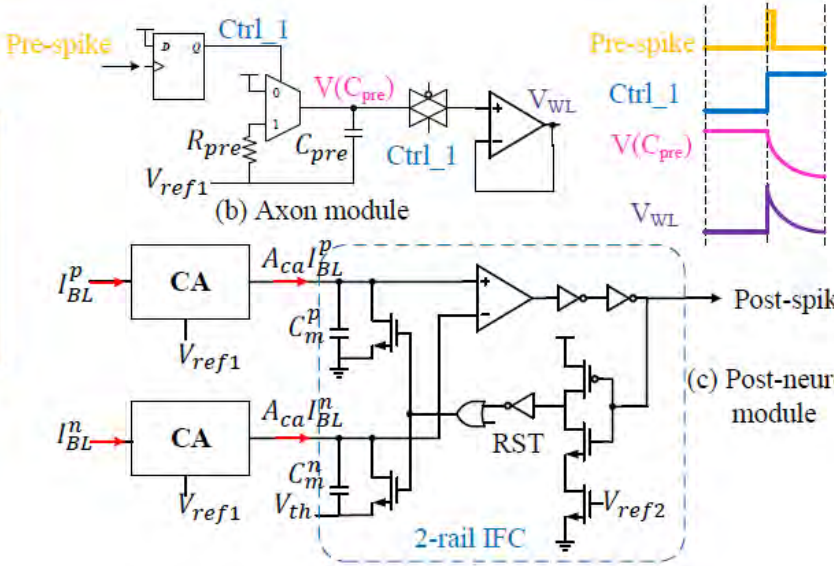
- How to map the excitatory and inhibitory synapses to the ReRAM crossbar array?
- How to leverage the event-driven property of SNNs to reduce energy consumption as most neurons are in the idle state?



Design Overview

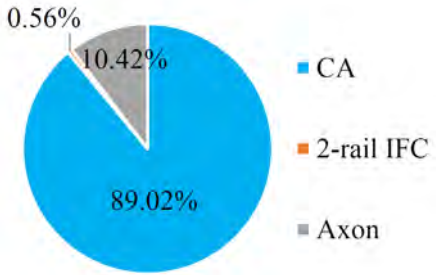


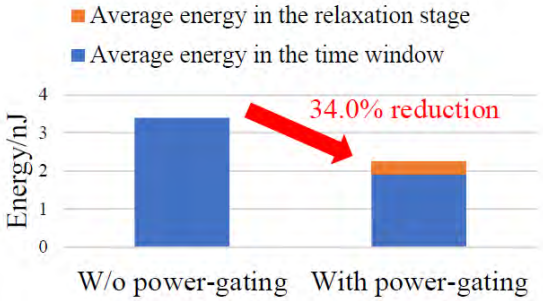




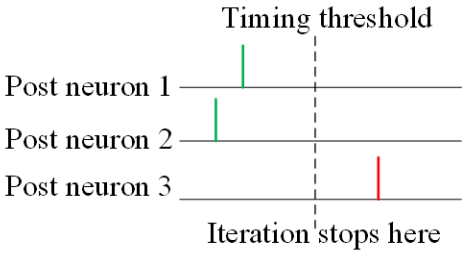
Design Details

- **Power-gating of Post-neuron Module**
- The CA dominates the overall power due to the power-consuming opamp within it.
- In the TTFS-based SNNs, each neuron fires only a single spike during each inference time window.
- Applying power-gating to the post-neuron module.



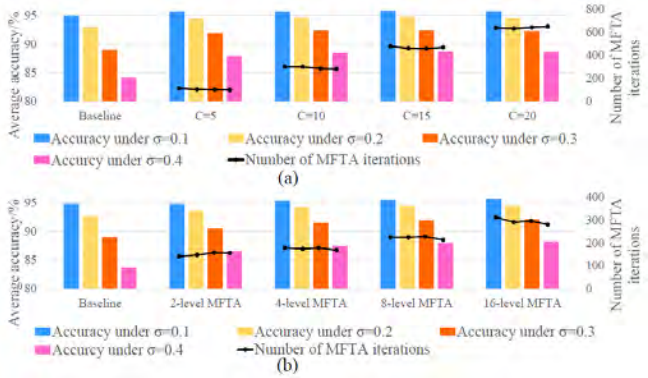


- **Timing threshold adjustment (TTA)**
- Earlier spikes represent stronger activations, while later spikes contribute less to neuron's potential accumulation in the next layer.
- TTA scheme speeds up one iteration by allowing only a portion of neurons in each layer to propagate their spikes.

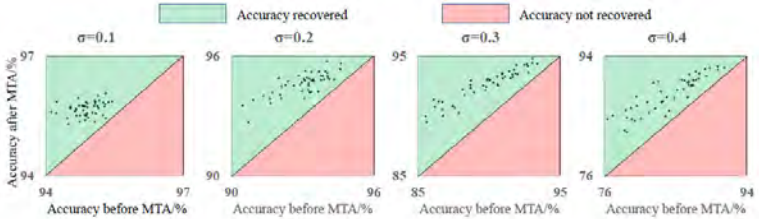


Evaluation

- **Accuracy analysis of MFTA**

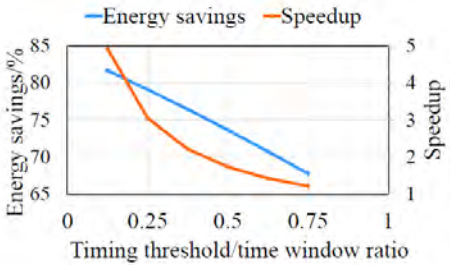


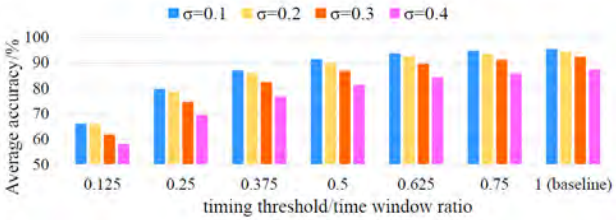
Design space exploration



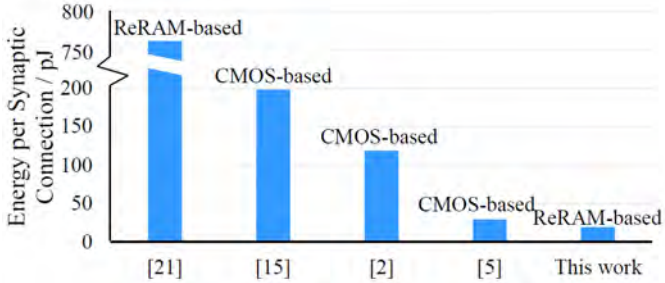
Monte-Carlo simulation results

- **Tradeoff in TTA**





- **Energy & Speed**



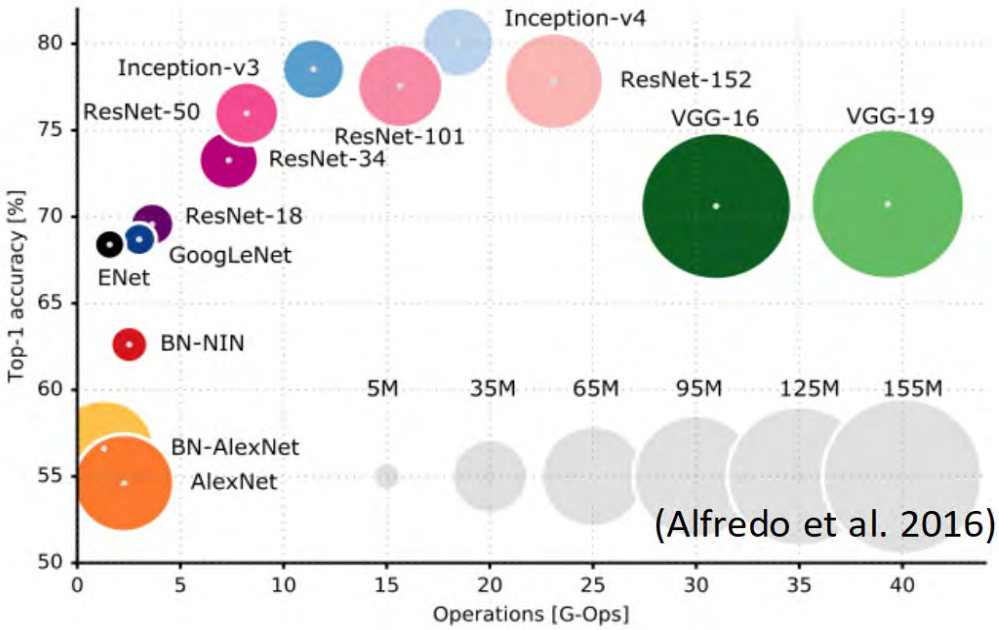
Cros-Layer Co-Design for Efficiency & Reliability

Duke UNIVERSITY

Arches beneath the Chapel,
Duke campus, Durham, NC, USA

Why Sparse DNN Models?

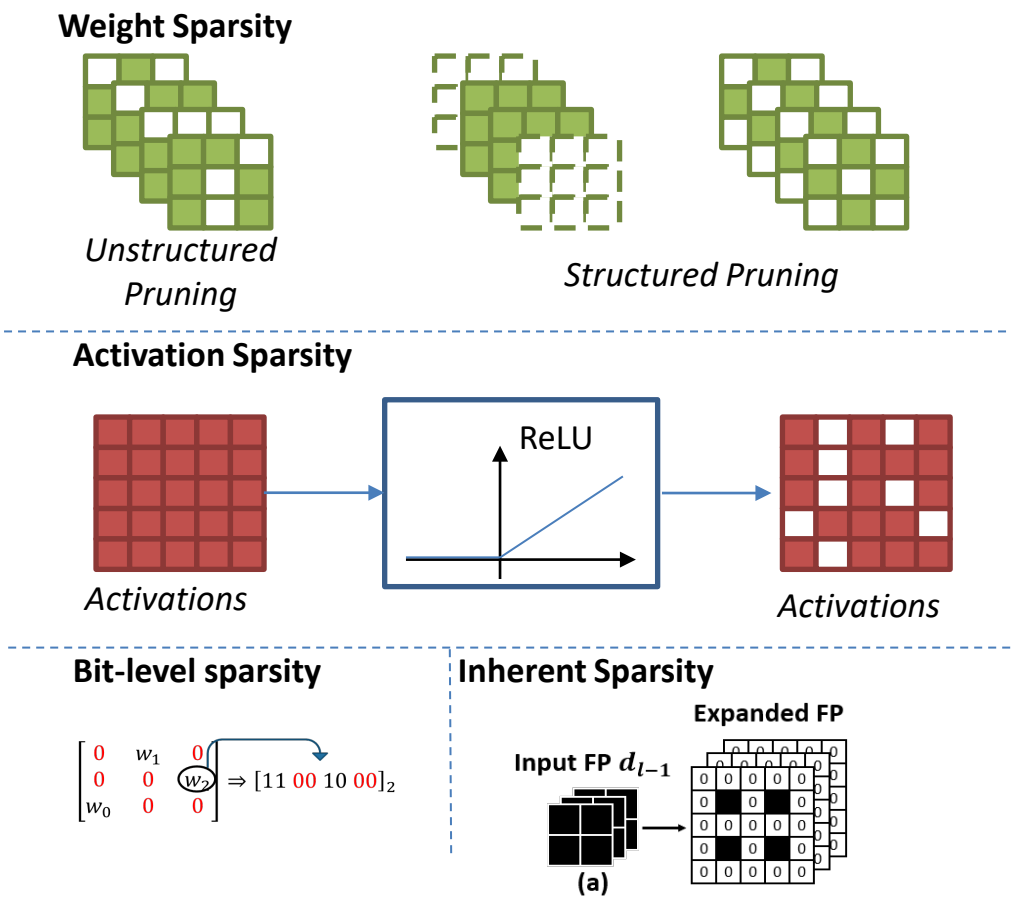
- Ever-growing size brough challenges to the deployment of the DNN models
- The parameters of DNNs are redundant
- We can reduce the computation and bandwidth requirement by eliminating those redundancy



Canziani, et al. "An analysis of deep neural network models for practical applications." *arXiv preprint arXiv:1605.07678* (2016).

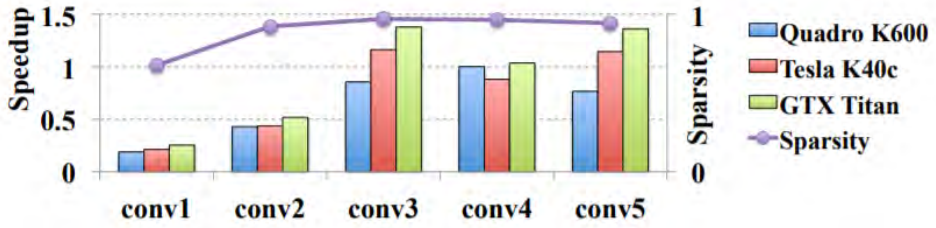
Sparse DNNs

- Weight Sparsity
  - Unstructured Pruning
  - Structured Pruning
- Activation Sparsity
- Bit-level Sparsity
- Inherent Sparsity



The Need of Structural Sparsity

- Non-structured sparsity may not bring much speedup on traditional platforms like GPUs

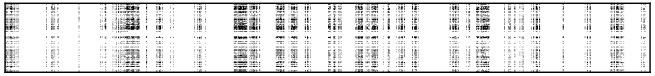


Non-structured sparsity

conv2_1: weight sparsity (col:8.7% row:19.5% elem:94.6%)

Structured sparsity

conv2_1: weight sparsity (col:75.2% row:21.9% elem:91.5%)



5.17X speedup

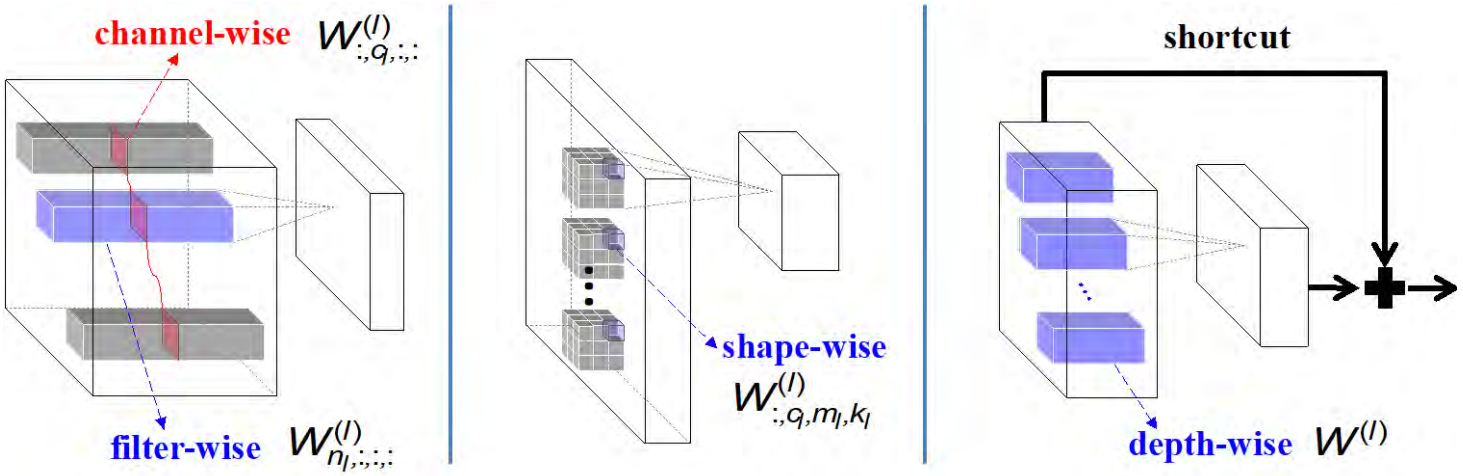
- Structured sparsity is more hardware-friendly
- Structured sparsity can be achieved by having all the parameters within a structured group become zero or nonzero simultaneously

[NeurIPS'16, W. Wen et. al.]

Structurally Sparse Deep Neural Networks

In a nutshell, a group can be ***<u>any</u>*** form of a weight block, depending on what sparse structure you want to learn

In CNNs, a group of weights can be a <u>channel</u>, a <u>3D filter</u>, a <u>2D filter</u>, a <u>filter shape fiber</u> (i.e., a weight column), and even a <u>layer</u> (in ResNets).



Group Lasso Regularization is *ALL* You Need

Step 1: Weights are split to G groups $\mathbf{w}^{(1..G)}$

e.g. (w_1, w_2, w_3, w_4, w_5) -> group 1: (w_1, w_2, w_3), group 2: (w_4, w_5)

Step 2: Add group Lasso on each group $\mathbf{w}^{(g)}$ $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$ i.e. vector length

sqrt($w_1^2 + w_2^2 + w_3^2$), sqrt($w_4^2 + w_5^2$)

Step 3: Sum group Lasso over all groups as a regularization: $R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g,$

sqrt($w_1^2 + w_2^2 + w_3^2$) + sqrt($w_4^2 + w_5^2$)

Step 4: SGD optimizing $\arg\min_{\mathbf{w}} \left\{ E(\mathbf{w}) \right\} = \arg\min_{\mathbf{w}} \left\{ E_D(\mathbf{w}) + \lambda_g \cdot R_g(\mathbf{w}) \right\}$

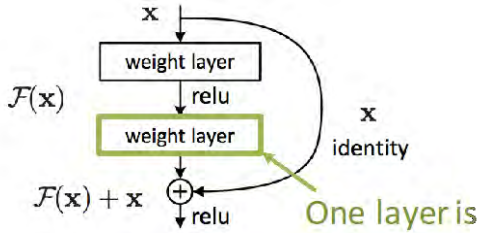
We refer to our method as ***Structured Sparsity Learning (SSL)***

NIPS'16; ICLR'18; ICASSP'19

- is supported by the library of Intel Nervana Neural Network Processors.
- is adopted by Intel's newest NLP accelerator.
- applied to Microsoft Bing
- is adopted by SF-Technology, achieving 2X performance improvement in their datacenter.

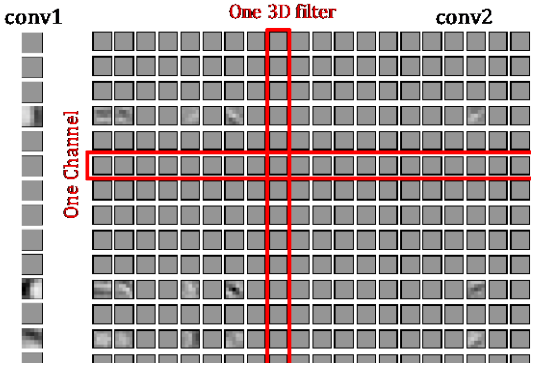
Experiments

Removing layers in ResNets

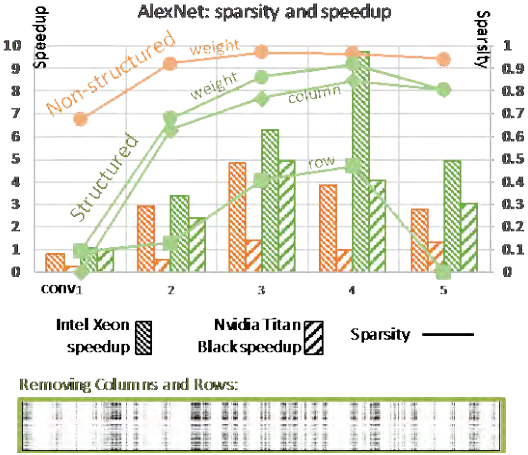


K. He et al. CVPR 2016. one group.

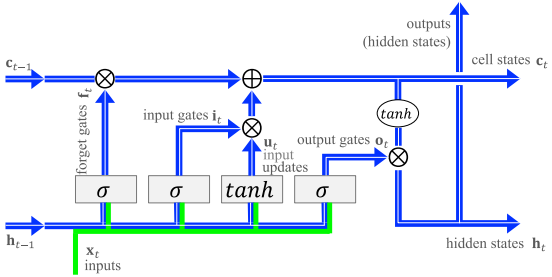
Structurally Sparse *LeNet* – Removing Channels and Filters



Structurally Sparse Matrices



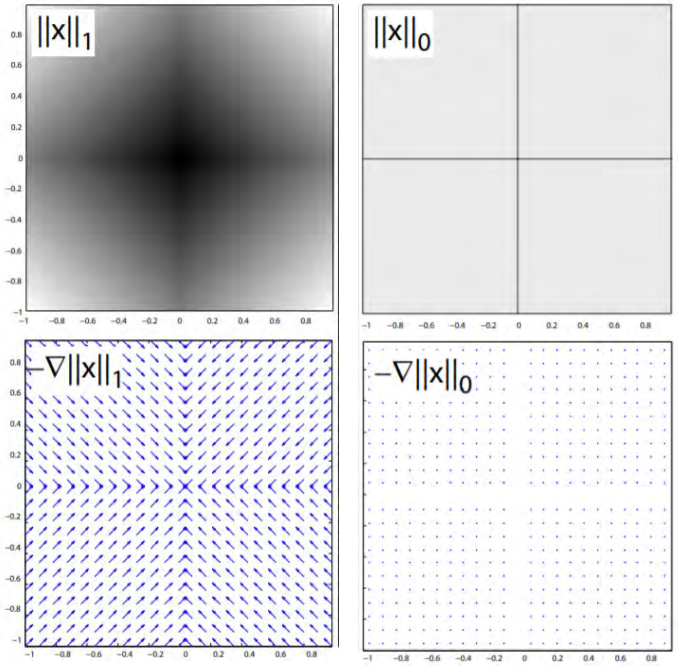
Structurally Sparse LSTMs: Removing Hidden Structures



[ICLR'18 W. Wen et. al.] , [ICASSP'19 J. Zhang et.al.]

Sparsity-inducing Regularizer for DNN

- **L1 regularizer (sum of absolute values)**
  - Used for sparsity since 1996
  - Differentiable, convex, easy to optimize
  - Proportional to the scaling, can only “scale down” all the elements with the same speed, undesired penalty on large elements
- **L0 regularizer (number of nonzero values)**
  - Reflect the sparsity by definition
  - Scale-invariant
  - No useful gradients
  - Need additional tricks for applying on DNN pruning (i.e. stochastic approximation, ADMM), making the problem complicated

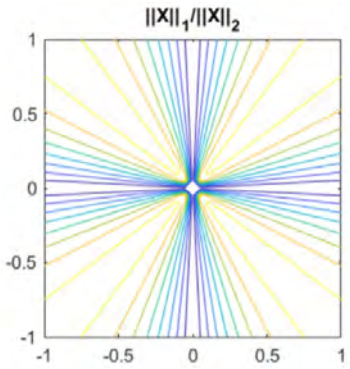


Hastie, Trevor, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

Moving Beyond Lasso

- **Goal:** Find a sparsity-inducing regularizer that is both differentiable and scale-invariant
- **Hoyer-square regularizer** $H_S(W) = \frac{(\sum_i |w_i|)^2}{\sum_i w_i^2}.$
  - Square of the L1/L2 ratio, differentiable, scale-invariant, same range and similar minima structure as L0
- **Group-HS regularizer for structural pruning**
  - Apply Hoyer-Square regularizer over the L2 norm of the group
  - Group weights along channels and filters

$$G_H(W) = \frac{(\sum_{g=1}^{G} ||w^{(g)}||_2)^2}{\sum_{g=1}^{G} ||w^{(g)}||_2^2} = \frac{(\sum_{g=1}^{G} ||w^{(g)}||_2)^2}{||W||_2^2}.$$



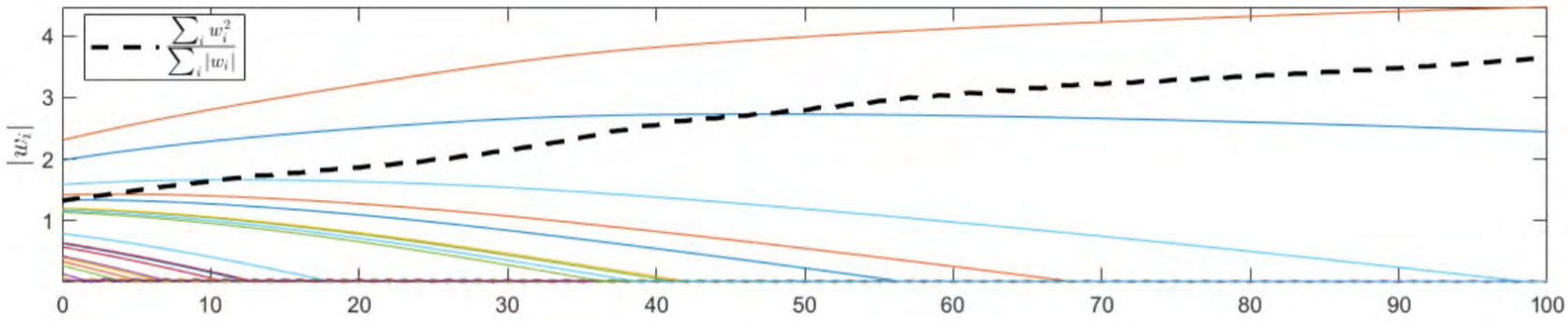
[ICLR'20 H. Yang et.al.]

Optimizing with Hoyer-Square Regularizer

- **Gradient**

$$\partial_{w_j} H_S(W) = 2sign(w_j)\frac{\sum_i |w_i|}{(\sum_i w_i^2)^2}(\sum_i w_i^2 - |w_j|\sum_i |w_i|).$$

- **Induced auto-trimming**
  - Turn weights with smaller absolute value to zero while protecting larger weights
  - Gradually extend trimming threshold as more weights coming close to zero



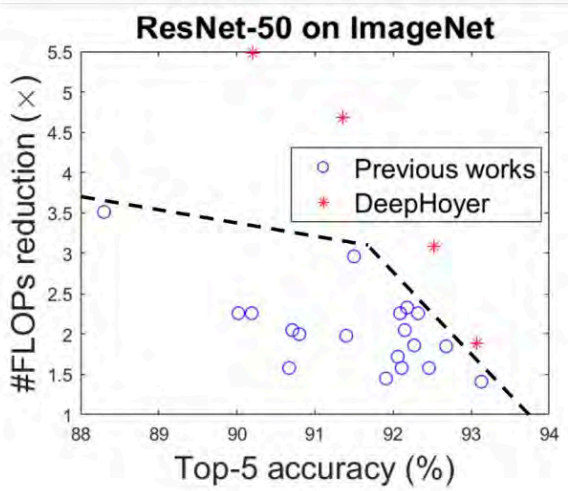
High Element-wise Sparsity & Large FLOPs Reduction

- Further improve the model compression rate of element-wise pruning w/o accuracy loss
- Outperform the Pareto frontier of performance-#FLOPs tradeoff

Element-wise pruning results on AlexNet w/o accuracy loss

Layer	Nonzero wights left after pruning					
	Baseline	Han et al.	Zhang et al.	Ma et al.	Hoyer	HS
CONV1	34.8K	29.3K	28.2K	24.2K	**21.3K**	31.6K
CONV2	307.2K	116.7K	**61.4K**	109.9K	77.2K	148.4K
CONV3	884.7K	309.7K	**168.1K**	241.2K	192.0K	299.3K
CONV4	663.5K	245.5K	**132.7K**	207.4K	182.6K	275.6K
CONV5	442.2K	163.7K	**88.5K**	134.7K	116.6K	197.1K
FC1	37.7M	3.40M	1.06M	**0.763M**	1.566M	**0.781M**
FC2	16.8M	1.51M	0.99M	1.070M	0.974M	**0.650M**
FC3	4.10M	1.02M	0.38M	0.505M	0.490M	**0.472M**
Total	60.9M	6.8M	2.9M	3.05M	3.62M	**2.85M**

Especially effective for large FC layers



Mix-Precision with Bit-Level Sparsity

- Selecting the optimal precision for each layer introduced a large and discrete design space.
- For a fixed-point quantized matrix, when can its precision be reduced?
  - MSB=0 for all elements: precision can reduce directly

$$\begin{bmatrix}6\\3\end{bmatrix} \equiv \begin{bmatrix}0 & 1 & 1 & 0\\0 & 0 & 1 & 1\end{bmatrix}_2 \equiv \begin{bmatrix}1 & 1 & 0\\0 & 1 & 1\end{bmatrix}_2$$

  - LSB=0 for all elements: precision can be reduced with scaling factor 2

$$\begin{bmatrix}10\\4\end{bmatrix} \equiv \begin{bmatrix}1 & 0 & 1 & 0\\0 & 1 & 0 & 0\end{bmatrix}_2 \equiv 2 \times \begin{bmatrix}1 & 0 & 1\\0 & 1 & 0\end{bmatrix}_2$$

- MP quantization scheme can be explored by inducing structural bit-level sparsity

[ICLR'21, H. Yang et al.]

Mix-Precision with Bit-Level Sparsity

Bit representation conversion

FP/BP loop: each bit as **float** trainable variable

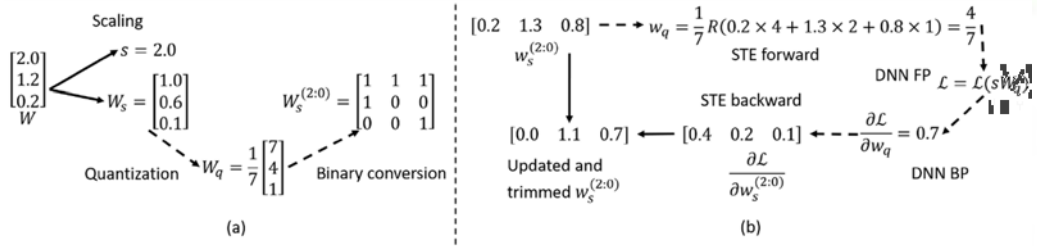


Table 2: Quantization results of ResNet-20 models on the CIFAR-10 dataset. BSQ is compared with DoReFa-Net (Zhou et al., 2016), PACT (Choi et al., 2018), LQ-Net (Zhang et al., 2018), DNAS (Wu et al., 2019) and HAWQ (Dong et al., 2019). "MP" denotes mixed-precision quantization.

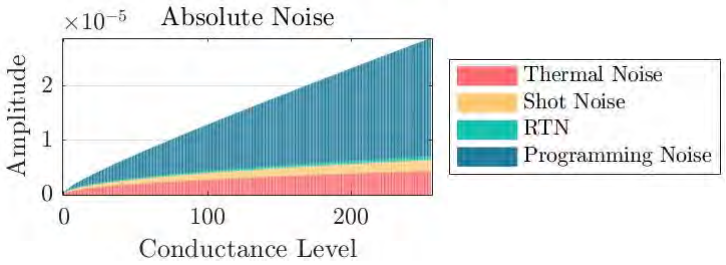
Benchmarks					BSQ		
Act. Prec.	Method	Weight Prec.	Comp (×)	Acc (%)	α	Comp (×)	Acc (%)
32-bit	Baseline	32	1.00	92.62			
	LQ-Nets	3	10.67	92.00	5e-3	14.24	92.77
	DNAS	MP	11.60	92.72	7e-3	19.24	91.87
	LQ-Nets	2	16.00	91.80			
4-bit	HAWQ	MP	13.11	92.22	5e-3	14.24	92.32
3-bit	LQ-Nets	3	10.67	91.60	2e-3	11.04	92.16
	PACT	3	10.67	91.10	5e-3	16.37	91.72
	DoReFa	3	10.67	89.90			
2-bit	LQ-Nets	2	16.00	90.20			
	PACT	2	16.00	89.70	5e-3	18.85	90.19
	DoReFa	2	16.00	88.20			

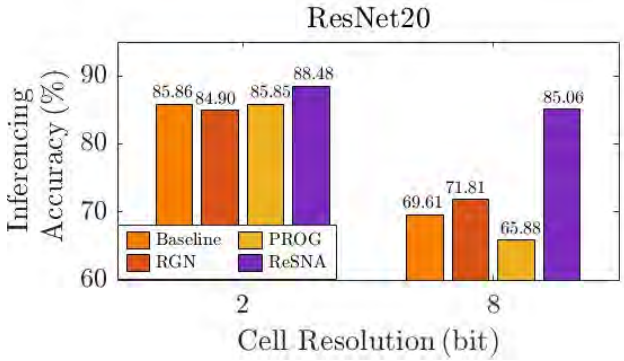
- Start with 8-bit quantized model
- Bit-level group LASSO : $B_{GL}(W^g) = \sum_{b=0}^{n-1} \left\| \left[W_p^{(b)}; W_n^{(b)} \right] \right\|_2$
- Overall training objective : $\mathcal{L} = \mathcal{L}_{CE}(W_q^{(1:L)}) + \alpha \sum_{l=1}^{L} \frac{\#Para(W^l) \times \#Bit(W^l)}{\#Para(W^{(1:L)})} B_{GL}(W^l)$
- Apply periodic re-quantization and precision adjustment throughout the training process

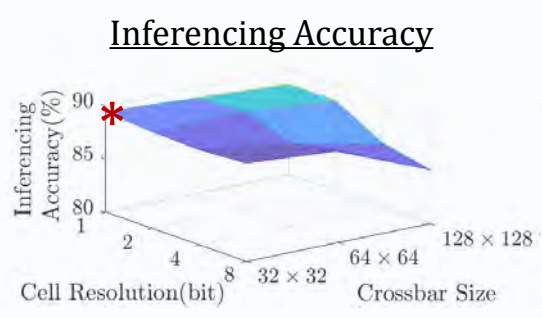
[ICLR'21, H. Yang et al.]

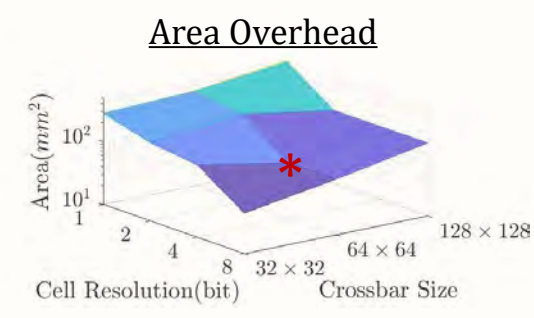
System-level Reliability Improvement

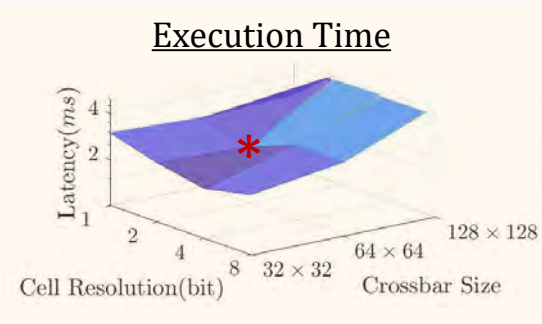
- **ReSNA: ReRAM-based Stochastic-Noise-Aware Training**

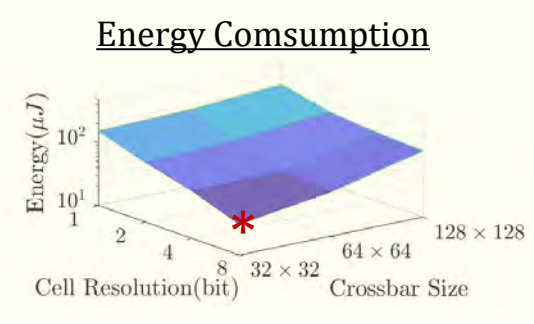




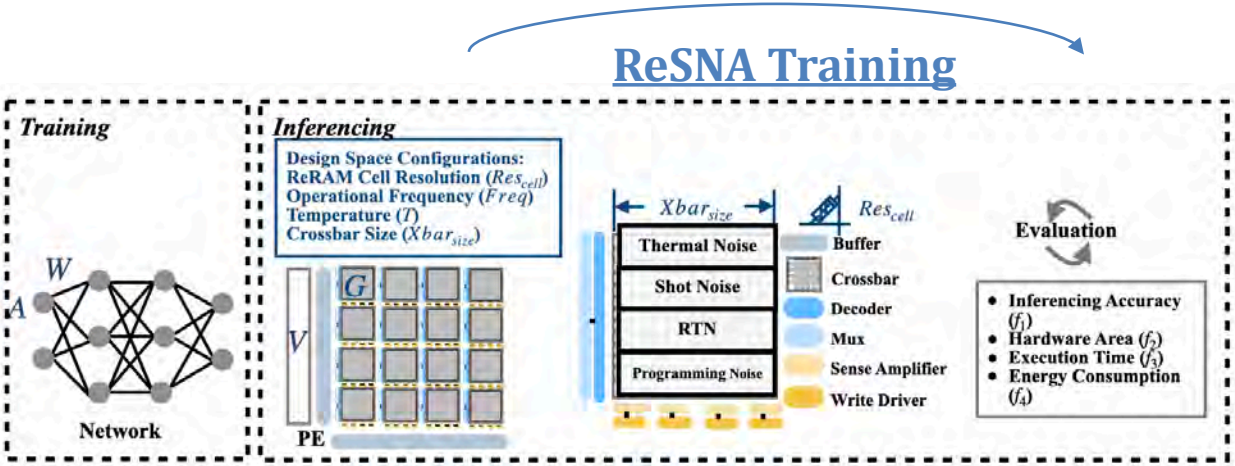


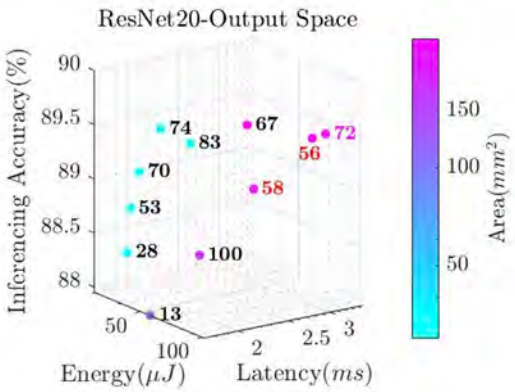


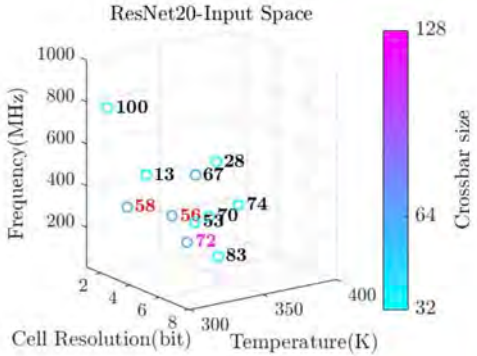




Robust and Efficient ReRAM-based System







Conclude & Our Lab

Inside Fitzpatrick Center, top floor,
Duke campus, Durham, NC, USA

Conclusion

- The progress of hardware development needs to match up with the upscaling of DNN models at software level.
- A holistic scheme integrating the efforts on device, circuit and algorithm levels is important.
- Execution acceleration, energy efficiency, and design flexibility will greatly benefit from device-circuit-algorithm co-designs.

Thank you and Q&A

Asian Pond at Duke Gardens,
Duke campus, Durham, NC, USA