PERMCNN: Energy-efficient Convolutional Neural Network Hardware Architecture with Permuted Diagonal Structure

Chunhua Deng, Siyu Liao, Bo Yuan

Abstract—In the emerging artificial intelligence (AI) era, efficient hardware accelerator design for deep neural networks (DNNs) is very important to enable real-time energy-efficient DNN model deployment. To this end, various DNN model compression approaches and the corresponding hardware architectures have been intensively investigated. Recently, PERMDNN, as a permuted diagonal structure-imposing model compression approach, was proposed with promising classification performance and hardware performance. However, the existing PERMDNN hardware architecture is specifically designed for fully-connected (FC) layer-contained DNN models; while its support for convolutional (CONV) layer is missing. To fill this gap, this paper proposes PERMCNN, an energy-efficient hardware architecture for permuted diagonal structured convolutional neural networks (CNNs). By fully utilizing the strong structured sparsity in the trained models as well as dedicatedly leveraging the dynamic activation sparsity, PERMCNN delivers very high hardware performance for inference tasks on CNN models. A design example with 28nm CMOS technology shows that, compared the to state-of-the-art CNN accelerator, PERMCNN achieves 3.74× and 3.11× improvement on area and energy efficiency, respectively, on AlexNet workload, and 17.49× and 14.22× improvement on area and energy efficiency, respectively, on VGG model. After including energy consumption incurred by DRAM access, PERMCNN achieves 2.60× and 9.62× overall energy consumption improvement on AlexNet and VGG workloads, respectively.

Index Terms—Deep Learning, Model Compression, Hardware Accelerator, Convolutional Neural Network

1 INTRODUCTION

In the emerging artificial intelligence (AI)-centric era, deep neural networks (DNNs) have become the most important and powerful intelligence-enabled technique. Thanks to the availability of massive amount of training data and much stronger computing power than before, DNNs models can now be well trained with very deep and wide architecture, and thereby being able to achieve unprecedented high classification accuracy and/or prediction quality in various practical application domains, such as computer vision, speech recognition, natural language processing, recommendation systems etc.

However, the current prosperity of DNNs is not free but comes from the significant increase on model sizes. Today's DNN models commonly have tens or hundreds of layers and each layer contains hundreds or thousands of neurons/filters. Such large-size architecture, consequently, causes DNNs inherently suffer high computational and storage complexity, and thereby posing severe challenge for their real-time and energy-efficient deployment, especially in the resource-constrained and latency-sensitive applications such as embedded and mobile platforms.

To address these challenges, numerous solutions from both machine learning and computing hardware communities have been proposed [1]–[25], and these prior efforts can be roughly categorized to two orthogonal directions: *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. Currently the most popular and well-investigated model compression approaches are weight pruning [26]–[28] and precision reduction [29], [30]. On the other hand, efforts on hardware acceleration aim to achieve high-performance execution of DNN models via designing DNN-specific computing platforms. These platforms, namely DNN accelerators, provide customized hardware support for the kernel computations in DNN executions, especially for inference, and hence they can achieve orders-of-magnitude improvement on hardware performance as compared with general purpose CPUs and GPUs.

Recently, PERMDNN, as a work that performs model compression and hardware acceleration simultaneously, is proposed in [31] to provide an algorithm/hardware co-design solution for high-performance DNN model execution. By imposing *permutation diagonal structure* on the DNN models, PERMDNN develops low-complexity forward and backward propagation schemes for both permuted diagonal structure-based fully-connected (FC) layers and convolutional (CONV) layers, thereby ensuring that such structured DNN models can be trained and executed at the algorithm level. Then, PERMDNN develops a corresponding hardware engine that specifically supports the inference task of permuted diagonal structured DNN models. The strong structure in DNN models is fully leveraged in this hardware architecture to reduce indexing overhead, and hence such deep integration between algorithm and hardware brings very high hardware performance improvement over the state-of-the-art DNN hardware designs.

• *Chunhua Deng, Siyu Liao, and Bo Yuan are with the Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ, 08854.*
E-mail: bo.yuan@soe.rutgers.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981068, IEEE Transactions on Computers

Despite its promising performance, there exists an important missing in PERMDNN framework. The hardware architecture developed in [31] is specifically designed for accelerating the inference on FC layers and it is not well-suited for CONV layers. As illustrated in Fig. 1, the permuted diagonal structure has different meanings in FC layer and CONV layer cases. In FC layer such structure is imposed on the 2D weight matrix, while in the CONV layer it is imposed on the 4D weight tensor. Consequently, the original matrix-vector multiplication-based data processing scheme in PERMDNN hardware engine cannot be used for permuted diagonal structured CONV layer that uses sliding window-based 2D convolution as kernel function. Given the very widespread adoption of CONV layer-contained convolutional neural networks (CNNs) in many real-world AI tasks, such absence of CONV layer-suited hardware architecture will significantly hinder and limit the practical applications of PERMDNN framework.

To fill this gap and deliver the promise of permuted diagonal structured CNN models, this paper proposes PERMCNN, an energy-efficient CONV layer-targeted hardware architecture with permuted diagonal structure. By fully utilizing the strong structured sparsity in the trained models as well as dedicatedly leveraging the dynamic activation sparsity, PERMCNN delivers very high hardware performance for inference tasks on CNN models. A design example with 28nm CMOS technology shows that, compared to the state-of-the-art CNN accelerator, PermCNN achieves 3.74× and 3.11× improvement on area and energy efficiency, respectively, on AlexNet workload, and 17.49× and 14.22× improvement on area and energy efficiency, respectively, on VGG model. After including energy consumption incurred by DRAM access, PermCNN achieves 2.60× and 9.62× overall energy consumption improvement on AlexNet and VGG workloads, respectively.

The rest of this paper is organized as follows. Section 2 introduces the basics of CONV layer, the existing DNN hardware accelerators, and the motivation of our work. The general forward and backward propagation schemes on permuted diagonal structure-imposed CONV layer are briefly reviewed in Section 3. Section 4 describes the hardware architecture of the proposed PERMCNN in detail. The implementation and evaluation results of PERMCNN are presented in Section 5. Section 6 draws the conclusion.

2 Background and Motivation

2.1 CONV Layer

In general, CONV layer is the most important component of a CNN model since it consumes the largest portion of the overall computation. Mathematically, when the inference is performed in a batch way, the computation of a CONV layer is essentially the 2D convolution between a 4D input tensor (a batch of 3D inputs) and 4D weight tensor (a group of 3D filters), and the output of CONV layer is also a 4D tensor. The detailed computation procedure is described as follows:

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

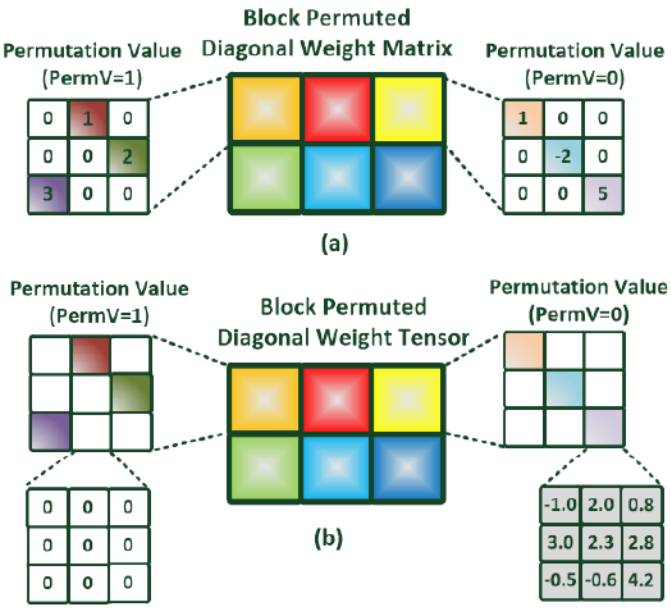

Fig. 1: Example of imposing block-diagonal permuted structure on (a) weight matrix of FC layer and (b) weight tensor of CONV layer. Here the block size p, as the compression ratio, is 3. The entire weight matrix or tensor contains blocks of component weight matrices or tensors, and each component permuted diagonal matrix or tensor is affiliated with a permutation value (PermV). PermV is selected from 0,1,...,p-1. The non-zero 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.

TABLE 1: The CNN parameters.

Parameter	Description
M	Number of 3D filters
C	Number of channels
H/W	Height/width of input
E/F	Height/width of output
R/S	Filter kernel height/width
N	Batch size of 3D input activations
U	Stride

where $\mathcal{X} \in \mathbb{R}^{N\times C\times H\times W}$, $\mathcal{Y} \in \mathbb{R}^{N\times M\times E\times F}$, $\mathcal{W} \in \mathbb{R}^{M\times C\times R\times S}$ are input tensor, output tensor, and weight tensor, respectively. And the meaning of parameters M, C, H/W, E/F, R/S, N and U are described in Table 1.

2.2 Related Work

To date the high-performance DNN hardware accelerators have been extensively investigated in prior work. The pioneering effort in this field is Diannao family [20]–[24], which includes a set of designs ranging from multi-machine configuration to embedded solution. In [32]–[35], efficient dataflows are studied and developed to improve the hardware performance of DNN accelerators. Together with different types of model compression approaches, especially weight pruning, [31], [36]–[41] propose several compressed model-oriented DNN inference hardware engine. Besides, utilizing new memory technology to address data movement bottleneck is another popular solution, and the related efforts using this strategy are reported in [7]–[10], [42], [43].

2.3 Motivation

Importance of Sparsity in DNN Accelerators. Among various existing DNN hardware designs, sparsity-aware architecture is particularly attractive and has gained a lot of attention from both academia and industry [36], [37], [44], [45] (as shown in Table 2). In general, the existing popularity on sparse DNNs is due to two reasons. First, weight pruning, as a widely adopted model compression approach, inherently brings high sparsity in DNN models. Second, activation sparsity, as the natural outcome after using rectified linear unit (ReLU) layer in neural network architecture, widely exists in different types of DNN models. Consequently, this co-existence of model sparsity and activation sparsity significantly enhances the importance and effectiveness of designing sparsity-aware DNN hardware accelerators.

Current Challenges. Despite its current prosperity, the research on sparsity-aware DNN hardware accelerators is still facing unsolved underlying challenges. For instance, part of existing work [37], [45] only exploit the utilization of activation sparsity while ignoring model sparsity. The all-sparsity utilization is studied in [36], but it suffers low multiplier utilization problem due to the inherent characteristics of its dataflow. The hardware architecture in [44] can efficiently deal with both model sparsity and activation sparsity; however, it requires extra indexing overhead and only targets for FC layers.

Recently, PERMDNN [31] is proposed to address the unstructured model sparsity problem in prior work with still fully utilizing two types of sparsity. Evaluation results show that PERMDNN can achieve both high task performance on different datasets and high hardware performance in terms of different metrics (throughput, area efficiency and energy efficiency). Unfortunately, similar to [44], the hardware architecture of PERMDNN is only designed for FC layers, thereby limiting its applications in CONV layer-centric scenarios.

Block-level sparsity, such as [46], is another approach to introduce structured sparsity to DNNs. However, similar to PERMDNN the state-of-the-art block-level sparsity work are mainly introduced to compress fully-connected neural network or recurrent neural network. Hence the corresponding block-level sparsity hardware architecture can only be used for accelerating FC layer instead of CONV layer.

Motivation of This Work. Notice that at the algorithm level the permuted diagonal structure-imposing approach, as the key idea in PERMDNN, already contains the low-complexity forward and backward propagation schemes for CONV layers with high sparsity ratio and negligible accuracy loss. Therefore, based on this already verified theoretical outcome and encouraged by the high performance demonstrated by FC layer-targeted PERMDNN hardware architecture, this paper aims to develop a CONV layer-targeted hardware architecture, namely PERMCNN, to deliver the promising advantages of imposing permuted diagonal structure on CONV layers. Such effort, if successful in fully utilizing the structured model sparsity as well as dynamic activation sparsity, would achieve significant hardware performance improvement over the existing CNN accelerators.

TABLE 2: Some existing sparsity-aware DNN accelerators. Here WS means weight sparsity, and IAS means input activation sparsity.

Accelerator	WS	IAS	DRAM Access Information Revealed	Target Layer
EIE [44]	Y	Y	N	FC
Cnvlutin [37]	N	N	N	CONV
Eyeriss [45]	N	Y	Y	CONV
SCNN [36]	Y	Y	N	CONV
PERMDNN [31]	Y	Y	N	FC
PERMCNN	Y	Y	Y	CONV

3 PERMUTED DIAGONAL ON CONV LAYER

In this section we briefly review the forward and backward propagation schemes on permuted diagonal structure-imposed CONV layers. More algorithm-level details, including classification accuracy on different datasets, can be referred to [31].

Forward Propagation. As illustrated in Fig. 1, 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. Therefore, we only need to store non-zero kernels $\mathcal{W}' \in \mathbb{R}^{\frac{M\times C}{p}\times R\times S}$, and the mapping between original $\mathcal{W}$ and non-zero kernels $\mathcal{W}'$ is defined as following:

$$\mathcal{W}(m,c,r,s) = \begin{cases} \mathcal{W}'(l\times p+i,r,s) & (i+k_l)\equiv c \mod p \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where p is the block size, $i \equiv m \mod p$, $l = \lfloor \frac{m}{p} \rfloor \times \frac{C}{p} + \lfloor \frac{c}{p} \rfloor$. The k_l is the permuted offset for diagonals, also denoted as PermV. Eqn. 2 describes the indexing design of the PERMCNN. Each block only takes p kernels. The $l \times p$ indicates the offset of current block and i stands for current kernel within the block. The modulo condition is required for the permuted diagonal. Based on this mapping, the convolution is only performed for non-zero kernels. Overall, the forward propagation can be summarized as:

$$\mathcal{Y}(n,m,e,f) = \sum_{g=0}^{\frac{C}{p}-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1} \mathcal{X}(n, gp+(i+k_l \mod p), Ue+r, Uf+s) \times \mathcal{W}'(k_l\times p+i,r,s). \tag{3}$$

Backward Propagation. Besides forward propagation, the corresponding backward propagation is also developed in [31] to ensure the trained CONV layer exhibits permuted diagonal structure as follows:

$$\frac{\partial J}{\mathcal{W}(m,c,r,s)} = \sum_{n=0}^{N-1}\sum_{e=0}^{E-1}\sum_{f=0}^{F-1} \mathcal{X}(n,c,Ue+r,Uf+s) \times \frac{\partial J}{\partial \mathcal{Y}(n,m,e,f)}, \forall\, \mathcal{W}(m,c,r,s) \neq 0 \tag{4}$$

TABLE 3: Model compression results [31] on AlexNet, VGG-16 over ImageNet [47] and ResNet on CIFAR-10 [48].

Model	Dataset	p	Acc.	CR of CONV layers
ResNet-20	CIFAR-10	1	91.25%	1.09MB (1×)
ResNet-20	CIFAR-10	2	90.85%	0.70MB (1.55×)
Wide ResNet-48	CIFAR-10	1	95.14%	190.2MB (1×)
Wide ResNet-48	CIFAR-10	4	94.92%	61.9MB (3.07×)
AlexNet	AlexNet	1	80.2%	8.9MB (1×)
AlexNet	AlexNet	4	79.85%	2.3MB (3.83×)
VGG-16	AlexNet	1	88.7%	56.12MB(1×)
VGG-16	AlexNet	4	88.27%	14.03MB(3.99×)

$$\frac{\partial J}{\partial \mathcal{X}(n,c,x,y)} = \sum_{m=1}^{M-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1} \mathcal{W}(n,m,r,s) \times \frac{\partial J}{\partial \mathcal{Y}(n,m,,(x-r)/U,(y-s)/U)}. \quad (5)$$

Notice that here Eqn. 5 is used to aid the gradient computation described in Eqn. 4 since $\mathcal{X}(n,c,x,y)$ in the current layer is the output of previous layer as $\mathcal{Y}(n,m,e,f)$.

Table 3 lists the model compression results over the AlexNet, VGG, ResNet and Wide ResNet models. The second column is the block size p setting and the last column is the compression ratio (CR) over the corresponding convolution layers. More detailed algorithm-level experimental configurations can be found in our prior PERMDNN work [31]. It can be noted that permuted diagonal structure can achieve high sparsity and with negligible accuracy drop.

4 PERMCNN ARCHITECTURE

In this section, we develop the hardware architecture of PERMCNN for the permuted diagonal structure-imposed CONV layers. Specifically, we will describe data processing scheme, key component units, critical inter-unit routing in detail. Notice that similar to most of the existing CNN hardware accelerators, PERMCNN architecture is designed for inference tasks.

4.1 Data Processing Scheme

In general, PERMCNN adopts a partial-parallel data processing scheme that utilizes multiple processing elements (PEs) to perform 2D convolution between a batch of 3D input tensors and a group of 3D weight filters. Fig. 2 shows an example data processing scheme for a $4\times2\times2\times2$ CONV weight tensor on a $1 \times 2 \times 2 \times 3$ input tensor, which is essentially the activation results of the previous layer. Here each PE is in charge of computations that belong to a specific filter, and different PEs perform independent processing on the input activations to maximize the throughput. Notice that though each filter can only be affiliated with one PE, different channels of the input activations can share the same PE. The details of such data routing scheme will be described in Section 4.5.

Specifically, the entire processing is performed on the input activations in the left-to-right and top-to-bottom direction to map the desired sliding window-style 2D convolution. As shown in Fig. 2, in each clock cycle each PE multiplies one non-zero entry in a filter kernel and non-zero entry in one channel of input activations, and then accumulate their product. After finishing the computation between the kernel and the current corresponding same-size partial input activations (e.g. 2×2 filter kernel and 2×2 part in the entire 2×3 channel of activations in Fig. 2), each PE stores the final accumulated result in the register (marked as red in Fig. 2, and then continues to "slide window" on the input activation map. Accordingly, there are multiple registers contained in each PE to store different final accumulated results.

Notice that the above described data processing scheme skips the zero entries in both filter kernels and input activations to save computation and time. For the zero-value weights existed in the filters, it is very convenient to identify and skip them because of the structured sparsity characteristics in the permuted diagonal CONV layer. On the other side for activations, as will be described in detail in Section 4.4, the dedicatedly designed APU utilizes two types of indexing information to ensure that only the non-zero values are processed and correctly paired with the corresponding non-zero weight values.

It should be noticed that the processing scheme illustrated in Fig. 2 may have the potential imbalance problem. Such imbalance problem does not result from uneven non-zero weight distribution since the inherent regularity of weight tensor in PermCNN automatically enables balanced distribution for non-zero weights across different PEs. Instead, such imbalance comes from the uneven distribution of input activation sparsity. Fortunately, such imbalanced input activation sparsity can be easily and solved by using FIFOs in APU, which will be described in detail in Section 4.4.

4.2 Overall Hardware Architecture

Based on the data processing scheme described in Section 4.1, we develop the corresponding hardware architecture of PERMCNN. Fig. 3 shows the overall architecture of the proposed computing engine. Here different from PERMDNN, PERMCNN stores both the compressed models and activation results of the component layers in the off-chip DRAM. Such arrangement is based on two reasons: 1) The compression ratios of CONV layers are typically much lower than FC layers, and hence even the compressed CNN models cannot fit to on-chip SRAM; and 2) the activation results of CONV layers, in the format of 4D tensors, can consume much higher storage costs than the 1D vector-format activations of FC layers, thereby also exceeding the capacity of SRAM.

Next we describe the dataflow in the proposed architecture. After the activation values are read from DRAM, they are first processed by sparse-to-dense conversion unit (S2DCU) to be converted back to dense format (see Fig. 4). This is because though the sparsity format (e.g. compressed sparse row (CSR) [49]) of activations can help save memory cost in the off-chip DRAM, in the on-chip processing the original dense format is still required to identify the original positions of non-zero values in the activations, thereby ensuring functional validity in the later computations. Then, these dense-format activation values are sent to activation processing units (APUs) to prepare the desired non-zero values and index information for the next-stage convolution operation. Notice that here the extracted values and indices,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981068, IEEE Transactions on Computers

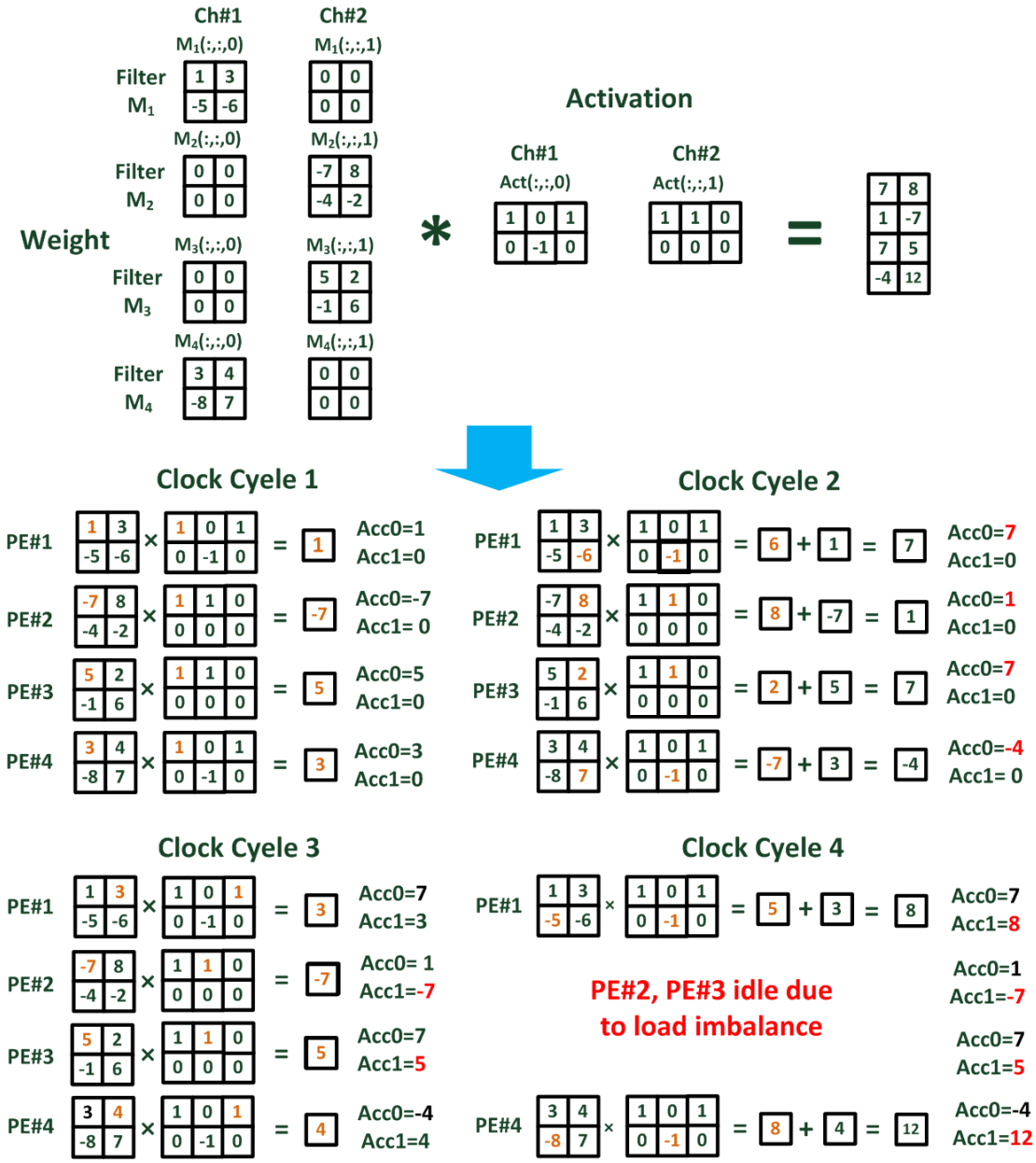

Fig. 2: Example data processing scheme. Here filter kernel size is 2×2.

which are used to ensure the valid computation of 2D convolution, are different from those used in the sparse format in DRAM. Then, a crossbar module is used to route the non-zero values and index information to the corresponding PEs, and the PEs are in charge of 2D convolutions between different filter kernels and different channels of input activations. Notice that the weights of filter kernels are first read from off-chip DRAM, and then they are stored in the SRAMs in PEs for local reuse. After all the PEs finish their computations, the final results will be converted back to sparse format via dense-to-sparse conversion unit (D2SCU) to save the memory costs in DRAM.

4.3 Sparse/Dense Conversion Units

As described before, the interface between PERMCNN computing engine and DRAM are S2DCU and D2SCU modules. Here the sparse encoding format used in this two types of conversion units are compressed sparse row (CSR), which is the same data representation adopted in [45]. Specifically, the S2DCU reads one channel of input activations from DRAM in a row-wise way and then converts it to dense format. Similarly, the D2SCU compresses the dense-format results of PEs to the sparse format and sends them back to DRAM.

4.4 Activation Processing Unit

As mentioned in Section 4.2, APU is used to extract non-zero values and their index information for the later 2D convolution. Specifically, there are three types of information that are generated by APU: convolution ID (CID), position ID (PID) and extended activation vector (ACT). Fig. 4 illustrates the function of APU on a 3×3 input activations. After receiving the converted dense-format activation from S2DCU, APU identifies each non-zero value that is needed in the 2D

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981068, IEEE Transactions on Computers

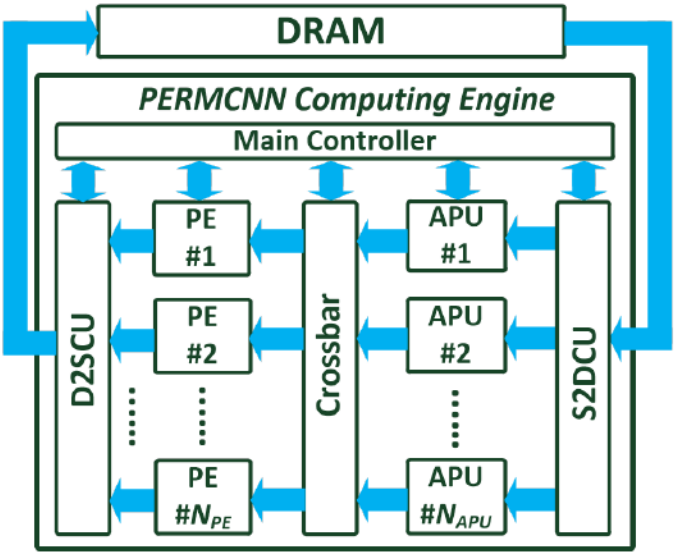

Fig. 3: Overall architecture of PERMCNN hardware.

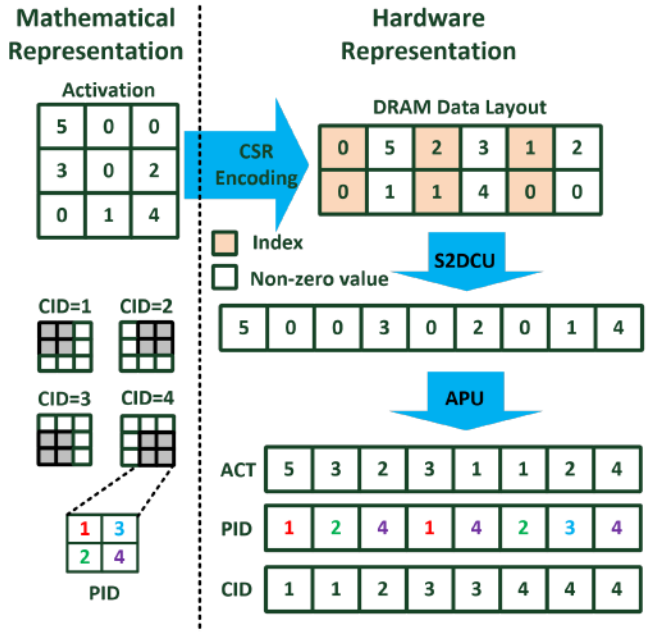

Fig. 4: Data processing example in S2DCU and APU. Here activation map is 3×3, and filer kernel size is 2×2.

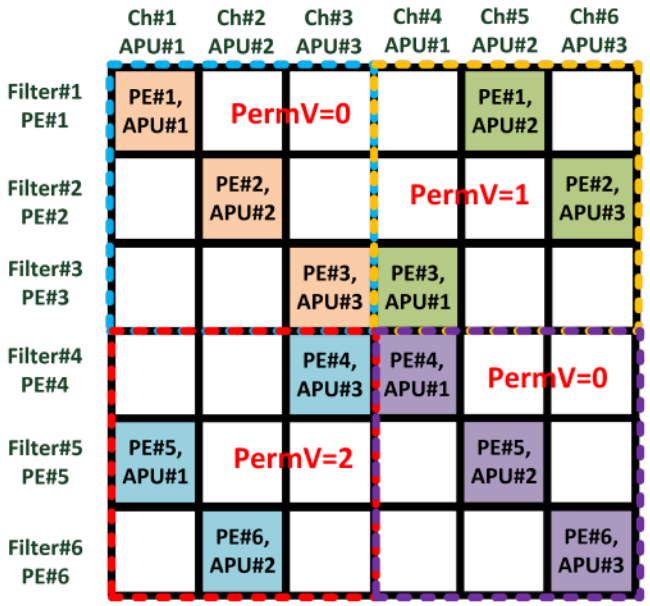

Fig. 5: An example of interconnection between APUs and PEs with block size p=3. A crossbar module is in charge of realizing such interconnection.

convolution and sequences them. Notice that there exists data repeat in the sequence due to the fact that the filter kernel processes the same activation value in the different dot product-format computation steps when it slides over the activation map. Accordingly, in order to ensure that each activation value in the ACT sequence, no matter it has repeat or not, can always be used for the correct computation step in the 2D convolution, each activation value in the ACT is affiliated its individual CID and PID. Here CID indicates the specific computation step when the current activation value will be used, and PID indicates the correct position where the activation value should be placed in the dot product computation step. Notice that the index principles for computation step and the positions are pre-determined as illustrated in Fig. 4. After all the entries in ACT sequence and their corresponding PIDs and CIDs have been generated, all of them will be buffered in a FIFO. The reason for using this FIFO is to reduce load imbalance problem incurred by uneven input activation sparsity distribution. In the worst case scenario, some PEs will receive N_{acc} zero activation values; while some other PEs will receive N_{acc} non-zero activation values for the current computation steps. Accordingly, in order to avoid stalling PEs in this worst case scenario, the FIFO depth in the PE of PERMCNN is set as N_{acc} to ensure the continuous processing of each PE. Please notice that such strategy is similar to the one used in EIE [44] that targets sparse fully-connected layer.

4.5 Crossbar Module

To ensure each PE receives its desired activation values from APUs, a dedicatedly designed crossbar module, as the underlying hardware to realize such interconnection, is needed in the PERMCNN architecture. Fig. 5 shows an example interconnection among multiple PEs and APUs for a block-permuted diagonal weight tensor with block size $p = 3$. Here different PEs process different filters of weight model and different APUs provide different channels of activations to PEs. Notice that within each component diagonal permuted tensor, the specific PermV is a very important parameter to help pair the correct PE and APU by utilizing the inherent structure existed in the permuted diagonal model. Algorithm 1 describes our proposed routing scheme in detail. The key idea here is to utilize the information of PermV in different component permuted diagonal tensors to quickly identify the non-zero filters via modulo operation, and thereby paring the correct non-zero filter kernels and activations. Consider the modulo operation can be easily implemented by comparator and adder, and all the other routing operations are simply based on multiplexers, the overall hardware cost of crossbar module is very low.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981068, IEEE Transactions on Computers

Algorithm 1: Crossbar Routing Scheme

Input : M, C, N_{PE}, N_{APU}, compression ratio p, 1-D permutation value array PermVArray with size B × D, where B=$\lceil \frac{M}{p} \rceil$, D=$\lceil \frac{C}{p} \rceil$

Output: PE-APU relation

for $j = 0$ **to** *D-1* **do**
  for $i = 0$ **to** *M-1* **do**
    //Calculate address of PermVArray for PE i
    $BlockRowBase = \lfloor \frac{i}{p} \rfloor$
    $BlockRowOffset = i \,\%\, p$
    $Address = BlockRowBase \times \mathrm{D} + j$
    $PermV_{ij}$ = PermVArray[$Address$]
    $APU_{id} = (PermV_{ij} + BlockRowOffset) \,\%\, p$
    $Channel_{id} = j \times p + APU_{id}$
    When processing channel $Channel_{id}$, connect PE # $(i+1)$ with APU # (APU_{id}+1).

4.6 Processing Element

After receiving the desired weight values and activation values, an array of PEs performs parallel multiply-and-accumulation processing to realize 2D convolution. Fig. 8 shows the inner architecture of one PE. Here the weight values are first read from off-chip DRAM and stored in the PE's own SRAM for data reuse. Notice that in order to ensure the next-stage multiplication in the 2D convolution is performed on the correct pair of weight and activation values, the layout of weight in the on-chip SRAM needs to follow a pre-defined principle. Fig. 7 shows an example of this data layout principle. Here for the weight filters belonging to the same PE, they should be accessed in the left-to-right and top-to-bottom order over the entire block-permuted diagonal weight tensor, and then stored into the SRAM one by one. In addition, within the same filter, the different weight values should be stored using the order determined by the pre-defined PID index principle. For instance, as shown in Fig. 7 for 2 × 2-size kernel the value with PID index as 1 should be stored first while the value with PID index as 4 should be stored last. Notice that here the pre-defined PID index principle is exactly the same to that is used in APU (see Fig. 4).

During the process of 2D convolution, each PE first loads the weight values from its SRAM to registers, and then the proper weight values, with the help of PID index sent from APU, is selected to be multiplied with the corresponding activation values. Fig. 8 illustrates the computing procedure of one PE in detail. Here once receiving the PID, CID and ACT information generated by APU, the PE utilizes PID information to select the corresponding registers. Notice that the data layout in PE's weight SRAM (see Fig. 7) and PID sequence in APU's processing (see Fig. 4) share the same PID index principle. This arrangement ensures that the element-wise multiplication between weight values and activation values are always performed on the correct pairs. Then, using CID information, which indicates the current multiplication should belong to which dot product in the overall 2D convolution, PE accumulates the products of the multiplication to calculate the final results of 2D convolution.

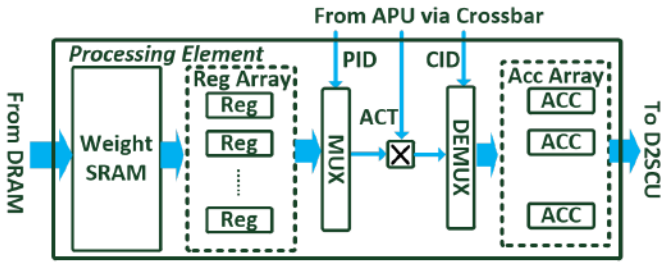

Fig. 6: Inner structure of PE.

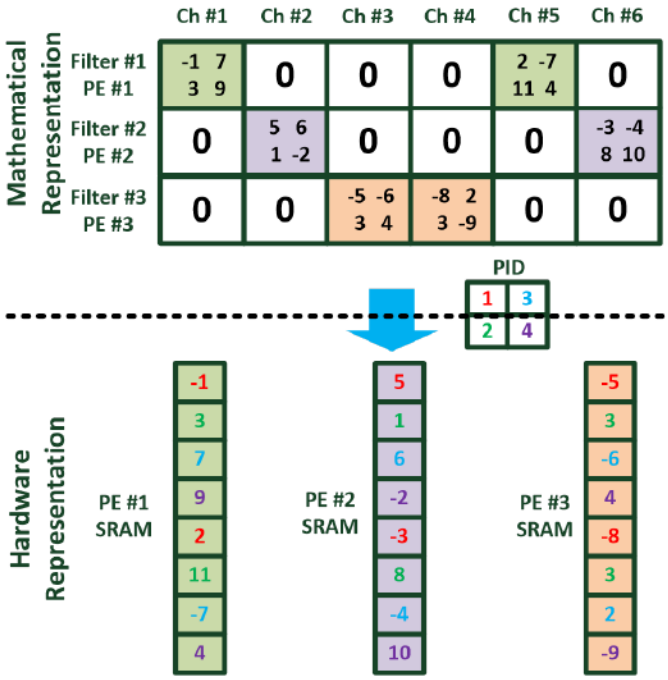

Fig. 7: Example of data layout in PE's weight SRAM.

5 Evaluation

5.1 Experimental Methodology

Simulation and CAD Tools. We first develop a bit-accurate cycle-accurate software model to simulate the behavior of PermCNN hardware architecture. Then, we develop RTL model using Verilog HDL to implement the proposed architecture. This RTL model is synthesized with 28nm CMOS technology via using Synopsys Design Compiler. Then we use Synopsys IC Compiler to perform place and route (see Fig. 9). To estimate power consumption, we use Synopsys Prime-Time PX with considering the switch activity and toggle rate extracted from simulation.

5.2 Design Configuration and Hardware Performance

Design Configuration. In general, the hardware resource in the PermCNN, such as the number of PEs and APUs can be very flexible for different applications. Similarly, the configuration for each PE, such as the number of accumulators and number precision, can also be very flexible. Table 4 shows the configuration parameters for our design example.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981068, IEEE Transactions on Computers

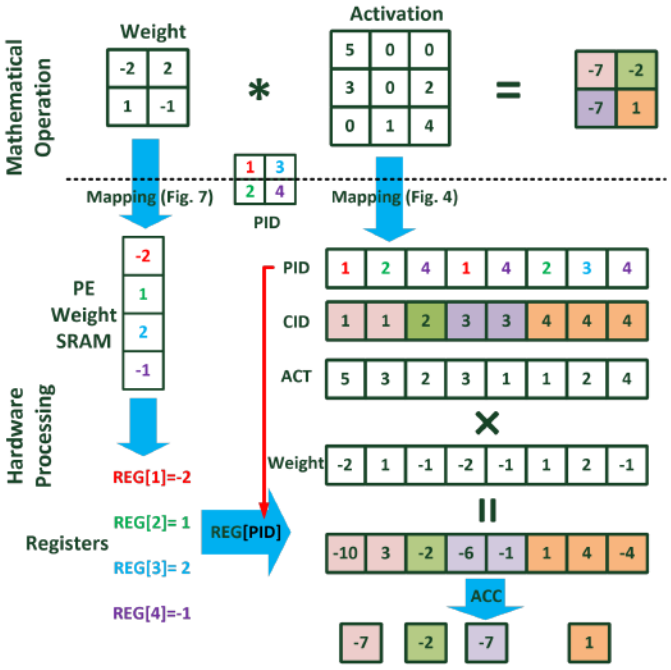

Fig. 8: Example of processing procedure of PE.

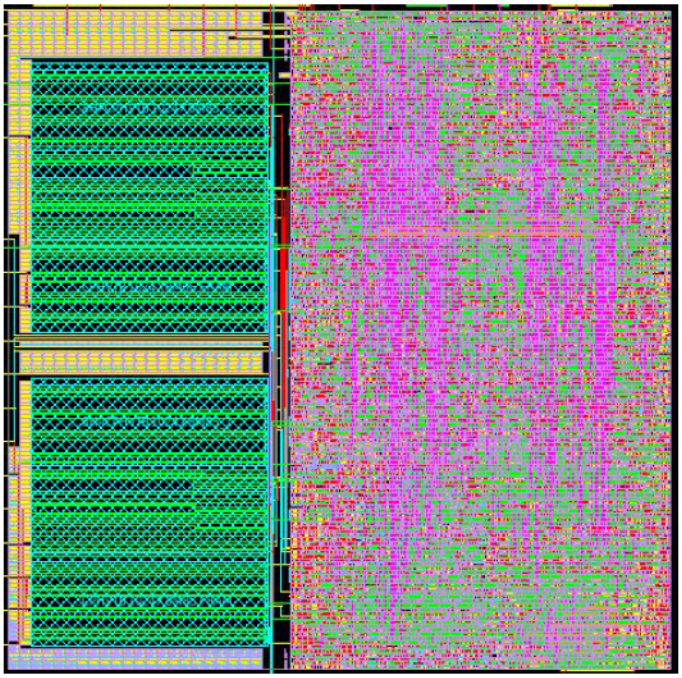

Fig. 9: Layout of one PE.

Here one PE is equipped with one 16-bit multiplier and 64 24-bit accumulators. The rationale for such configuration is to conservatively avoid overflow. Also, there is a 1152×16 weight SRAM contained in each PE. Accordingly, the total weight SRAM in the entire PERMCNN is 288KB. This size can store up to 147,456 weights in the sparse models, and it is sufficient to accommodate most CONV layers in the popular CNN models with a moderate compression ratio of 4.

Hardware Performance. According to the reports from EDA tools, the total area of the PERMCNN accelerator is 3.44 mm^2, and the power consumption is 506 mW for AlexNet [50] and 442 mW for VGG [51], respectively. Here the power consumption is averaged across all the CONV layers in these two models, and Table 5 and Table 6 show the information of these two workloads as well as the hardware performance breakdown on each layer. Also, notice that the power consumption reported by EDA tools refers to on-chip computing engine part while excluding the consumption incurred by off-chip DRAM part. In next subsection, the contribution from DRAM access will be considered and included in the overall system evaluation.

TABLE 4: Design configuration parameters.

PE Parameter		Value
Multiplier	Amount (N_{MUL})	1
	Width	16 bits
Accumulator	Amount (N_{ACC})	64
	Width	24 bits
Weigth SRAM	Amount	1
	Width	16 bits
	Depth	1152
PERMCNN Computing Engine Parameter		Value
Amount of PEs (N_{PE})		128
Quantization scheme		16 bits
Number of APUs (N_{APU})		4

TABLE 5: Performance breakdown of 5 CONV layers in AlexNet. Batch size is 4. Here CR means compression ratio (block size), and IAS means input activation sparsity.

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

5.3 Comparison with Eyeriss

Rationale. In this subsection, we compare our proposed PERMCNN with the seminal CNN accelerator Eyeriss [45]. It should be noticed that though there have been plenty of

TABLE 6: Performance breakdown of 13 CONV layers in VGG. Batch size is 3. Here CR means compression ratio (block size), and IAS means input activation sparsity.

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

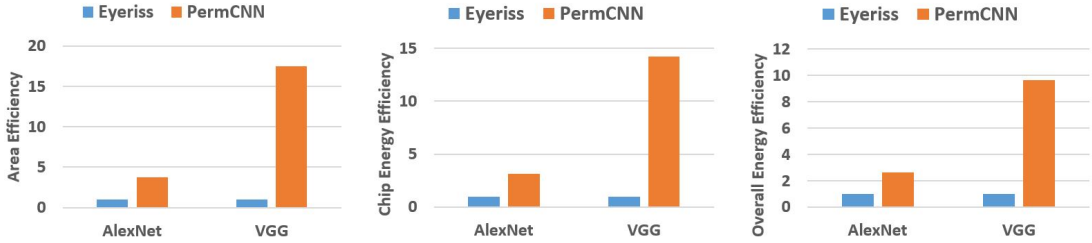

Fig. 10: Hardware performance comparison between Eyeriss and PERMCNN.

published work on CNN accelerators, Eyeriss is the most suitable design for a fair and comprehensive comparison because of the three reasons: 1) it develops a novel row stationary dataflow, thereby representing the state-of-the-art at architecture level; 2) it reveals very detailed layer-wise hardware performance on two practical workloads (AlexNet and VGG); and 3) most importantly, unlike most of the existing work, Eyeriss reports its performance including the energy consumption incurred by DRAM access (see Table 2), which in practice occupies a significant portion of the overall energy consumption of the entire CNN accelerator system. Based on these considerations, we adopt the same workloads and energy modeling approach used in Eyeriss to perform a fair and comprehensive hardware performance comparison between PERMCNN and Eyeriss.

Performance Comparison. Table 7 summarizes the hardware performance metrics of PERMCNN and Eyeriss. Considering the different technology nodes used in these two work, here we project the Eyeriss to 28nm technology following the project principle in [44]: frequency, area and power are scaled as linear, quadratic and constant way, respectively. In order to perform the comprehensive evaluation on the overall energy consumption, DRAM access, including the operation of reading weight and activation values from DRAM as well as writing activation values to DRAM, is listed in the table as well. And the overall power is the addition of DRAM power and the chip power. Notice that the DRAM access energy we used in the evaluation is 21 pJ/bit, which was reported for an LPDDR3 model [52].

From Table 7 it is seen that, PERMCNN achieves 3.74× and 3.11× improvement over Eyeriss in terms of area efficiency and energy efficiency, respectively, on the workload of 5 CONV layers of Alexnet model. On the workload of 13 CONV layers of VGG model, PERMCNN achieves 17.49× area efficiency improvement and 14.22× energy efficiency improvement. Moreover, the desired DRAM access for PERMCNN is reduced by 2.0× and 3.16× than Eyeriss for AlexNet and VGG workloads, respectively. Consequently, for the overall system energy efficiency, which includes both on-chip computing engine part and off-chip DRAM part, PERMCNN achieves 2.6× and 9.62× improvement on AlexNet and VGG workloads, respectively. Fig. 10 summarizes the comparison result between Eyeriss and PERMCNN in terms of area efficiency, chip energy efficiency, and overall energy efficiency. It is seen that that PERMCNN provides significant performance improvement with respect to these hardware performance metrics.

TABLE 7: Comparisons of PERMCNN and Eyeriss. AXN stands for AlexNet.

Design	Eyeriss		PERMCNN
CMOS Tech.	65nm (reported)	28nm (projected)	28nm
Frequency (MHz)	200	464	800
Area (mm^2)	12.25	2.27	3.44
Chip Power (mW)	278 (AXN) 236 (VGG)	278 (AXN) 236 (VGG)	506 (AXN) 442 (VGG)
Overall Power (mW)	300 (AXN) 249 (VGG)	300 (AXN) 249 (VGG)	654 (AXN) 689 (VGG)
Processing Time (ms)	28.83 (AXN) 1437 (VGG)	12.42 (AXN) 619 (VGG)	2.19 (AXN) 23.09 (VGG)
Throughput (frame/s)	34.7 (AXN) 0.70 (VGG)	80.55 (AXN) 1.625 (VGG)	456.6 (AXN) 43.31 (VGG)
Area Efficiency (frame/s/mm^2)	2.83 (AXN) 0.057 (VGG)	35.48 (AXN) 0.72 (VGG)	132.73(AXN) 12.59(VGG)
Chip Energy Efficiency (frame/J)	124.8 (AXN) 2.97 (VGG)	289.7 (AXN) 6.89 (VGG)	902.4 (AXN) 97.99(VGG)
DRAM Access (MB)	3.85 (AXN) 107.0 (VGG)	3.85 (AXN) 107.0 (VGG)	1.93 (AXN) 33.87(VGG)
Overall Energy Efficiency (frame/J)	114.5 (AXN) 2.80 (VGG)	268.5 (AXN) 6.53 (VGG)	698.2 (AXN) 62.86(VGG)

6 CONCLUSION

This paper proposes PERMCNN, a CNN hardware architecture to execute the hardware-friendly permuted diagonal structured CNN models. PERMCNN can fully leverage the benefits provided by the structured models to deliver very high hardware performance. Evaluation results show that, compared with the state-of-the-art CNN accelerator, PERMCNN achieves 3.74×, 3.11× and 2.60× improvement on area efficiency, on-chip energy efficiency and overall energy efficiency, respectively, when executing AlexNet models. And it achieves 17.49×, 14.22× and 9.62× improvement on area efficiency, on-chip energy efficiency and overall energy efficiency, respectively, when executing VGG model.

REFERENCES

[1] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016.

[2] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017.

[3] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "Sc-dcnn: highly-scalable deep convolutional neural network using stochastic computing," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017.

[4] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018.

[5] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *arXiv preprint arXiv:1805.03718*, 2018.

[6] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. W. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," *arXiv preprint arXiv:1804.06508*, 2018.

[7] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016.

[8] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017.

[9] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017.

[10] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016.

[11] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh, "Tabla: A unified template-based framework for accelerating statistical machine learning," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016.

[12] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing dma engine: Leveraging activation sparsity for training deep neural networks," in *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 2018.

[13] M. Song, J. Zhang, H. Chen, and T. Li, "Towards efficient microarchitectural design for accelerating unsupervised gan-based deep learning," in *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 2018.

[14] C. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp : Adaptive residual gradient compression for data-parallel distributed training," *CoRR*, vol. abs/1712.02679, 2017. [Online]. Available: http://arxiv.org/abs/1712.02679

[15] N. Wang, J. Choi, D. Brand, C. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," *CoRR*, vol. abs/1812.08011, 2018. [Online]. Available: http://arxiv.org/abs/1812.08011

[16] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghunathan, "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017.

[17] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "ESE: efficient speech recognition engine with compressed LSTM on FPGA," *CoRR*, vol. abs/1612.00694, 2016. [Online]. Available: http://arxiv.org/abs/1612.00694

[18] S. Wang, Z. Li, C. Ding, B. Yuan, Y. Wang, Q. Qiu, and Y. Liang, "C-LSTM: enabling efficient LSTM using structured compression techniques on fpgas," *CoRR*, vol. abs/1803.06305, 2018. [Online]. Available: http://arxiv.org/abs/1803.06305

[19] S. Liao, A. Samiee, C. Deng, Y. Bai, and B. Yuan, "Compressing deep neural networks using toeplitz matrix: Algorithm design and fpga implementation," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019.

[20] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM Sigplan Notices*, vol. 49, 2014.

[21] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014.

[22] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015.

[23] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1. ACM, 2015.

[24] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016.

[25] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "Tie: Energy-efficient tensor train-based inference engine for deep neural network," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: ACM, 2019. [Online]. Available: http://doi.acm.org/10.1145/3307650.3322258

[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[27] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," *CoRR*, vol. abs/1804.03294, 2018. [Online]. Available: http://arxiv.org/abs/1804.03294

[28] T. Yang, Y. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *CoRR*, vol. abs/1611.05128, 2016. [Online]. Available: http://arxiv.org/abs/1611.05128

[29] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[30] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016. [Online]. Available: http://arxiv.org/abs/1603.05279

[31] C. Deng, S. Liao, Y. Xie, K. K. Parhi, X. Qian, and B. Yuan, "Permdnn: Efficient compressed dnn architecture with permuted diagonal matrices," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018.

[32] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017.

[33] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016.

[34] A. S. Hyoukjun Kwon and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[35] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016.

[36] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017.

[37] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016.

[38] P. Hill, A. Jain, M. Hill, B. Zamirai, C.-H. Hsu, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars, "Deftnn: addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission," in *Proceedings of the 50th Annual*

IEEE/ACM International Symposium on Microarchitecture. ACM, 2017.

[39] C.-E. Lee, Y. S. Shao, J.-F. Zhang, A. Parashar, J. Emer, S. W. Keckler, and Z. Zhang, "Stitch-x: An accelerator architecture for exploiting unstructured sparsity in deep neural networks."

[40] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016.

[41] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017.

[42] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016.

[43] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "Rana: Towards efficient neural acceleration with refresh-optimized embedded dram," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018.

[44] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016.

[45] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, 2016.

[46] S. Narang, E. Undersander, and G. Diamos, "Block-sparse recurrent neural networks," *arXiv preprint arXiv:1711.02782*, 2017.

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009.

[48] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[49] R. W. Vuduc, "Automatic performance tuning of sparse matrix kernels," Ph.D. dissertation, 2003, aAI3121741.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[52] M. Schaffner, F. K. Gürkaynak, A. Smolic, and L. Benini, "Dram or no-dram? exploring linear solver architectures for image domain warping in 28 nm cmos," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015.

Chunhua Deng received his Bachelor degree from China University of Petroleum in 2005, and his Master degree from Beijing Institute of Technology in 2007. From 2007 to 2017, he worked at ZTE Corportation as a senior IC design engineer. He is currently pursuing his PhD degree in the Department of Electrical and Computer Engineering, Rutgers University. His research interests include machine learning, VLSI design.

Siyu Liao received the BE degree in information security from the Hefei University of Technology, Hefei, China, in 2014. He is now working toward the PhD degree in the Department of Electrical & Computer Engineering at the Rutgers, the State University of New Jersey. His research interests include machine learning and high performance computing.

Bo Yuan received his bachelor and master degrees from Nanjing University, China in 2007 and 2010, respectively. He received his PhD degree from Department of Electrical and Computer Engineering at University of Minnesota, Twin Cities in 2015. His research interests include algorithm and hardware co-design and implementation for machine learning and signal processing systems, error-resilient low-cost computing techniques for embedded and IoT systems and machine learning for domain-specific applications. He is the recipient of Global Research Competition Finalist Award in Broadcom Corporation and Doctoral Dissertation Fellowship in University of Minnesota. Dr. Yuan serves as technical committee track chair and technical committee member for several IEEE/ACM conferences. He is the technical member for VSA and CASCOM technical committees in IEEE Circuits and Systems society and DISPS technical committee in IEEE Signal Processing society. He is the associated editor of Springer Journal of Signal Processing System.