Advanced topics in computer architectures – 236509

System Aspects of machine learning

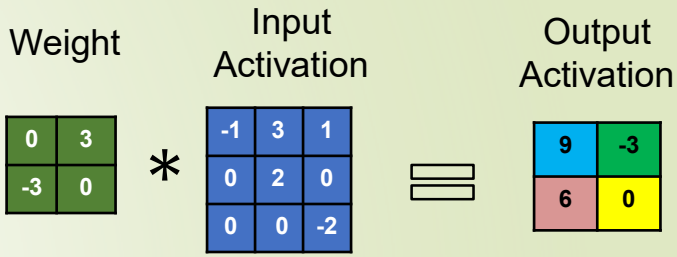
Sparse-CNN Accelerators

Student: Muhammad Dahamshi
Prof. Avi Mendelson
TA: Mr. Nicolay Valdman

Intro, motivation & background

- Convolutional Neural Networks (CNNs) have emerged as a fundamental technology for machine learning. Thus, high performance, high Throughput and extreme energy efficiency are critical for deployments of CNNs in a wide range of situations.
- The co-existence of activation sparsity and model sparsity in convolutional neural network (CNN) models raised from zero-valued weights/activations, padding, ReLU operator, etc. makes sparsity-aware CNN hardware designs very attractive.
- Meanwhile, CNN hardware accelerators, especially for inference-only, are also actively investigated by many startup companies because of the huge market of low-power embedded vision.

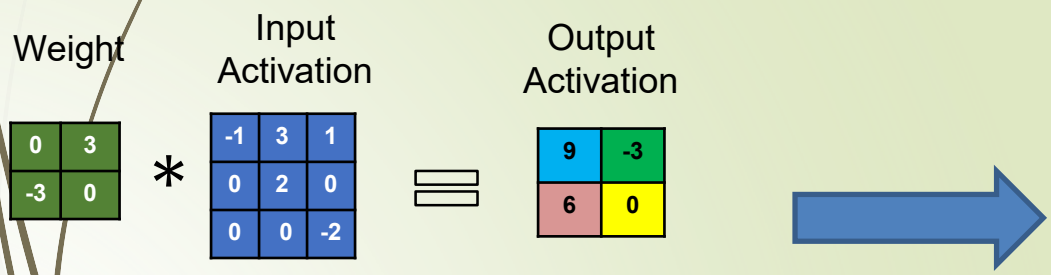
2-D Convolution

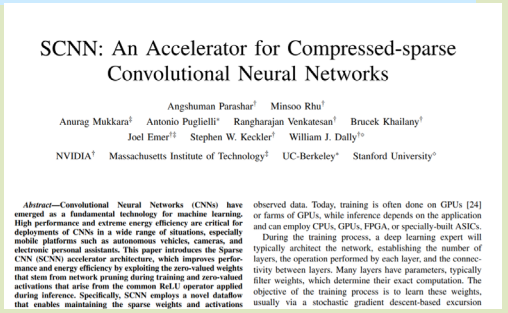


First novel approach: SCNN

https://arxiv.org/abs/1708.04485

- SCNN : An Accelerator for Compressed-sparse CNN: is the first novel dataflow that **considers both activation and weight sparsity**, thereby achieving high hardware performance.
- To avoid the unnecessary multiplications due to the zero operand (zero-value multiplications), SCNN directly performs the Cartesian Product among the non-zero activations and weights.





SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks

Angshuman Parashar† Minsoo Rhu†
Anurag Mukkara‡ Antonio Puglielli* Rangharajan Venkatesan† Brucek Khailany†
Joel Emer†‡ Stephen W. Keckler† William J. Dally†◇
NVIDIA† Massachusetts Institute of Technology‡ UC-Berkeley* Stanford University◇

***Abstract*—Convolutional Neural Networks (CNNs) have emerged as a fundamental technology for machine learning. High performance and extreme energy efficiency are critical for deployments of CNNs in a wide range of situations, especially mobile platforms such as autonomous vehicles, cameras, and electronic personal assistants. This paper introduces the Sparse CNN (SCNN) accelerator architecture, which improves performance and energy efficiency by exploiting the zero-valued weights that stem from network pruning during training and zero-valued activations that arise from the common ReLU operator applied during inference. Specifically, SCNN employs a novel dataflow that enables maintaining the sparse weights and activations**

observed data. Today, training is often done on GPUs [24] or farms of GPUs, while inference depends on the application and can employ CPUs, GPUs, FPGA, or specially-built ASICs.

During the training process, a deep learning expert will typically architect the network, establishing the number of layers, the operation performed by each layer, and the connectivity between layers. Many layers have parameters, typically filter weights, which determine their exact computation. The objective of the training process is to learn these weights, usually via a stochastic gradient descent-based excursion

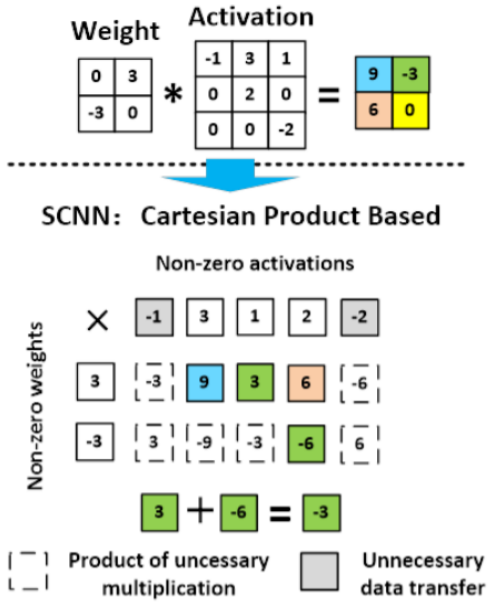


Fig. 1: Example of SCNN dataflow.

Why SCNN is not optimal ?

- The Cartesian product-based introduces many **unnecessary multiplications** by design, which correspondingly incurs **unnecessary data transfer**. Thus, SCNN is not optimal since it incurs architecturally-wasted multiplications and unnecessary data transfer.
- With the same example: it is easy to see that the weight 3 and -3 should never be multiplied with activation -1 and -2, since the weight can never cover these activations no matter which position the kernel is at.

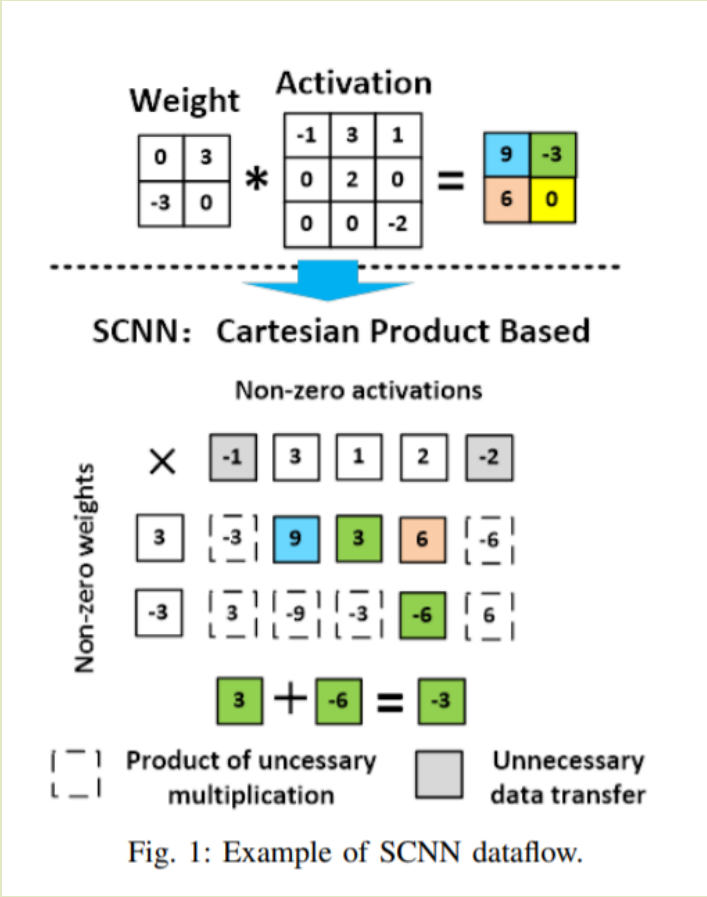
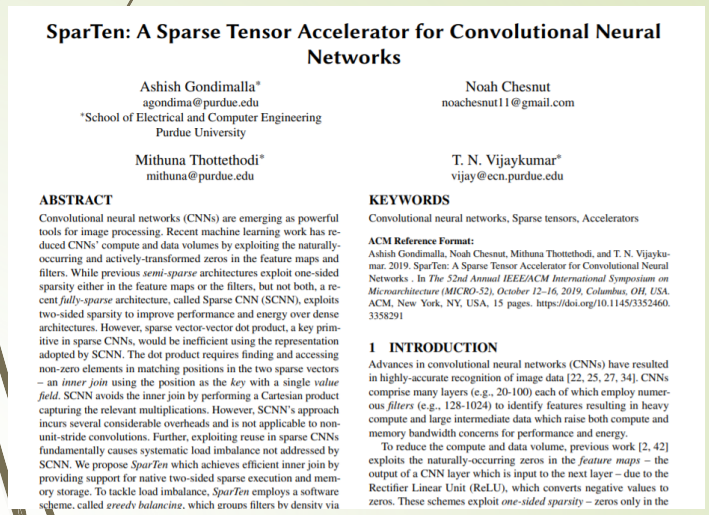


Fig. 1: Example of SCNN dataflow.

SparTen & ExTensor: Intersection Based Method

- To achieve the optimal scenario with only necessary multiplications in the original algorithm, SparTen and ExTensor attempt to directly search and identify the positions of the matched entries between two sparse vectors.
- Both solutions identify the matching non-zero value pairs between kernel weights and the corresponding activations with a **conceptual intersection operation**.
- The two schemes differ in how the intersection operation is implemented.



SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks

Ashish Gondimalla*
agondima@purdue.edu
*School of Electrical and Computer Engineering
Purdue University

Noah Chesnut
noachesnut11@gmail.com

Mithuna Thottethodi*
mithuna@purdue.edu

T. N. Vijaykumar*
vijay@ecn.purdue.edu

ABSTRACT

Convolutional neural networks (CNNs) are emerging as powerful tools for image processing. Recent machine learning work has reduced CNNs' compute and data volumes by exploiting the naturally-occurring and actively-transformed zeros in the feature maps and filters. While previous *semi-sparse* architectures exploit one-sided sparsity either in the feature maps or the filters, but not both, a recent *fully-sparse* architecture, called Sparse CNN (SCNN), exploits two-sided sparsity to improve performance and energy over dense architectures. However, sparse vector-vector dot product, a key primitive in sparse CNNs, would be inefficient using the representation adopted by SCNN. The dot product requires finding and accessing non-zero elements in matching positions in the two sparse vectors – an *inner join* using the position as the *key* with a single *value field*. SCNN avoids the inner join by performing a Cartesian product capturing the relevant multiplications. However, SCNN's approach incurs several considerable overheads and is not applicable to non-unit-stride convolutions. Further, exploiting reuse in sparse CNNs fundamentally causes systematic load imbalance not addressed by SCNN. We propose *SparTen* which achieves efficient inner join by providing support for native two-sided sparse execution and memory storage. To tackle load imbalance, *SparTen* employs a software scheme, called *greedy balancing*, which groups filters by density via

KEYWORDS

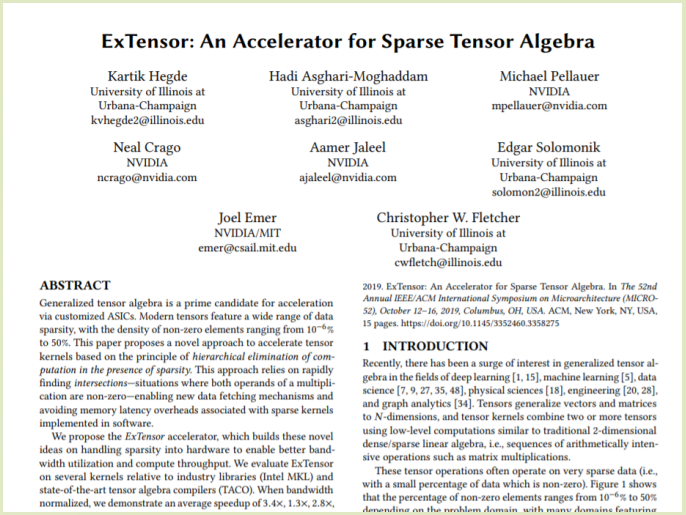
Convolutional neural networks, Sparse tensors, Accelerators

ACM Reference Format:
Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and T. N. Vijaykumar. 2019. SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks . In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3352460.3358291

1 INTRODUCTION

Advances in convolutional neural networks (CNNs) have resulted in highly-accurate recognition of image data [22, 25, 27, 34]. CNNs comprise many layers (e.g., 20-100) each of which employ numerous *filters* (e.g., 128-1024) to identify features resulting in heavy compute and large intermediate data which raise both compute and memory bandwidth concerns for performance and energy.

To reduce the compute and data volume, previous work [2, 42] exploits the naturally-occurring zeros in the *feature maps* – the output of a CNN layer which is input to the next layer – due to the Rectifier Linear Unit (ReLU), which converts negative values to zeros. These schemes exploit *one-sided sparsity* – zeros only in the



ExTensor: An Accelerator for Sparse Tensor Algebra

Kartik Hegde
University of Illinois at Urbana-Champaign
kvhegde2@illinois.edu

Hadi Asghari-Moghaddam
University of Illinois at Urbana-Champaign
asghari2@illinois.edu

Michael Pellauer
NVIDIA
mpellauer@nvidia.com

Neal Crago
NVIDIA
ncrago@nvidia.com

Aamer Jaleel
NVIDIA
ajaleel@nvidia.com

Edgar Solomonik
University of Illinois at Urbana-Champaign
solomon2@illinois.edu

Joel Emer
NVIDIA/MIT
emer@csail.mit.edu

Christopher W. Fletcher
University of Illinois at Urbana-Champaign
cwfletch@illinois.edu

ABSTRACT

Generalized tensor algebra is a prime candidate for acceleration via customized ASICs. Modern tensors feature a wide range of data sparsity, with the density of non-zero elements ranging from 10^{-6}% to 50%. This paper proposes a novel approach to accelerate tensor kernels based on the principle of *hierarchical elimination of computation in the presence of sparsity*. This approach relies on rapidly finding *intersections*—situations where both operands of a multiplication are non-zero—enabling new data fetching mechanisms and avoiding memory latency overheads associated with sparse kernels implemented in software.

We propose the *ExTensor* accelerator, which builds these novel ideas on handling sparsity into hardware to enable better bandwidth utilization and compute throughput. We evaluate ExTensor on several kernels relative to industry libraries (Intel MKL) and state-of-the-art tensor algebra compilers (TACO). When bandwidth normalized, we demonstrate an average speedup of 3.4×, 1.3×, 2.8×,

2019. ExTensor: An Accelerator for Sparse Tensor Algebra. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3352460.3358275

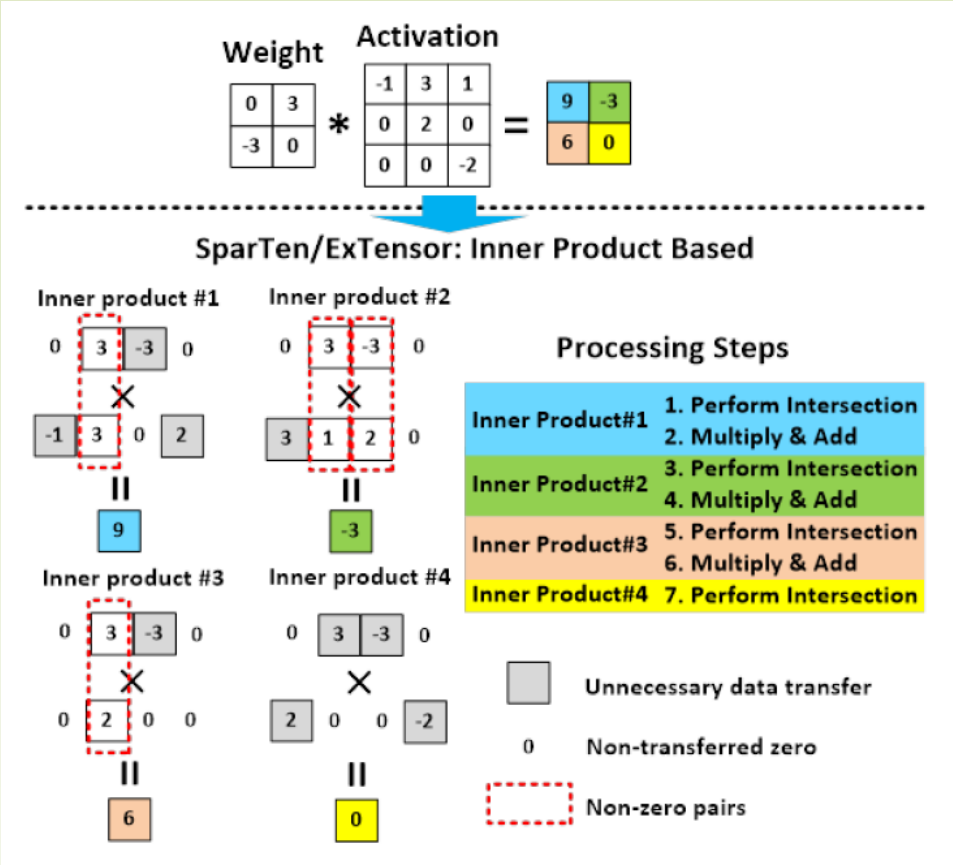
1 INTRODUCTION

Recently, there has been a surge of interest in generalized tensor algebra in the fields of deep learning [1, 15], machine learning [5], data science [7, 9, 27, 35, 48], physical sciences [18], engineering [20, 28], and graph analytics [34]. Tensors generalize vectors and matrices to *N*-dimensions, and tensor kernels combine two or more tensors using low-level computations similar to traditional 2-dimensional dense/sparse linear algebra, i.e., sequences of arithmetically intensive operations such as matrix multiplications.

These tensor operations often operate on very sparse data (i.e., with a small percentage of data which is non-zero). Figure 1 shows that the percentage of non-zero elements ranges from 10^{-6}% to 50% depending on the problem domain, with many domains featuring

SparTen & ExTensor: Intersection Based Method

Example of SparTen/ExTensor dataflow:



Why SparTen & ExTensor is not optimal ?

- The two designs suffer from three drawbacks, including:

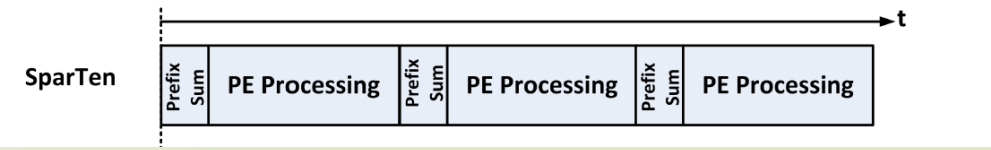
1) hardware cost for the intersection operation is high:

   SparTen use prefix sum; while ExTensor use content addressable memory(CAM)

   Both methods incur high hardware and energy overhead.

2) frequent stalls of computation phase due to strong data dependency between intersection and computation phases.

   (forming a conceptual producer and consumer data dependency)



3) unnecessary data transfer incurred by the explicit intersection operation.

GoSPA https://ieeexplore.ieee.org/gospa

- First, an implicit **on-the-fly intersection** is proposed to realize the optimal:

  intersection operation can be implemented much more efficiently when the knowledge of specific computation structure is considered.

- Observation: Dynamic and Static Streams

  Terminology:

  - Stream: operands of an intersection operation
  - dynamic stream: elements are not known before executing the computation
  - static stream: elements determined beforehand

Our key observation is that for 2-D convolution

in DNNs, the stream for sparse weights is static.

=> information needed is just a bit map of weights.

GoSPA

- **on-the-fly intersection:**

  For a pair of dynamic and static stream, it is possible and better to perform the logically equivalent intersection on-the-fly when the dynamic stream is brought to the compute unit.

- Since the static stream is known and does not change, based on its sparsity information, we can generate a conceptual static sparsity filter (SSF)

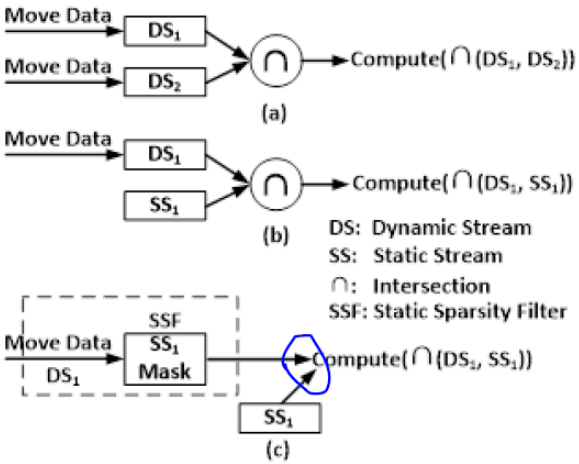


Fig. 3: (a) Intersection between two DS. (b) Intersection between DS and SS (SparTen/ExTensor's method). (c) On-the-fly intersection between DS and SS (Our method).

GoSPA

➨ **on-the-fly intersection:**

The simply but effective idea avoids the three drawbacks:

1) Clearly, it replaces the expensive intersection hardware cost, e.g., prefix-sum in SparTen and CAM in Extensor, with lightweight information embedded in dataflow.
2) by embedding SSF in the data path of dynamic stream, we can avoid transferring the unnecessary element as early as possible, instead of performing intersection after elements in both streams are staged in the intersection unit.
3) The design leads to a more streamlined dataflow. Hence, With common techniques such as pipelining, the execution can be made very efficient.

GoSPA

- **specialized computation reordering**:

  another optimization when applying on-the-fly intersection in 2-D convolution.

- **Why computation reordering?**

  If we produce one output at a time, we unavoidably only look at a subset of activations, we are losing the opportunity of **leveraging global computation structure** and also incurring additional data transfer (one of the drawbacks of SparTen and ExTensor).

- given activation can be paired at different times with different weight elements, and **will be fetched multiple times**. Why not performing all the checking relevant to an activation together?

GoSPA

- **specialized computation reordering: example**

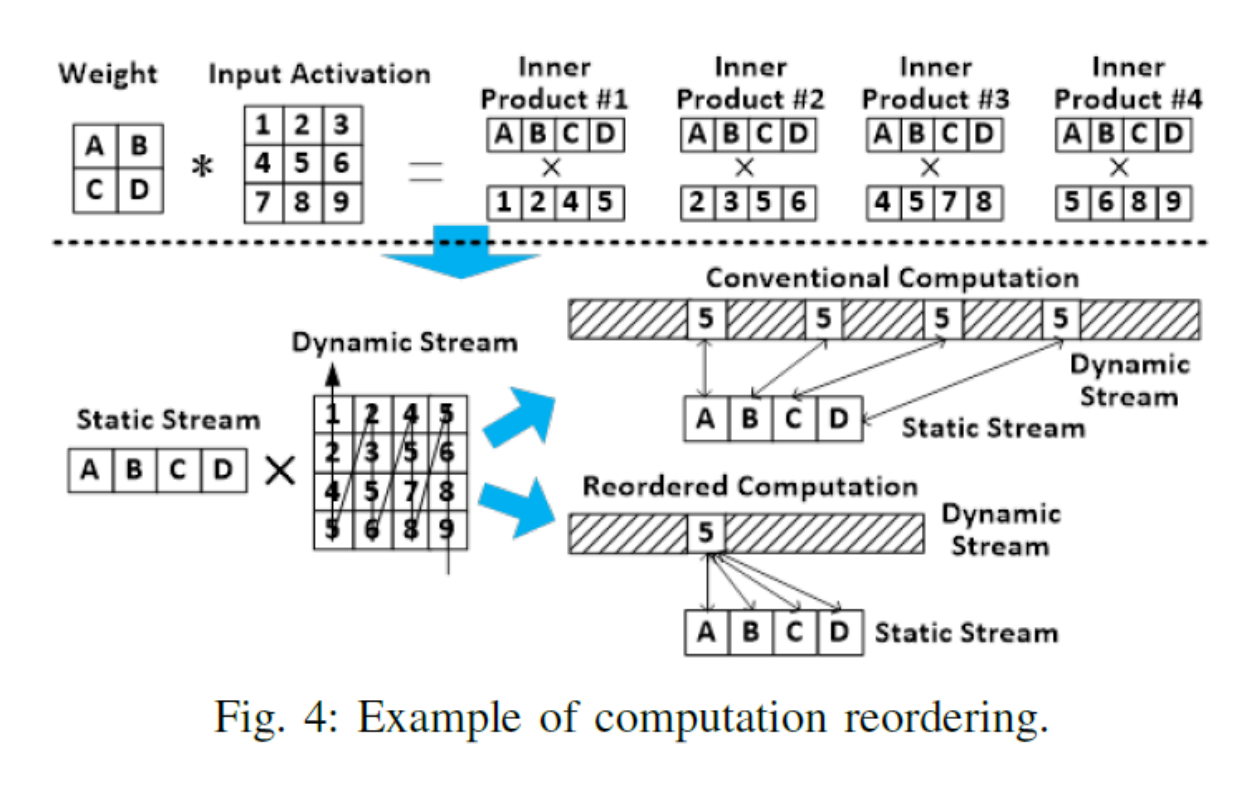
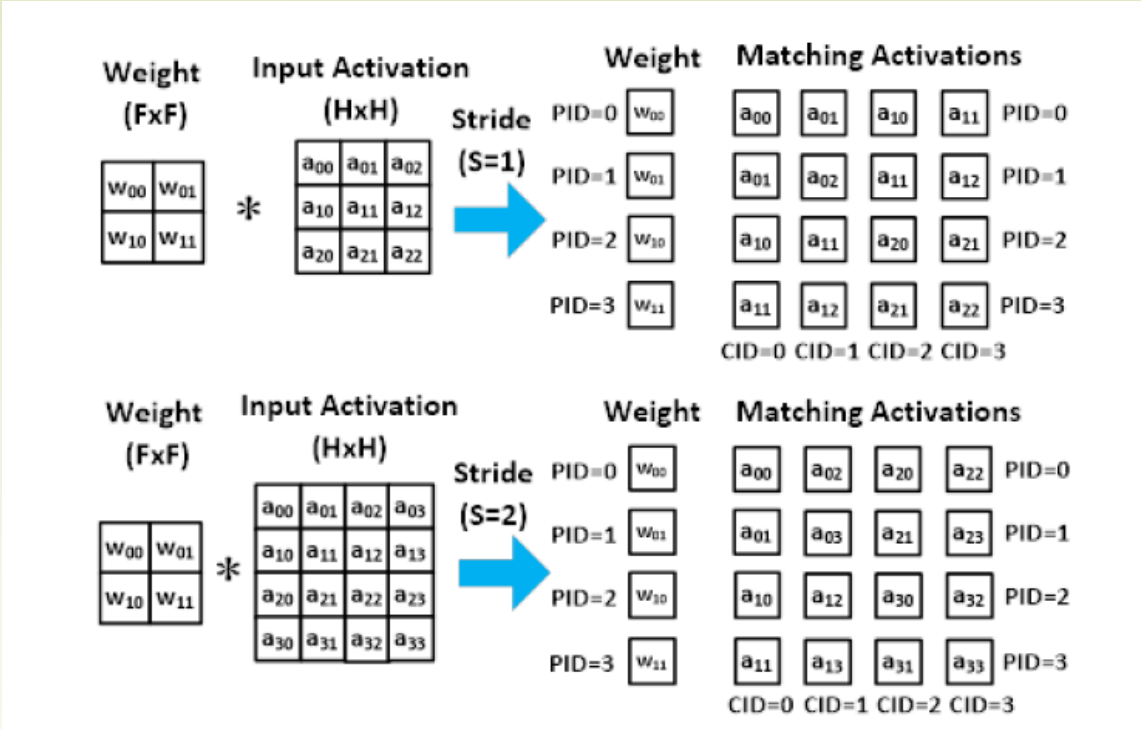


Fig. 4: Example of computation reordering.

Based on this property, the architecture is named as GoSPA, an energy efficient high-performance **G**lobally **O**ptimized **SP**arse CNN accelerator

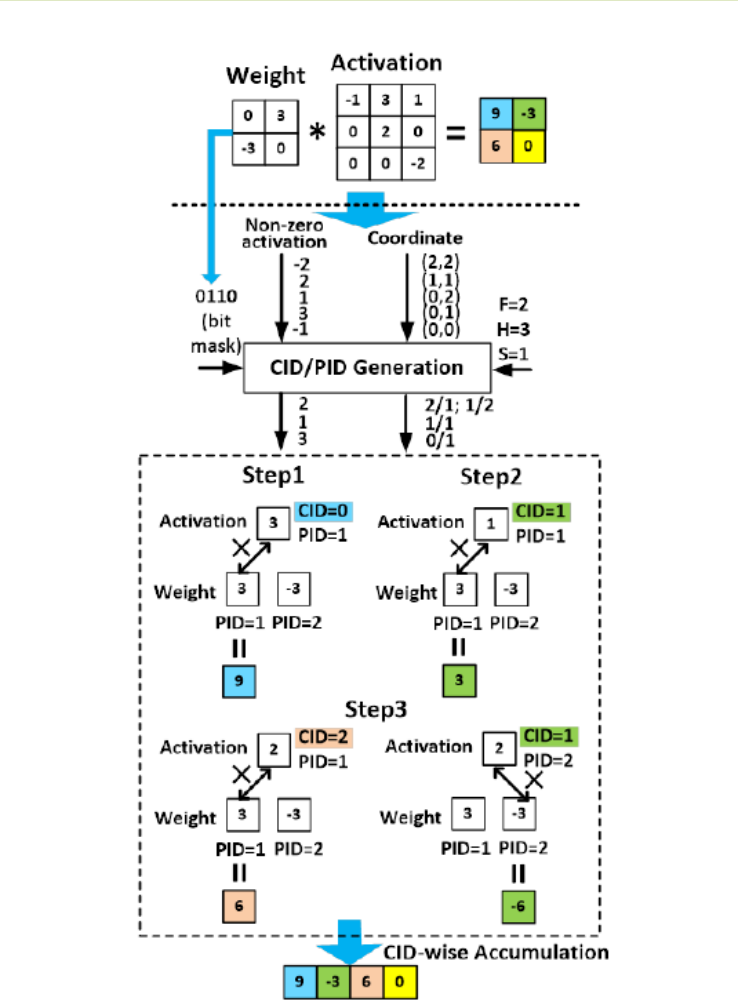
GoSPA

- **specialized computation reordering: how?**
- To conveniently perform the multiplication between matched weights and activations in processing units, we use **convolution ID (CID)** and **position ID (PID)** to jointly represent each activation, and use PID to solely represent each weight



GoSPA

- **Example of the optimized processing procedure:**



GoSPA Architecture

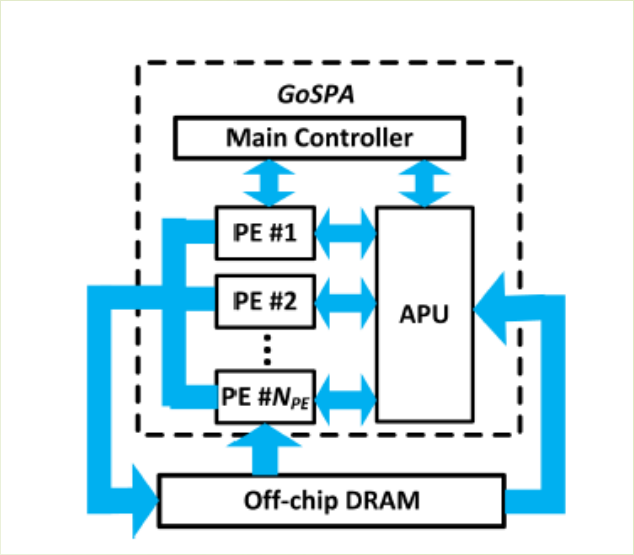
- activation processing unit (APU):

APU processes the sparse data and generate ID information as well as preparing correct non-zero activations to the corresponding PEs.

- Processing Elements (PEs):

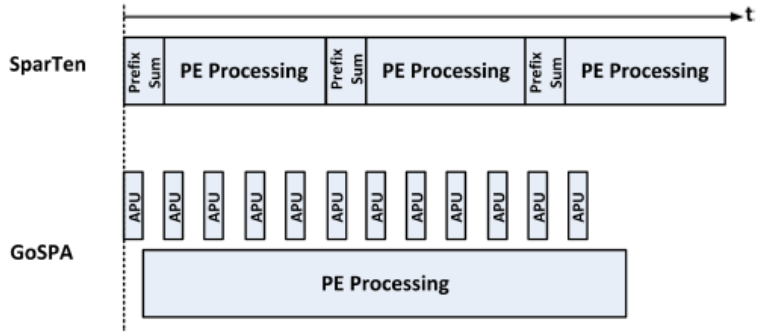
After receiving ACT, CID and PID from APUs, PEs identify the correct non-zero weights for the current input non-zero activations and perform multiplication and accumulation.

The overall hardware architecture of GoSPA.



Why GoSPA is optimal?

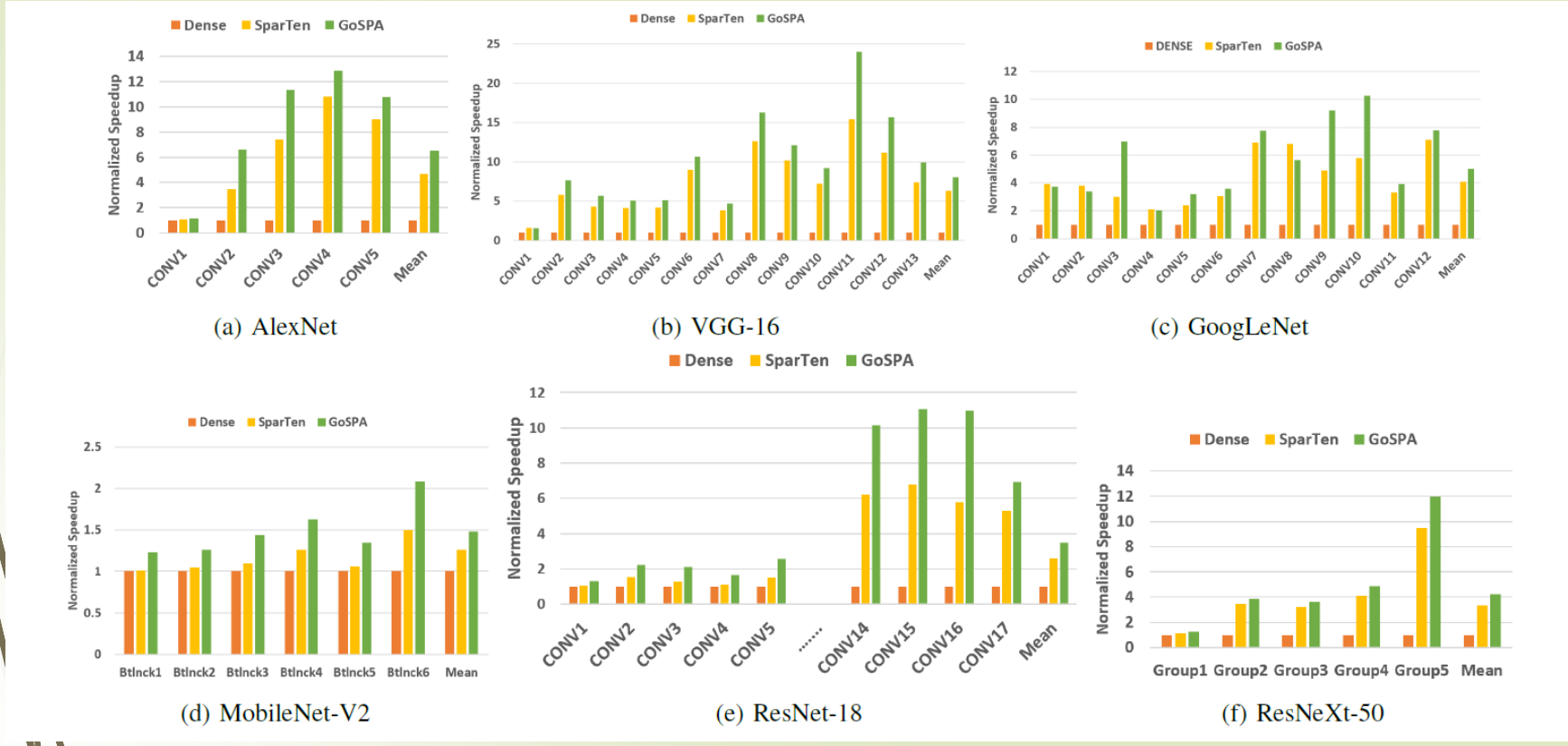
- because the same PID values always occur in the adjacent cycles, the corresponding PID-indexed weight value always stay in the registers at its maximum possible period, therefore significantly improving register-level weight reuse and reducing SRAM access.
- On the other hand, the dataflow of GoSPA makes a pipeline-like computing procedure become possible.



The processing scheduling of SparTen and GoSPA.

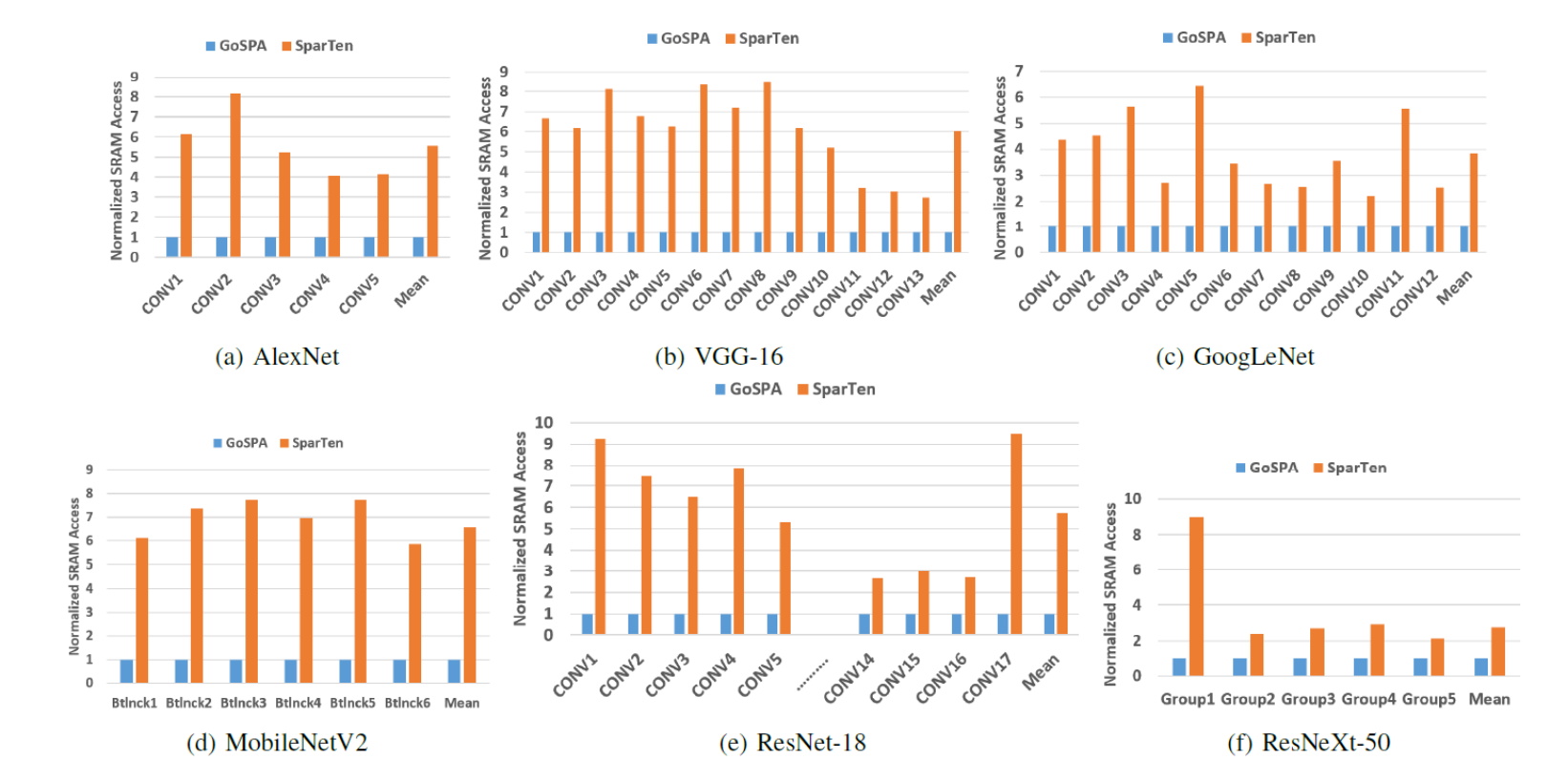
GoSPA: How much it is worth?

- Let's start with **Speedup** comparison between GoSPA and SparTen.
- For MobileNet, "Btlnck" denotes the bottleneck block of stacked CNN layers.
- For ResNeXt, "Group" means the stage of stacked CNN layers with cardinality=32



GoSPA: How much it is worth?

- **Normalized SRAM access comparison**.
- For MobileNet, ”Btlnck” denotes the bottleneck block of stacked CNN layers.
- For ResNeXt, ”Group” means the stage of stacked CNN layers with cardinality=32

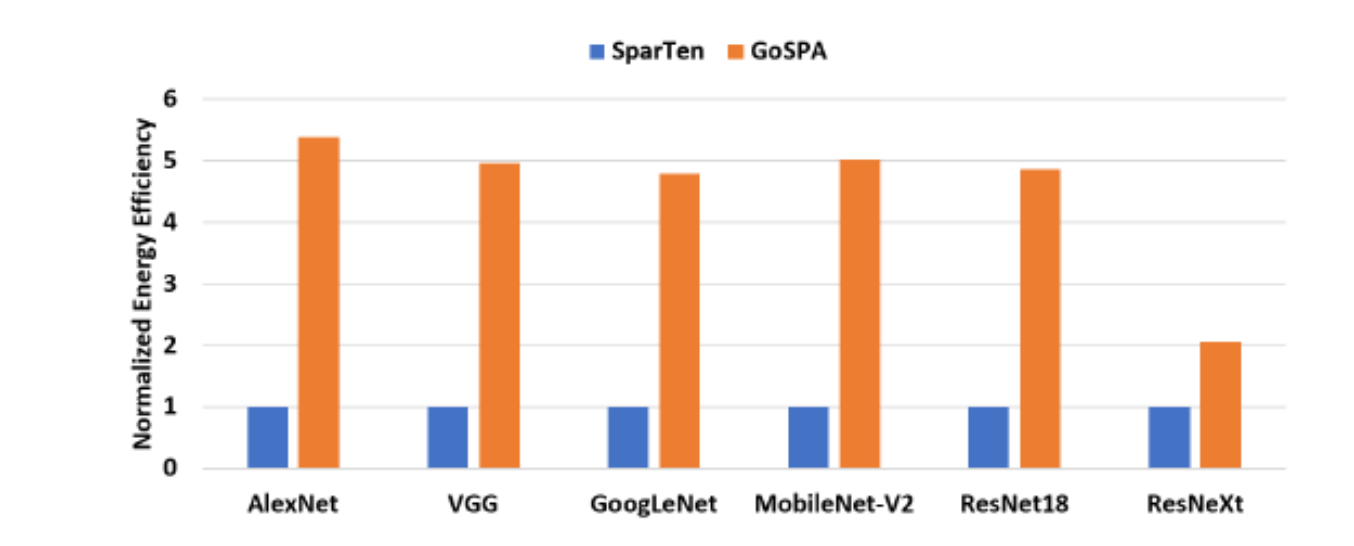


GoSPA: How much it is worth?

- **Normalized energy efficiency comparison**.

In depth numerical analysis verifies such significant improvement on energy efficiency mainly come from the two architecture level advantages of GoSPA over SparTen:

1) GoSPA has much less SRAM access than SparTen.
2) GoSPA needs much less hardware overhead for handling sparsity than SparTen.



GoSPA: recap

- **Evaluation results:** the proposed 2-D convolution-centered design principle brings significant hardware performance improvement than the inner product-based designs:

- Compared with SparTen, GoSPA achieves average x1.38, x1.28, x1.23, x1.17, x1.21, x1.28 **speedup** on AlexNet, VGG, GoogLeNet, MobileNet, ResNet and ResNeXt, respectively.

- More importantly, GoSPA achieves 5.38, 4.96, 4.79, 5.02, 4.86 and 2.06 **energy efficiency improvement** on AlexNet, VGG, GoogLeNet, MobileNet, ResNet and ResNeXt, respectively

CONCLUSION

This paper proposes GoSPA, a sparse CNN accelerator architecture. By using a novel on-the-fly intersection and reordering computation, GoSPA globally optimizes sparse 2-D convolution and hence avoids the limitations of the state-of-the-art sparse CNN designs.

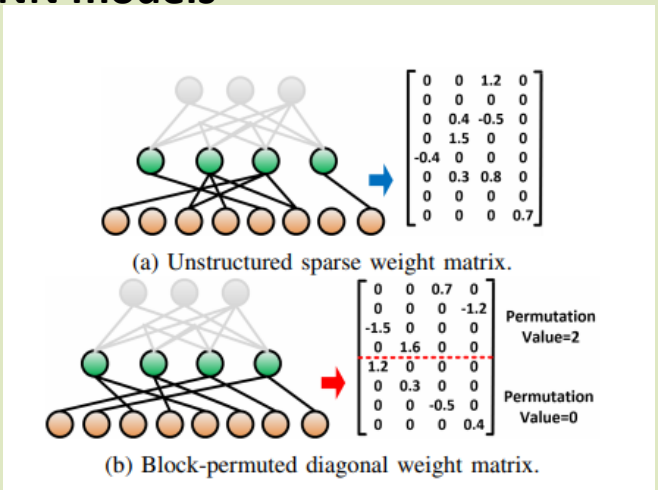
PermCNN link

- Motivation: model compression and hardware acceleration.
- Model compression, as an algorithm-level solution, targets to reduce the DNN model sizes without accuracy drop or only negligible loss.
- efforts on hardware acceleration aim to achieve high-performance execution of DNN models via designing DNN-specific computing platforms.

Recently, PermDNN, as a work that performs model compression and hardware acceleration simultaneously, is proposed in to provide **an algorithm/hardware co-design solution** for high-performance DNN model execution.

By imposing **permutation diagonal structure on the DNN models**

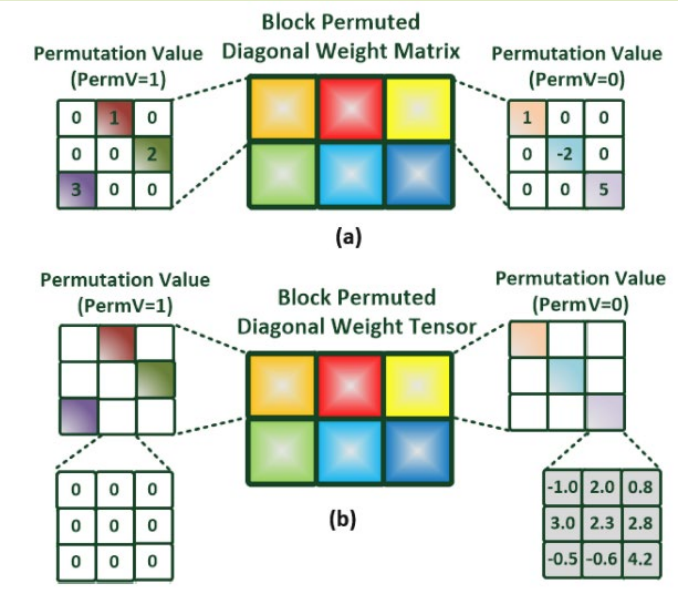
Mathematically, such approximation is the optimal approximation in term of l_2 norm measurement on the approximation error



PERMCNN

➨ **permutation diagonal structure, illustration:**

Example of imposing block-diagonal permuted structure on (a) weight matrix of FC layer and (b) weight tensor of CONV layer. Here the block size p, as the compression ratio, is 3. The entire weight matrix or tensor contains blocks of component weight matrices or tensors, and each component permuted diagonal matrix or tensor is affiliated with a permutation value (PermV). PermV is selected from 0,1,...,p-1. The nonzero weight values or weight filter kernels can only be placed in the main diagonal or permuted diagonal positions in the component weight matrices or tensors.



PermCNN

Storage requirement comparison:

***p*-by-*p* Sparse Matrix**

Sparsity Ratio: 1/*p* Quantization: *q*-bit

Unstructured Matrix	Permuted Diagonal Matrix
$\begin{bmatrix} 0 & 1.2 & 0 & 0 \\ 0 & 0 & 0 & 0.3 \\ 0 & 0 & -0.5 & 0 \\ 0.4 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & -1.2 \\ -1.5 & 0 & 0 & 0 \\ 0 & 1.6 & 0 & 0 \end{bmatrix}$
Col_Index=(1, 2, 3, 4)	Col_Index=(1, 2, 3, 4)
Store: Value=(0.4, 1.2, -0.5, 0.3)	Store: Value=(-1.5, 1.6, 0.7, -1.2)
Row_Index=(4, 1, 3, 2)	Permutation Value (PermV)=2
	Compute: Row_Index= (Col_Index+PermV) mod *p*
Total Space Cost: 2*pq* bits	Total Space Cost: (pq+$\log_2 p$) bits

For conv layers: $x \in \mathbb{R}^{NxCxHxW}, x \in \mathbb{R}^{NxMxExF}, W \in \mathbb{R}^{NXCXHXW}$ are input tensor, output tensor, and weight tensor, respectively.

$$\mathcal{Y}(n,m,e,f) = \sum_{c=0}^{C-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1}\mathcal{W}(m,c,r,s) \times \mathcal{X}(n,c,Ue+r,Uf+s),$$

$$\mathcal{Y}(n,m,e,f) = \sum_{g=0}^{\frac{C}{p}-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1}\mathcal{X}(n,gp+(i+k_l \bmod p), Ue+r,Uf+s) \times \mathcal{W}'(k_l \times p+i,r,s).$$

PermCNN

Forward Propagation: As illustrated before, the key idea of designing permuted diagonal CONV layer is to impose such structure along the two dimensions that are defined by number of 3D filters and number of channels. Based on this mechanism, the non-zero kernels can only be placed in the main diagonal or permuted diagonal positions Overall, the forward propagation can be summarized as:

$$\mathcal{Y}(n,m,e,f) = \sum_{g=0}^{\frac{C}{p}-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \mathcal{X}(n, gp + (i + k_l \bmod p), Ue + r, Uf + s) \times \mathcal{W}'(k_l \times p + i, r, s).$$

Backward Propagation: is also developed in PermDNN to ensure the trained CONV layer exhibits permuted diagonal structure

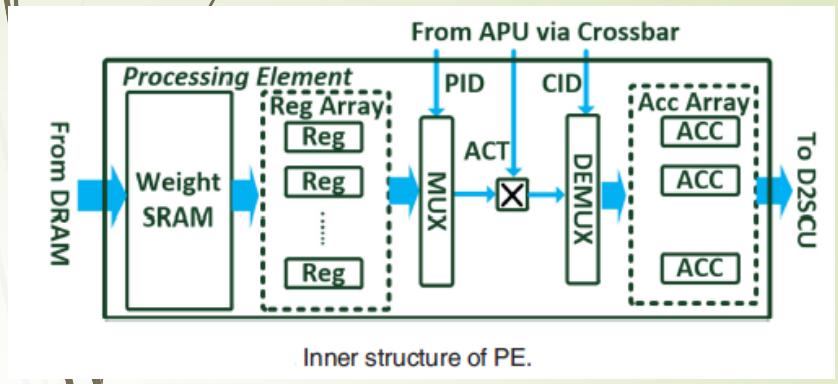
$$\frac{\partial J}{\mathcal{W}(m,c,r,s)} = \sum_{n=0}^{N-1} \sum_{e=0}^{E-1} \sum_{f=0}^{F-1} \mathcal{X}(n,c,Ue+r,Uf+s) \times \frac{\partial J}{\partial \mathcal{Y}(n,m,e,f)}, \forall\ \mathcal{W}(m,c,r,s) \neq 0$$

$$\frac{\partial J}{\partial \mathcal{X}(n,c,x,y)} = \sum_{m=1}^{M-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \mathcal{W}(n,m,r,s) \times \frac{\partial J}{\partial \mathcal{Y}(n,m,,(x-r)/U,(y-s)/U)}.$$

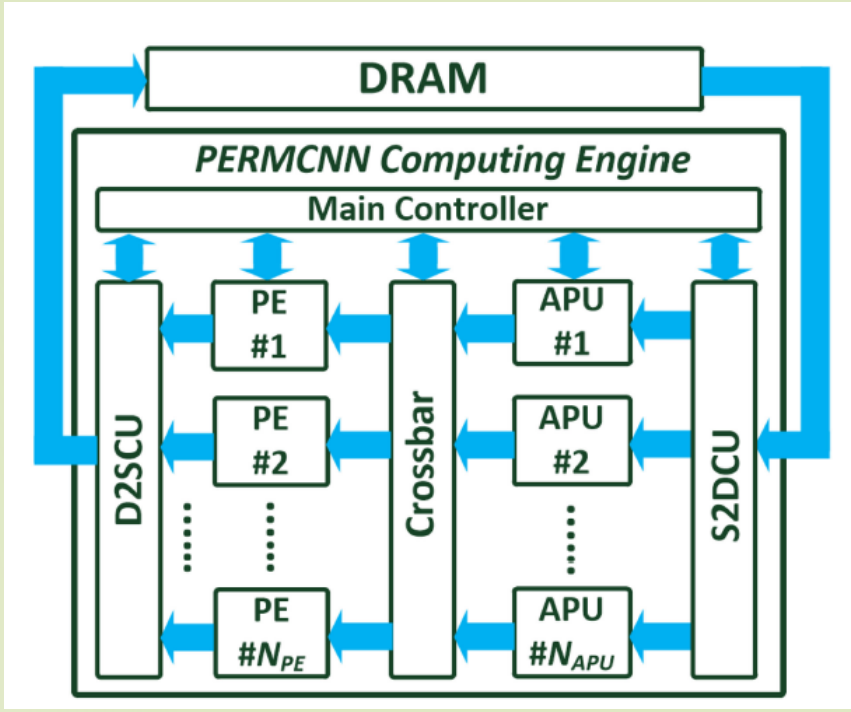
PermCNN Architecture

- D2SCU: dense-to-sparse conversion unit.
- S2DCU: sparse-to-dense conversion unit.

- APU & PE: Activation Processing Unit & Processing Unit (Similar to GoSPA)
- Crossbar Module: ensures each PE receives its desired activation values from APUs

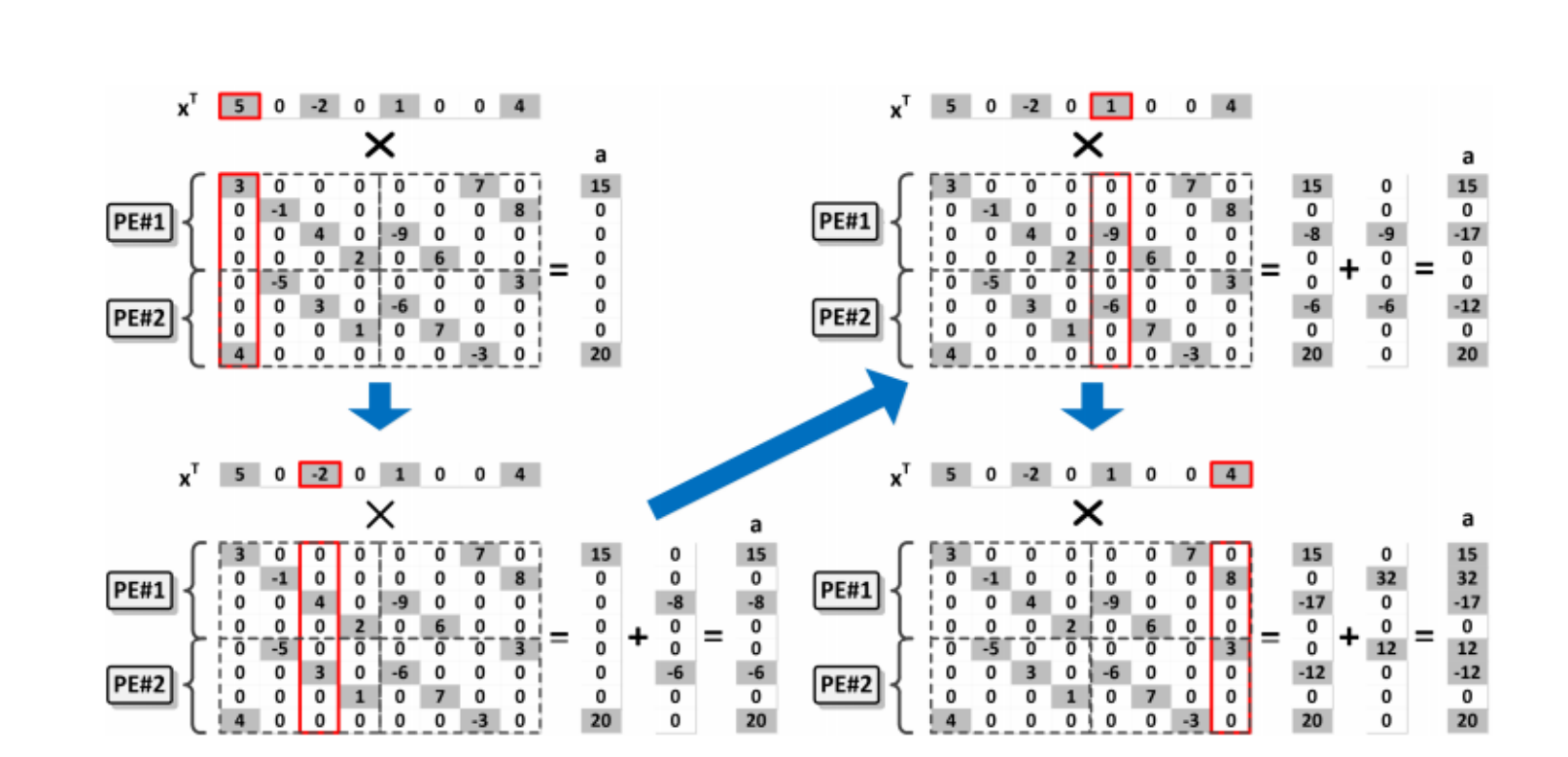


Inner structure of PE.



PermCNN dataflow

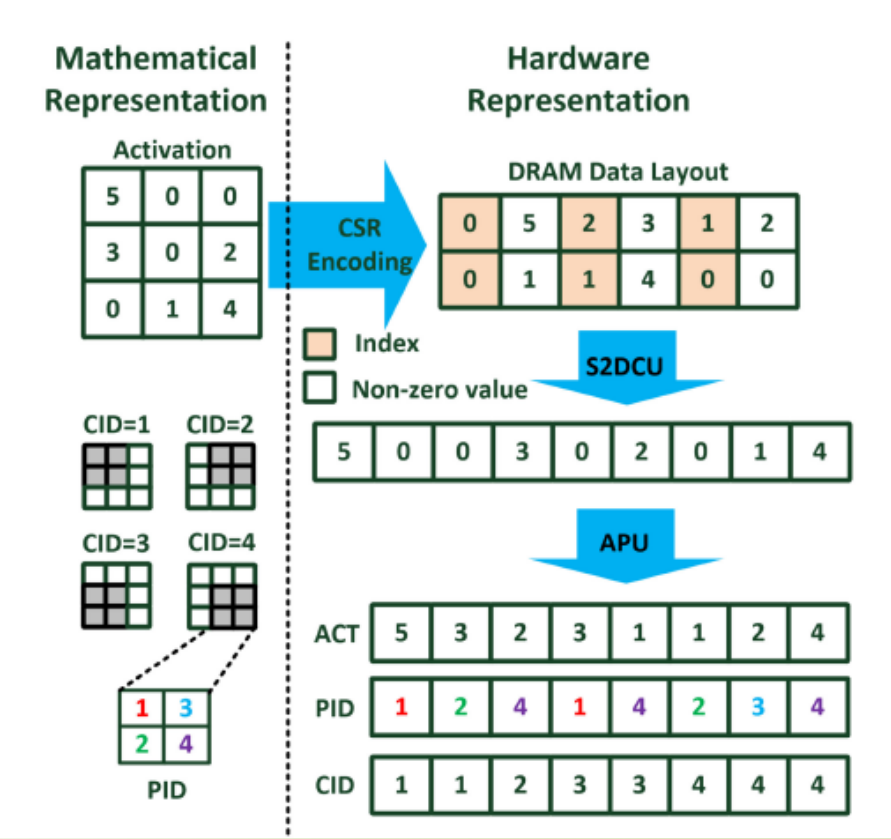
- Example column-wise processing procedure with input zero-skipping scheme



PermCNN dataflow

- Data processing example in S2DCU and APU.

Here activation map is 3x3, and filer kernel size is 2x2



PermCNN Evaluation

- Performance Breakdown of in AlexNet and VGG:

CR = compression ratio (block size), and IAS = input activation sparsity

5 CONV Layers in AlexNet
Batch size = 4

Layer	CR	IAS	Processing Time (ms)	Power (mW)	DRAM Access (MB)
CONV1	1	0%	5.49	511	3.1
CONV2	4	20%	1.85	489	1.4
CONV3	4	50%	0.77	500	1.5
CONV4	4	69%	0.36	511	1.1
CONV5	4	62%	0.29	506	0.7
Total	3.83	24%	8.76	506	7.7

13 CONV Layers in VGG
Batch size = 3

Layer	CR	IAS	Processing Time (ms)	Power (mW)	DRAM Access (MB)
CONV1-1	1	0%	5.09	303	10.3
CONV1-2	4	49%	14.65	315	24.1
CONV2-1	4	20%	5.68	493	9.8
CONV2-2	4	34%	9.38	496	12.2
CONV3-1	4	35%	4.64	493	5.6
CONV3-2	4	50%	7.13	500	7.5
CONV3-3	4	49%	7.24	499	7.3
CONV4-1	4	56%	3.17	498	3.9
CONV4-2	4	66%	4.85	506	6.1
CONV4-3	4	68%	4.59	508	5.8
CONV5-1	4	75%	0.89	519	3.0
CONV5-2	4	74%	0.94	516	3.0
CONV5-3	4	72%	1.01	514	3.0
Total	4.00	45%	69.25	442	101.6

GoSPA vs PermCNN (28nm CMOS)

	GoSPA (=)	PermCNN (=)	
Flexibility	Inference oriented Hardware	Inference & Training Hardware+alg' co-design	(=)
DRAM Access (MB)	58.8 (AXN) 4.1 (VGG)	101.6 (AXN) 7.7 (VGG)	(=)
Power (mW)	429.4 (AXN) 445.3 (VGG)	442 (AXN) 506 (VGG)	(=)
Throughput (frame/s)	460.3 (AXN) 29.7 (VGG)	456.6 (AXN) 43.31 (VGG)	-
Area Efficiency (frame/s/mm2)	172.4 (AXN) 11.1 (VGG)	132.73 (AXN) 12.59 (VGG)	(=)

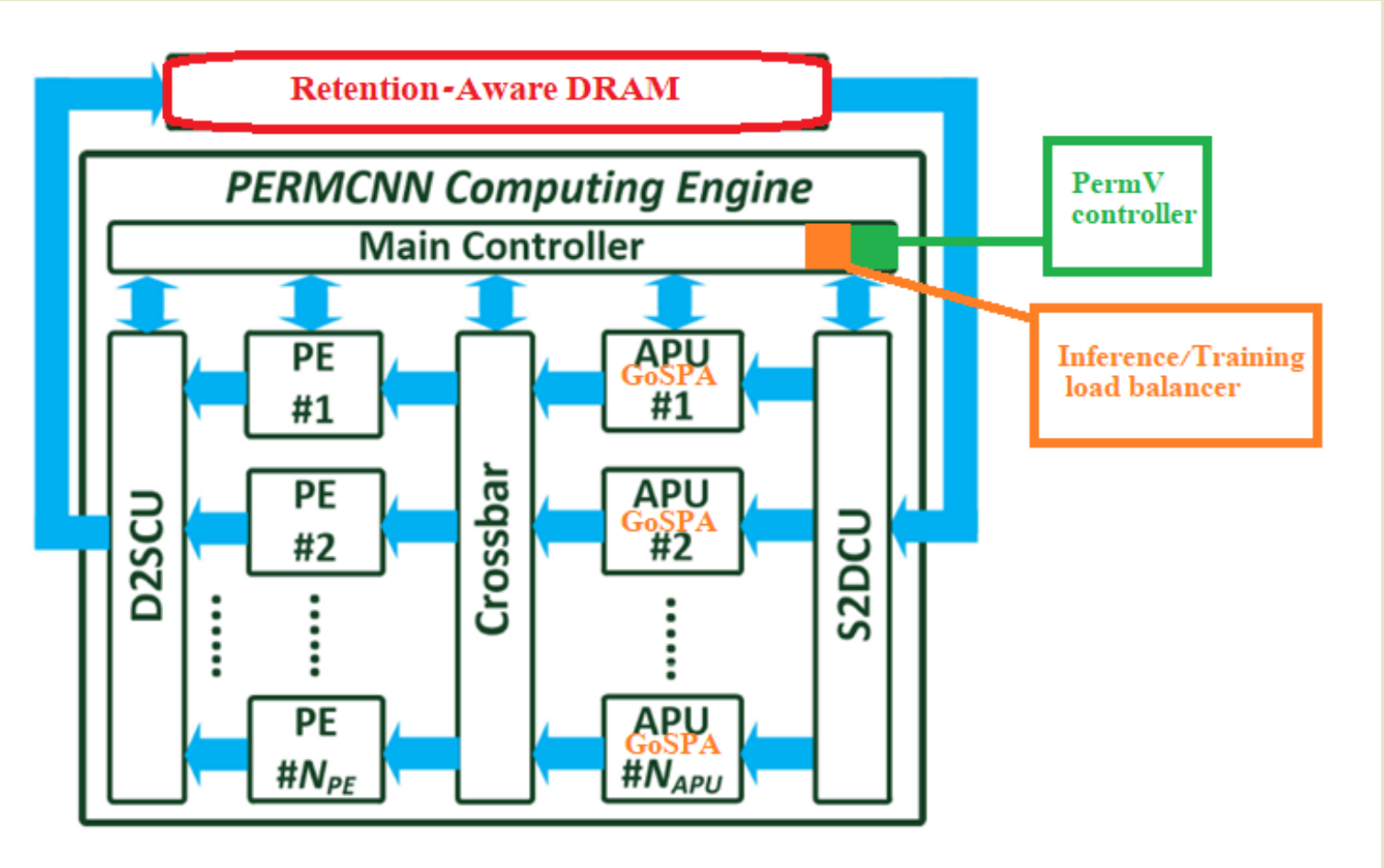
GoPerm ?

GoPerm

- **Taking the key-ideas from both GoSPA and PermCNN, we propose GoPerm:**

- Permutation diagonal structure
- Inference & Training Architecture: by using Forward/Back Propagation in PermDNN
- Using the more robust PID/CID algorithm in GoSPA (with reordirng)
- hyper-parameter fine-tuning, more dynamically configuration
- RANA [slides] : Refresh is unnecessary if the data's lifetime in eDRAM is shorter than the eDRAM's retention time, so more refresh can be removed.

GoPERM conceptual architecture design



References

[SCNN] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, “Scnn: Anaccelerator for compressed-sparse convolutional neural networks,” in Proceedings of the 44th Annual International Symposium on Computer Architecture. ACM, 2017, pp. 27–40.

[SparTen] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, “Sparten: A sparse tensor accelerator for convolutional neural networks,” in Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO ’52. New York, NY, USA: ACM, 2019, pp. 151–165.

[ExTensor] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, “Extensor: An accelerator for sparse tensor algebra,” in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO ’52. New York, NY, USA: Association for Computing Machinery, 2019, p.319–333.

[PermDNN] C. Deng, S. Liao, Y. Xie, K. K. Parhi, X. Qian, and B. Yuan, “PermDNN: Efficient compressed DNN architecture with permuted diagonal matrices,” in Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit., 2018, pp. 189–202

[PermCNN] C. Deng, S. Liao, and B. Yuan, “Permcnn: Energy-efficient convolutional neural network hardware architecture with permuted diagonal structure,” IEEE Transactions on Computers, vol. 70, no. 2, pp. 163–173, 2021.

[GoSPA] [ISCA] C. Deng, Y. Sui, S. Liao, X. Qian, and B. Yuan, “GoSPA: An Energy-efficient High-performance Globally Optimized SParse Convolutional Neural Network Accelerator,” accepted by ACM Intl. Symp. on Computer Architecture (ISCA), June 2021.

Questions?
THANK YOU ☺