S^2 Engine: A Novel Systolic Architecture for Sparse Convolutional Neural Networks

Jianlei Yang [ID], *Senior Member, IEEE*, Wenzhi Fu, Xingzhou Cheng [ID], Xucheng Ye, Pengcheng Dai, *Student Member, IEEE*, and Weisheng Zhao [ID], *Fellow, IEEE*

Abstract—Convolutional neural networks (CNNs) have achieved great success in performing cognitive tasks. However, execution of CNNs requires a large amount of computing resources and generates heavy memory traffic, which imposes a severe challenge on computing system design. Through optimizing parallel executions and data reuse in convolution, systolic architecture demonstrates great advantages in accelerating CNN computations. However, regular internal data transmission path in traditional systolic architecture prevents the systolic architecture from completely leveraging the benefits introduced by neural network sparsity. Deployment of fine-grained sparsity on the existing systolic architectures is greatly hindered by the incurred computational overheads. In this work, we propose S^{2}Engine – a novel systolic architecture that can fully exploit the sparsity in CNNs with maximized data reuse. S^{2}Engine transmits compressed data internally and allows each processing element to dynamically select an aligned data from the compressed dataflow in convolution. Compared to the naïve systolic array, S^{2}Engine achieves about $3.2\times$ and about $3.0\times$ improvements on speed and energy efficiency, respectively.

Index Terms—Systolic array, sparsity, convolution neural network, accelerator

---

1 INTRODUCTION

CONVOLUTIONAL neural networks (CNNs) have made remarkable success in modern artificial intelligence (AI) applications [1]. The required training data size and model complexity, however, keep increasing for better performance in a large variety of applications. The incurred high computational cost and data movement bandwidth are hardly supported by conventional computing platforms and hence, motivating recent huge investment on the corresponding accelerators [2]. Among these designs, systolic architecture [3] – a specialized processing element network designed for massive parallelization and extensive data reuse, has been proved efficient in performing CNN computations. In addition to academic research [4], [5], systolic architectures were also adopted in some industrial practices of deep neural network (DNN) accelerators [6], [7], [8], [9].

The highly regularized layout and internal data movement path of systolic arrays make engineering realization of the design very efficient. However, such a regularity also prevents exploiting irregular computation patterns that frequently appear in sparse CNNs. Sparsity of deep CNNs has been proven important to minimize computation workloads and

model size [10], [11], [12]. State-of-the-art pruning algorithms can reduce the model size by $> 10\times$ [13] and computational cost by $> 4\times$ [11] with negligible accuracy loss. However, due to the large variety of the irregularities, the sparsity is not fully exploited by the existing accelerators and introduces significant design overheads. For example, Cambricon-X only considers the weight sparsity [14] while Cnvlutin only deploys the feature sparsity [15]. Cambricon-S can fully deploy both the weight and feature sparsity but requires the sparsity pattern to be coarse-grained, which greatly limits the application scope [16]. SCNN supports fine-grained sparsity in both feature and weight but introduces significant computational overhead due to the required additional coordinates transformation [17]. SparTen [18] also utilizes both feature and weight sparsity, but the energy efficiency is significantly degraded due to the required additional logic for inner-join operations.

In this work, we propose S^2 Engine – a novel **S**ystolic architecture for **S**parse convolutional neural networks. Different from the existing systolic approaches [4], [6], compressed feature and weight flows are fed into the systolic array and the aligned pairs can be dynamically selected from the compressed dataflow by each processing element (PE). Our approach solves the contradiction between the regularity of data transmission and the irregularity of sparsity such that the sparsity in CNNs can be fully exploited. Data reuse could be efficiently implemented in S^{2}Engine by introducing an associated collective element (CE) array for the PE array to reduce external buffer access. Furthermore, fine-grained mixed-precision data processing is also supported by S^{2}Engine to satisfy the varying precision requirement of CNNs even within the same kernel [19]. Experimental results show that S^{2}Engine can achieve about $3.2\times$ speedup and about $3.0\times$ energy efficiency improvement compared with the existing systolic approaches.

● *Jianlei Yang, Wenzhi Fu, Xingzhou Cheng, and Xucheng Ye are with the School of Computer Science and Engineering, Beihang University, Beijing 100191, China. E-mail: {jianlei, xingzhou, ZY1806121}@buaa.edu.cn, minister_f@163.com.*
● *Pengcheng Dai and Weisheng Zhao are with the School of Integrated Circuits and Engineering, Beihang University, Beijing 100191, China. E-mail: dpc_work@163.com, weisheng.zhao@buaa.edu.cn.*
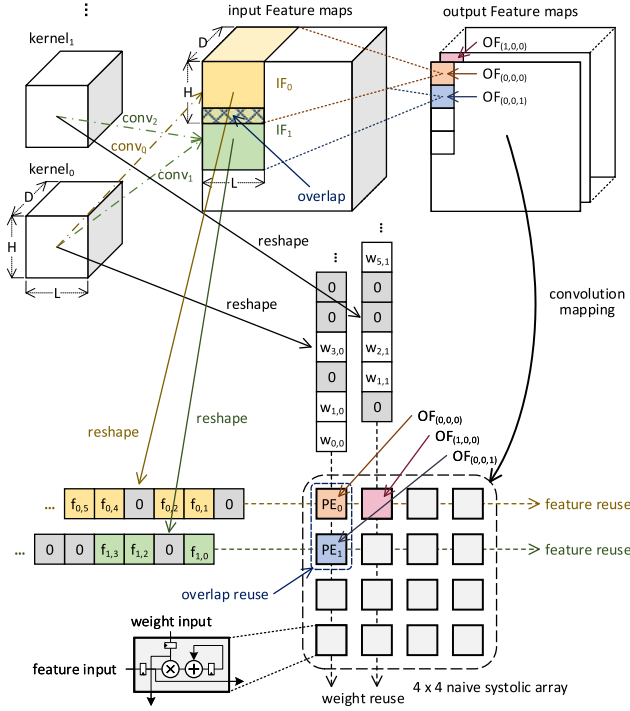
Fig. 1. Illustration of data reuse manners among convolutions.

The main contributions of our work are:

- We propose a novel systolic architecture, namely, S^2Engine. By allowing each PE to select the aligned data pairs dynamically from the compressed dataflows, S^2Engine could fully exploit the sparsity during the execution of CNNs with low overhead.
- We introduce a collective element (CE) array to further allow data reuse during convolution procedures. This technique also works for native systolic array.
- S^2Engine solves the contradiction between the regularity of data transmission and the irregularity of sparsity, showing good robustness for different sparsity degrees.

The rest of the paper is organized as follows. Section 2 provides a preliminary on CNN sparsity and a brief introduction of systolic architecture for CNN accelerators. Section 3 discusses the motivation of our design by considering data reuse manner and sparsity irregularities in CNNs. Section 4 illustrates the detailed architectures of proposed S^2Engine. Section 5 explains the experimental methodology and Section 6 presents the experimental results. Several related works are discussed in Section 7, and concluding remarks are given in Section 8.

2 PRELIMINARIES

In this section, we give preliminaries of both CNN and systolic architecture.

2.1 CNN Sparsity and Quantization

The main computation task of CNN algorithms is performing convolution operations layer by layer. As illustrated in Fig. 1, convolution procedure is carried out between different kernels and input feature maps. Take the convolution of

$conv_0$ as an example, the dimensionality of $kernel_0$ and IF_0 (input feature) could be represented as $H \times L \times D$, and $conv_0$ is defined as

$$\sum_{i=0}^{H-1} \sum_{j=0}^{L-1} \sum_{k=0}^{D-1} kernel_0[i][j][k] \times IF_0[i][j][k], \quad (1)$$

where $i \in [0, H-1]$, $j \in [0, L-1]$ and $k \in [0, D-1]$ represent the indices of three dimensions, respectively. After the convolution operations, nonlinear activation function (such as rectified linear unit, ReLU) is usually applied to obtain the output feature $OF_{(i',j',k')}$, while $i' \in [0, H'-1]$, $j' \in [0, L'-1]$ and $k' \in [0, D'-1]$ represent the indices of three dimensions $H' \times L' \times D'$ for output feature maps, respectively.

In deep CNNs, sparsity exists both in weights and features due to the inherent redundancy of the networks. Weights sparsity denotes the zeros in convolution kernels and is often obtained by various pruning algorithms. Feature sparsity denotes the zeros in feature maps and is caused by zero activation functions. Sparsity levels of weights and features are defined by the percentages of the zeros existing in kernels and feature maps, respectively. With negligible loss on accuracy, the state-of-the-art pruning algorithms [11], [13] can significantly increase the sparsity level of CNNs and lead to reduction in both model size and computation workloads. Therefore, the sparsity of CNNs greatly affects the computing efficiency of the relevant neural network accelerators. However, due to unstructured pruning and the randomness of input data, the sparsity is irregular and unpredictable, which is difficult for accelerators to leverage.

Another strategy to reduce network redundancy is called quantization, that is, using low-precision fixed-point data representation instead of using high-precision float-point one to perform inference [13], [20], [21]. This strategy can also significantly reduce the model footprint, memory access and computational overhead of the CNN while maintaining its accuracy. Many recent designs of neural network accelerators and GPUs already support 8/16-bit fixed-point data in CNN inference. In addition to the one-precision-fits-all approaches, mixed-precision quantization algorithms have been also developed by assigning different precision to different layers of CNNs according to the different sensitivities of each layer [22]. A fine-granularity quantization approach is also proposed by [19], where most data is represented with low precision (i.e., 4-bit) while only a small portion of the data (i.e., 3 percent) is represented with high precision (i.e., 16-bit). These mixed-precision approaches further reduce the CNN computation cost and memory consumption.

2.2 Systolic Architecture

Systolic array is a specialized network of homogeneous PEs that is designed for massive parallel computing in a special-purpose system. Both the structure of PEs and the communication in the systolic array keep simple and regular, offering great convenience to practical implementations. In a typical systolic array design, all the internal PEs can get their input data from the neighboring PEs and do not need to access the external memory. Therefore, systolic array becomes an efficient dataflow-driven architecture and can achieve high throughput with relatively low memory bandwidth.

TABLE 1
Average Accesses per Parameter by MACs in Various CNNs

	AlexNet [27]	VGG16 [28]	ResNet50 [29]
Total MACs	666M	15.3G	3.86G
Parameters	2.33M	14.7M	23.5M
Avg. Usage of Param.	572	2082	336

TABLE 2
Weight and Feature Sparsity of Different CNNs,
Represented by the Percentage of Zeros

	AlexNet	VGG16	ResNet50
Average Weight Sparsity	64%	68%	76%
Average Feature Sparsity	61%	72%	66%

Several previous works adopt systolic array for CNNs acceleration with different dataflows. According to the taxonomy in [23] and [24], the dataflow adopted by the accelerator in [4] belongs to output stationary. Under this dataflow, as demonstrated in Fig. 1, each PE undertakes the computation of a convolution. The convolution in the same channel are allocated to the PEs in the same row. Input features and kernels are both reshaped into one-dimensional vectors and then fed into each row and column of the systolic array, respectively. TPU [6] adopts weight stationary dataflow. The feature of each convolutional layer is loaded into the systolic array before the weight is fed into the array. Different from [4], each PE in TPU does not complete an entire convolution. Instead, the partial accumulation is transported between adjacent PEs and the convolution computation is completed by the accumulation in the PEs along the transmission path. Despite the difference in the dataflow, all of those designs retain the basic characteristics of the systolic array and obtain significant advantages of both speed and energy efficiency.

3 MOTIVATION

The efficiency of CNN accelerators is largely determined by how to exploit the network sparsity and data reuse. In systolic architecture, however, the regular structure greatly limits its capability of supporting the network sparsity and data reuse. In this section, we analyze the potential optimization space of systolic architecture from the perspectives of *data reuse* and *sparsity*.

3.1 Optimization With Data Reuse

During the computation of deep CNNs, each parameter may be accessed for many times by MAC operations, as depicted by the statistics in Table 1. Therefore, repeatedly loading these data from a separate memory (e.g., a DRAM or a global buffer) introduces excessive memory accesses. Since the energy consumption of memory accesses can be much larger than that of normal logic computations [25], [26], reducing memory accesses, e.g., by improving data reuse, can substantially reduce the energy consumption of DNN accelerators.

We can divide the data reuse strategies into three types: *weight reuse*, *feature reuse* and *overlap reuse*. As illustrated in Fig. 1, weight reuse is defined as performing convolutions between the same kernel (weights) and different input feature maps, e.g., $conv_0$ and $conv_1$ sharing the same $kernel_0$. Correspondingly, feature reuse denotes convolutions between different kernels (weights) and the same feature, such as $conv_1$ and $conv_2$ sharing the same IF_1. Overlap reuse also exists between $conv_0$ and $conv_1$, which denotes convolutions between the same kernel (weights) and the overlapped input feature maps.

As illustrated in Fig. 1, the dataflow in naïve systolic array can naturally utilize both the weight and feature reuses by fetching data from adjacent PEs instead of external buffers. However, it does not support the overlap reuse which needs to access the adjacent rows of PE array. Therefore, the insufficient exploration of overlap reuse leads to excessive accesses on external buffers and extra buffer capacity. As a supplement to the naïve design, the proposed S^2Engine can exploit overlap reuse and hence, support all three types of data reuse strategies.

3.2 Optimization With Sparsity

Sparsity widely exists in modern deep CNNs, including weight sparsity and feature sparsity. For example, the intrinsic redundancy of deep CNNs allow us to prune majority of the weights with negligible accuracy loss by retraining the models [11], [13]. Moreover, feature sparsity is introduced during inference by ReLU function that converts negative inputs to zeros. The average weight sparsity of three typical CNNs are listed in Table 2. Because feature sparsity varies with different input images, we randomly select 50000 images from ImageNet [30] and calculate their corresponding feature sparsity. Since all the MACs with zero operand(s) can be skipped without affecting the convolution result, only the MACs on aligned data pairs have to be computed and defined as must-be-performed MACs as shown in Fig. 2. The distributions of feature density and must-be-performed MAC ratios of the three CNNs are depicted in Fig. 3. According to these statistics, the sparsity of both weights and features is not trivial and provides a considerable opportunity to reduce computational cost and memory footprint.

Although the sparsity of CNNs is not trivial, its irregularity makes it difficult to effectively accelerate the executions of the sparse CNNs [14]. Fig. 2 demonstrates the fundamental challenge brought by such irregularity. To eliminate all the unnecessary MAC operations (with one or two zero operands), only the aligned data pairs need to be sent to PEs. As a result, although $conv_0$ and $conv_1$ share the same kernel, the actual weights they access in the kernel can be totally different due to the difference between the sparsity patterns of IF_0 and IF_1. Fig. 1 demonstrates that how such an irregularity brings challenge to systolic array. The weights required by PE_0 ($w_{1,0}$) and PE_1 ($w_{0,0}$ and $w_{3,0}$) for convolution are different, which breaks the data transmission path inside the systolic array. This characteristic also prevents the accelerators with explicitly planned dataflow [6], [31], [32] from fully leveraging the network sparsity. It also becomes challenging to optimize both data reuse and sparsity at the same time.

Table 3 summarizes the sparsity strategies explored by existing accelerators. For example, TPU does not explore any
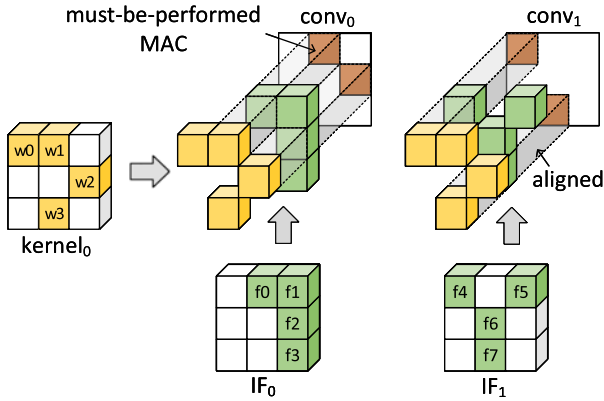
Fig. 2. A demonstration of two convolutions with sparse feature and weight. The MAC operation must be performed only when the corresponding positions of the weight and feature are both non-zero, i.e., aligned-pair (w, f).

of the two sparsity strategies in systolic arrays [6], since each zero would inevitably occupy a PE. Eyeriss only exploits the feature sparsity [31]. By utilizing feature sparsity, Cnvlutin achieves performance improvement by skipping the operations with zero elements in feature maps [15]. Cambricon-X [14] and [5], however, only exploit weight sparsity within the network trained by their proposed pruning algorithm. Cambricon-S fully deploys the sparsity of both features and weights, but only supports them at coarse granularity and requires additional pruning algorithms [16]. EIE also exploits these two types of sparsity [33] but is only designed for fully-connected networks. SCNN exploits these two types of sparsity on both convolutional and fully-connected layers [17]. However, SCNN requires lots of coordinate computations and introduces significant overhead. As an evidence, SCNN only achieves 79 percent of the speed but consumes 33 percent more energy when processing dense CNNs [17] . SparTen [18] supports sparse vector-vector multiplication and load balance optimization to improve the hardware utilization, but the required additional computation resources degrade the energy efficiency significantly.

TABLE 3
Comparison of Sparsity Considerations on
Different CNN Accelerators

Accelerator	Gate MAC	Skip MAC	Skip Buffer or DRAM Accesses
TPU [6]	Unmentioned	-	Unmentioned
Eyeriss [31]	$\mathcal{F}$	-	$\mathcal{F}$
Cnvlutin [15]	$\mathcal{F}$	$\mathcal{F}$	$\mathcal{F}$
Cambricon-X [14]	$\mathcal{W}$	$\mathcal{W}$	$\mathcal{W}$
[5]	$\mathcal{W}$	$\mathcal{W}$	$\mathcal{W}$
Cambricon-S [16]	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$
EIE [33]	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$
SCNN [17]	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$
SparTen [18]	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$
S2Engine	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$	$\mathcal{F} + \mathcal{W}$

Here $\mathcal{F}$ means only feature sparsity is considered and $\mathcal{W}$ means only weight sparsity is considered.

4 S2 ENGINE ARCHITECTURE

4.1 Architecture Overview

The top-level architecture of S2Engine and the schematic of its internal dataflow are shown in Fig. 4. S2Engine consists of a homogeneous PE array and an associated CE (collective element) array. Sparse features are compressed and stored in feature buffer (FB) while sparse weights are compressed and stored in weight buffer (WB).

As aforementioned, weight stationary dataflow adopted in TPU [6] prevents it from deploying the sparsity. Therefore, S2Engine utilizes an output stationary dataflow as shown in Fig. 4. Similar to Fig. 1, each PE undertakes the computation of a separate convolution while the compressed feature and weight flow among the systolic array in two directions in order to realize data reuse. The sparse features are fetched by CE array and sent to PE array. The feature overlap is processed
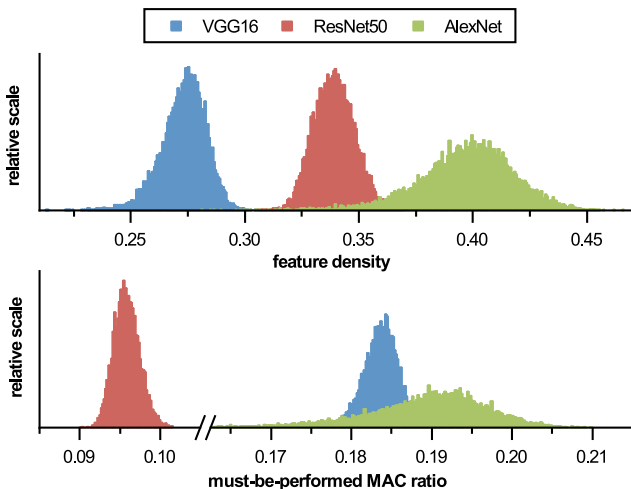

Fig. 3. The distribution of feature density and must-be-performed MAC ratio on ImageNet dataset. The feature density is defined as the proportion of non-zero elements in all feature maps, and the must-be-performed MAC ratio is defined as the proportion of MAC operations with two non-zero operands (the aligned data pairs) in all convolutions.
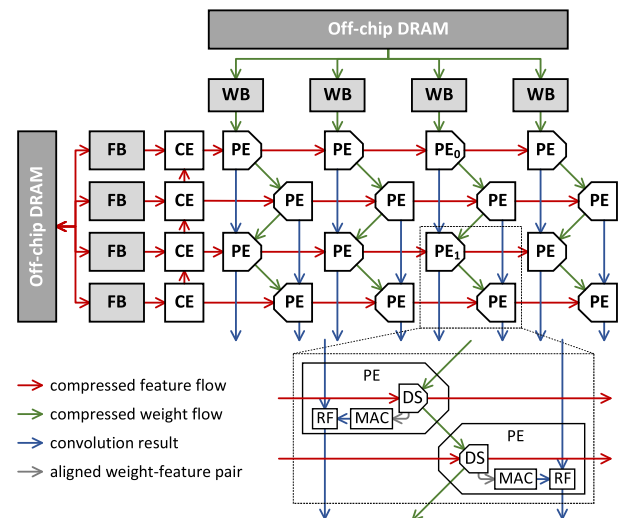

Fig. 4. Architecture overview of S2Engine and demonstration of internal dataflow.

between different PE rows to achieve overlap reuse. The data transmission path shown in Fig. 4 allows the S^2Engine to transmit the convolution results out of the systolic array faster compared to the naïve design adopted in [4].

In this work, we enhance the PE design from the naïve design [4], [6] to exploit the sparsity of both weight and feature. Our PE can dynamically select the aligned data pairs from the two dataflows that passing across it for sparse convolution. As shown in Fig. 4, each PE can be decomposed into three components: <u>D</u>ynamic <u>S</u>election (DS), <u>M</u>ultiplication and <u>AC</u>cumulation (MAC), <u>R</u>esult <u>F</u>orwarding (RF). DS component performs dynamic selection and then sends aligned feature-weight pair to MACs. The design of MAC component is trivial as it just simply achieves the multiplication and accumulation. The function of RF component is illustrated with PE_0 and PE_1 marked in Fig. 4. Due to the irregularity of sparsity, the workload allocated to each PE might not be equal. For example, PE_1 might generate the convolution results before PE_0. To guarantee the convolution results are transmitted out from the systolic array sequentially, in addition to the naïve design, the RF component in PE_1 needs to stall and wait until the convolution results from PE_1 have been forwarded.

Similar to the naïve design shown in Fig. 1, the convolution process would be projected to GEMM (general purpose matrix multiplication) operation for the PE array to process. However, different from the naïve im2col() operation provided in Caffe [34], the three dimensional input feature map is divided into groups and then reshaped into one-dimensional vector at this granularity. Such a reshaping manner is adopted for the convenience of deploying overlap reuse which would be detailed illustrated in Section 4.4. After being reshaped into one-dimensional data (illustrated in Fig. 1), the sparse data are further compressed before feeding into PE array as demonstrated in Fig. 4.

4.2 Dataflow Compression

To select all the aligned data pairs, e.g., $(f_{0,1}, w_{1,0})$ in Fig. 1, from the compressed feature and weight flow, the indices of non-zero feature and weight need to be extracted and compared with each other in PEs. However, several popular sparse formats are not suitable for this purpose because they usually introduce additional overheads. COO [35] format encodes two absolute indexes for each element, which contains redundant information and cannot represent an arbitrary large matrix with limited bit width for offset. The CSC format [33] stores the number of zeros between two non-zeros, therefore non-zeros accessing has to perform coordinate transformation from the compressed format. In this work, as a variation of COO, Enhanced COO format (ECOO) is introduced to overcome these limitations. Instead of achieving a higher compression ratio, the primary goal of adopting ECOO format is to simplify the design of DS.

As described before, since the input data is reshaped at the granularity of groups, the one-dimensional vectors fed into the systolic array (feature or weight) have been naturally divided into groups as shown in Fig. 5, where the group length is fixed at 6 in this example. The absolute position of each element inside each group is stored as offset. To avoid the mismatch of feature and weight elements
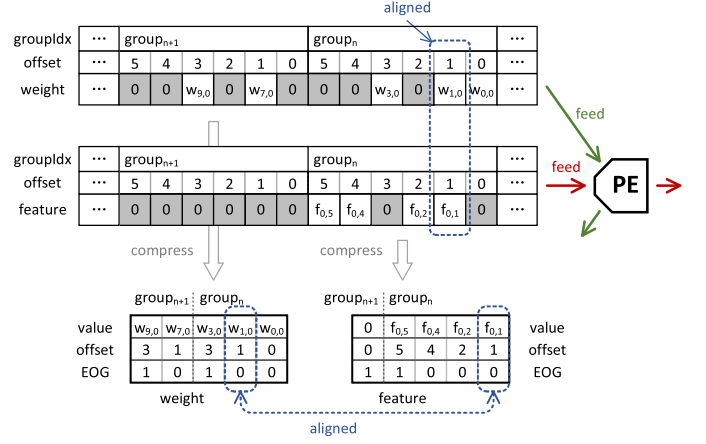

Fig. 5. A toy example to illustrate the dataflow compression.

between different groups, an extra EOG (end-of-group) flag is defined at the last element of each group. If all elements in a group are zeros, one zero is kept and marked as EOG as a placeholder. Hence, ECOO format could be denoted as triplets (value, offset, EOG). As we have observed that 4 bits are enough to represent offset which is also used in several previous works [13], [33], the group length is set to 16 in this work. Additional 1 bit is required for EOG and each nonzero feature would be represented by 13 bits in total. Another 1 bit is required to encode nonzero weight (14 bits in total) to represent the end-of-kernel. As shown from Fig. 5, the aligned weight-feature pair, e.g., $(w_{1,0}, f_{0,1})$, would have the same offset after being compressed. In Section 4.3, we will show that this property makes it easier for the DS component to select all the aligned weight-feature pairs from the compressed dataflow.

4.3 Dynamic Selection Component

By adopting the ECOO format described above, DS component only needs to select all weight-feature pairs with the same offset and feed them into MACs. The selecting process is scheduled by the controller to find the weight-feature pairs with the same offset. As illustrated in Fig. 6, weight and feature flows are first buffered in W-FIFO and F-FIFO, respectively. Then the controller could select the aligned weight-feature pairs (stored in w_{value} and f_{value} respectively) according to their offset (stored in w_{offset} and f_{offset} respectively) and EOG (stored in w_{EOG} and f_{EOG} respectively), and store them into WF-FIFO. After that, the weight and feature flows are exported to the adjacent PEs for data reuse among the entire systolic array, as demonstrated in Fig. 4.

A detailed illustration of this dynamical selection process with the toy example in Fig. 5 is shown in Fig. 7. In this example, utilizing the ECOO format can significantly simplify the selection process. For the convenience of the following description, we define the push of dataflow as an action that transmitting one data from FIFO to both the registers and the succeed PE in its transmission path. For example, weight flow is pushed in $cycle_0$ while feature flow is pushed in $cycle_2$. In $cycle_0$, feature flow is pushed since $w_{offset} < f_{offset}$. Then in $cycle_1$, because of $w_{offset} = f_{offset}$, the weight-feature pair stored in registers is aligned and is sent to MAC. Meanwhile, both weight and feature flow are pushed to process the subsequent data. In $cycle_3$,
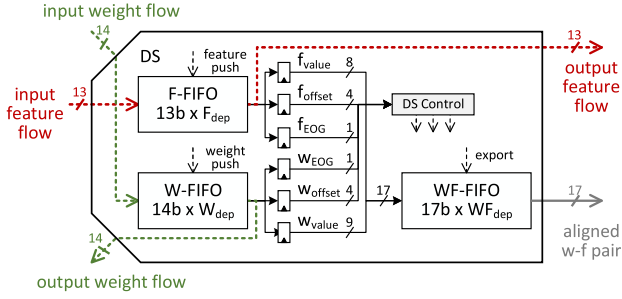
Fig. 6. Datapath and functional components of DS design, where `w-f` represents weight-feature.

weight meets `EOG` but feature does not. Therefore, feature flow is pushed until its meets `EOG` too in `cycle_4`. After that, DS begins to process the next group of data in `cycle_5`.

This toy example also illustrates how S^2Engine achieves speedup by deploying network sparsity. As demonstrated in Fig. 7, the processing of one group (`group_n`) is completed in five cycles while the two groups (`group_n` and `group_{n+1}`) is processed in seven cycles. It can be seen that more cycles (six cycles for each group) would be required to process them in a naïve design [4], [6]. Moreover, because there is only one aligned weight-feature pair is selected in this process, the MAC component would be idle for most of time if it runs at the same frequency as DS component. It can be further inferred that a higher utilization of MAC component and a higher throughput can be achieved when DS runs at a higher frequency than MAC component. The impact of the frequency ratio between these two components on throughput is thoroughly evaluated in Section 6.

On the other hand, as demonstrated in Fig. 7, such a dynamical selection process can also result in noncontinuous movement of both feature and weight flows. The performance of the succeed PE on data transmission path will be degraded. Hence, the FIFOs inserted in DS component can provide such a discontinuity in order to make the whole systolic array run smoothly. Based on our observations, several tens of bits are enough to implement the required FIFOs

Fig. 7. Demonstration of the dynamical selection process (from `cycle_0` to `cycle_5`) using a toy example in PE(0,0).
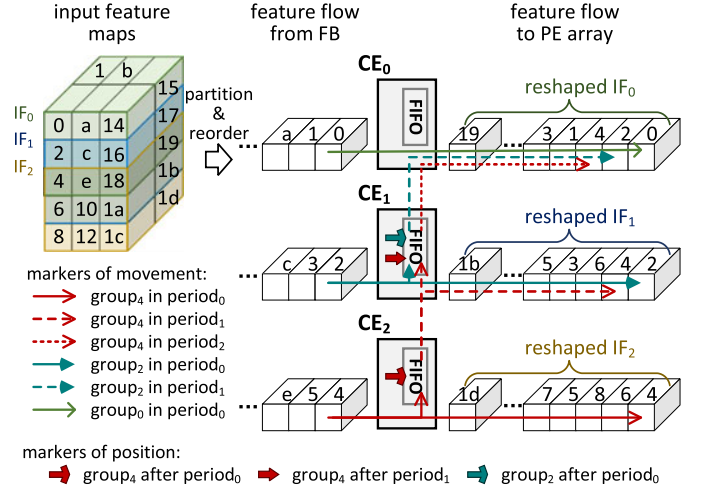

Fig. 8. Internal data transmission of CE array.

(represented by registers). The impact of FIFO size on total performance will be evaluated in Section 6.

4.4 Collective Element

CE array is designed to exploit the overlap reuse in adjacent rows of PE array as shown in Fig. 1. The compressed data groups are fed into CE array and broadcasted according to the internal data transmission path as shown in Fig. 8. The input three-dimensional data (both feature and weight) are divided into groups along the channels (cubes in Fig. 8), and each group contains up to 16 elements. After that, they are reshaped as one-dimensional dataflow and fed into different rows of PE array.

Considering a convolutional layer with a kernel size of 3×3 and a stride of 1, as shown in Fig. 8, the feature maps required for the three convolutions, `IF_0`, `IF_1` and `IF_2` is overlapped with each other. When this layer is processed on a naïve design shown in Fig. 1, the overlapped part of these three feature maps would be stored in three separate FBs as three copies, resulting in a waste of memory storage. On the other hand, since the same group of data would be loaded from the three FBs separately, unnecessary power overheads are introduced by these repeated buffer accesses.

The working mechanism of CE array is illustrated in Fig. 8. The data transmission procedure is divided into three periods, where data movement path and data groups are highlighted. In `period_0`, `CE_2` loads `group_4` from FB and then send this group to PE array while holding a copy of the group in its internal FIFO. During the same period, `group_2` is sent to PE array by `CE_1` in the same manner. In the next period, `group_4` is loaded from the internal FIFO in `CE_2` and sent to PE array by `CE_1`. Fig. 8 shows that each CE only holds one group of data at a time. Since the input feature map are divided into groups along channels, the above dataflow generation manner could guarantee that such a CE array could process all scales of CNNs by using internal FIFOs with fixed depth.

In such an approach, CE array mainly involves lots of internal FIFO accessing (small register files) instead of frequently FB accessing (large SRAM) so that its energy efficiency could be improved accordingly. Meanwhile, each CE only stores one group of data as shown in Fig. 8, i.e., `CE_2`
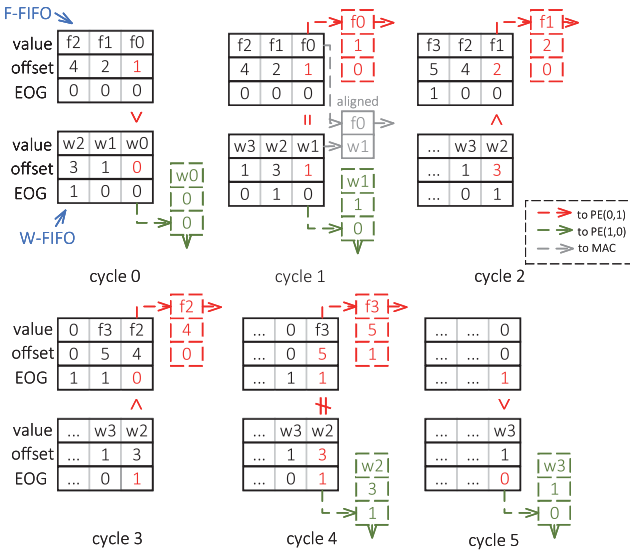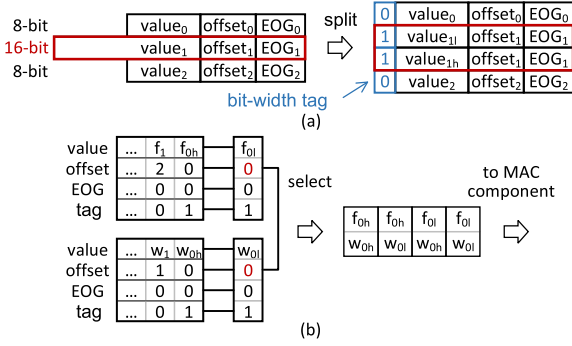
Fig. 9. (a) the unified representation for both 8-bit and 16-bit data with an extra flag bit. (b) disassembling 16-bit multiplication into four 8-bit multiplications.

only stores $group_4$ in its internal FIFO, so that the deployed CE array could make S^2Engine very memory efficient.

The CE array runs at the same frequency as DS component and the evaluation in Section 6 shows that it does not cause a performance bottleneck of S^2Engine. Different from the strategy adopted in [36] which introduces auxiliary structure in each PE, the CE array adopted in this design can deploy the overlap reuse with negligible extra resources. Without loss of generality, our CE array can be also applied to naïve systolic array shown in Fig. 1.

4.5 Mixed-Precision Data Processing

S^2Engine can further support mixed-precision computing at a fine-grained manner. The work in [37] introduces extra data path to process higher precision data. However, our PE is only designed with an 8-bit data path. During the dataflow compression procedure, data will be divided into two regions, 8-bit and 16-bit ones, according to the given threshold. Then a 8-bit data will be marked with tag 0, and a 16-bit data $value_1$ has to be split into two 8-bit data and marked with tag 1, as shown in Fig. 9a. After that, the two 8-bit data could be fed into PEs for normal processing. Fig. 9b also demonstrates a special situation when two 16-bit data meet at the same PE. This situation could be solved by dividing the data into four pairs and feeding the pairs into the PEs. It is obvious that the mixed-precision data processing in our approach will not degrade the throughput of PE array, as validated by the experimental results in Section 6.

5 EXPERIMENTAL METHODOLOGY

The kernel parts of S^2Engine, including PE, CE and FIFO, are implemented as Verilog RTL and synthesized by Synopsys DC with Global Foundry 14nm LP FinFET technology. Gate level simulations are performed by Synopsys VCS simulator by setting the frequency of MAC component as 500MHz. The area cost and energy consumption of PE, CE and FIFO are analyzed by PrimeTime. The area and energy consumption of buffers (both WB and FB) are estimated by PCACTI [38] while the energy consumption of off-chip DRAM is estimated by CACTI [39].

5.1 Performance Evaluation

The utilized sparse CNN models are trained by the pruning algorithm proposed in [11]. An in-house compiler is designed with C++ language to translate the sparse CNN

models into compressed dataflow for S^2Engine simulations. Meanwhile, the required buffer capacity and the total amount of buffer accessing is evaluated by analyzing the generated compressed dataflow. A cycle-by-cycle accurate simulator is implemented with C++ language, which is practical to evaluate the performance of S^2Engine under various configurations. Our simulator models the cycle-by-cycle behaviors of each atomic component, including (1) RF/FIFO in PE and CE, (2) MAC in PE, (3) DS in PE, (4) WB and FB, etc. The generated dataflow is fed into the simulator to obtain the involved execution cycles as well as the statistics on the behaviors of the atomic components for latency and energy efficiency estimation.

5.2 Architecture Configurations

Several kinds of configurable parameters are explored to comprehensively evaluate the performance of S^2Engine. First, inheriting the modularity and expandability of systolic architecture, S^2Engine can be configured as different scales for various throughput requirements. Moreover, both the size of FIFOs and frequency ratio of DS-MAC component in each PE can be also configured as mentioned in Section 4. The naïve systolic array with output stationary dataflow demonstrated in Fig. 1 is adopted as a baseline, which can be also basically regarded as the performance of TPU. Similar to SCNN [17], the naïve systolic array is configured with total 2MB SRAM for FB and WB. This capacity is sufficient to hold 66 out of 71 convolution layers we evaluated. Since both feature and weight are compressed while the required buffer capacity can be further reduced by CE array, 1MB SRAM is sufficient for S^2Engine to hold 68 out of 71 layers. With a similar conclusion in [17], the DRAM bandwidth is configured as 50GB/s which will not become as a performance bottleneck. Aiming to estimate the speedups brought by sparsity, the naïve systolic array runs at the same frequency as the MAC component in S^2Engine. Besides, the naïve design adopts the same convolution mapping strategy as S^2Engine, which provides a fair comparison. In order to evaluate the performance degradation incurred by the blocking behaviors of finite-depth FIFO in PE, an infinite-depth FIFO is adopted as an ideal situation which is corresponding to the upper bound of performance.

5.3 Benchmarks

S^2Engine is evaluated on both synthetic and actual CNN models. Synthetic models are utilized to analyze some typical characteristics of S^2Engine while actual models are adopted to evaluate the realistic impacts of different design aspects. A series of CNN models are synthesized by different designated sparsity levels both on features and weights to evaluate the sensitivity of our design as the sparsity changes. Similar to [11], actual sparse CNN models are generated by training on ImageNet [30] and pruning with a neural network distiller [40]. Since the feature maps are resulted from input images, their sparsity can vary significantly with different input images and greatly affect the evaluation results. The ImageNet is divided into three subsets according to their resulted different feature sparsity (maximum, average and minimum) for comprehensive evaluation. If not mentioned, the following evaluation results are obtained by average feature sparsity.
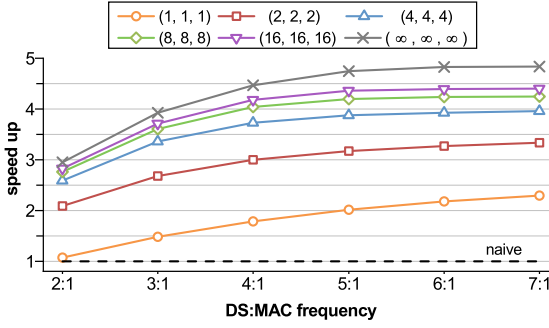
Fig. 10. Speedups with different FIFOs depth and DS:MAC frequency ratio.

6 EVALUATION RESULTS

This section first evaluates the characteristics of PE array in S^2Engine separately, including speedup under different configurations, and sensitivity to different sparsity levels. Then, the benefits brought by adopted CE array in S^2Engine are separately measured on actual models. At last, the complete S^2Engine is evaluated on different actual sparse CNN models to demonstrate the performance improvement in speedup, energy and area efficiency versus the naïve systolic array.

6.1 Design Space Exploration

As discussed before, both FIFOs depth and frequency of DS component can largely affect the throughput of S^2Engine. These impacts are evaluated on actual sparse models by fixing the scale of PE array as 16×16. The evaluation results also provide guidance for parameters selection in the following experiments.

Fig. 10 illustrates the average speedups of PE array when evaluating AlexNet/VGG16/ResNet50 under different configurations: FIFOs depth and frequency ratio of DS:MAC. Several typical combinations of FIFOs depth are selected and evaluated while the performance of other settings can be inferred from these results. Corresponding to the symbols in Fig. 6, the FIFOs depth are denoted as (W_{dep}, F_{dep}, WF_{dep}) in Fig. 10. Also, the infinite depth of FIFOs are denoted as (∞, ∞, ∞).

The results in Fig. 10 show that the benefits brought by higher DS frequency and deeper FIFOs diminish on marginal. When FIFOs depth is increased from (2,2,2) to (4,4,4), there are about $1.2\times$ speedups achieved. When FIFOs depth is further increased to (8,8,8), only about $1.1\times$ additional speedups are obtained. As a result, relatively small FIFOs with affordable area overhead are sufficient for data buffering. The similar situation could be found when varying DS: MAC frequency ratio. There are about $1.5\times$ speedups achieved when DS:MAC frequency ratio is doubled from 2:1 to 4:1. If the ratio is further doubled to 8:1, only about $1.1\times$ additional speedups could be obtained. Since the speedups are almost saturated when frequency ratio reaches to 4:1, i.e., DS component runs at 2000MHz, DS: MAC frequency ratio is set as 4:1 in the following experiments. Meanwhile, such a frequency ratio can be certainly realized by frequency division with a single clock domain.

6.2 Sensitivity to CNN Sparsity

Since sparsity levels usually affect the performance of CNN accelerators, we also evaluate different sparsity levels in

actual CNN models. Aiming to compare with SCNN, our PE array is fixed as 32×32. As shown in Fig. 11, a series of synthetic AlexNet models are evaluated by varying the sparsity levels both on features and weights from 10 to 100 percent. The sparsity level is defined as density, i.e., the percentage of non-zeros in features or weights. In general, our approach could obtain significant speedups compared with the naïve design because of the benefits from sparsity. Additionally, we could achieve better performance or efficiency than SCNN in most scenarios.

On-chip energy is evaluated separately under different configurations as shown in Fig. 11. Our proposed PE array achieves better on-chip energy efficiency versus the naïve design when the density is lower than 0.5/0.5. Additionally, the energy efficiency would be further improved by introducing CE array which will be illustrated in Section 6.5. For area efficiency, a metric of area/ops, i.e., the required chip area per operation, is defined to evaluate the area overhead introduced by DS component. Considering that smaller SRAM is required by S^2Engine, its area consumption will be even smaller than the naïve design (see the area breakdown in Table 5) which leads to a significant improvement in area efficiency as shown in Fig. 11. Since a large portion of resources are required by the involved crossbar and accumulator buffers in SCNN, our proposed PE array could have much more benefits on area efficiency than SCNN.

In conclusion, our adopted PE array behaves better than SCNN and the naïve design under most of the configurations when feature/weight density is lower than 0.5/0.5. Such a density can be easily satisfied by actual CNNs according to the statistics in Table 2 and Fig. 3. However, the utilized non-zero patterns of features and weights are uniformly distributed, which is different from the actual situation where the large data tends to concentrate [16]. In the following experiments, we will evaluate S^2Engine by using actual sparse CNN models.

6.3 Sensitivity to 16-Bit Data Ratio

Similar to the sensitivity to sparsity, the proposed mixed-precision data processing strategy is evaluated on a series of generated dense AlexNet models with 16-bit data ratio growing from 10 to 100 percent where the remaining data are 8-bit. Since 8-bit and 16-bit data are processed in the same datapath, the results in Fig. 12 show that our strategy is robust to various 16-bit data ratio. The evaluation in Table 4 shows that our strategy can process mixed precision data more efficiently than the work in [37].

6.4 Memory Efficiency

Memory efficiency is demonstrated by evaluating the reduction of required buffer capacity and accessing when overlap reuse is introduced, which is exploited by CE array. The average reduction ratios of all convolution layers in different CNN models are illustrated in Fig. 13. A significant reduction on buffer capacity and accessing is achieved on AlexNet and VGG16 where 3×3 kernels are widely adopted. However, CE array cannot achieve such a significant reduction on ResNet50 due to the widely adopted 1×1 kernel. Experimental results also show that S^2Engine with larger scale of PE array could obtain a slightly higher
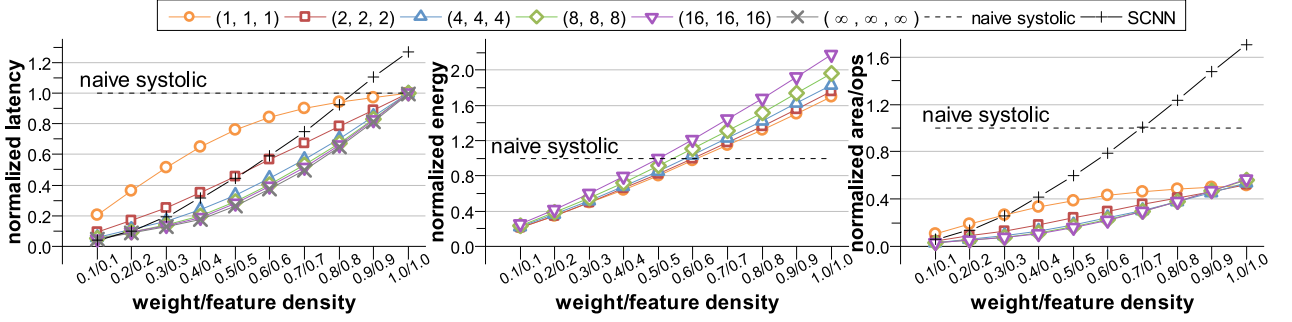
Fig. 11. Normalized latency, energy and area efficiency with different sparsity levels (`density`) and different FIFOs depth.

reduction both on buffer capacity and buffer accessing, which is reasonable because the inner data transmission between CEs are easier broken in smaller scale of PE array. The following evaluation will show that such a reduction on memory accessing could largely improve its on-chip energy efficiency.

6.5 Speed, Energy and Area Efficiency

At last, actual sparse CNN models are evaluated on S^2Engine with different scales of PE array to identify the speedups and energy/area efficiency improvements.

Fig. 14 illustrates the speedups of S^2Engine with different scale of PE array compared with naïve design. S^2Engine with larger scale of PE array will degrade the speedups because the dataflow usually becomes discontinuous after the DS procedure (as demonstrated in Fig. 7) and have to be accumulated PE by PE. Compared with VGG16 and ResNet50, AlexNet has a larger upper bound of speedups than average speedups because it has a larger variance in feature density distribution according to the statistics in Fig. 3. In summary, S^2Engine achieves about *3.2× speedups* on average versus the naïve design for all configurations and models. The largest speedup (3.6×) is achieved when the FIFOs depth is (8,8,8). Such a

speedup has almost reached the upper bound which is obtained with the FIFOs depth of (∞, ∞, ∞). The main bottleneck left in this case is the frequency of DS component as evaluated in Fig. 10.

The energy consumption of DRAM access can be simply reduced by utilizing compressed dataflow, however, such an improvement is not very relevant to our proposed architectures. Therefore, without considering the energy consumed by DRAM, the on-chip energy breakdown of S^2Engine with 16×16 PE array across various CNN models is illustrated in Fig. 15. It shows that the utilized CE array could significantly reduce the energy consumption, and the reduction on energy consumption mainly comes from MAC and SRAM (FBs and WBs). A large part of computations involved in MACs have been skipped which reduces the energy consumption significantly. The energy consumption of SRAM access is reduced by utilizing the compressed dataflow while the energy consumption of FB could be further reduced by CE array. It can be seen that the energy reduction achieved by these two aspects is much larger than the overhead introduced by FIFOs and therefore leads to better on-chip energy efficiency. Fig. 16 further evaluates the benefits on energy consumption brought by S^2Engine in different scales of PE array and different FIFOs depth across various CNN models when compared with naïve design.
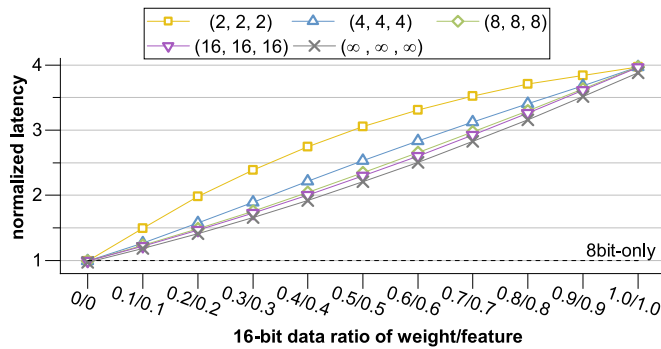

Fig. 12. Normalized latency versus 16-bit data ratio for different FIFOs depth.

TABLE 4
Required Additional Running Cycles of Mixed-Precision Data Processing Compared With 8-Bit Only Strategy for Different FIFOs Depth, Which are Evaluated on the Generated Model

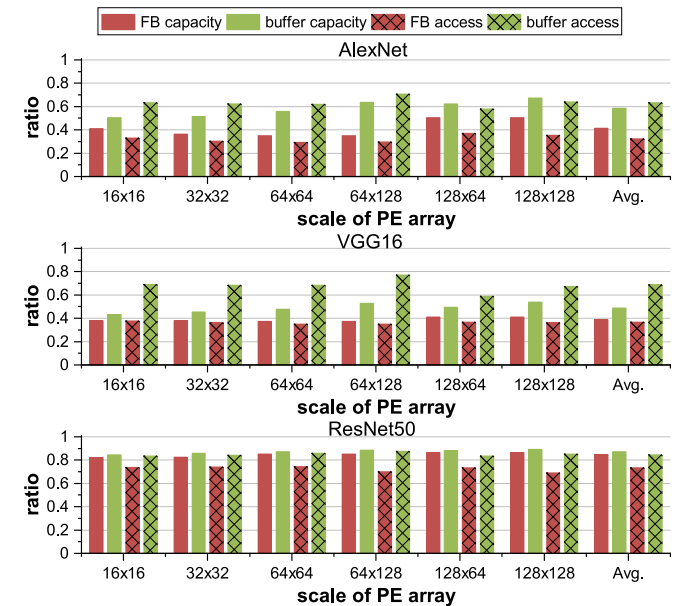| 16-bit Ratio	(2,2,2)	(4,4,4)	(8,8,8)	(16,16,16)	[37]
3.5%	16.3%	9.1%	8.4%	8.2%	10%
5%	24.1%	13.1%	11.9%	11.7%	∼ 20%

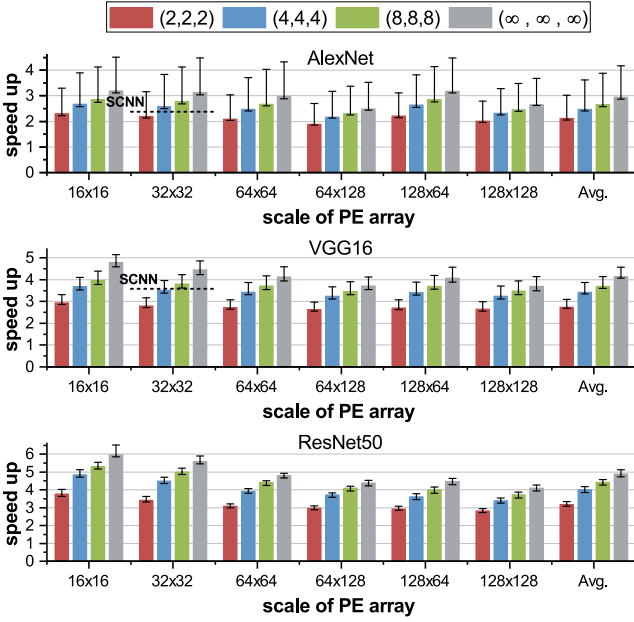Fig. 13. Analysis of reduction on buffer accessing and buffer capacity.

Fig. 14. Speedups of S^{2}Engine with different scale of PE array and different FIFOs depth, when compared with naïve design across various CNN models. The histograms represent the speedups with average feature sparsity. The upper bound and lower bound for each histogram represents the speedup with maximum feature sparsity and minimum feature sparsity, respectively.

Actually, S^{2}Engine with CE array could achieve about $1.8\times$ improvement on average in energy efficiency than naïve design, and such an improvement has a good scalability as the PE array scales up. CE array contributes about $1.3\times$ to the improvement and it is more significant for smaller PE array because a large proportion of energy is consumed by FBs access. The largest improvement ($1.9\times$) is achieved when FIFOs depth is set as (2,2,2). If further taking DRAM into consideration, S^{2}Engine could achieve about $3.0\times$ *improvement* of on-chip *energy efficiency* on average for different configurations and models.

The area efficiency improvement brought by S^{2}Engine in different scale is shown in Fig. 17. The introduced CE array brings little impact on chip area since a small CE array is only enough. The working manner of PE array as illustrated in Fig. 8 shows that it would not introduce additional latency. S^{2}Engine achieves significant area efficiency improvement especially for the case with smaller PE array in the same reason as analyzed in Section 6.2. Consistent with the evaluation illustrated in Fig. 13, Fig. 17 also shows S^{2}Engine can achieve better area efficiency compared to SCNN on actual sparse networks. However, as PE array scales up, the proportion of area
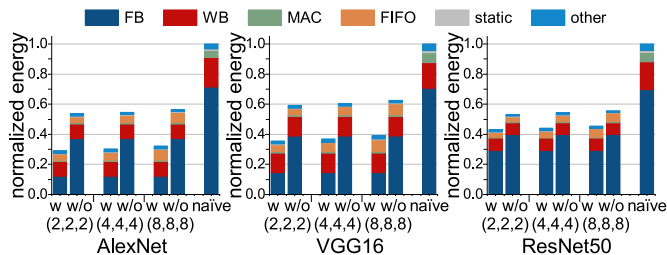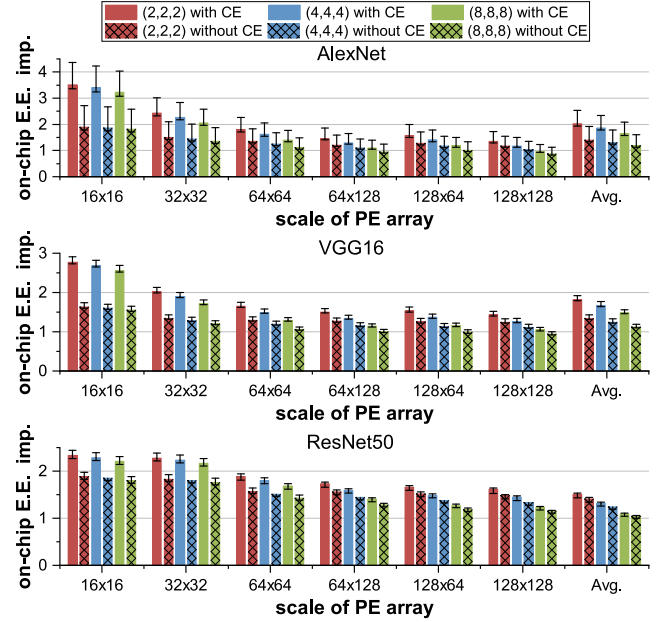

Fig. 16. On-chip energy efficiency improvement (E.E. imp.) of S^{2}Engine with different scale of PE array and different FIFOs depth across various CNN models.

occupied by PE array is gradually increased which diminishes the benefit brought by smaller SRAM and makes the overhead introduced by FIFOs more significant. Therefore, S^{2}Engine achieves about $2.9\times$ area efficiency improvement on average for different PE array scales but only achieves about $1.2\times$ at the scale of 128×128. As a tradeoff between speedup and area efficiency, the largest area efficiency ($3.1\times$) can be achieved when FIFOs depth of is set as (4,4,4).

As a conclusion of the above evaluations, with different FIFOs depth, the average *speedup* (across different scales and networks) achieved by S^{2}Engine ranges from **2.7×** to

Fig. 15. On-chip energy breakdown of S^{2}Engine with 16×16 PE array across various CNN models, where w means S^{2}Engine with CE array, w/o means S^{2}Engine without CE array.
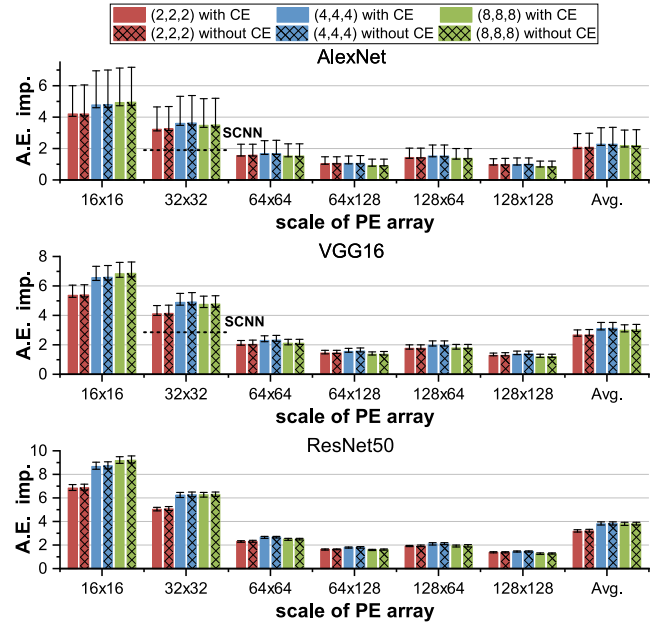

Fig. 17. Area efficiency improvement (A.E. imp.) of S^{2}Engine with different scale of PE array and different FIFOs depth across various CNN models.

TABLE 5
Comparison of Resources, Area Breakdown (mm^2), Speedup, Energy Efficiency (E.E.) and Area Efficiency (A.E.) Improvement (imp.) Among S^{2}Engine, Naïve Design, SCNN and SparTen

		S^2 Engine (32 × 32)			Naïve (32 × 32)	SCNN	SparTen
FIFO	Depth	2	4	8			
	Cap.	12KB	22KB	32KB	-	32KB	31KB
	Area	0.43	0.56	0.81	-	0.26	3.20
MULs	Num.	1024				1024	1024
	Area	0.12				0.51	17.8
SRAM	Cap.	1MB			2MB	1MB	-
	Area	1.44			2.89	1.98	-
Total Area		2.03	2.15	2.39	3.04	7.9	24.5
Technology		14nm				16nm	45nm
Speedup		2.49×	3.05×	**3.29×**	1×	2.94×	5.60×
E.E. imp.		**2.70×**	2.66×	2.59×	1×	2.21×	1.4×/0.5×*
A.E. imp.		3.67×	**4.23×**	4.11×	1×	2.20×	-

∗ *means* 1.4× *and* 0.5× *is E.E. imp. for partially memory and computation respectively.*

3.6×, on-chip *energy efficiency* improvement ranges from **1.5×** to **1.9×**, *area efficiency* improvement ranges from **2.6×** to **3.1×**. Therefore, it can be inferred that the performance would roughly fall into these intervals with various FIFOs depth, which shows the robustness of our approach.

Since SCNN and SparTen only report their improvement on speedup and energy efficiency versus their corresponding versions for processing dense networks, we can only provide a rough comparison here as presented in Table 5. The scale of S^{2}Engine is set to 32 × 32 since it has the same number of multipliers as SCNN and SparTen. Meanwhile, only AlexNet and VGG16 are considered since they are evaluated for all designs. With different configurations, S^{2}Engine could achieve **3.29×** speedup or **2.70×** energy efficiency improvement compared with the naïve design while SCNN could achieve up to 2.94× speedup and 2.21× energy efficiency improvement compared with its dense version. Although S^{2}Engine's speedup is not as good as that of SparTen (5.60×), S^{2}Engine's energy efficiency improvement is much higher than SparTen's. Table 5 shows that our design can deploy the sparsity more efficiently and outperform SCNN under most of configurations, and is more energy efficient than SparTen. It can be also seen that our work is a kind of *lightweight* design from the area breakdown. It is obtained by utilizing the sparsity, e.g., dynamic selection and output-stationary dataflow. Our design is much more hardware friendly without introducing components with large overhead, compared with SCNN (involved scatter network and accumulator buffers) and SparTen (involved prefix-sum circuit and permute network).

7 RELATED WORKS

Before the beginning of the era of machine learning, several systolic architecture-based accelerators were proposed for different algorithms [3], [41]. After the era of deep learning begins by [27], many accelerators have been proposed for deep neural networks. ShiDianNao [42] utilized a systolic array with kernel elements broadcast for convolutions. The work in [43] implemented AlexNet [27] with a 1-D systolic array. UCLA [4] proposed an end-to-end automation flow from high level C code to a 2-D systolic array-based FPGA implementation. Google's TPU [6] implemented an unprecedented 256 × 256 systolic architecture and achieved a great success in industry. The work in [5] deployed the feature sparsity on a 2-D systolic array through column combining but the network could be only trained by their own pruning method. Sparse-TPU [44] was similar to [5] and could exploit the feature sparsity as well as the weight sparsity, but its dataflow was not optimized for DNN applications. Although these works have taken many explorations on some aspects, none of them could provide a fully supporting for sparse neural networks.

Eyeriss [31] only optimized energy efficiency for memory access with the consideration of feature sparsity and was enhanced in [45] to process sparse and compact networks. Furthermore, Cnvlutin [15] only utilized feature sparsity and Cambricon-X [14] only considered weight sparsity. As somewhat an extension of Eyeriss, ZeNA [46] skipped all of the unnecessary computations but did not optimize the memory access. SCNN [17] indeed fully exploited the sparsity with general pattern in both feature and weight, but required additional output coordinates transformation. Cambricon-S [16] also fully deployed the feature and weight sparsity but was only applicable for coarse-grained sparsity pattern. DUET [47] could utilize both input and output feature sparsity through output speculation, but cannot support weight sparsity. SparTen [18] could support sparse vector-vector multiplication and improve the hardware utilization, but its energy efficiency was significantly degraded due to the existed additional computation resources. Eyeriss-V2 [45] was also capable of processing compressed sparse data of both weight and feature, but its NoC consumed a large number of routers and the data transmission or dispatching mechanism was much complicated, which was a burden on performance and energy consumption. In summary, none of these works could fully utilize the fine-grained sparsity both in feature and weight.

In regard to the mixed-precision quantization, Stripes [48] only supported one kind of precision for each layer and TPU [6] could support 8/16-bit processing in a very coarse granularity. The work in [37] introduced additional datapath for higher precision processing rather than reusing the computation logic resources, which significantly degraded the performance for processing higher precision cases.

8 CONCLUSION

In this paper, S^{2}Engine is proposed as a novel systolic architecture for sparse CNNs acceleration. By allowing each PE to select the aligned weight-feature pairs from the passing cross compressed dataflow dynamically, S^{2}Engine solves the contradiction between the regularity of data transmission path and the irregularity of the sparsity. Consequently, full exploitation of the sparsity is achieved without restriction on sparse patterns. Furthermore, a CE array is exploited for efficient data reuse in systolic array as well as the extension of PEs for supporting fine-grained mixed-precision processing. S^{2}Engine is evaluated on ImageNet with real sparse CNN models. The experimental results have shown that it could achieve about 3.2× speedup and about 3.0× energy-efficiency improvement compared with the naïve systolic array.

ACKNOWLEDGMENTS

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[2] "NVIDIA deep learning accelerator," 2018. [Online]. Available: http://nvdla.org/

[3] J.-N. Hwang, J. A. Vlontzos, and S.-Y. Kung, "A systolic neural network architecture for hidden Markov models," *IEEE Trans. Acoustics, Speech, Signal Process.*, vol. 37, no. 12, pp. 1967–1979, Dec. 1989.

[4] X. Wei, *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proc. ACM/IEEE Des. Automat. Conf.*, 2017, p. 29.

[5] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proc. Int. Conf. Architectural Support Program. Lang. Operat. Syst.*, 2019, pp. 821–834.

[6] N. P. Jouppi, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE Int. Symp. Comput. Architecture*, 2017, pp. 1–12.

[7] "Google TPU v2," 2018. [Online]. Available: https://cloud.google.com/tpu/

[8] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Commun. ACM*, vol. 61, pp. 50–59, 2018.

[9] "White paper: Accelerating DNNs with Xilinx Alveo accelerator cards," Xilinx, Inc., Tech. Rep., White Paper, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

[10] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[11] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[12] H. Mao *et al.*, "Exploring the regularity of sparse structure in convolutional neural networks," 2017, *arXiv: 1705.08922*.

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

[14] S. Zhang, *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2016, p. 20.

[15] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM SIGARCH Comput. Architecture News*, 2016, pp. 1–13.

[16] X. Zhou, *et al.*, "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2018, pp. 15–28.

[17] A. Parashar, *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. ACM SIGARCH Comput. Architecture News*, 2017, pp. 27–40.

[18] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "SparTen: A sparse tensor accelerator for convolutional neural networks," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2019, pp. 151–165.

[19] E. Park, S. Yoo, and P. Vajda, "Value-aware quantization for training and inference of neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 580–595.

[20] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[22] P. Judd *et al.* "Reduced-precision strategies for bounded memory in deep neural nets," 2015, *arXiv:1511.05236*.

[23] Y.-H. Chen, J. Emer, and V. Sze, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE Micro*, vol. 37, no. 3, pp. 12–21, Jun. 2017.

[24] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN accelerator," 2018, *arXiv: 1811.02883*.

[25] R. Hameed, *et al.*, "Understanding sources of inefficiency in general-purpose chips," in *Proc. ACM SIGARCH Comput. Architecture News*, 2010, pp. 37–47.

[26] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2014, pp. 10–14.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

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

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[31] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[32] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[33] S. Han, *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Architecture*, 2016, pp. 243–254.

[34] Y. Jia *et al.* "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*.

[35] R. Vuduc, "Automatic performance tuning of sparse matrix kernels," Ph.D. dissertation, Univ. California, Berkeley, CA, USA, 2003.

[36] C. Xin, *et al.*, "COSY: An energy-efficient hardware architecture for deep convolutional neural networks based on systolic array," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2017, pp. 180–189.

[37] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *Proc. ACM/IEEE Int. Symp. Comput. Architecture*, 2018, pp. 688–698.

[38] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "FinCACTI: Architectural analysis and modeling of caches with deeply-scaled FinFET devices," in *Proc. IEEE Comput. Soc. Annu. Symp.*, 2014, pp. 290–295.

[39] N. Muralimanohar, R. Balasubramanian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," Hewlett-Packard Lab., School Comput., Univ. Utah, Tech. Rep. TR-HPL-2009–85, 2009, pp. 22–31.

[40] N. Zmora, G. Jacob, and G. Novik, "Neural network distiller," 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1297430

[41] H. Asgari, Y. S. Kavian, and A. Mahani, "A systolic architecture for Hopfield neural networks," *Procedia Technol.*, vol. 17, pp. 736–741, 2014.

[42] Z. Du, *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM SIGARCH Comput. Architecture News*, 2015, pp. 92–104.

[43] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL™ deep learning accelerator on Arria 10," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2017, pp. 55–64.

[44] X. He, *et al.*, "Sparse-TPU: Adapting systolic arrays for sparse matrices," in *Proc. ACM Int. Conf. Supercomput.*, 2020, pp. 1–12.

[45] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Selected Top. Circuits Syst.*, 2019, pp. 292–308.

[46] D. Kim, J. Ahn, and S. Yoo, "ZeNA: Zero-aware neural network accelerator," *IEEE Design and Test*, vol. 35, no. 1, pp. 39–46, Feb. 2018.

[47] L. Liu, *et al.*, "DUET: Boosting deep neural network efficiency on dual-module architecture," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2020, pp. 738–750.

[48] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–12.

Jianlei Yang (Senior Member, IEEE) received the BS degree in microelectronics from Xidian University, Xi'an, China, in 2009, and the PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2014. He is currently an associate professor with the School of Computer Science and Engineering, Beihang University, Beijing, China. From 2014 to 2016, he was a postdoctoral researcher with the Department of ECE, University of Pittsburgh, PA, USA. His research interests include computer architectures and neuromorphic computing systems. Dr. Yang was the recipient of the First/Second place on ACM TAU Power Grid Simulation Contest in 2011/2012. He was also a recipient of IEEE ICCD Best Paper Award in 2013, ACM GLSVLSI Best Paper Nomination in 2015, IEEE ICESS Best Paper Award in 2017, and ACM SIGKDD Best Student Paper Award in 2020.

Wenzhi Fu received the BS degree in computer science from Beihang University, Beijing, China, in 2019. He is currently working toward the PhD degree with the School of Informatics, University of Edinburgh, Edinburgh, U.K. His research interests include computing architectures, database theory, and systems.

Xingzhou Cheng received the BS degree in computer science from Beihang University, Beijing, China, in 2019. He is currently working toward the graduation degree with the School of Computer Science and Engineering, Beihang University, Beijing. His research interests include computing architectures for deep learning and machine vision.
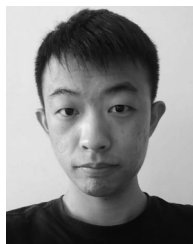
Xucheng Ye received the BS degree in computer science in 2018 from Beihang University, Beijing, China, where he is currently working toward the graduation degree with the School of Computer Science and Engineering. His research interests include deep learning algorithms and systems.
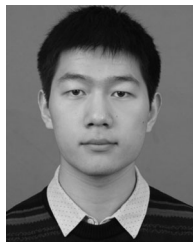
Pengcheng Dai (Student Member, IEEE) received the BS and MS degrees in electronic engineering from Beihang University, Beijing, China, in 2017 and 2020, respectively. He is currently a software development engineer with ByteDance Ltd. Inc., Beijing, China. His research interests include computing architectures for deep learning and machine vision.

Weisheng Zhao (Fellow, IEEE) received the PhD degree in physics from the University of Paris Sud, Paris, France, in 2007. He is currently the professor with the School of Microelectronics, Beihang University, Beijing, China. In 2009, he joined the French National Research Center(CNRS), as a Tenured Research Scientist. Since 2014, he has been a distinguished professor with Beihang University, Beijing, China. He has authored or coauthored more than 200 scientific articles in leading journals and conferences, such as *Nature Electronics*, *Nature Communications*, *Advanced Materials*, IEEE Transactions, ISCA, and DAC. His research interests include the hybrid integration of nano-devices with CMOS circuit and new nonvolatile memory (40-nm technology node and below) like MRAM circuit and architecture design. Dr. Zhao is currently the editor-in-chief for the *IEEE Transactions on Circuits and Systems I: Regular Paper*. He is an IEEE Fellow.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.