Advanced topics in computer architectures - 236509

System Aspects of machine learning

Student Name	ID	E-mail
Muhammad Dahamshi	318396926	muhammad.dah@campus.technion.ac.il

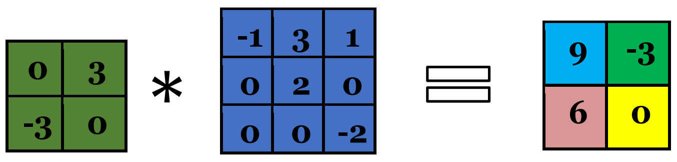
Sparse-CNN Accelerators - Final project

1 Introduction, 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



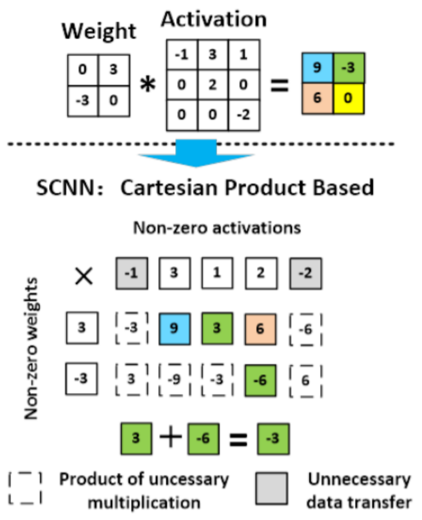
2 Contents and Organization

The rest of this paper is organized as follows. Related works in sparse-aware CNN acceleration are presented in Section 3. Section 4 is more involved discussion in GoSPA as a more robust architecture based in these related works, including evaluation results. PermCNN is presented in Section 5 as nn-model compression design. While Section 6 are devoted to the compression between GoSPA and PermCNN. In Section 7, we propose our new architecture, which we named GoPerm, and improvements from taking the key-ideas from both GoSPA and PERMCNN.

3 RELATED WORK

3.1 First novel approach: SCNN

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.

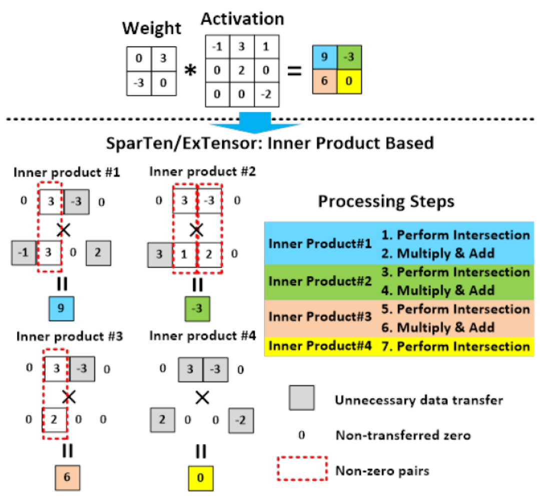

Weight
Activation
SCNN: Cartesian Product Based
Non-zero activations
Non-zero weights
Product of uncessary multiplication
Unnecessary data transfer
Fig. 1: Example of SCNN dataflow.

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.

3.2 SparTen & ExTensor: Intersection Based Method

Example of SparTen/ExTensor dataflow:


Weight
Activation
SparTen/ExTensor: Inner Product Based
Inner product #1
Inner product #2
Inner product #3
Inner product #4
Processing Steps
Inner Product#1 1. Perform Intersection 2. Multiply & Add
Inner Product#2 3. Perform Intersection 4. Multiply & Add
Inner Product#3 5. Perform Intersection 6. Multiply & Add
Inner Product#4 7. Perform Intersection
Unnecessary data transfer
Non-transferred zero
Non-zero pairs

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.

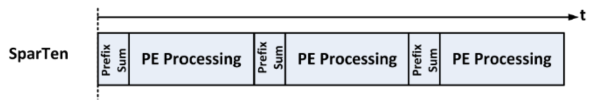
Both solutions identify the matching non-zero value pairs between kernel weights and the corresponding activations with a **conceptual intersection operation**.

The two schemes differ in how the intersection operation is implemented.

3.2.1 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

4 GoSPA

[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

The 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.

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)

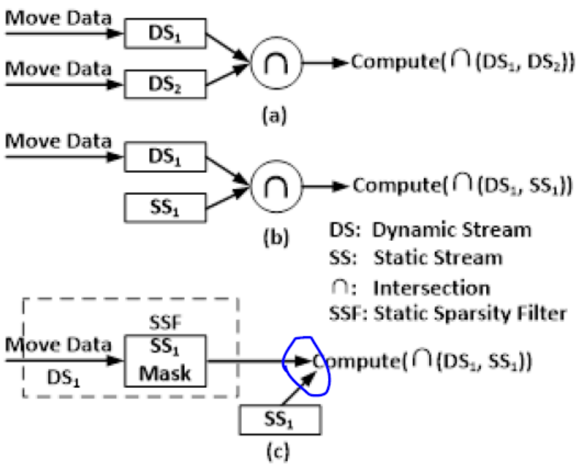


Fig. 3: (a) Intersection between two DS. (b) Intersection between DS and SS (SparTen/ExTensor's method). (c) On-the-fly intersection between DS and SS (Our method).

The simple 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.

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?

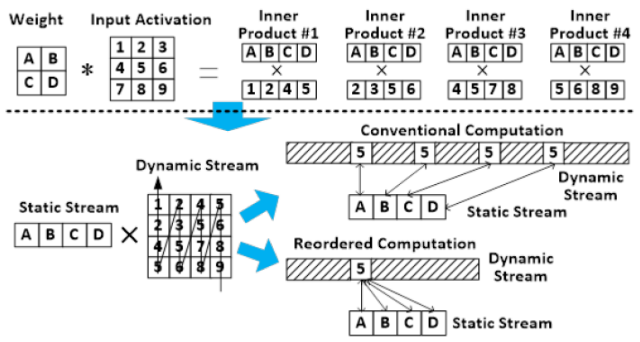
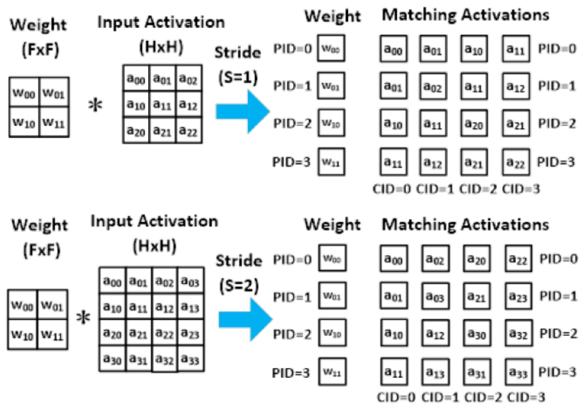
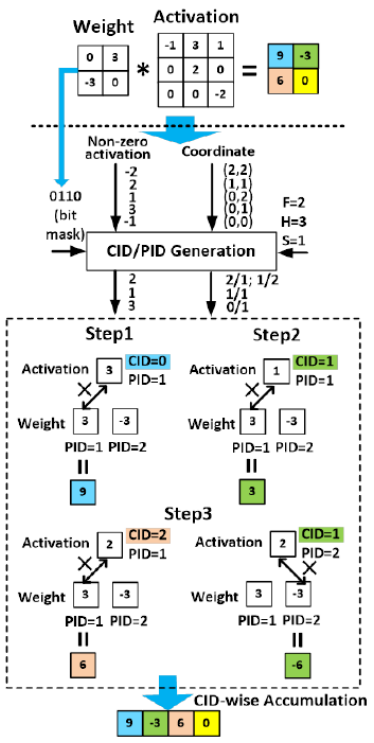


Fig. 4: Example of computation reordering.

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

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





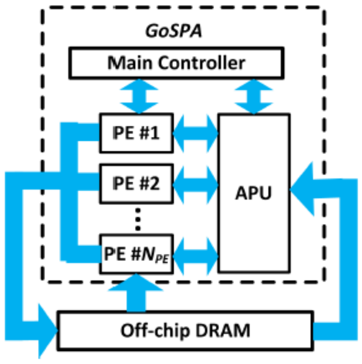
⇐ **Example of the optimized processing procedure**

4.1 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.

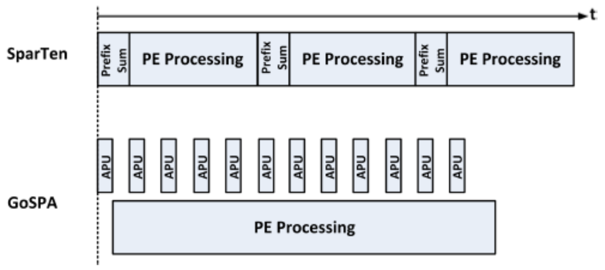
Processing Elements (PEs): After receiving ACT, CID and PID from APUs, PEs identify the correct non-zero weights for the current input non-zero activations and perform multiplication and accumulation.

The overall hardware architecture of GoSPA.



4.2 Why GoSPA is optimal?

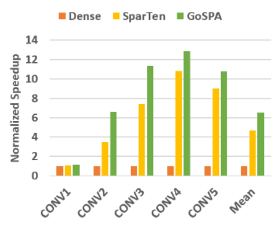
because the same PID values always occur in the adjacent cycles, the corresponding PID-indexed weight value always stay in the registers at its maximum possible period, therefore significantly improving register-level weight reuse and reducing SRAM access. On the other hand, the dataflow of GoSPA makes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



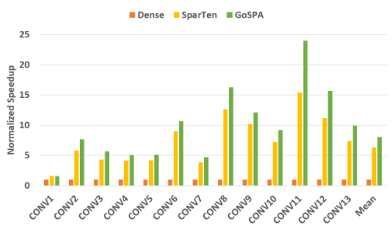
The processing scheduling of SparTen and GoSPA.

4.3 GoSPA: How much it is worth?

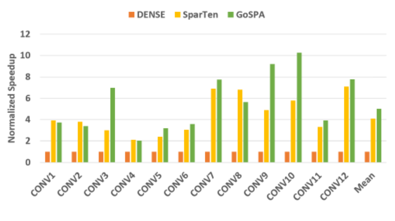
Let's start with Speedup comparison between GoSPA and SparTen.
For MobileNet, "Btlnck" denotes the bottleneck block of stacked CNN layers.
For ResNeXt, "Group" means the stage of stacked CNN layers with cardinality=32



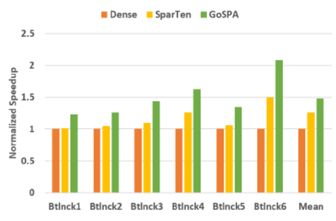
(a) AlexNet



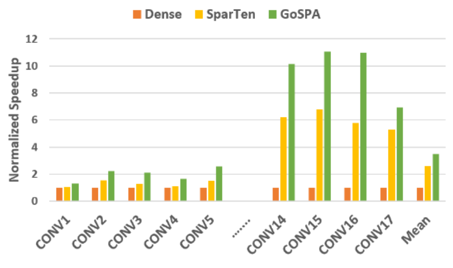
(b) VGG-16



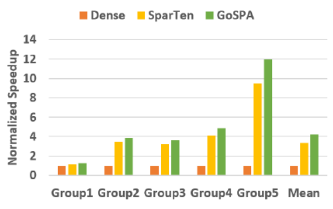
(c) GoogLeNet



(d) MobileNet-V2

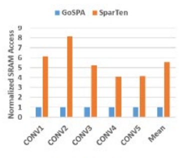


(e) ResNet-18

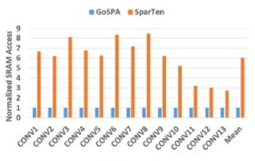


(f) ResNeXt-50

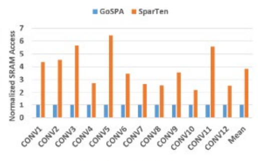
Normalized SRAM access comparison:



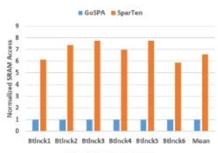
(a) AlexNet



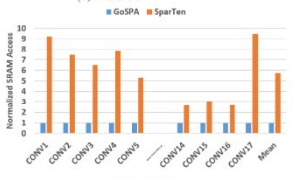
(b) VGG-16



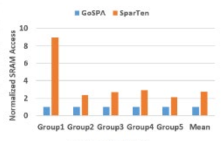
(c) GoogLeNet



(d) MobileNetV2

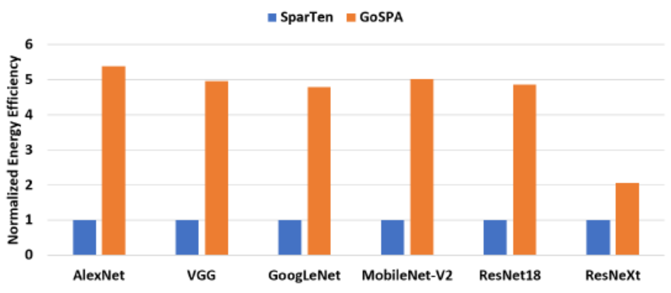


(e) ResNet-18



(f) ResNeXt-50

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.

Evaluation summary: 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

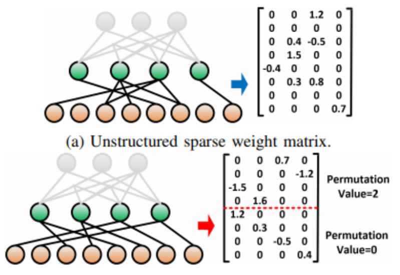
5 PERMCNN link

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**

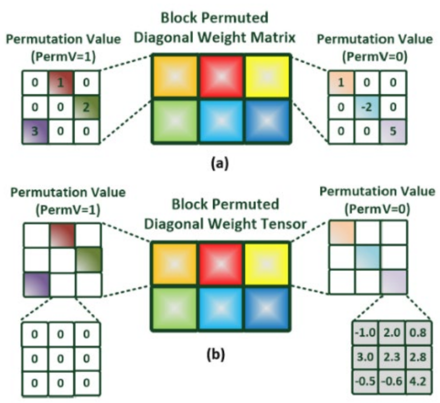
Mathematically, such approximation is the optimal approximation in term of l_2 norm measurement on the approximation error



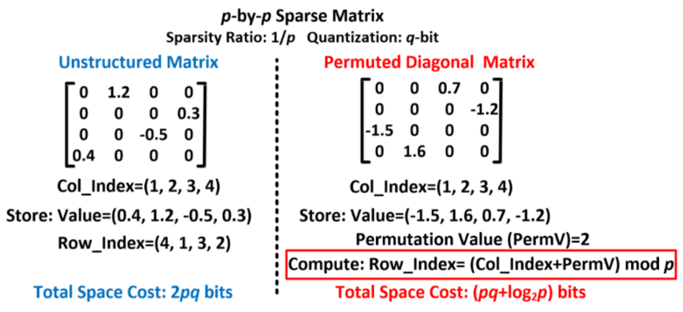
(a) Unstructured sparse weight matrix.
(b) Block-permuted diagonal weight matrix.

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.


Block Permuted Diagonal Weight Matrix
Permutation Value (PermV=1)
Permutation Value (PermV=0)
(a)
Block Permuted Diagonal Weight Tensor
Permutation Value (PermV=1)
Permutation Value (PermV=0)
(b)

Storage requirement comparison:



p-by-*p* Sparse Matrix
Sparsity Ratio: $1/p$ Quantization: q-bit

Unstructured Matrix

$$\begin{bmatrix} 0 & 1.2 & 0 & 0 \\ 0 & 0 & 0 & 0.3 \\ 0 & 0 & -0.5 & 0 \\ 0.4 & 0 & 0 & 0 \end{bmatrix}$$

Col_Index=(1, 2, 3, 4)
Store: Value=(0.4, 1.2, -0.5, 0.3)
Row_Index=(4, 1, 3, 2)
Total Space Cost: $2pq$ bits

Permuted Diagonal Matrix

$$\begin{bmatrix} 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & -1.2 \\ -1.5 & 0 & 0 & 0 \\ 0 & 1.6 & 0 & 0 \end{bmatrix}$$

Col_Index=(1, 2, 3, 4)
Store: Value=(-1.5, 1.6, 0.7, -1.2)
Permutation Value (PermV)=2
Compute: Row_Index= (Col_Index+PermV) mod p
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.

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

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

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:

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

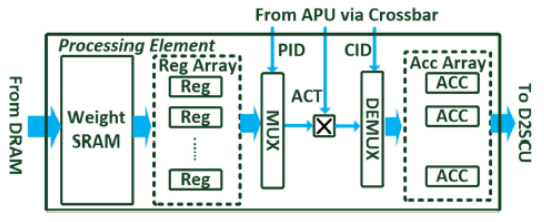
Backward Propagation: is also developed in PermDNN to ensure the trained CONV layer exhibits permuted diagonal structure

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

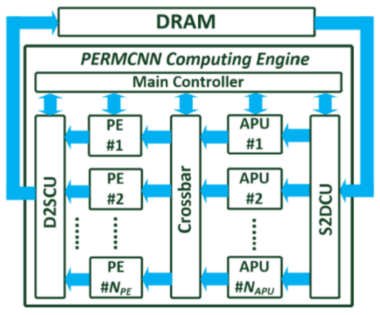
$$\frac{\partial J}{\partial\mathcal{X}(n,c,x,y)}=\sum_{m=1}^{M-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1}\mathcal{W}(n,m,r,s)\times\frac{\partial J}{\partial\mathcal{Y}(n,m,,(x-r)/U,(y-s)/U)}.$$

5.1 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.



5.2 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

6 GoSPA vs PERMCNN (28nm

CMOS)

	GoSPA (=)	PermCNN (=)	
Flexibility	Inference oriented Hardware	Inference & Training Hardware + algorithm 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)	(=)

7 GoPERM

Taking the key-ideas from both GoSPA and PERMCNN, we propose GoPERM:

- Permutation diagonal structure

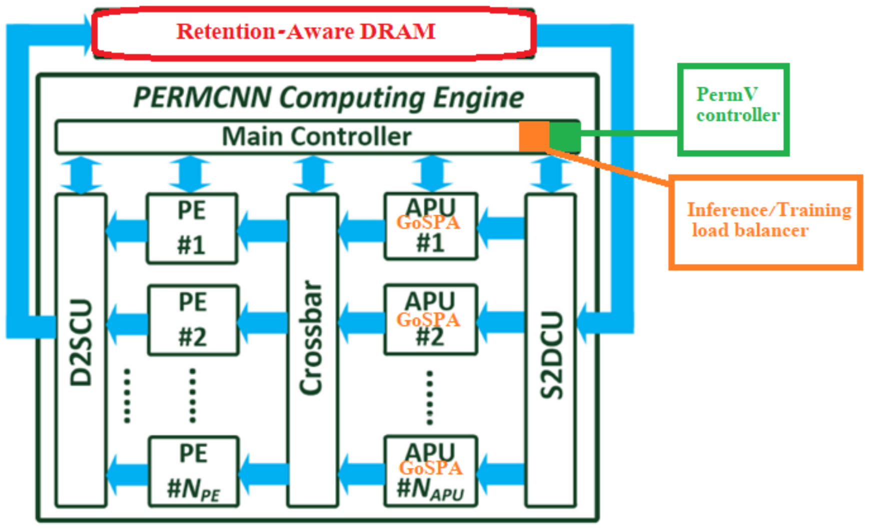
- Inference & Training Architecture: by using Forward/Back Propagation in PermDNN

- Using the more robust PID/CID algorithm in GoSPA (with reordering)

- 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:



8 References

[SCNN] 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] 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.

Thanks for reading :)