Accelerator Architectures for Machine Learning (AAML)

Lecture 9: Sparse DNN Accelerator

Tsung Tai Yeh
Department of Computer Science
National Yang-Ming Chiao Tung University

Acknowledgements and Disclaimer

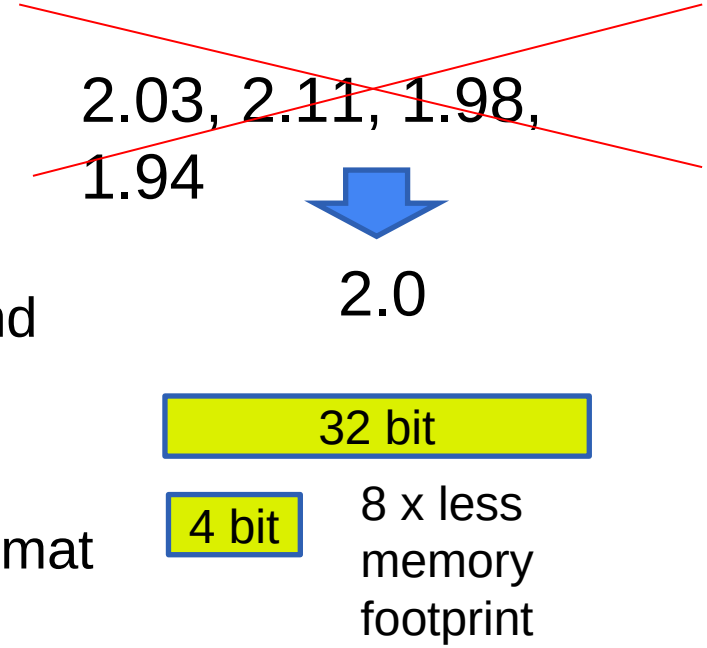
- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020
  CS224W: Machine Learning with Graphs, Stanford University, 2021

Outline

- Efficient Inference Engine (EIE)
- Cnvlutin Sparse Accelerator
- Nvidia Tensor Core: M:N Sparsity
- TorchSparse: Sparse CONV on the GPU

Approaches to Reduce Model Sizes

- **Weight sharing**
  - Trained quantization
- **Quantization**
  - Quantizing the weight and activation
  - Fine-tune in float format
  - Reduce to fixed-point format



Compressed Sparse Row (CSR) Format

- A matrix M (m * n) is represented by three 1-D vectors
- **The A vector**
  - Store values of non-zero elements
  - **Row-by-row** traversing order
- **The IA vector**
  - Store the cumulative number of non-zero elements with size m + 1
  - IA[0] = 0
  - IA[i] = IA[i - 1] + # of non-zero elements in (i-1) th row of the M
- **The JA vector**
  - Store the column index of each element in the A vector

CSR Case Study

- **A vector is [3, 4, 2, 1]**
- JA vector stores column indices of element in A
- **JA = [0, 1, 2, 1]**
- IA[0] = 0, IA[1] = IA[0] + # of non-zero elements in row 0 = 0
- IA[2] = IA[1] + 2 = 2, IA[3] = IA[2] + 1 = 3, IA[4] = IA[3] + 1 = 4
- **IA = [0, 0, 2, 3, 4]**

Index 0	1	2	3
0	**0**	**0**	**0**
3	4	0	0
0	0	2	0
0	1	0	0

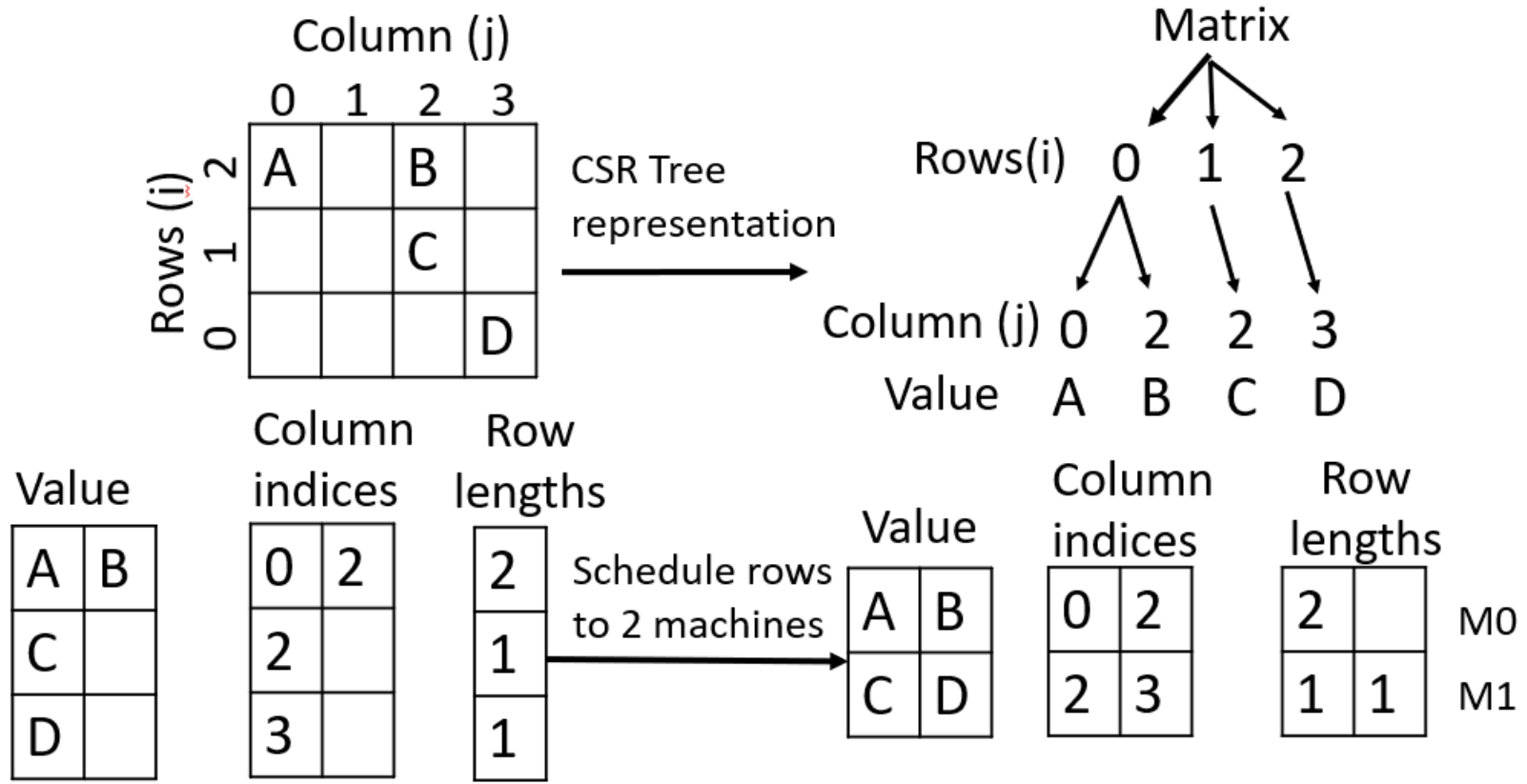
Analysis of CSR Format

- **The sparsity of the matrix**
  - (Total # of elements - # of non-zero elements) / Total # of element
- The direct array based representation required memory
  - **3 * NNZ (Number of Non-Zero)**
- CSR format required memory: **2 * NNZ + m + 1**
- CSR matrices are memory efficient as long as
  - **NNZ < (m * (n - 1) – 1) / 2**

Compressed Sparse Column (CSC) Format

- A matrix M (m * n) is represented by three 1-D vectors
- **The A vector**
  - Store values of non-zero elements
  - **Column-by-column** traversing order
- **The IA vector**
  - Store the cumulative number of non-zero elements with size n + 1
  - IA[0] = 0
  - IA[i] = IA[I - 1] + # of non-zero elements in (i-1) th column of the M
- **The JA vector**
  - Store the row index of each element in the A vector

Sparse Matrix Vector Multiplication (SpMV)



Efficient Inference Engine (EIE)

- The first DNN accelerator for sparse data, compressed model
  - The special-purpose hardware for sparse operations with matrices that are up to 50% dense
  - Exploit both weight sparsity and activation sparsity
  - Saves energy by skipping zero weights
  - Saves cycle by not computing it
  - Aggressive weight quantization (4 bit) to save memory footprint
  - EIE decodes the weight to 16 bit and uses 16 bit arithmetic

EIE: DNN Accelerator for Sparse

Han et. al., ISCA 2016

0 * A = 0

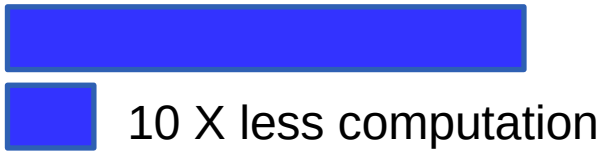
Sparse Weight
90% static sparsity

W * 0 = 0

Sparse Activation
70% dynamic sparsity

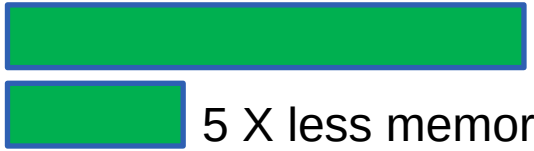
3.01, 2.99 => 3

Weight Sharing
4-bit weights

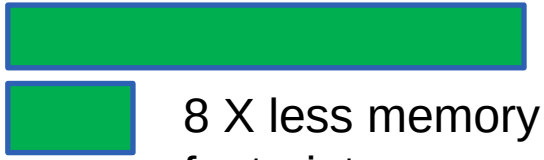


10 X less computation

3 X less computation



5 X less memory footprint



8 X less memory footprint

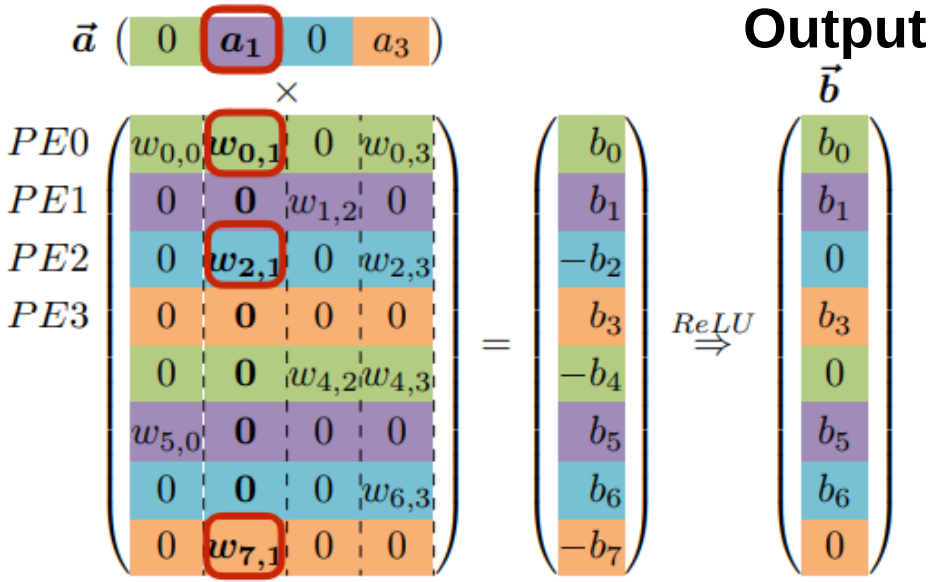
EIE: Reduce Memory Access by Compression

- Compress data based on CSC format
- Rule of thumb: 0 * A = 0, W * 0 = 0

Input

Weights

Output

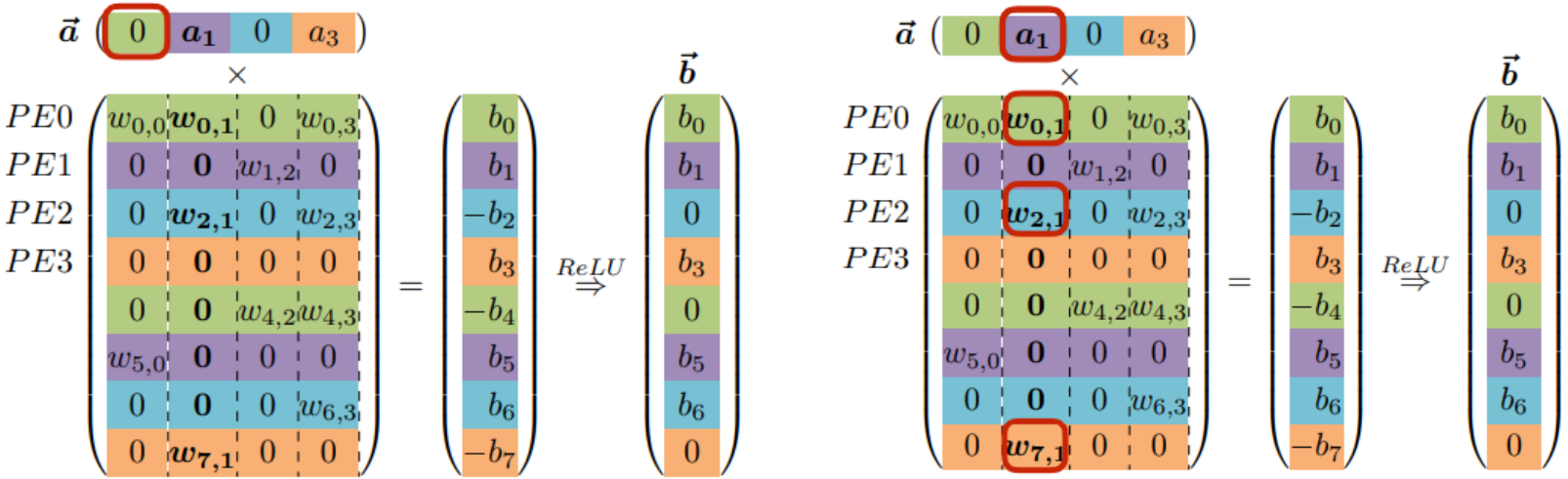


Virtual Weight	$W_{0,0}$	$W_{0,1}$	$W_{4,2}$	$W_{0,3}$	$W_{4,3}$
Relative Index	0	1	2	0	0
Column Pointer	0	1	2	3	

Han et. al., ISCA 2016

EIE Dataflow

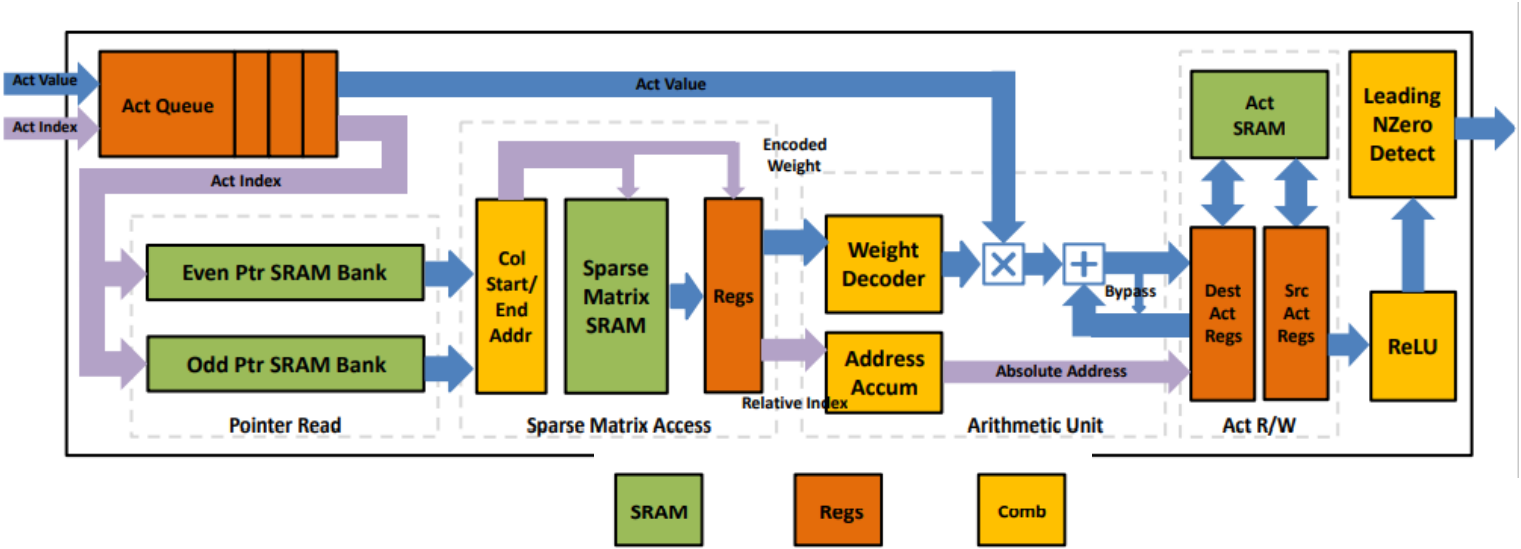
- Skip the execution when activation = 0
- Scan through each activation and only calculate non-zero values



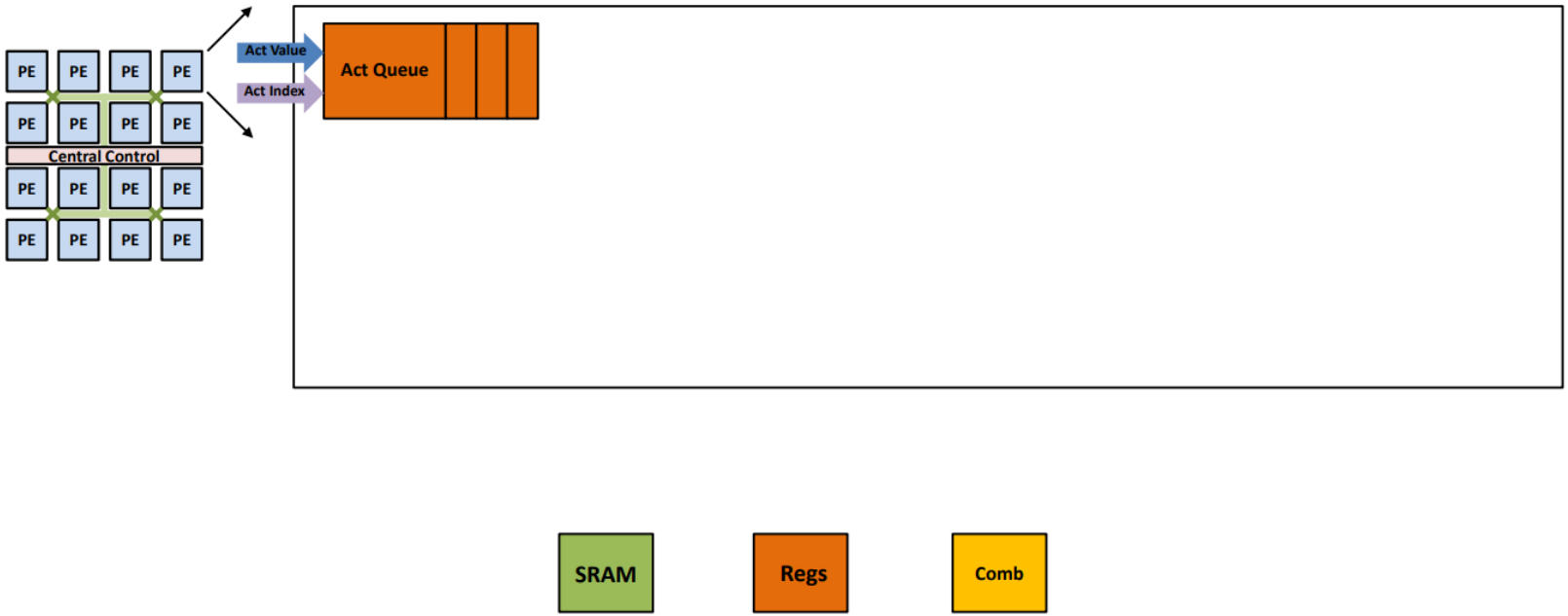
Han et. al., ISCA 2016

EIE: Micro Architecture for each PE

- Process Fully Connected Layers (after deep compression)
- Store weights column-wise in Run Length format (CSC format)
- Read relative column when input is non-zero

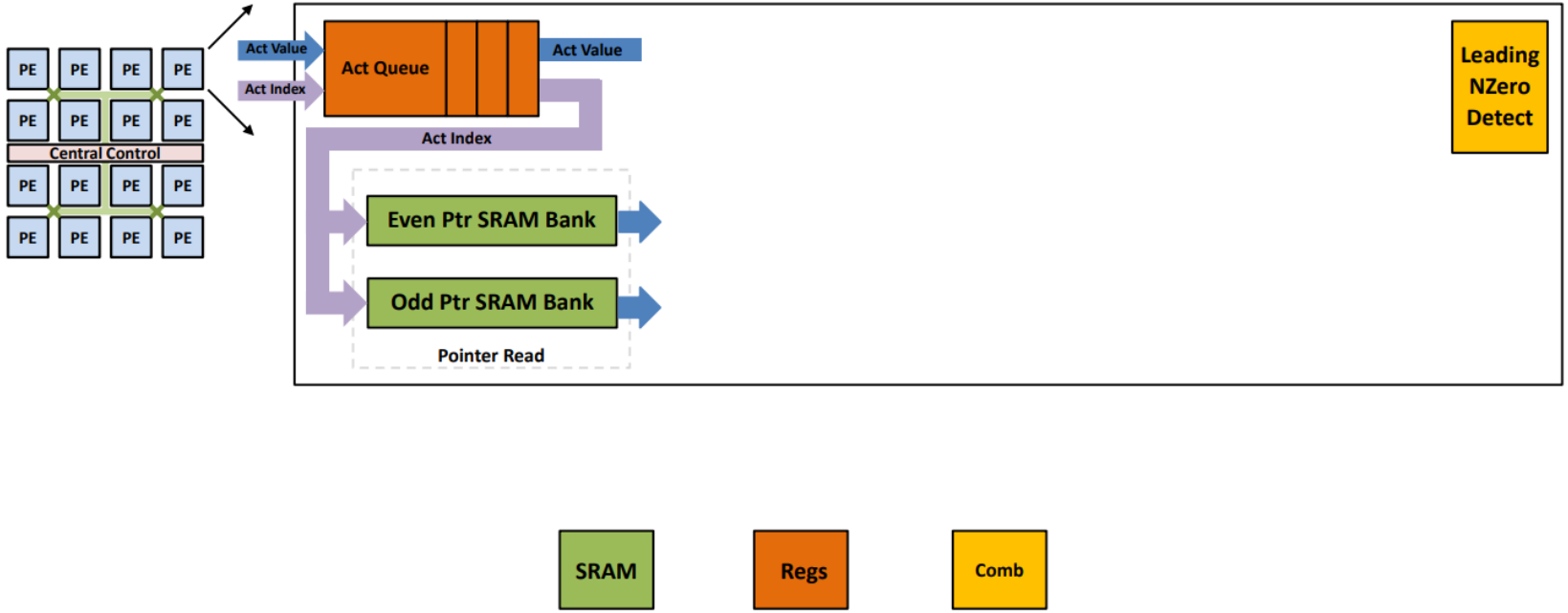


Load Balance

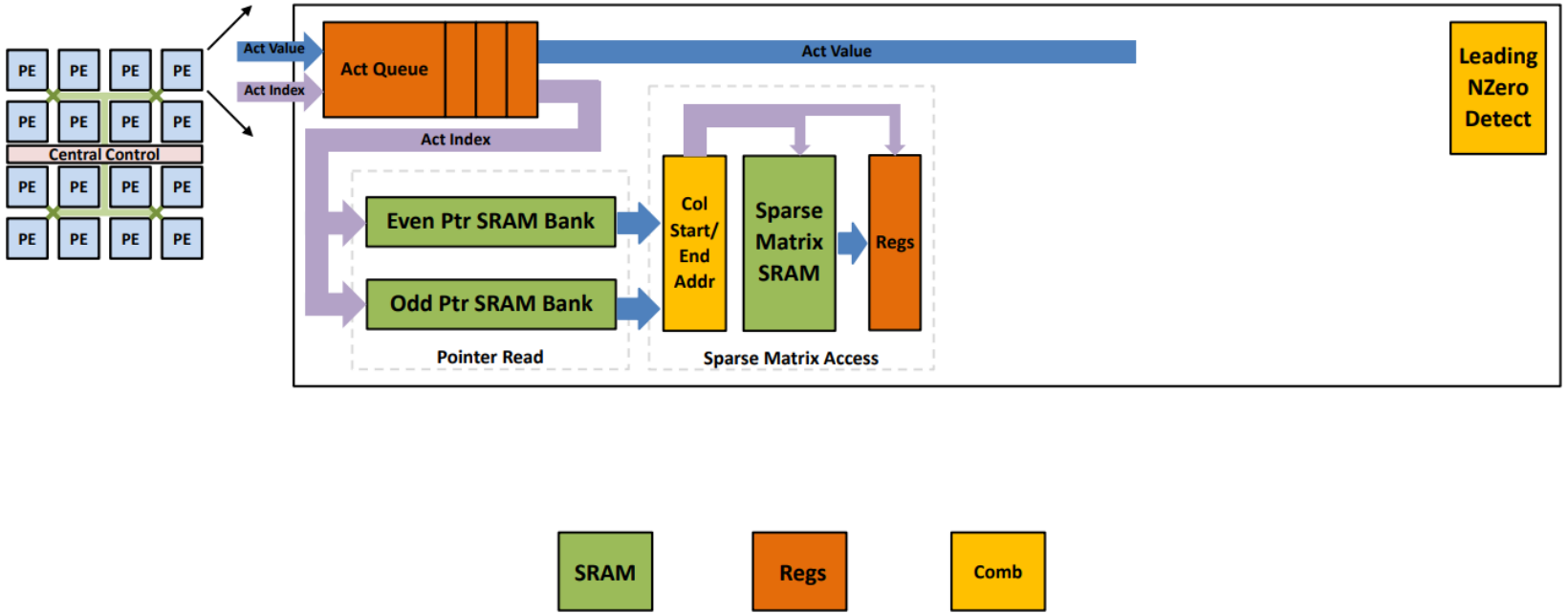


EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.*, ISCA 2016]

Activation Sparsity

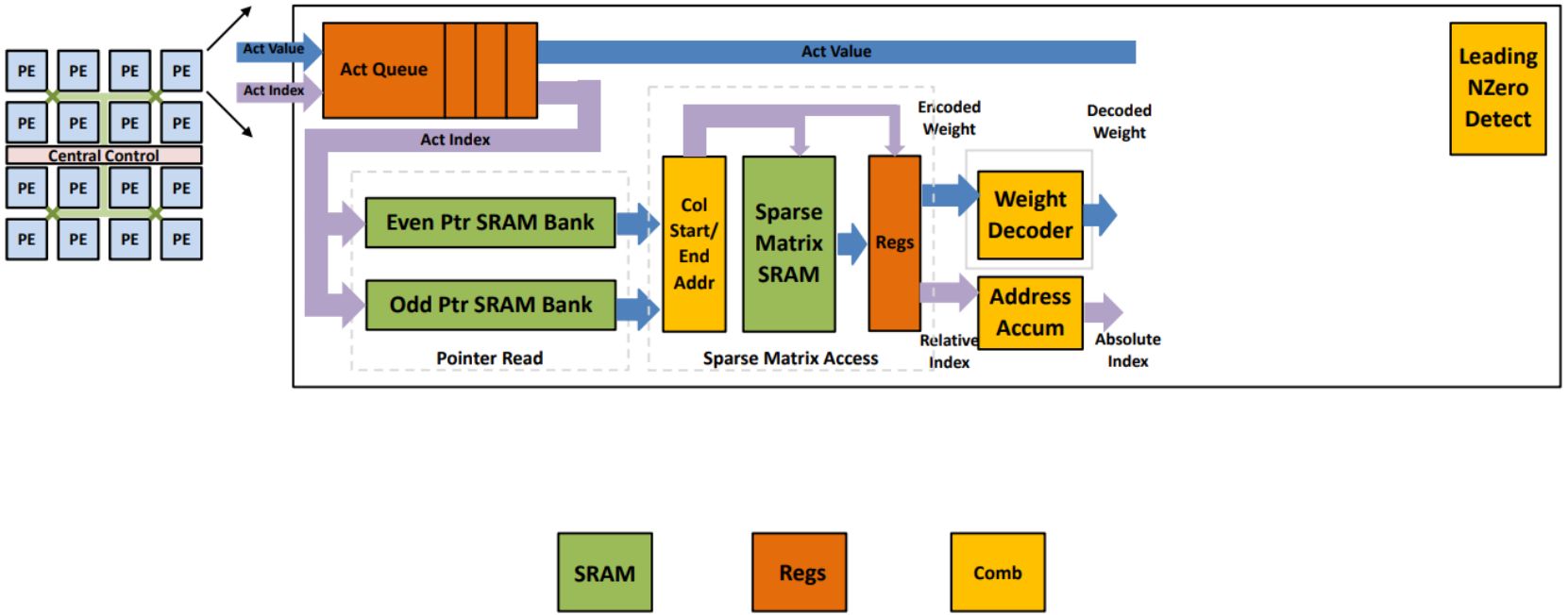


Weight Sparsity



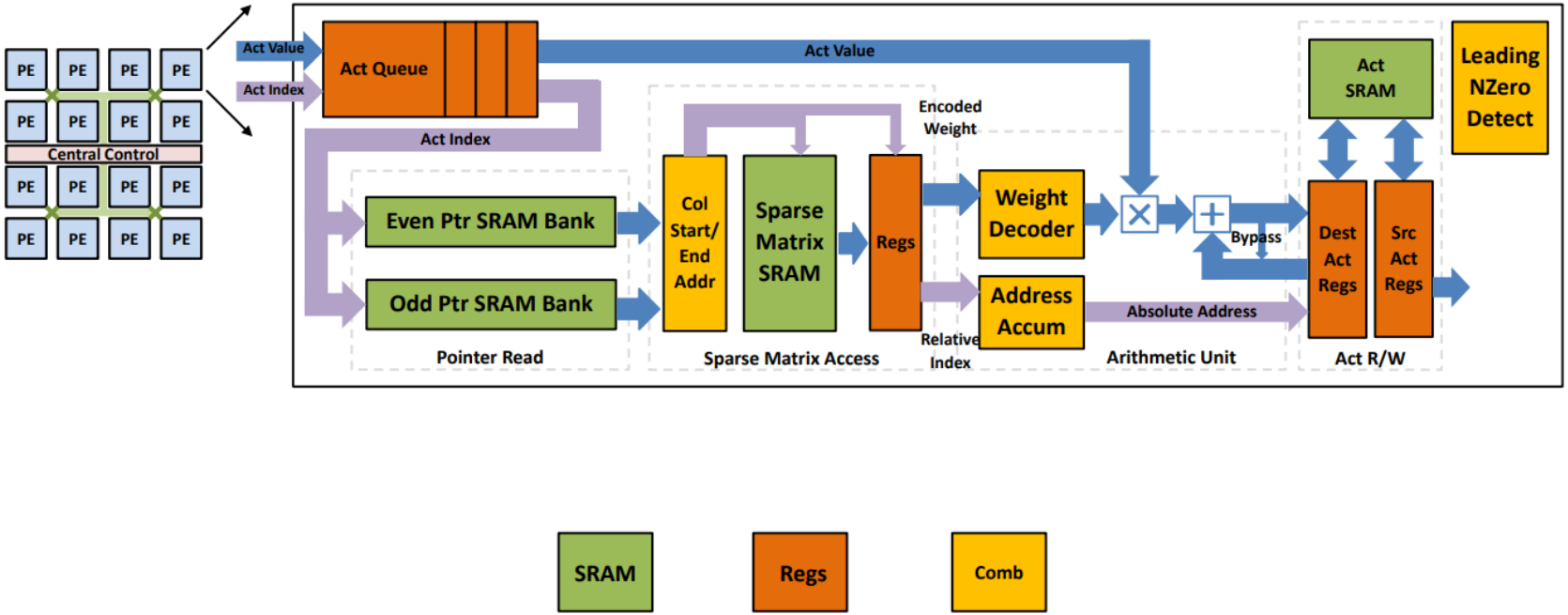
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.*, ISCA 2016]

Weight Sharing

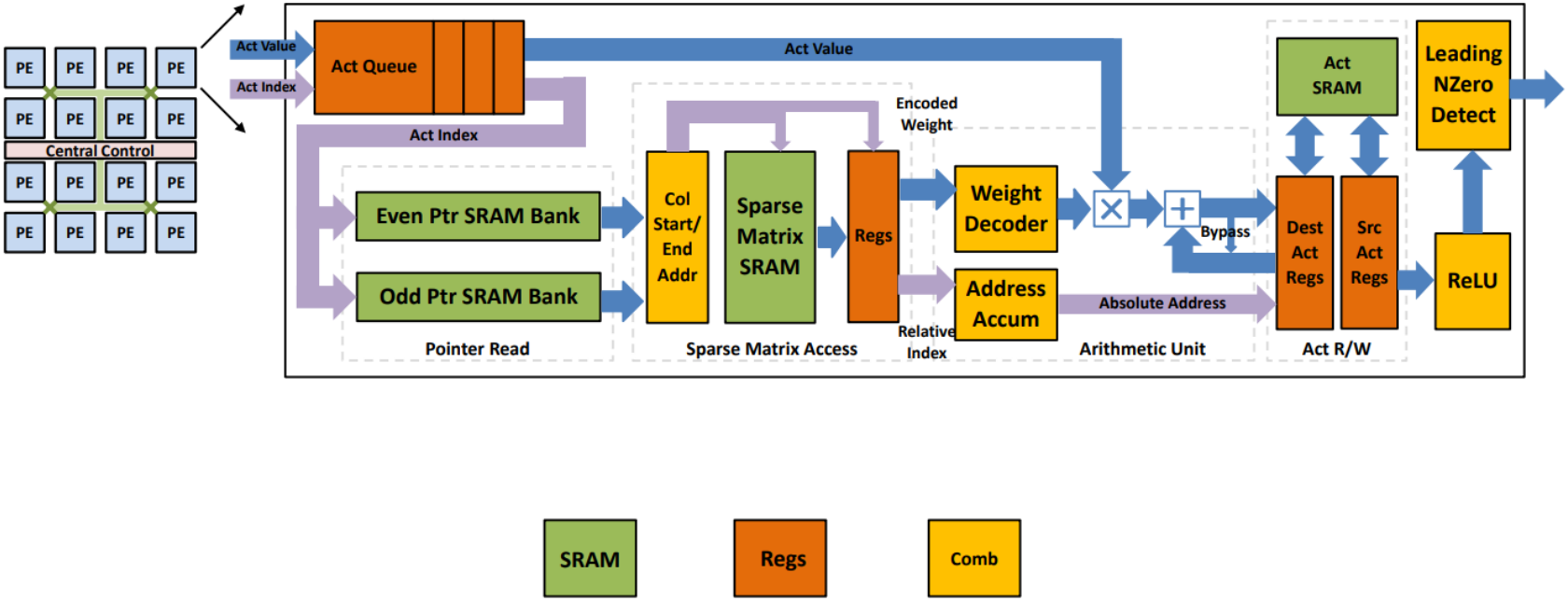


EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.*, ISCA 2016]

Arithmetic & Write Back



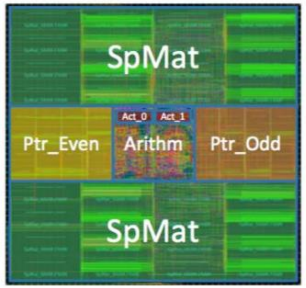
ReLU & Non-zero Detection



Post Layout Result of EIE

- CPU: Intel Core-i7 5930k
- GPU: NVIDIA TitanX
- Mobile GPU: NVIDIA Jetson TK1

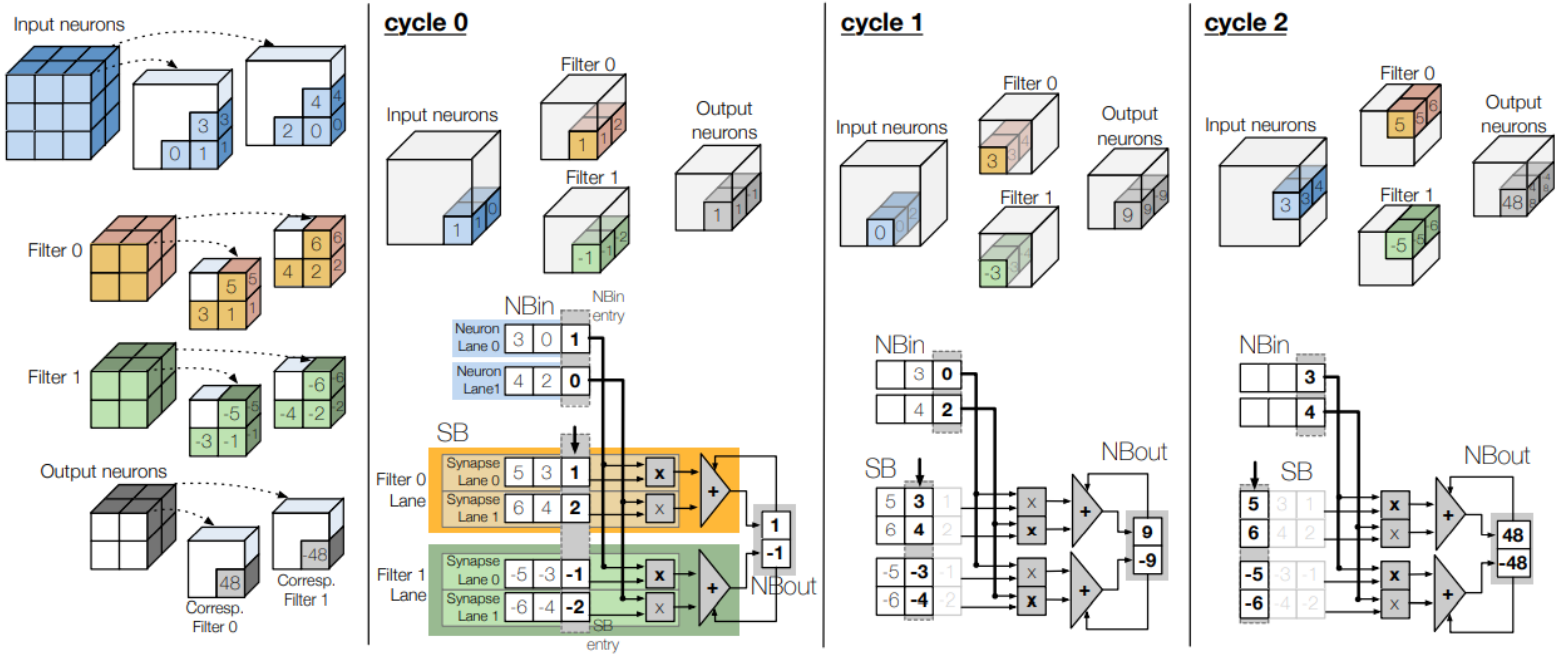
Layer	Size	Weight Density	Activation Density	FLOP Reduction	Description
AlexNet-6	4096 × 9216	9%	35%	33x	AlexNet for image classification
AlexNet-7	4096 × 4096	9%	35%	33x	
AlexNet-8	1000 × 4096	25%	38%	10x	
VGG-6	4096 × 25088	4%	18%	100x	VGG-16 for image classification
VGG-7	4096 × 4096	4%	37%	50x	
VGG-8	1000 × 4096	23%	41%	10x	
NeuralTalk-We	600 × 4096	10%	100%	10x	RNN and LSTM for image caption
NeuralTalk-Wd	8791 × 600	11%	100%	10x	
NeuralTalk-LSTM	2400 × 1201	10%	100%	10x	



Technology	40 nm
# PEs	64
on-chip SRAM	8 MB
Max Model Size	84 Million
Static Sparsity	10x
Dynamic Sparsity	3x
Quantization	4-bit
ALU Width	16-bit
Area	40.8 mm^2
MxV Throughput	81,967 layers/s
Power	586 mW

Cnvlutin

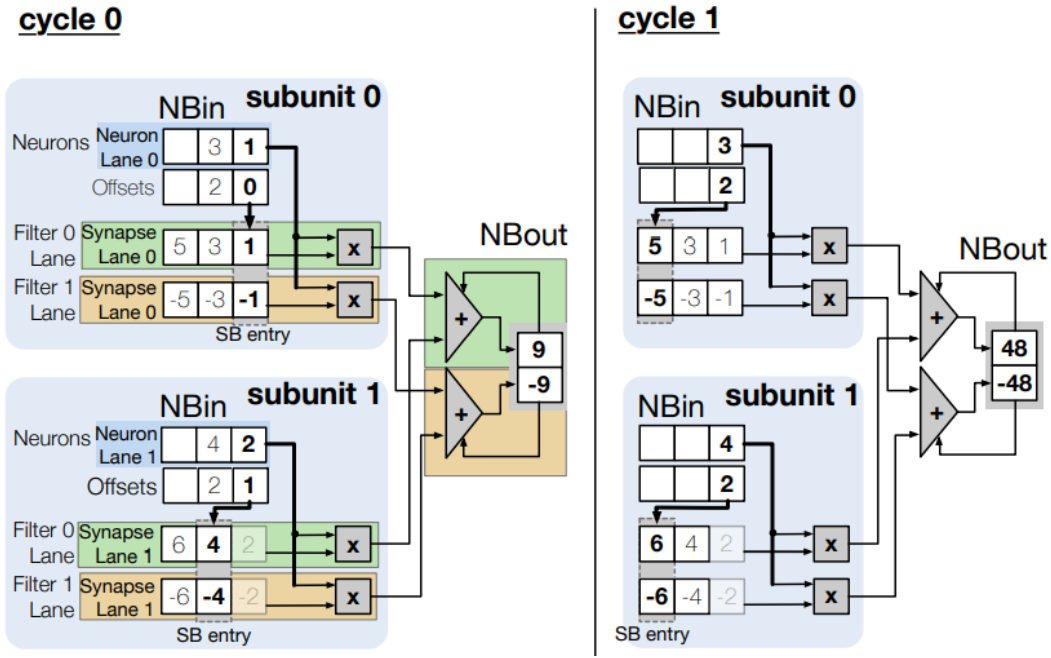
- Baseline does not skip zero and takes three cycles to complete



[Albericio et al., ISCA 2016]

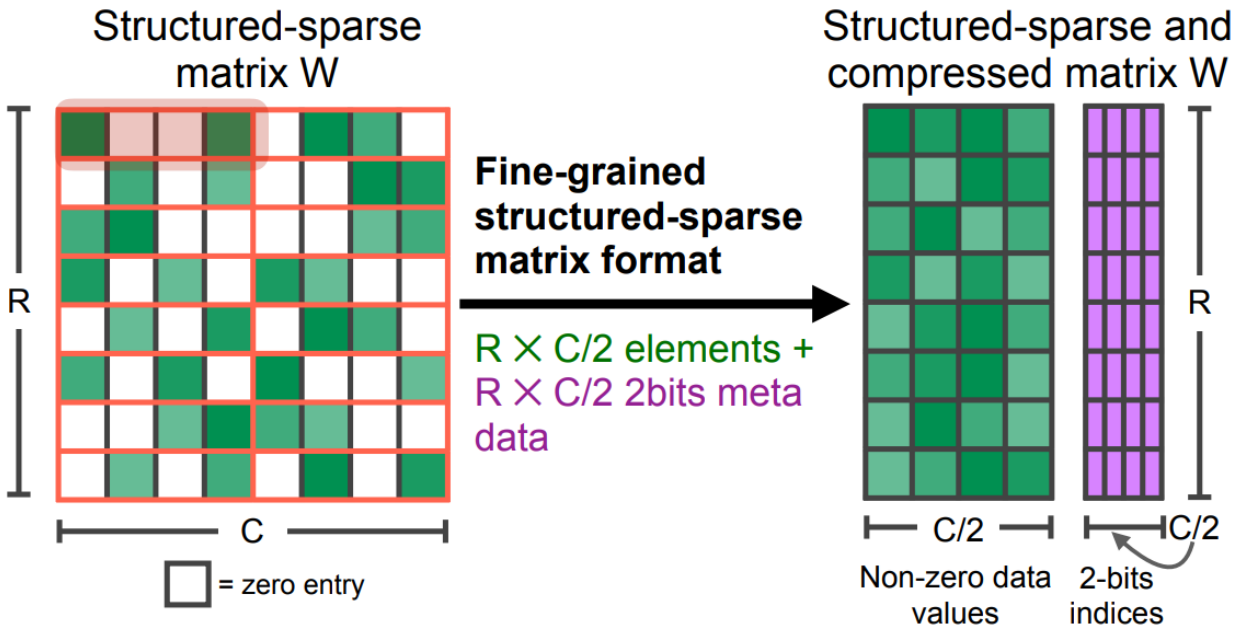
Cnvlutin

- Work on CONV layer
- Cnvlutin skips zero to shorten the execution time
- Add **offset bit** to indicate the proper filter to read



[Albericio et al., ISCA 2016]

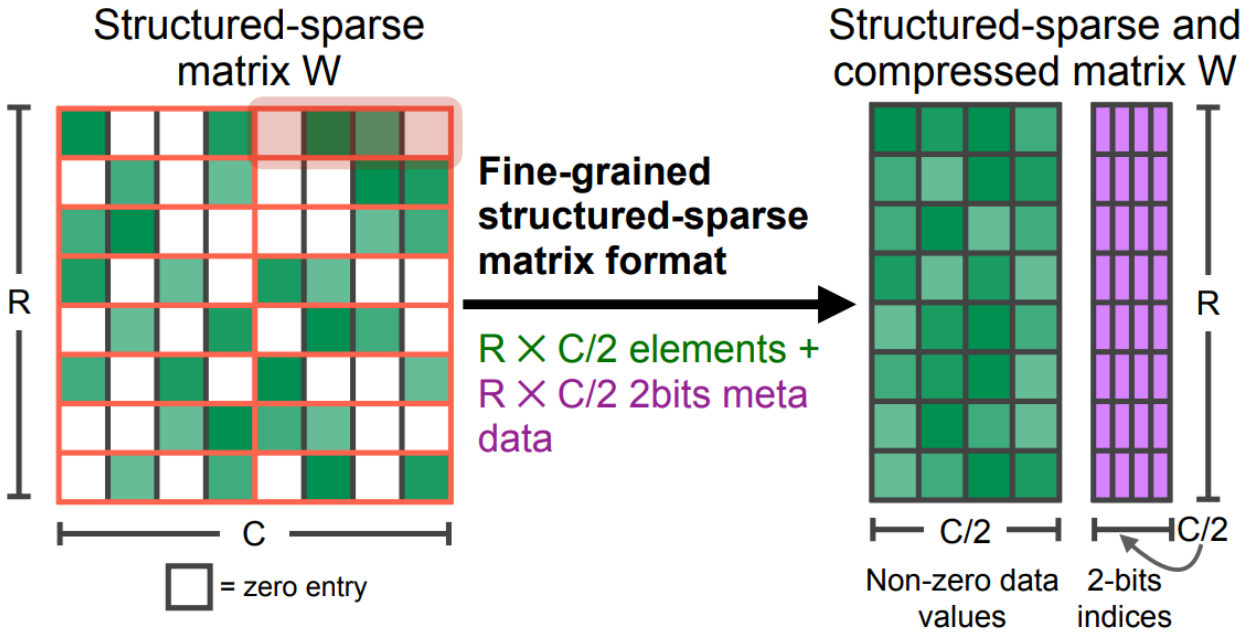
Nvidia Tensor Core: M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

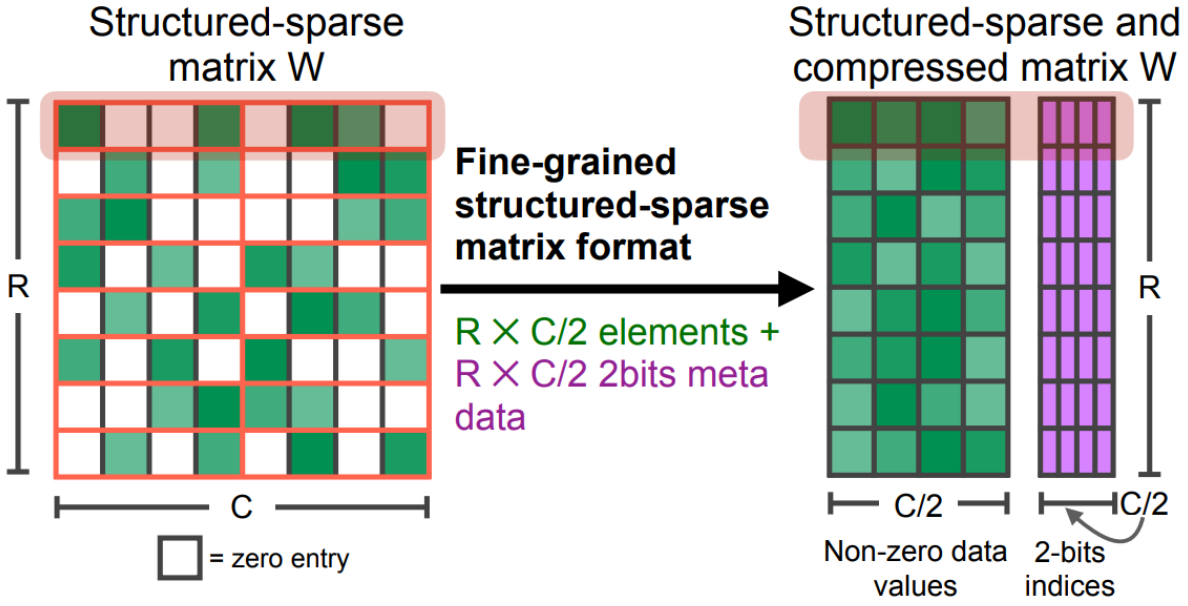
Nvidia Tensor Core: M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

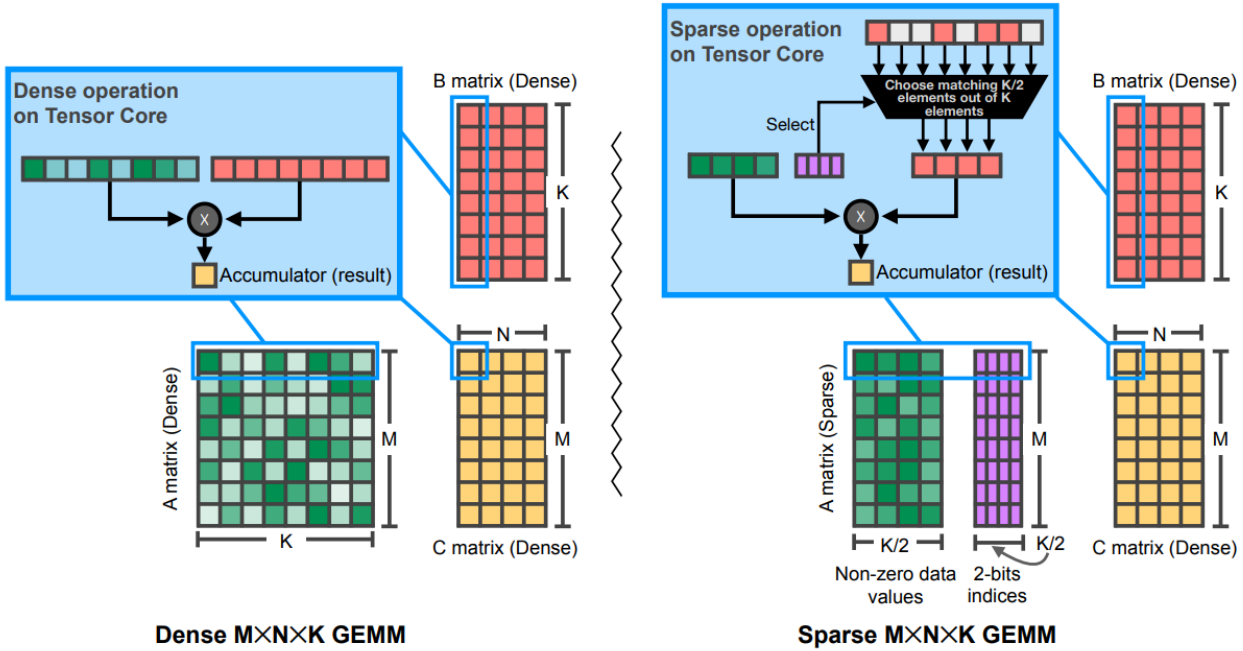
Nvidia Tensor Core: M:N Sparsity



Push all the nonzero elements to the left in memory: save storage and computation.

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

Nvidia Tensor Core: M:N Sparsity



The indices are used to mask out the inputs. Only 2 multiplications will be done out of four.

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

Nvidia Tensor Core: M:N Sparsity

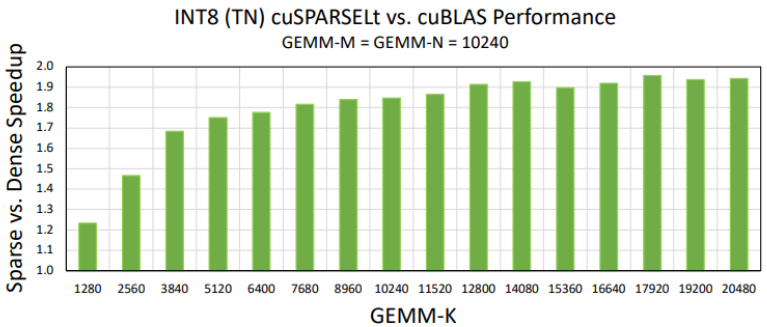


Fig. 3. Comparison of sparse and dense INT8 GEMMs on NVIDIA A100 Tensor Cores. Larger GEMMs achieve nearly a 2× speedup with Sparse Tensor Cores.

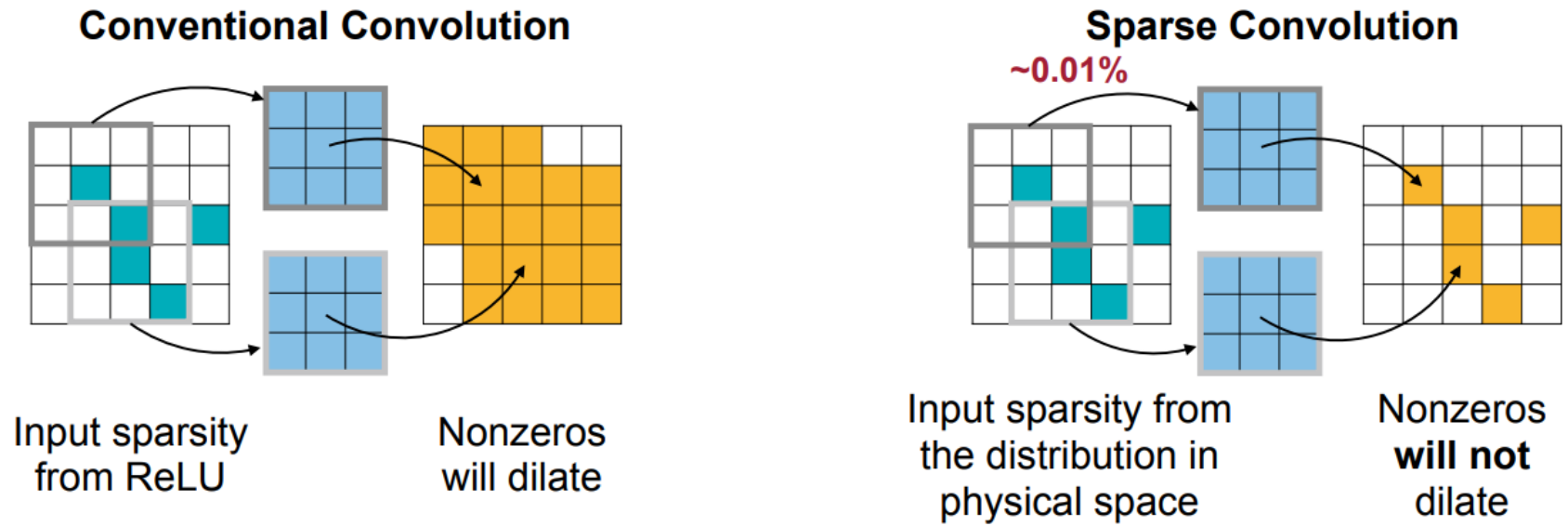
Network	Accuracy		
	Dense FP16	Sparse FP16	Sparse INT8
ResNet-34	73.7	73.9	73.7
ResNet-50	76.1	76.2	76.2
ResNet-50 (SWSL)	81.1	80.9	80.9
ResNet-101	77.7	78.0	77.9
ResNeXt-50-32x4	77.6	77.7	77.7
ResNeXt-101-32x16	79.7	79.9	79.9
ResNeXt-101-32x16 (WSL)	84.2	84.0	84.2
DenseNet-121	75.5	75.3	75.3
DenseNet-161	78.8	78.8	78.9
Wide ResNet-50	78.5	78.6	78.5
Wide ResNet-101	78.9	79.2	79.1
Inception v3	77.1	77.1	77.1
Xception	79.2	79.2	79.2
VGG-11	70.9	70.9	70.8
VGG-16	74.0	74.1	74.1
VGG-19	75.0	75.0	75.0
SUNet-128	75.6	76.0	75.4
SUNet-7-128	76.4	76.5	76.3
DRN26	75.2	75.3	75.3
DRN-105	79.4	79.5	79.4

Pruning CNNs with 2:4 sparsity will bring about large speedup for GEMM workloads and it will not incur performance drop for DNN models.

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

TorchSparse: Sparse CONV on the GPU

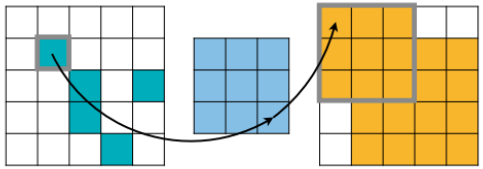
- Sparse convolution on sparse inputs



TorchSparse: Sparse CONV on the GPU

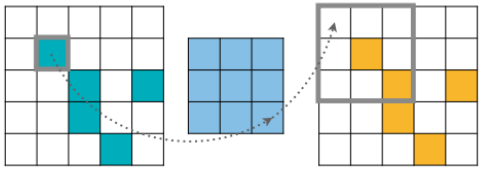
- Sparse convolution on sparse inputs

Conventional Convolution



(P_0, Q_0, $W_{1,1}$)

Sparse Convolution



No compute

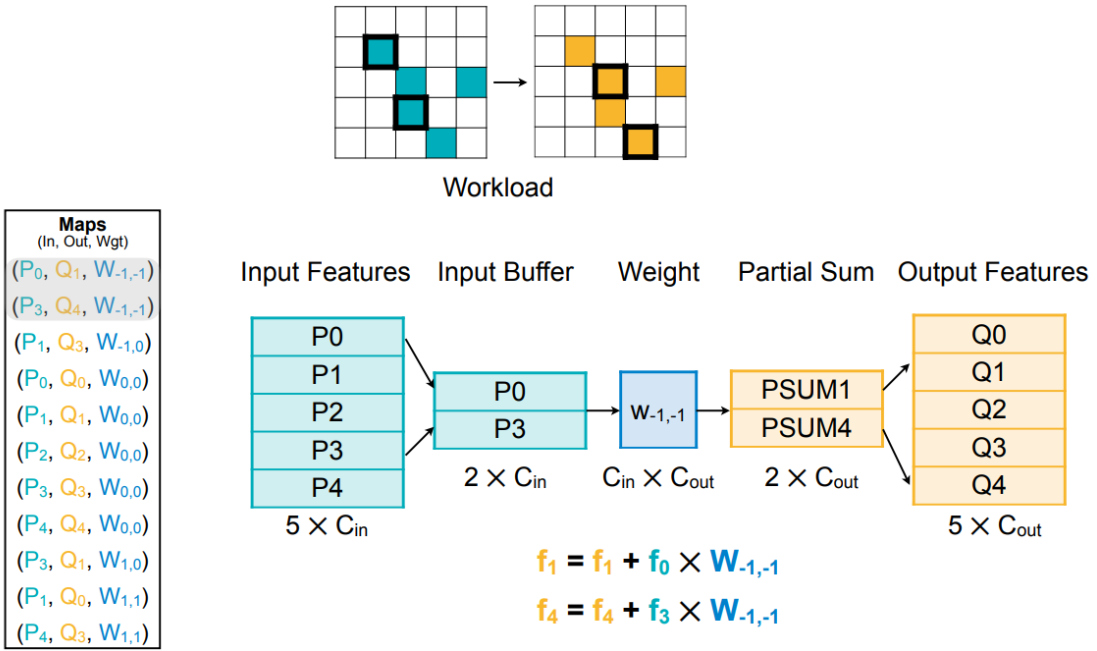
Maps
(In, Out, Wgt)

Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

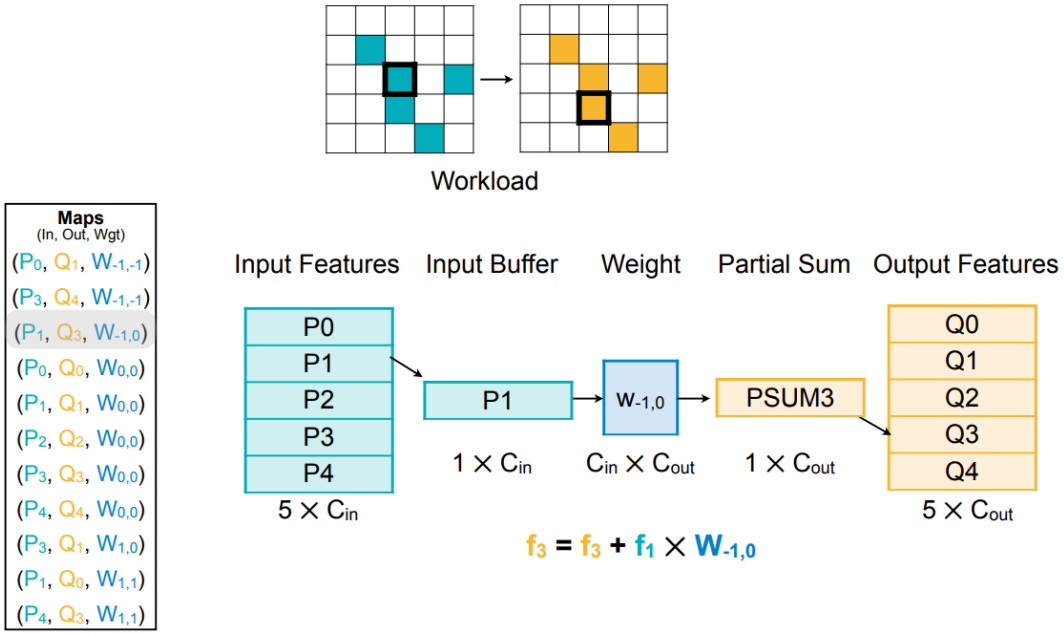
- Weight-stationary computation, separate matmul for different weights



TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

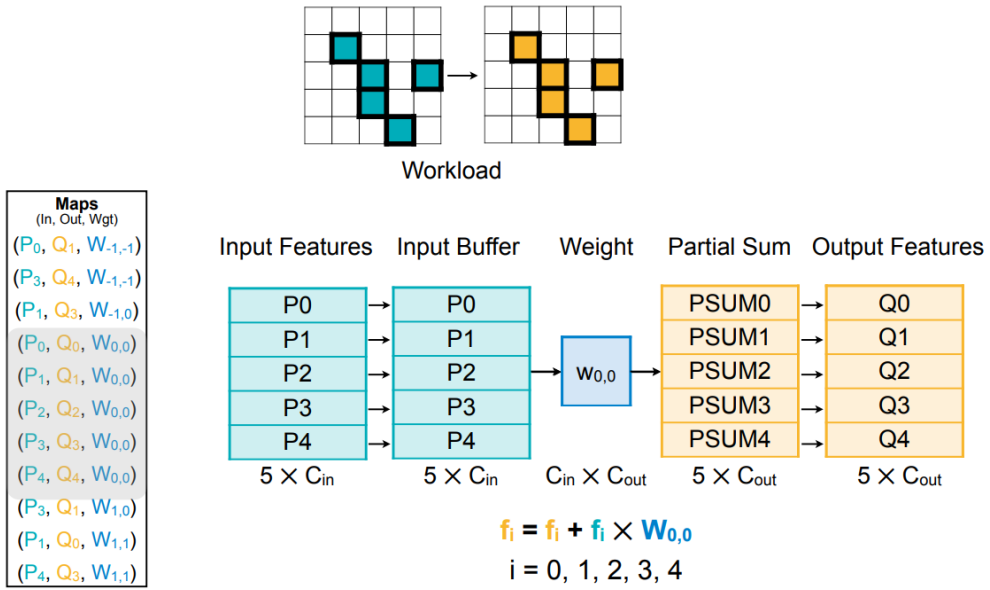
- Weight-stationary computation, separate matmul for different weights



TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

- Weight-stationary computation, separate matmul for different weights

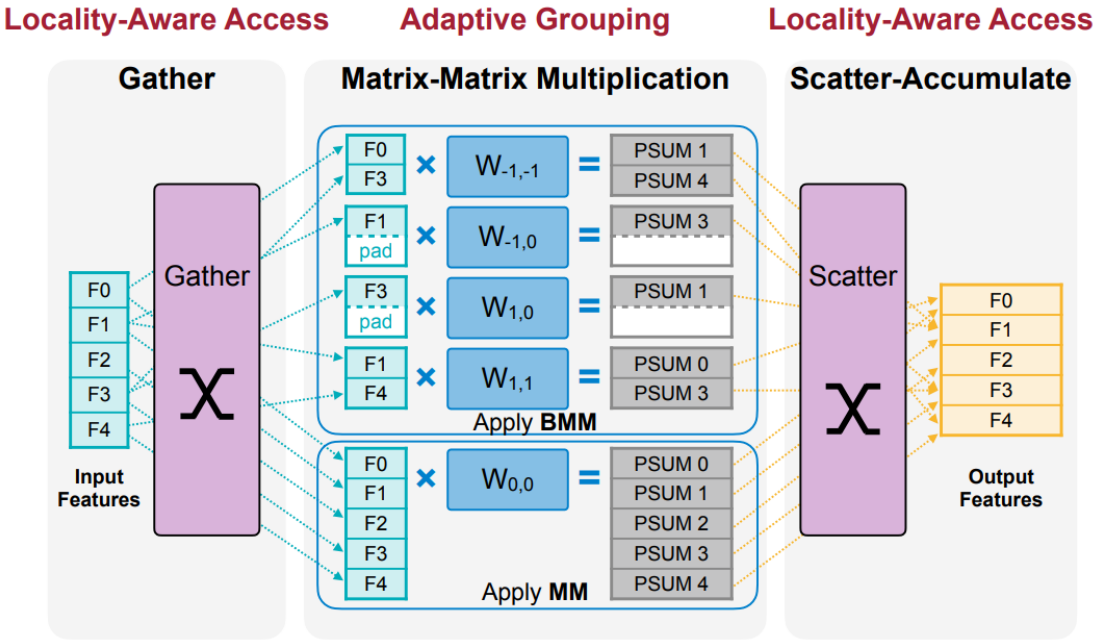


Note: maps for $W_{0,0}$ contains all entries.

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

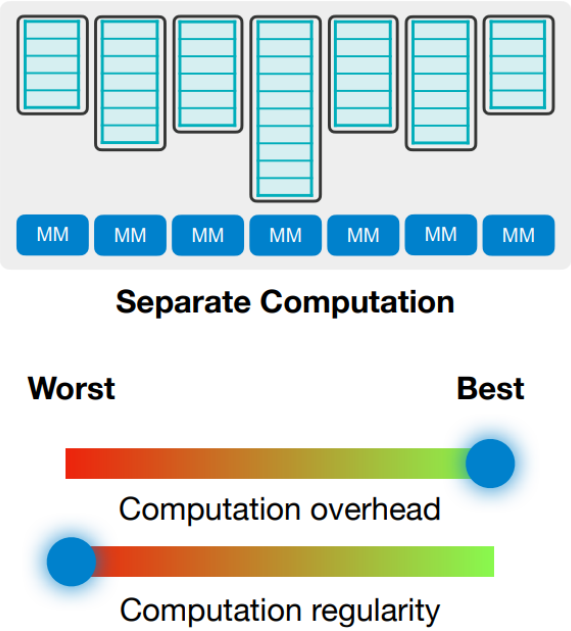
- Weight-stationary computation, separate matmul for different weights



TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

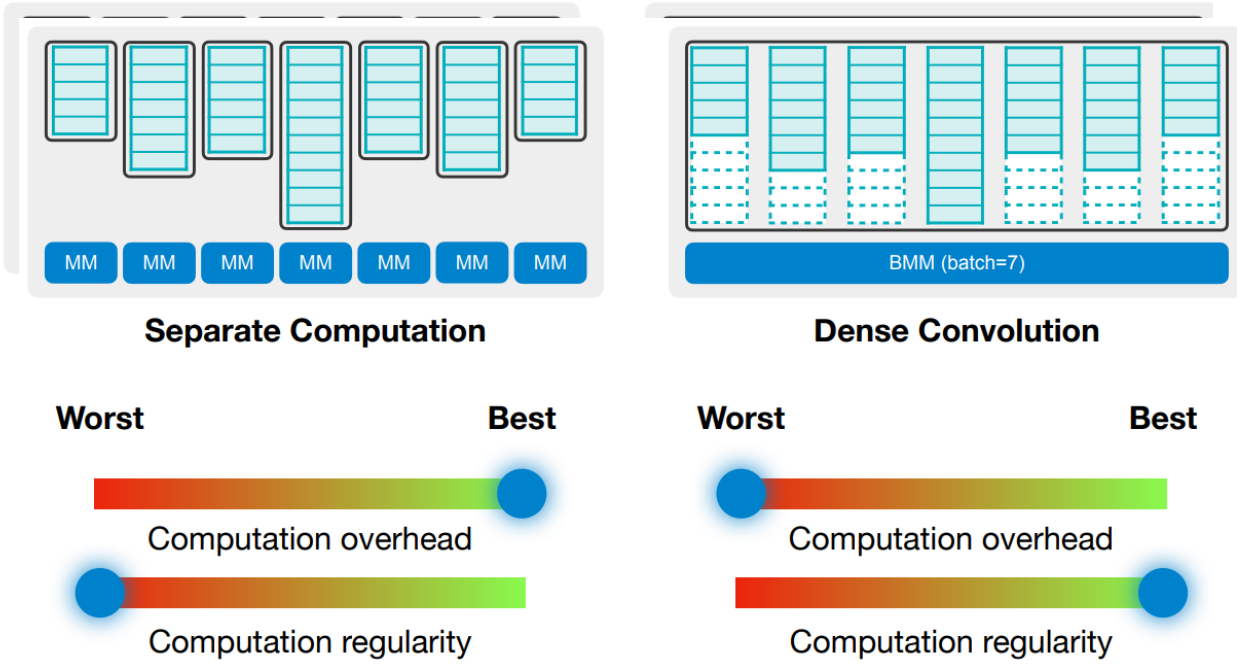
- Separate computation: many kernel calls, low device utilization



TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

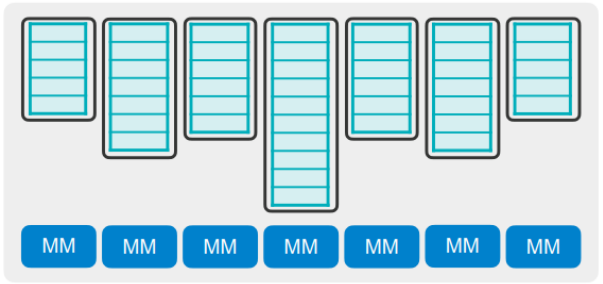
- Dense convolution: best regularity but load imbalance



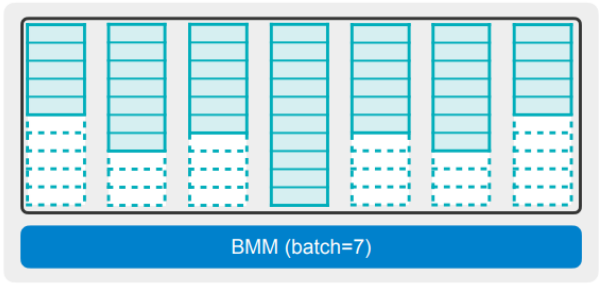
TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

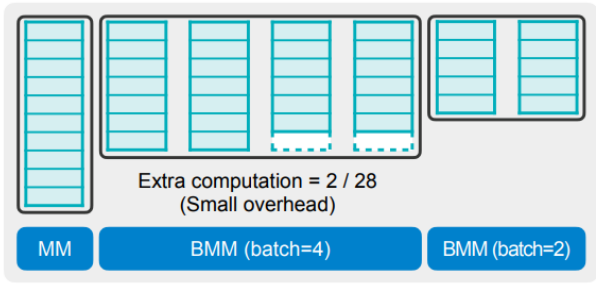
- Computation with grouping: balancing overhead and regularity



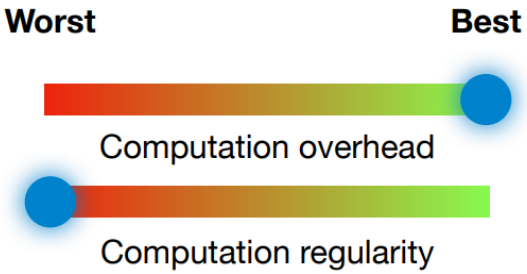
Separate Computation

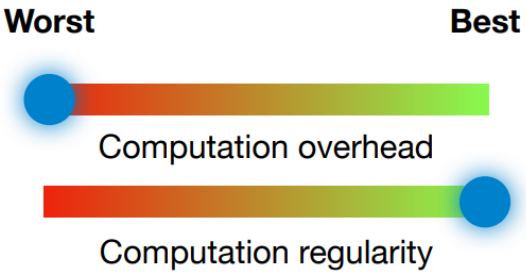


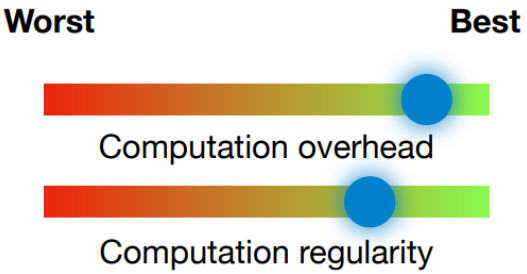
Dense Convolution



Computation with grouping



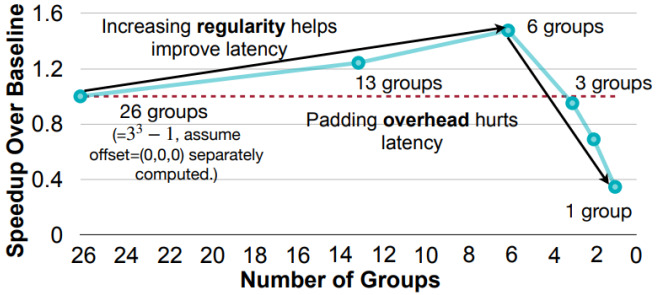


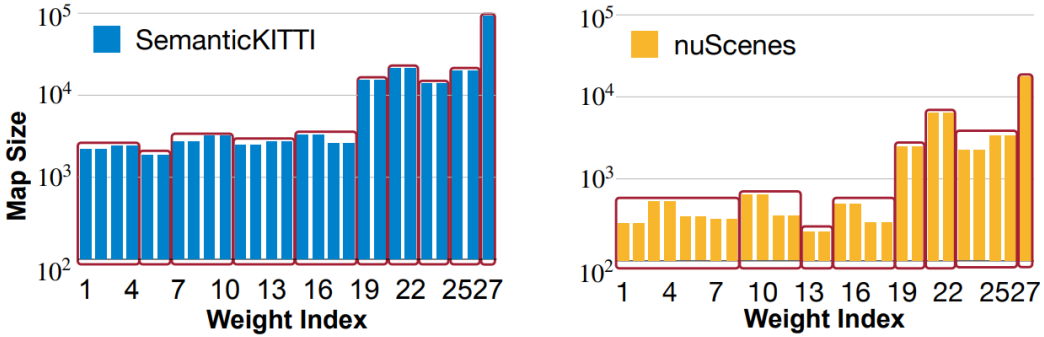


TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse: Sparse CONV on the GPU

- Searching customized strategy for different model and datasets





TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Takeaway Questions

- What are values in “A”, “JA”, “IA” vector in CSR format?
  - (A) [5, 6, 7, 4, 3, 2, 1, 8], [0, 1, 1, 1, 2, 3, 4, 5], [0, 1, 3, 4, 6, 7, 8]
  - (B) [5, 6, 7, 4, 3, 2, 1, 8], [0, 1, 1, 3, 2, 3, 4, 5], [0, 1, 3, 4, 6, 7, 8]
  - (C) [5, 6, 7, 4, 3, 2, 1, 8], [0, 1, 1, 2, 2, 3, 4, 5], [0, 1, 3, 3, 6, 7, 8]

5	6	0	0	0	0
0	7	0	4	0	0
0	0	3	2	1	0
0	0	0	0	0	8

Takeaway Questions

- What are critical issues when designing a sparse DNN accelerator?
  - (A) Compressed data overhead
  - (B) Sparse data mapping
  - (C) Hard to prefetch data