Configurable Multi-directional Systolic Array Architecture for Convolutional Neural Networks

RUI XU, SHENG MA, YAOHUA WANG, XINHAI CHEN, and YANG GUO,
National University of Defense Technology, China

The systolic array architecture is one of the most popular choices for convolutional neural network hardware accelerators. The biggest advantage of the systolic array architecture is its simple and efficient design principle. Without complicated control and dataflow, hardware accelerators with the systolic array can calculate traditional convolution very efficiently. However, this advantage also brings new challenges to the systolic array. When computing special types of convolution, such as the small-scale convolution or depthwise convolution, the **processing element (PE)** utilization rate of the array decreases sharply. The main reason is that the simple architecture design limits the flexibility of the systolic array.

In this article, we design a **configurable multi-directional systolic array (CMSA)** to address these issues. First, we added a data path to the systolic array. It allows users to split the systolic array through configuration to speed up the calculation of small-scale convolution. Second, we redesigned the PE unit so that the array has multiple data transmission modes and dataflow strategies. This allows users to switch the dataflow of the PE array to speed up the calculation of depthwise convolution. In addition, unlike other works, we only make a few changes and modifications to the existing systolic array architecture. It avoids additional hardware overheads and can be easily deployed in application scenarios that require small systolic arrays such as mobile terminals. Based on our evaluation, CMSA can increase the PE utilization rate by up to 1.6 times compared to the typical systolic array when running the last layers of ResNet-18. When running depthwise convolution in MobileNet, CMSA can increase the utilization rate by up to 14.8 times. At the same time, CMSA and the traditional systolic arrays are similar in area and energy consumption.

CCS Concepts: • **Computer systems organization → Neural networks;**

Additional Key Words and Phrases: Systolic array, convolutional neural network, hardware accelerator

ACM Reference format:
Rui Xu, Sheng Ma, Yaohua Wang, Xinhai Chen, and Yang Guo. 2021. Configurable Multi-directional Systolic Array Architecture for Convolutional Neural Networks. *ACM Trans. Archit. Code Optim.* 18, 4, Article 42 (July 2021), 24 pages.
https://doi.org/10.1145/3460776

This work is supported in part by the National Key Research and Development Project No. 2018YFB0204301, in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61672526, and in part by the Science and Technology Innovation project of Hunan Province No. 2018RS3083, No. 2019RS2027.

Authors' address: R. Xu, S. Ma (corresponding author), Y. Wang, X. Chen, and Y. Guo, National University of Defense Technology, 109 Deya road Kaifu, Changsha, Hunan, China, 410073; emails: nudtxurui@gmail.com, masheng@nudt.edu.cn, yaowangeth@gmail.com, chenxinhai16@nudt.edu.cn, guoyang@nudt.edu.cn.

1544-3566/2021/07-ART42 $15.00
https://doi.org/10.1145/3460776

1 INTRODUCTION

Recently, many hardware-customized accelerators with systolic array architecture for **convolutional neural networks (CNNs)** [11, 25] have been proposed [1–3, 5, 7, 8, 16, 21, 24, 32, 33, 36]. The systolic array architecture helps to improve data reuse opportunities in hardware accelerators, thus greatly reducing the memory traffic between the accelerators and external storage. There have also been many discussions about the systolic array architecture, such as dataflow, on-chip network, data sparsity, and the like [22, 23, 43]. These modifications and improvements greatly improve the efficiency of the systolic array in processing traditional CNN models.

However, with the evolving of CNNs, two trends make hardware-customized accelerators with the systolic array architecture inefficient: first, as the structure of CNNs continues to be deeper, the feature map size and the scale of convolution are becoming smaller and smaller [12] (For convenience, this article uses *small-scale convolution* to refer to this type of convolution.). Second, some CNN models introduce **depthwise convolution (DWConv)** [4, 14]. When dealing with small-scale convolution or DWConv, the processing elements (PEs) in the systolic array are significantly under-utilized.

Figure 1 shows the PE utilization rate of an accelerator with a 16 × 16 systolic array when processing a typical CNN model – MobileNet [14]. The systolic array has a high PE utilization rate when computing traditional convolution, but the utilization rate decreases sharply when computing small-scale convolution and DWConv. For some extreme cases, the utilization rate is less than 6%. Low utilization indicates that a large number of PEs are idle, which is a disaster for the performance of the hardware accelerator. This is because the simple architecture and fixed dataflow make the systolic array lack flexibility, and the data of workload or CNN models cannot completely drive the PE in the array.

To solve these problems, we need to make the systolic array architecture more flexible, like having more strategies of dataflow and multiple data transmission modes. Previous works usually choose to add extra routers, control units, and data paths in a systolic array [3, 27]. These changes can indeed increase the flexibility of the systolic array. However, the cost is a substantial increase in design complexities and hardware overheads. It may be acceptable for a large-scale hardware accelerator for the servers used in data-centers, but it is absolutely intolerable for the mobile/edge platform with small-scale systolic arrays. The scarce on-chip resource of the mobile/edge platform limits the space for modifications [8].

In this paper, We first established a mathematical model for the PE utilization rate of the systolic array. Through this mathematical model, we can easily find the reason for the decrease in the PE utilization rate during the convolution process. Based on this analysis, we further propose our novel design to efficiently improve the PE utilization rate for the convolution computation.

We propose a configurable multi-directional systolic array (CMSA). CMSA has the following properties. First, CMSA adds a new data path and transmission direction in the systolic array. So that we can split the array to change data mapping for small-scale convolution. Second, by redesigning the PEs and using the multi-directional data paths, we can switch the dataflow flexibly to solve the problem in DWConv. In addition, unlike [3, 27], CMSA maintains the systolic array architecture, and keeps the original intention of simple and efficient designs of the systolic array.

This article makes the following contributions:

- We analyze the effect of data mapping on the utilization rate of two-dimensional systolic arrays. By establishing a mathematical model, we can get the utilization rate in the process of systolic array operation. Then, we summarize the reasons for the decrease in the utilization rate of systolic arrays when computing special types of convolution.

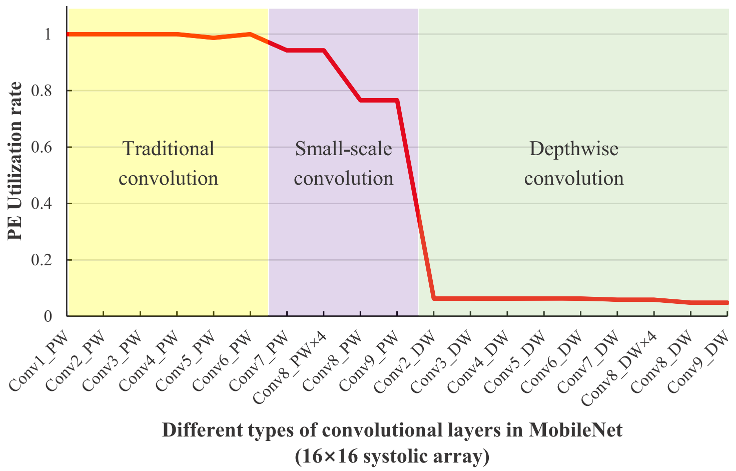
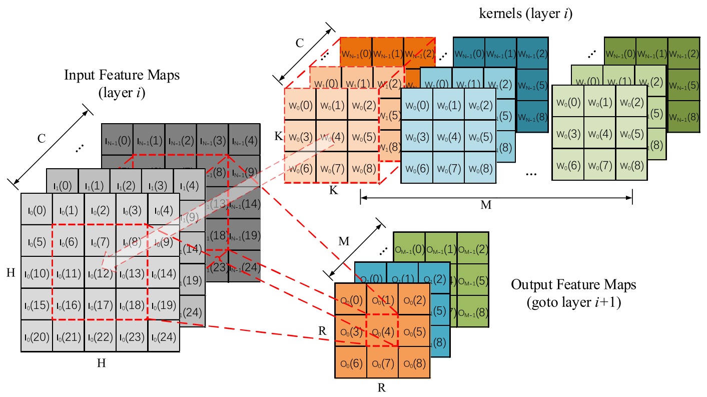


Fig. 1. The PE utilization rate of an accelerator with a 16 × 16 systolic array when processing different types of convolutional layers in MobileNet. It is obvious that the systolic array has a high PE utilization rate when computing traditional convolution, but the utilization rate decreases when computing small-scale convolution and DWConv.



(a)

```
for(m=0; m<M; m++){//loop L1, Output channels
    for(c=0; c<C; c++){//loop L2, Input channels
        for(r=0; r<R; r++){//loop L3, Output rows
            for(r=0; r<R; r++){//loop L4, Output columns
                for(k=0; k<K; k++){//loop L5, Kernel rows
                    for(k=0; k<K; k++){//loop L6, Kernel columns
                        O[m,r,r]=K[m,c,k,k]*I[c,r+k,r+k]
                        //Multiply-accumulate operation
}   }   }   }   }   }
```

(b)

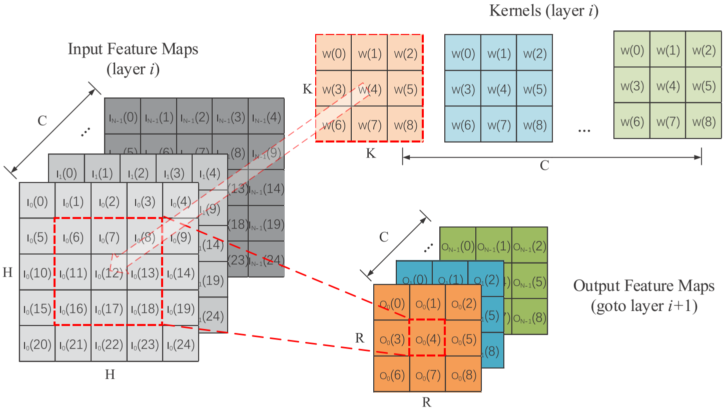
Fig. 2. (a) The process of traditional convolution and (b) The simple C code of it.

Table 1. Structures of Different CNN Models

Name	Layer	Kernels size	Ofmaps size
ResNet-18 (Cifar-10)	Conv1	3 × 3	32 × 32
	Conv2_1 × 2	3 × 3	32 × 32
	Conv2_2 × 2	3 × 3	32 × 32
	Conv3_1 × 2	3 × 3	16 × 16
	Conv3_2 × 2	3 × 3	16 × 16
	Conv4_1 × 2	3 × 3	8 × 8
	Conv4_2 × 2	3 × 3	8 × 8
	Conv5_1 × 2	3 × 3	4 × 4
	Conv5_2 × 2	3 × 3	4 × 4
ResNet-18 (ImageNet)	Conv1	3 × 3	112 × 112
	Conv2_1 × 2	3 × 3	56 × 56
	Conv2_2 × 2	3 × 3	56 × 56
	Conv3_1 × 2	3 × 3	28 × 28
	Conv3_2 × 2	3 × 3	28 × 28
	Conv4_1 × 2	3 × 3	14 × 14
	Conv4_2 × 2	3 × 3	14 × 14
	Conv5_1 × 2	3 × 3	7 × 7
	Conv5_2 × 2	3 × 3	7 × 7
MobileNet (ImageNet)	Conv1	3 × 3	112 × 112
	Conv2_dw/pw	3 × 3/1 × 1	112 × 112
	Conv3_dw/pw	3 × 3/1 × 1	56 × 56
	Conv4_dw/pw	3 × 3/1 × 1	56 × 56
	Conv5_dw/pw	3 × 3/1 × 1	28 × 28
	Conv6_dw/pw	3 × 3/1 × 1	28 × 28
	Conv7_dw/pw	3 × 3/1 × 1	14 × 14
	Conv8_dw/pw×5	3 × 3/1 × 1	14 × 14
	Conv9_dw/pw	3 × 3/1 × 1	7 × 7

- We design a novel systolic array. Our design can segment the array and use PEs efficiently, when calculating small-scale convolution, to improve the utilization rate of the array.
- We propose a novel dataflow suitable for depthwise convolution. We integrate it into a two-dimensional systolic array to improve the PE utilization rate when computing depthwise convolution.
- We implement our design and verify it through experiments. Experimental results show that CMSA can increase the utilization rate by up to 1.6 times compared to typical systolic arrays when computing small-scale convolution. When computing depthwise convolution, CMSA can increase the utilization rate by up to 14.8 times.

The rest of this article is organized as follows: Section 2 introduces the CNN basics and related work. In Section 3, we analyze the systolic array architecture through a mathematical model.
Section 4 introduces the CMSA architecture. We present the experimental results in Section 5.



(a)

```
for(m=0; m<1; m++){//loop L1 does not exist, M=1
    for(c=0; c<C; c++){//loop L2, Input channels
        for(r=0; r<R; r++){//loop L3, Output rows
            for(r=0; r<R; r++){//loop L4, Output columns
                for(k=0; k<K; k++){//loop L5, Kernel rows
                    for(k=0; k<K; k++){//loop L6, Kernel columns
                        O[c,r,r]=K[c,k,k]*I[c,r+k,r+k]
                        //Multiply-accumulate operation
}   }   }   }   }   }
```

(b)

Fig. 3. (a) The process of depthwise convolution and (b) the simple C code of it.

2 BACKGROUND AND RELATED WORK

In this section, we first introduce a primer on the convolution, which is the key operation in the process of the CNNs. We also discuss the recent developments and evolution of the convolution, including the small-scale convolution and depthwise convolution. Then we introduce the systolic array architecture and also introduce one of its commonly used dataflow – *Output Stationary* dataflow. Finally, we introduce the recent work of systolic arrays and show their advantages and disadvantages.

2.1 Convolution

The core part of CNNs is the convolution [18, 39]. Figure 2(a) shows the process of typical convolution in a traditional convolution neural network. It uses $K \times K$ (K represents the height (rows) and width (columns) of kernels) kernels to convole input feature maps (ifmaps), and get $R \times R$ (R represents the height (rows) and width (columns) of feature maps) output feature maps (ofmaps). Meanwhile, ifmaps have C channels, and ofmaps have M channels. Each channel has corresponding kernels [12]. Figure 2(b) shows the simple C code for this procedure.

However, the design of CNN models is always developing. The two current changes in the structure of CNNs deserve our attention.

Small-Scale Convolution. As the network structure of CNNs becoming further deepened, the feature map size of the last convolutional layers becomes particularly small. Table 1 lists the structure of several typical CNN models. It can be seen that as the number of network layers increases,

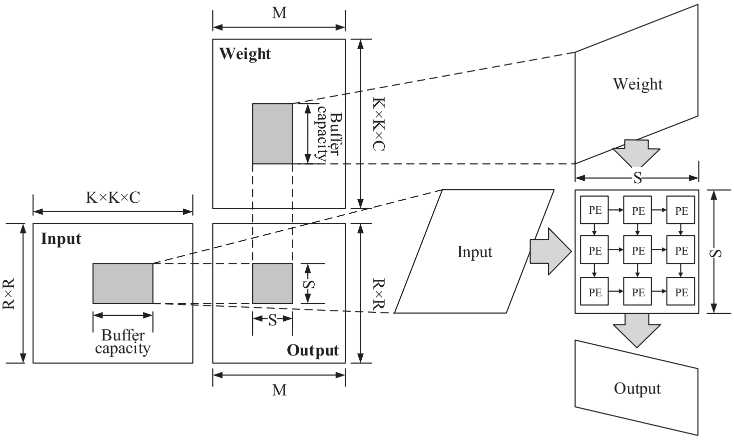


Fig. 4. The systolic array architecture with *OS* dataflow. The properties of the architecture and dataflow determine how to tile and process GEMM on the array.

the size of ofmaps gradually shrinks. For example, the ofmaps size of the last layers of ResNet-18 [12] and MobileNet [14] is only 7 × 7. These CNN models are all trained using the ImageNet dataset [6]. If users choose the small-scale dataset (like Cifar-10 and Cifar-100 [17]), the size of ofmaps can be further reduced to 4 × 4 [12].

Depthwise Convolution. Depthwise convolution (DWConv) is a special kind of convolution. It was first proposed by Xception [4] and MobileNet [14], and is now widely used in CNN models [40]. The DWConv is a spatial convolution performed independently over each channel of an ifmap. In other words, the DWConv only allows one kernel-channel to convolve with only one ifmap to produce one ofmap [14]. Like kernels, ofmaps are correlated to the C channels of ifmaps, and the original M channels are dismissed (Equivalent to $M = 1$). Figure 3 shows the calculation process and the simple C code. It can be clearly seen that the loop of the output channel does not exist.

The direct benefit of the disappearance of one the ofmap data dimension is the reduction of the amount of data and calculation. At the same time, with the pointwise convolution, the CNN models using DWConv can maintain a high accuracy rate. However, due to the fixed dataflow and architecture, computing DWConv is a disaster for the systolic array.

2.2 Systolic Array

Figure 4 shows the architecture and workflow of the *Output Stationary* (*OS*) systolic array [8, 16, 35]. Input data are delivered to the peripheral PEs and each PE relays these data to its neighbor [5]. At the same time, the workload is divided into multiple tiles, which are sequentially sent to the systolic array for calculation. *OS* dataflow means a pixel of ofmaps is generated completely before being pushed out of the array. It can prevent the movement of partial ofmaps and avoid the necessity of wasting clock cycles in preloading kernel data into the array [5]. There are other systolic array designs with different dataflow strategies. But their architecture and working principle are similar to the *OS* systolic array [35].

The systolic array is originally used in hardware accelerators for the general matrix multiplication (GEMM) [35]. By *im2col* algorithm, the convolution can be converted to the GEMM. Figure 5 shows the GEMM obtained from the convolution with 5 × 5 ifmaps, 3 × 3 kernels, and 3 × 3 ofmaps. Compared with the process of convolution shown in Figure 2, *im2col* associates multiple dimensions/loops of data and unrolls them on a two-dimensional matrix.

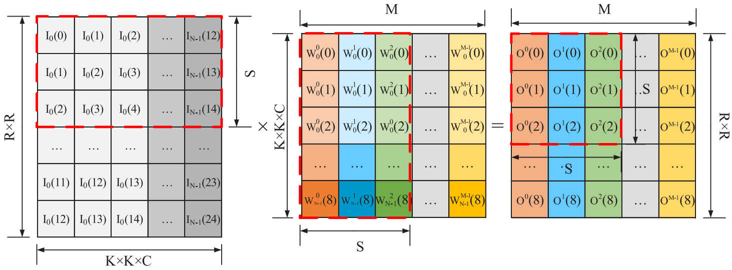
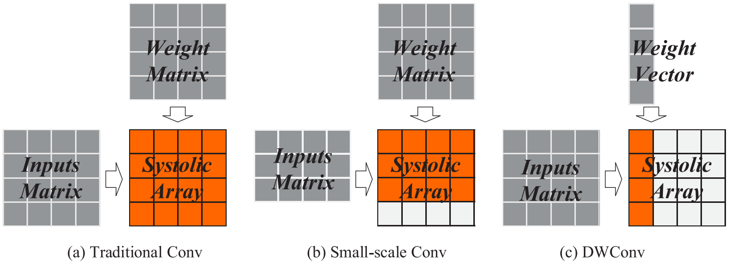


Fig. 5. The equivalent matrix multiplication for the convolution with 5 × 5 ifmaps, 3 × 3 kernels, and 3 × 3 ofmaps. *OS* maps the pixels of ofmaps on a *S*×*S* array



(a) Traditional Conv (b) Small-scale Conv (c) DWConv

Fig. 6. The number of active PEs in systolic array is different with different types of convolution. Orange boxes indicate active PEs and white boxes indicate idle PEs.

Figure 5 also shows how these matrices are segmented and mapped to the systolic array, and the ofmaps matrix is fixed on a $S \times S$ array. Because the dimensions of the GEMM of typical convolutional layers are both large and multiples of sizes of the systolic array, tiling, and processing this GEMM can fully utilize PEs in the systolic array [29]. Thus, the systolic array is very efficient at dealing with traditional CNNs [35]. However, there are problems when a systolic array processes the special types of convolution. Because the size of the output feature map ($R \times R$) in the small-scale convolution is small, feature maps no longer provide enough data to drive all PEs. Also, in DWConv, due to $M = 1$, the GEMM shrinks to the matrix-vector multiplication. It leads to a serious loss of performance. Figure 6 vividly shows these problems.

2.3 Related Work

The research on the systolic array architecture is a hot topic in recent years, and there are many designs worth discussing. Neuflow [32] deploys the *Weight Stationary* dataflow in the PE array, and uses the systolic mode to reuse data. However, because the array scale is limited to the size of the kernels, the scalability of Neuflow is very poor. ShiDianNao [7] uses *OS* dataflow to make the PEs array more scalable. Its architecture is similar to a systolic array. By reusing data between PEs, ShiDianNao achieves high performance. But it is not optimized for depthwise separable convolution. Flexflow [27] and Eyeriss [2] propose new dataflows and change the systolic architecture. These designs are efficient in the calculation, but their data transmission is completely dependent on the bus interconnection. It means that the implementations of designs may have difficulty in achieving the timing closure for a high clock frequency, or even passing the place and route [42]. It also consumes additional bandwidth, and there are many long wires in the design, which makes the implementation more difficult. Since the emergence of TPU [16], the two-dimension systolic array (or the TPU-like systolic array) becomes the main trend in hardware CNNs accelerators deployed in both edge devices [8–10, 15, 16, 26, 30, 38] and servers used in data-centers [16, 28, 29, 31]. The architecture of this systolic array has been introduced in Section 2.2, and we consider it as a baseline design in this article.

However, we find that these designs only support one fixed dataflow, and all face challenges of low PE utilization rates when processing the special types of convolution [41]. Some proposals try to overcome these problems. Eyeriss V2 uses *RS+* dataflow to solve the problem [33]. Through the routing unit, flexible data forwarding between PEs can be realized. However, this dataflow unrolls the mini-batch to improve the ability of parallel computing. It is not optimized for the case of small mini-batch (such as video processing or lightweight tasks). Chen et al. [3] and Lu et al. [27] try to change dataflow of the systolic array to improve the PE utilization rate. However, they introduce a lot of storage units, routers, data paths and control units, and increase hardware overheads. Obviously, these works deviate from the original intention of simple and efficient designs of the systolic array.

There are other optimization work for the systolic array. For example, Kung et al. [19] and [13] use column combining to train and compress models. [29] solves the problem of reducing the number of channels in the compression training process by combining or splitting multiple small arrays. [20], [9] Gope et al. and Sharife et al. [37] deploy low-precision networks and use bit-wise calculations to achieve low-power designs. However, none of these designs consider the emerging special types of convolution, including the small-scale convolution and DWConv.

3 ANALYTICAL MODEL

In this section, we establish a mathematical model to analyze the PE utilization rate of the systolic array. Through this model, we can find the key factors that affect the performance of the systolic array. At the same time, it can explain the low utilization rate of the systolic array when processing special types of convolution, and motivate us to find a solution.

3.1 The PE Utilization Rate of the Systolic Array

The PE utilization rate of the systolic array refers to the ratio of the number of activated PEs to the total number of PEs in the calculation process. This ratio can reflect the current computing efficiency and directly affect the performance and energy consumption of the systolic array.

Equation (1) describes how PE utilization is calculated for a given cycle.

$$Util_{cycle} = \frac{Num.activated\ PEs}{Num.total\ PEs} \tag{1}$$

Then Equation (2) describes the average utilization rate during the computing process, where *cycles* refers to the total cycles spent in the calculation.

$$Util = \frac{\sum Util_{cycle}}{cycles} = \frac{\sum Num.activated\ PEs}{Num.total\ PEs \times cycles} \tag{2}$$

It is very difficult to count the number of PE activated in each cycle. But for a systolic array, each activation of a PE means a multiply-accumulate (MAC) operation. So, we just need to count the number of MAC operations required to complete convolution.

$$\sum Num.actived\ PEs = Num.MAC = R^2 \times M \times K^2 \times C. \tag{3}$$

It is also quite challenging to count the computing cycles. The computing procedure of the systolic array can be divided into three stages, *filling*, *calculating*, and *emptying*.

$$cycles = cycles_{filling} + cycles_{calculating} + cycles_{emptying}. \tag{4}$$

The value of calculating cycles is determined by the workload, dataflow, and systolic array architecture. As introduced in Section 2.2, the data needs to be divided into multiple tiles and sent to the systolic array. Therefore, we can obtain the following equations, where $cycles_{tile}$ is the number of cycles required for the systolic array to complete one tile calculation, and $Num.tiles$ is the total number of tiles divided from the workload:

$$cycles_{calculating} = cycles_{tile} \times Num.tiles. \tag{5}$$

The systolic array architecture decides the size of each tile, the dataflow decides the time it takes for the systolic array to calculate one tile, and the workload decides the number of the tiles. Taking the $S \times S$ systolic array with OS dataflow as an example, according to the description of Figure 5, the following equations can be easily obtained:

$$cycles_{tile} = K^2 \times C \tag{6}$$

$$Num.tiles = \left\lceil \frac{M}{S} \right\rceil \times \left\lceil \frac{R^2}{S} \right\rceil \tag{7}$$

Since $filling$ cycles and $emptying$ cycles are much smaller than $calculating$ cycles, we can get:

$$cycles \approx cycle_{calculating} = K^2 \times C \times \left\lceil \frac{M}{S} \right\rceil \times \left\lceil \frac{R^2}{S} \right\rceil \tag{8}$$

Substitute Equation (3) and Equation (8) into Equation (2),

$$Util = \frac{R^2 \times M \times K^2 \times C}{S \times S \times K^2 \times C \times \left\lceil \frac{M}{S} \right\rceil \times \left\lceil \frac{R^2}{S} \right\rceil} = \frac{R^2 \times M}{S^2 \times \left\lceil \frac{M}{S} \right\rceil \times \left\lceil \frac{R^2}{S} \right\rceil} \tag{9}$$

It can be seen that under the certain architecture (systolic array) and dataflow (OS), three factors affect the PE utilization rate, the PE array size, the number of output feature channels and the ofmaps size.

Figure 5 shows that the output feature channels are horizontally distributed in the array, which determines how many columns of PEs are activated. And ofmaps are distributed vertically in the array, which determines how many rows of PEs are activated. So we can further divide Equation (9) into:

$$Util_{horizontal} = \frac{M}{S} \Big/ \left\lceil \frac{M}{S} \right\rceil \tag{10}$$

$$Util_{vertical} = \frac{R^2}{S} \Big/ \left\lceil \frac{R^2}{S} \right\rceil \tag{11}$$

In other words, the input data affects the PE utilization rate in both the horizontal and vertical directions. it is also in line with the description of Figure 4 and Figure 5.

It should be noted that the systolic array can use other dataflows, such as WS or IS, and similar equations can also be obtained.

3.2 Performance and the PE Utilization Rate

Performance is actually directly related to the utilization rate. An important indicator to measure performance is throughput. Equation (12) shows how PE utilization affects the throughput of systolic arrays, where $Num.total\ PEs$ refers to the number of PEs in the array, $Throughput_{PE}$ refers to the throughput of a single PE, and $Util$ is the average PE utilization rate in the calculation process.

$$Throughput = Num.total\ PEs \times Throughput_{PE} \times Util. \tag{12}$$

Table 2. The Factors Affecting the PE Utilization Rate

Factors	Influences	Recommendation
The systolic array size (S)	It determines the conditions for the optimal utilization rate.	According to the statistical results, select the appropriate array size in the design stage.
The output feature channels number (M)	It affects the horizontal utilization rate of the array.	M is an integer multiple of S or M is much larger than S.
The output feature maps size (R)	It affects the vertical utilization rate of the array.	R^2 is an integer multiple of S or R^2 is much larger than S.

Therefore, in order to improve the performance of the systolic array, we should find a way to increase the PE utilization rate. It is a key factor and worth studying.

According to our analysis model (Equations (9), (10), and (11)), the PE utilization rate of a systolic array is first related to the array size S. Then the number of columns of PE that can be activated (in the horizontal direction of the array) is determined by the output channel number M of the CNNs model. And the number of rows of PE that can be activated is related to the output feature map size R. In fact, for a systolic array with the *OS* dataflow and the fixed array size, the final factors that affect the average PE utilization rate are the characteristics of the network model (M and R).

According to the Equations (10) and (11), if the total PE utilization rate is high, M and R must meet the following conditions:

(1) M and R^2 in the network model are integer multiples of S. At this time, $M/S = \lceil M/S \rceil$, $R^2/S = \lceil R^2/S \rceil$. Then, $Util_{horizontal} = 1, Util_{vertical} = 1, Util = 1$.

The PE utilization rate is optimal at this time. However, this situation is too ideal. The systolic array has to deal with various workloads, and the characteristics of the network models are also changing at any time. It is difficult to keep the number of output channels and the size of the feature map as integer multiples of the array size always.

(2) M and R^2 are much larger than S. Traditionally, the scale of the workload is very large, that is, $M, R^2 >> S$,so

$$\frac{M}{S} \to \left\lceil \frac{M}{S} \right\rceil, \frac{R^2}{S} \to \left\lceil \frac{R^2}{S} \right\rceil \tag{13}$$

Therefore,

$$Util_{horizontal} \to 1,\ Util_{vertical} \to 1,\ Util \to 1. \tag{14}$$

So the PE utilization rate of the systolic array is always high when computing convolutional layers of traditional CNNs. This is why many hardware accelerators choose the systolic array architecture.

If any one of M and R does not meet the above conditions, the PE utilization rate will decrease, thereby affecting the performance of the systolic array.

In addition, we can analyze the characteristics of common CNNs models to determine the size of the array during the design process to meet the condition (1) as much as possible. For example, the common array size is usually a multiple of 16. This is because the number of channels in most network models is a multiple of 16. Table 2 shows our conclusions. Through it, readers can understand our analysis more clearly.

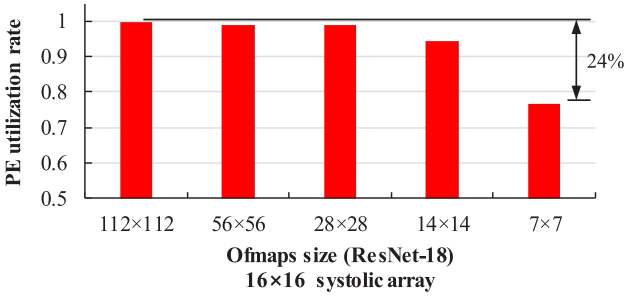


Fig. 7. The PE utilization rate of the systolic array when computing convolutional layers with different ofmaps sizes in ResNet-18.

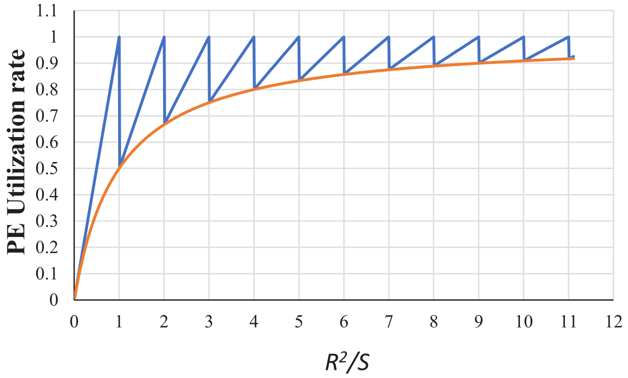


Fig. 8. The curve of PE utilization rates changing with R^2/S. The blue line represents the actual utilization rates, and the orange line is the trend line of the minimum utilization rates.

3.3 Challenges

With the evolution of neural networks, it is difficult for the characteristics of new network models to meet the above conditions. This leads to the low efficiency of the systolic array in the calculation process.

3.3.1 Challenges in Small-Scale Convolution. As introduced in Section 2.1, with the network structure of CNNs becoming further deepened, the feature map size of the last convolutional layers becomes particularly small. The small feature map means a small R, which will directly affect the vertical PE utilization rate. If R^2 is no longer much greater than S, the performance and efficiency of the systolic array will be greatly reduced.

Take a 16 × 16 systolic array as an example, and the ResNet-18 in Table 1 as the workload. Figure 7 shows the PE utilization rate of the systolic array when computing convolutional layers with different feature maps sizes. It can be seen that when the feature map size is large enough, the utilization rate is quite high. When the feature map size is reduced, the utilization rate also becomes smaller, which is 24% lower than the peak value.

In order to further illustrate the impact of small-size convolution on the PE utilization rate, we draw a curve of PE utilization with R^2/S. Figure 8 shows the results. The blue line represents the actual utilization rates and the orange line represents the trend line. It can be seen that the value of PE utilization rate fluctuates violently, but the overall trend is to increase with the increase of R^2/S. When R^2/S is greater than 4, the utilization rate is stable greater than 80%. When R^2/S is greater

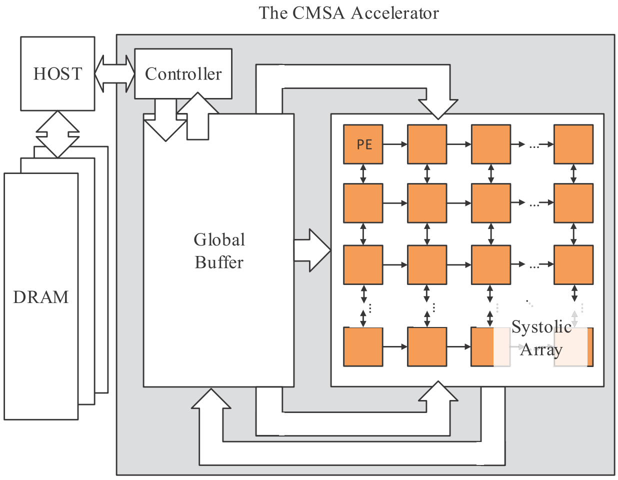


Fig. 9. The CMSA architecture. It is mainly composed of the controller, global buffer, and PEs array.

than 9, the utilization rate is stable greater than 90%. Therefore, the larger R^2/S is, the better the utilization rate is. In this article, we refer to the convolution that has a performance loss of more than 20% due to their small feature maps size as *small-size convolution*.

3.3.2 *Challenges in Depthwise Convolution.*

Depthwise convolution seriously reduces the PE utilization rate and computational efficiency of the systolic array. When calculating the depthwise convolutional layers, the PE utilization rate of the systolic array plummeted below 6%, as shown in Figure 1 in Section 1.

Section 2.1 introduces that depthwise convolution has one less output feature maps dimension than traditional convolution. This means there is no Loop 1 (Figure 3(b)) and M in Equation (10) is equal to 1. So the formula for PE utilization rate becomes:

$$Util_{horizontal} = \frac{1}{S}\ ,\ Util \leqslant \frac{1}{S} \tag{15}$$

This causes the PE utilization rate of the systolic array to be less than $1/S$, and the larger the array size, the lower the utilization rate.

In fact, the PE utilization rate of the systolic array is reduced because the size of the tiles divided from the workload is not enough to activate all the PEs in the array. Therefore, we must change the tile segmentation and mapping method, that is, change the traditional architecture and dataflow of the systolic array.

4 CMSA ARCHITECTURE

As introduced in Section 3, we decide to modify the architecture and dataflow of the systolic array. But the major difference between our work and others is that we try to preserve the original architecture of the systolic array as much as possible. In other words, our work is upgrading the systolic array to increase the flexibility. The advantage of this is that while improving the efficiency, it complies with the original simple design principles of the systolic array, and reduces the hardware overhead compared to the previous work.

Figure 9 shows an overview of our design. The CMSA accelerator is mainly composed of the controller, global buffer, and PEs array. CMSA still retains the main structure of the systolic array. Compared with the traditional systolic array accelerators, we only add a data path to the bottom of

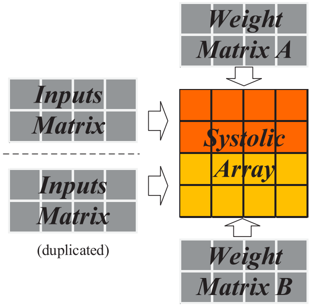


Fig. 10. A toy model to demonstrate how to split the PE array to further unroll the kernel data.

the array and allow two-way data transmission in the vertical direction. This simple modification provides enough flexibility for the systolic array.

4.1 Optimization for Small-Scale Convolution

Since R is close to or even smaller than the systolic array size in small-scale convolution, many idle PEs appear during the operation in a typical systolic array (Figure 6(b)). However, we observe the fact that, in the last few layers of the CNN model, although the size of the output feature map R is small, the number of output channels M is large. So we consider unrolling the data on the channels as much as possible instead of on the feature maps.

To achieve this optimization, we split the array into the top and bottom parts. Figure 10 shows the toy model. After we divide the array into two halves, we provide an additional data path for the weight data. At this time, the top data path can provide S sets of weight data, and the bottom data provides another S sets of weight data. So the horizontal PE utilization rate of the array becomes:

$$Util_{horizontal}^{split} = \frac{M}{2S} \bigg/ \left\lceil \frac{M}{2S} \right\rceil. \tag{16}$$

In addition, the input feature map is still passed in from the left side of the array, and the input data of the two arrays is the same. This means that the size of the input feature maps required by these arrays is reduced by half. We can also obtain the vertical PE utilization rate at this time:

$$Util_{vertical}^{split} = \frac{R^2}{S/2} \bigg/ \left\lceil \frac{R^2}{S/2} \right\rceil = \frac{2R^2}{S} \bigg/ \left\lceil \frac{2R^2}{S} \right\rceil. \tag{17}$$

Then, we compare the PE utilization rate of the split arrays with the original array.

$$\frac{Util^{split}}{Util} = \frac{Util_{horizontal}^{split} Util_{vertical}^{split}}{Util_{horizontal} Util_{vertical}} \tag{18}$$

Since M is much larger than S,

$$Util_{horizontal}^{split} = \frac{M}{2S} \bigg/ \left\lceil \frac{M}{2S} \right\rceil \rightarrow \frac{M}{S} \bigg/ \left\lceil \frac{M}{S} \right\rceil \rightarrow 1 \tag{19}$$

So the equation can be simplified to:

$$\frac{Util^{split}}{Util} = \frac{Util_{vertical}^{split}}{Util_{vertical}} = \frac{\frac{2R^2}{S} \Big/ \left\lceil \frac{2R^2}{S} \right\rceil}{\frac{R^2}{S} \Big/ \left\lceil \frac{R^2}{S} \right\rceil} = \frac{2 \left\lceil \frac{R^2}{S} \right\rceil}{\left\lceil \frac{2R^2}{S} \right\rceil} \tag{20}$$

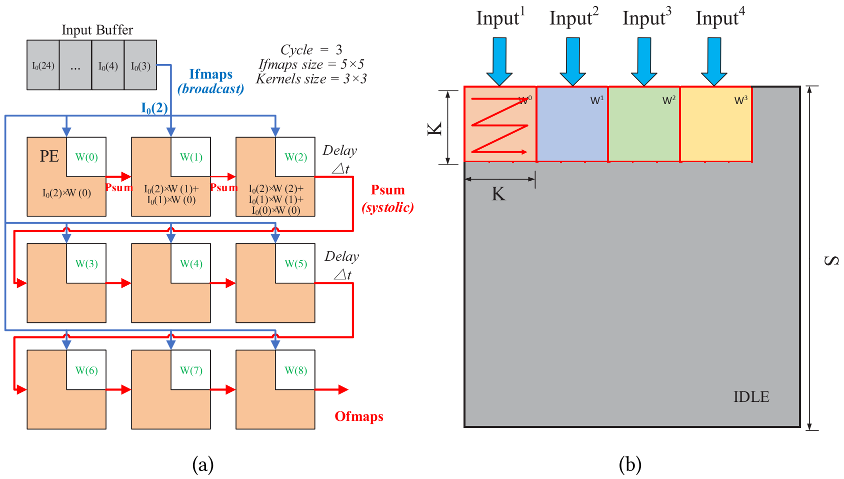


Fig. 11. A two-dimensional array with *WS* dataflow. (a) The 3 × 3 array uses the *WS* dataflow to process the convolution of the 5 × 5 ifmaps and the 3 × 3 kernels. (b) Due to the limitation of the 2-D structure, deploying *WS* dataflow in the systolic array directly will cause a large number of PEs to be idle.

Let $t = R^2/S$,

$$\frac{Util^{split}}{Util} = \frac{2\lceil t \rceil}{\lceil 2t \rceil} \tag{21}$$

$$2\lceil t \rceil = \begin{cases} 2t\ (t \in Z) \\ 2(t+1)(t \notin Z) \end{cases},\ \lceil 2t \rceil = \begin{cases} 2t\ (t = 0.5n, \exists n \in Z) \\ 2t+1(t \neq 0.5n, \forall n \in Z) \end{cases} \tag{22}$$

So we can get the final result,

$$\frac{Util^{split}}{Util} = \frac{2\lceil t \rceil}{\lceil 2t \rceil} \geqslant 1\ ,\ Util^{split} \geqslant Util. \tag{23}$$

Therefore, in theory, dividing the array can improve performance. CMSA is designed based on this observation. Since the data transmission is bidirectional in the array and there is an additional data path connecting the bottom of the array, CMSA can be easily divided into two parts for parallel operations, as shown in Figure 10.

4.2 Optimization for Depthwise Convolution

Because of the correlation between the output feature channels and the input feature channels, the conventional systolic array with *OS* dataflow cannot effectively calculate DWConv (Figure 6(c)). So we design a novel architecture to calculate DWConv, and it can be integrated with the existing systolic array architecture.

Figure 11(a) shows a two-dimensional (2-D) array with *WeightStationary* (*WS*) dataflow [32]. *WS* dataflow, as the name suggests, maps each element of kernels to a given PE. PEs store the weight data in advance and ifmaps data is broadcast to all PEs every cycle. Meanwhile, the partial sum needs to pass every PEs in each line and pass to the next line after the Δt delay, where $\Delta t = Ifmaps\ size - Kernels\ size$ [32].

It can be seen that the *WS* dataflow unrolls the weight data on the array (loop L5 and L6 in Figure 3(b)), but has nothing to do with loop L1. Meanwhile, due to the small size of the array(just $K \times K$), we can map multiple sets of the 2-D arrays in the original array. It is equivalent to unrolling the loop L2, which can further improve the efficiency of the array.

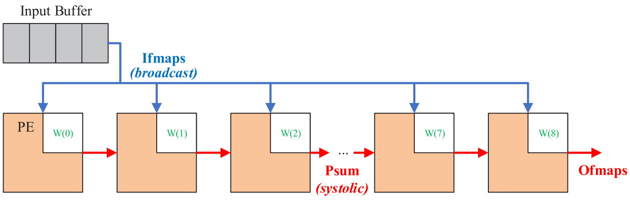
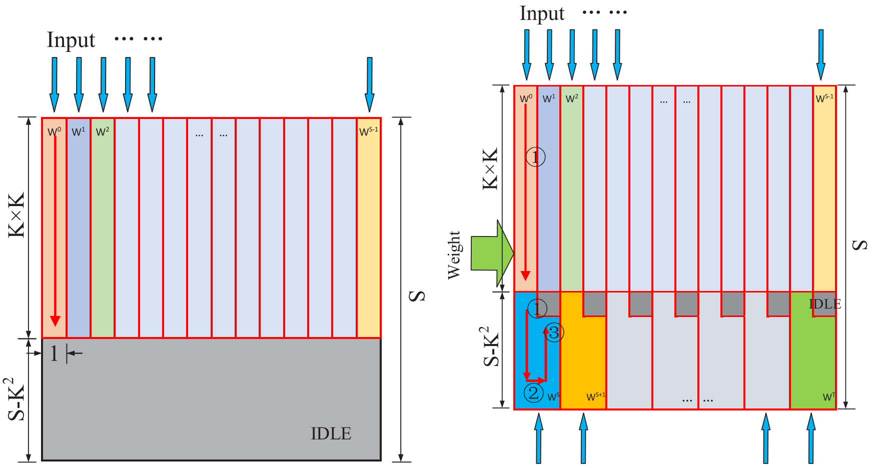


Fig. 12. The architecture of the PE array with 1-D *WS* dataflow.



(a)

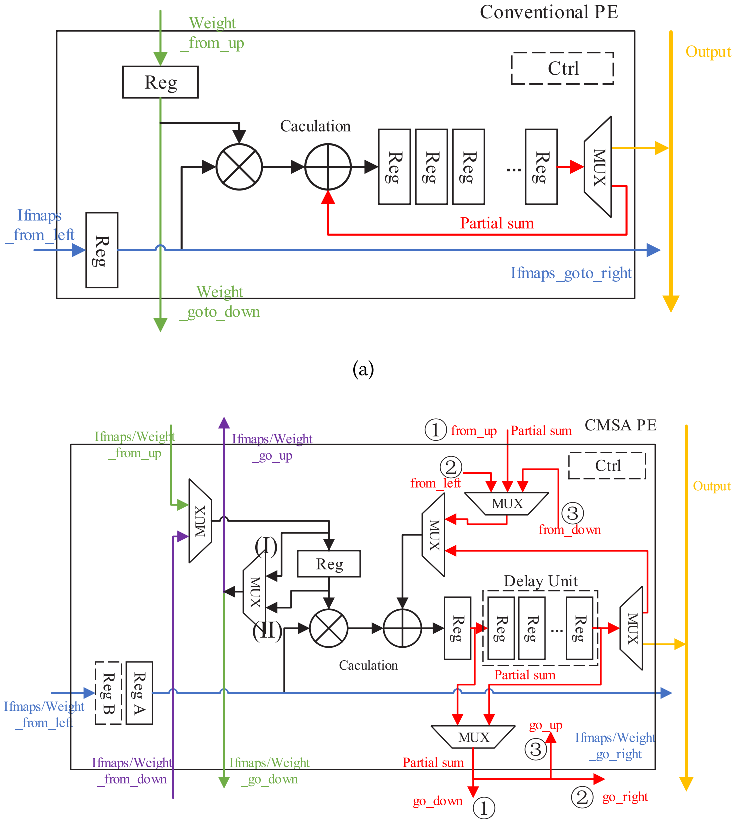
(b)

Fig. 13. CMSA uses the one-dimensional WS dataflow in the systolic array. (a) If we directly deploy the one-dimensional WS dataflow in the PE array, some idle PEs will appear. (b) We choose to fold the WS dataflow to utilize the PE array fully.

However, due to the limitation of data paths in the traditional systolic array, input data can only be passed in from the left or top of the array. If using conventional 2-D WS arrays directly, it leads to lots of idle PEs, as shown in Figure 11(b). Also, *WS* dataflow has a complex data stream direction, and partial sum needs to be transmitted to the next line, resulting in an unbalanced data transmission distance, which is not conducive to implementation.

Therefore, we proposed the one-dimensional (1-D) *WS* dataflow, as shown in Figure 12. We straighten the 2-D *WS* dataflow so that the data transmission direction of the partial sum becomes one-way, and the transmission distance between the PEs becomes balanced. Moreover, The size of a straightened *WS* array becomes $1 \times K^2$, which is more advantageous for scale-up, as shown in Figure 13(a). In addition, the register bank inside the PE is used as a delay unit.

In most cases, the kernel size of DWConv is just 3 [4, 14]. It means that the length of the WS array is only 9. However, in most designs, the array size S tends to be greater than 9, and the remaining height $(S - 9)$ usually cannot accommodate a new *WS* array. To utilize the remaining PE units, we fold the 1-D *WS* array, as shown in Figure 13(b). The cost of folding the array is that the partial sum transmission needs to support three transmission directions, respectively, ①up, ②down, and ③left. Although using three directions, PEs in the array still transmits data to their neighbors, so the data transmission distance is still balanced.



(a)

(b)

Fig. 14. The structure of the (a) conventional PE and (b) CMSA PE.

The WS dataflow costs the additional overhead of updating weight before computation. So we add an extra register in the PE to form the double buffers. It can use the computation process to hide the overhead of the weight update. We describe the details in Section 4.3.

4.3 Processing Elements Design

For a systolic array to support the above designs, we must make modifications to PEs in the array. Figure 14(a) shows the structure of a conventional PE [42]. It passes the weight data down and passes the ifmaps data left to adjacent PEs. The PE also accumulates the product of the ifmaps and weight data, and the internal register bank temporarily stores the partial sum [42]. The traditional PE design is simple and efficient, but it is not flexible and cannot meet our requirements to efficiently support small-scale and depthwise convolution.

We design a novel PE that allows flexible switching of the dataflow and data transmission direction (Figure 14(b)). We add more data paths in the PE to support the multi-directional transmission of data. Meanwhile, the type of data is no longer assigned to a given data path. For example, in *OS* dataflow, the ifmaps data can flow into the PE from the left to the right. In WS dataflow, the weight data flows from the left to the right.

In addition, there are two data transmission modes in the PE of CMSA. One is the broadcast mode, corresponding to the label(I) in Figure 14(b). It is used to support *WS* dataflow that needs

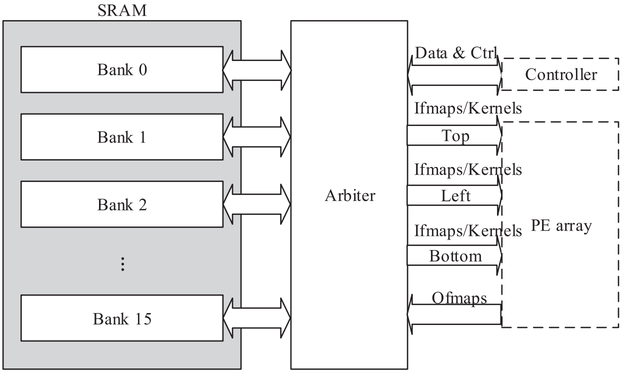


Fig. 15. The architecture of the global buffer.

to broadcast ifmaps data. The other is systolic mode, corresponding to the label(II) in Figure 14(b). It is used to support *OS* dataflow. We also add an extra data path for the partial sum. The position of the PE in the array determines its transmission direction. For example, if a PE is in the position that is indicated by the label① in Figure 13(b), then we choose the upward direction for this PE. Besides, we add an extra register in the horizontal data path to hide the overhead of updating the weight. Users can configure the controller units to switch the dataflow or data transmission direction.

4.4 Global Buffer

Figure 15 shows the architecture of the global buffer. The global buffer is mainly composed of SRAM and an arbiter. The SRAM consists of a set of banks, which can be configured and grouped into ifmaps, kernels, and ofmaps buffers according to the *Ctrl* signal from the controller. Unlike previous works, this design can improve the buffer utilization rate [5]. The arbiter is responsible for managing banks in SRAM. It is also responsible for reading/writing data and distributing data to the PE array. Since CMSA needs to transmit data to the PE array from three directions (top, left, and bottom), users can configure the arbiter to determine the type of data on the data path in different directions.

5 EVALUATION

In this section, we verify the mathematical model mentioned in Section 3. Then we present the detail of our experimental results. We implement the CMSA architecture in a cycle-accurate simulator and compare the performance with the traditional systolic array (the TPU-like systolic array). We get the PE utilization rate and performance of them under the typical workloads. We also compare the area and energy consumption between CMSA and the traditional systolic array. At the end of the experiment, we also discussed the impact of the large-size array on our design. The experimental results are divided into five parts: (1) Mathematical model, (2) Small-scale convolution (ResNet-18), (3) Depthwise convolution (MobileNet and MobileNet V2), (4) Area and energy consumption, and (5) The effect of the large-size array.

5.1 Experimental Setup

Workloads. In our experiments, we use three typical workloads—ResNet-18 [12], MobileNet, and MobileNet V2 [14]. ResNet-18 is very famous because of its high accuracy. Meanwhile, MobileNet and MobileNet V2 are widely used on mobile or embedded devices because of their small model

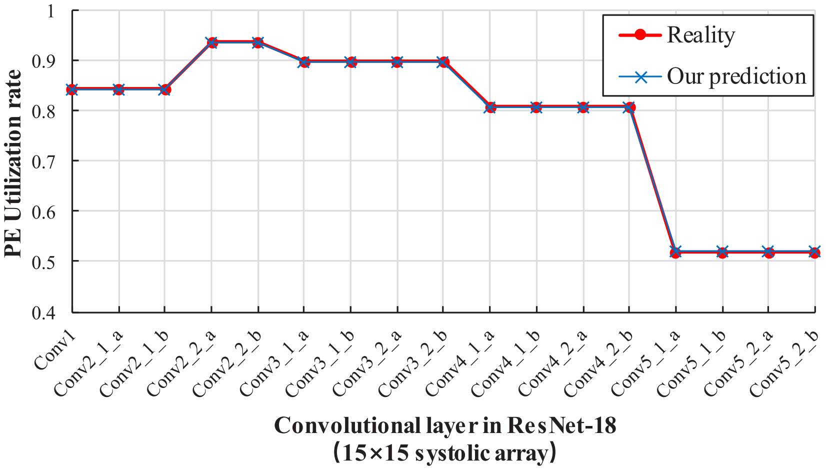


Fig. 16. Comparison between the predicted utilization rates and the actual rates. Our prediction results are all calculated through the mathematical model in Section 3.

size. They reflect two trends in the current development of CNNs – increasing the number of layers and becoming more complex. Table 1 lists the structure of these workloads. In order to better demonstrate the advantages of CMSA, we chose to use the ResNet-18 trained on the Cifar-10 dataset.

Evaluation. We simulate CMSA and the traditional systolic array with SCALE-Sim [35], which is a configurable systolic array based cycle-accurate DNN accelerator simulator. By using this simulator, we can get accurate information on the runtime of the array and the PE utilization rate. We also implement these designs in RTL, which is synthesized under a commercial 65-nm library by Synopsys Design Compiler to analyze area and energy consumption.

5.2 Experimental Results

Mathematical Model. We first verify our analytical model. We use the analytical model to predict the changes in the utilization rates of a 15 × 15 systolic array during processing ResNet-18. We record the utilization rates of each convolutional layer in ResNet-18 and then compare it to the actual results. Figure 16 shows the experimental results. The error between the predicted utilization rate and the actual rate is very small. It proves the effectiveness of our analytical model and verifies the correctness of our analysis for systolic arrays.

Small-Scale Convolution. To verify that CMSA can improve the efficiency of processing small-scale convolutions, we compare the PE utilization rate of the traditional systolic array and CMSA when processing ResNet. We also consider the effect of the size of the PE array on the PE utilization rate. Figure 17 shows the PE utilization rate of the traditional systolic array and CMSA in processing small-scale convolutional layers of ResNet with different array sizes. It can be seen that when a traditional systolic array computing small-scale convolution, the PE utilization rate fluctuates and decreases. And when the array size is larger, the decrease is more obvious. When the array size is 15 × 15, the difference in PE utilization rate of different convolutional layers is nearly 40%. The reason for this phenomenon is that the larger array size, the more idle PEs.

CMSA eliminates this decrease as much as possible. The experimental results show that CMSA improves the PE utilization rate when processing small-scale convolution. When the array size

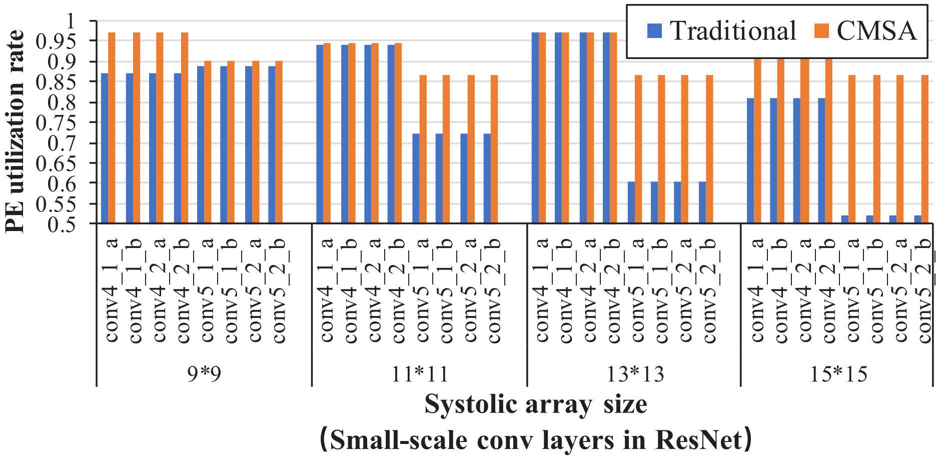


Fig. 17. The PE utilization rate of the traditional systolic array and CMSA in processing small-scale convolutional layers of ResNet with different array sizes.

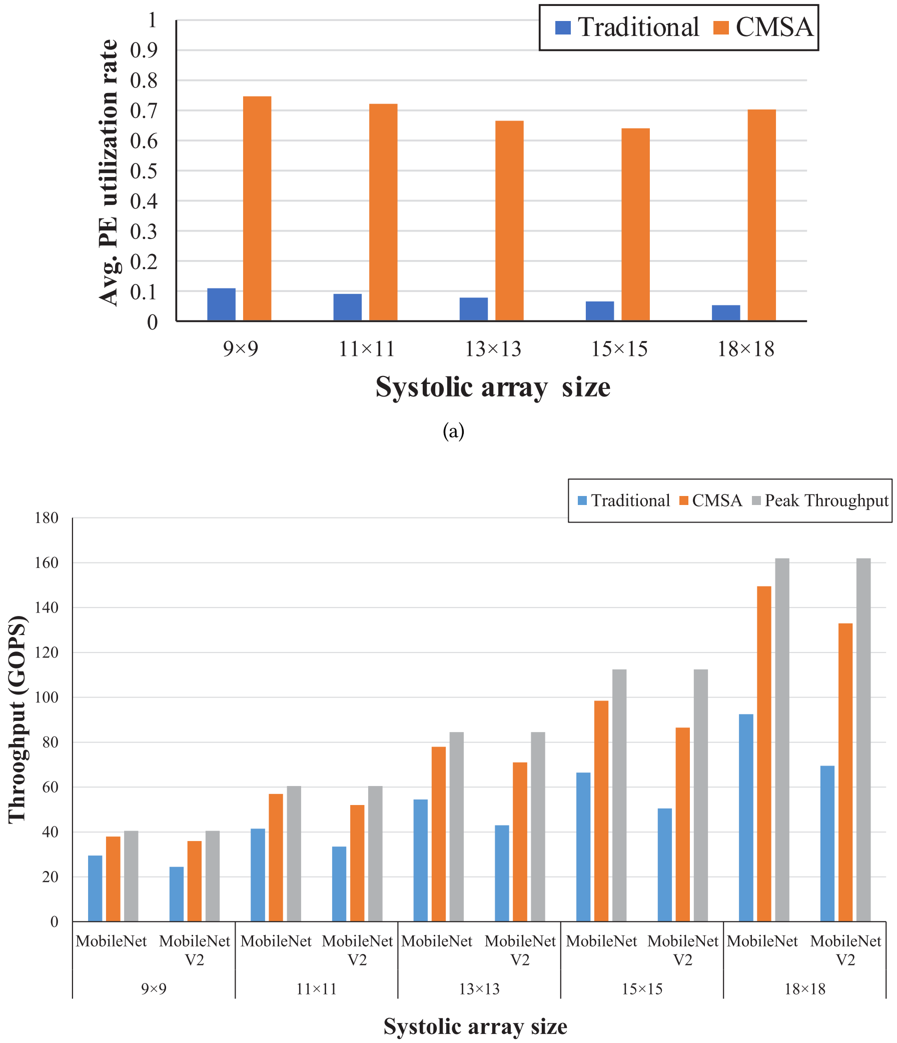
is 15 × 15, CMSA increases the utilization rate of last layers from the original 52% to 86%, nearly 1.67×. These results demonstrate the efficiency of CMSA in computing small-scale convolution.

Depthwise Convolution. We evaluate the performance of CMSA when processing depthwise convolution. Figure 18(a) shows the average PE utilization rate of the traditional systolic array and CMSA when processing DWConv layers of MobileNet with different array sizes. The traditional systolic array is inefficient in dealing with DWConv. The PE utilization rate of the typical systolic array is usually less than 10% and further decrease as the array size increases. This is due to that the horizontal utilization of the array is now $1/S$, and the larger the S, the lower the PE utilization rate.

CMSA significantly improves the PE utilization rate of the array. It increases the utilization rate from less than 10% to about 70%. When the array size is 18 × 18, it can even increase the utilization rate by 14.8 times. When CMSA calculates the depthwise convolution, it changes the dataflow used in the PE array, which significantly improves the PE utilization.

The increase in the PE utilization rate brings performance improvements. Figure 18(b) shows the comparison of throughput between the traditional systolic array and CMSA. In order to fully demonstrate the advantages of CMSA, we set the frequency of both systolic arrays to 500MHz and choose two CNNs models—MobileNet and MobileNet V2. Compared with the traditional systolic array, the throughput rate of CMSA is always the closest to the peak performance. Through the acceleration of the DWConv layers and the optimization for small-scale convolution, CMSA can achieve a total speedup of 1.63 times when computing MobileNet and 2 times when computing MobileNet V2 compared with the traditional systolic array. In addition, we observe an interesting phenomenon. When the array size increases, the speedup becomes larger. It is because the PE utilization rate of 1-D WS array is stable at about 70%, while the PE utilization rate of the traditional systolic array will decrease with the increase of array size.

Area and Energy. We compare the area and energy consumption of CMSA with the traditional systolic array. Figure 19(a) shows the experimental results of the area increasing with the size of the array in CMSA and traditional systolic array. Compared with the traditional design, the area of CMSA has increased by about 3.4% on average, because CMSA just adds additional registers and data paths in PEs. Compared to the huge area of SRAM or Buffers, these can be ignored.



(a)

(b)

Fig. 18. (a) The average PE utilization rate of the traditional systolic array and CMSA in processing DWConv layers of MobileNet with different array sizes. (b) The comparison of throughput between the traditional systolic array and CMSA in processing MobileNet and MobileNet V2.

Figure 19(b) shows the experimental results of the energy consumption increasing with the size of the array in CMSA and traditional systolic array. For the convenience of comparison, we normalized the experimental results. When the array is small, the energy consumption of CMSA is similar to the traditional systolic array. However, when the array is large, thanks to the increase of PE utilization, the energy consumption of CMSA has been reduced by up to 20% compared with the traditional array.

The Large-Size Array. In Section 3.1, our analytical model shows that when the array size S becomes larger, the utilization rate will be correspondingly smaller. In actual design, the large-size array is the challenge for both the traditional systolic array and CMSA. Figure 20 shows the

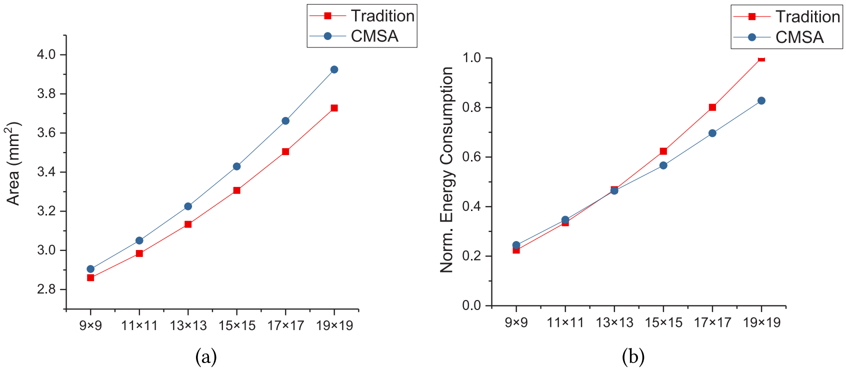


Fig. 19. Comparison of (a) area and (b) energy consumption between CMSA and the traditional systolic array.

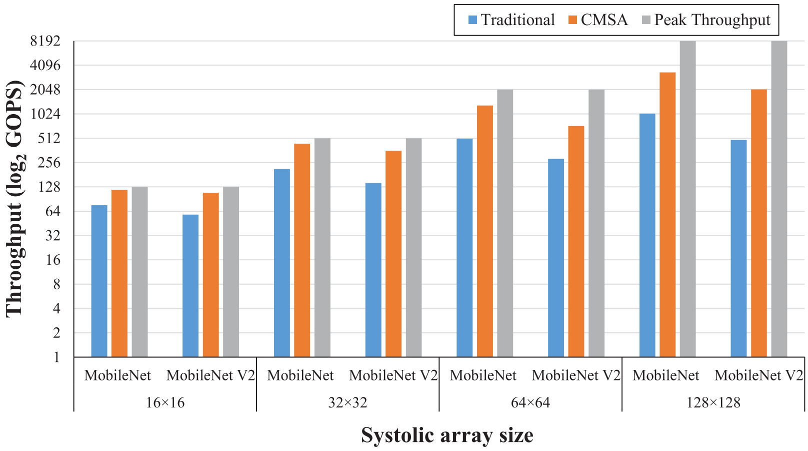


Fig. 20. The throughput of the traditional systolic array and CMSA with different array sizes.

the throughput of the traditional systolic array and CMSA with different array sizes. Obviously, the larger the array size, the greater the gap between the throughput and peak performance of the array when calculating MobileNet or MobIieNet V2. But CMSA is always better than the traditional systolic array, and the throughput rate has been increased by 1.6x-4.2x compared with the traditional systolic array.

It should be noted that since MobileNet mostly runs on embedded or mobile terminals/devices, the array size would not be so large. At the same time, this scale-up method of the systolic array is not recommended because it will bring great performance loss [34].

6 CONCLUSION

Due to the simple and efficient architecture characteristics of the systolic array, its performance is high when calculating the traditional convolution, but it faces challenges when calculating the emerging special types of convolution.

In order to analyze these problems, we first establish a mathematical model to measure the PE utilization rate of the systolic array. Based on this mathematical model, we can explain why the performance of the systolic array declines when calculating small-scale and depthwise convolution.

At the same time, we can also accurately obtain the PE utilization rate of the systolic array in various situations.

Then, we design a CMSA to solve these problems. We add an additional data path to the systolic array to split the array for improving efficiencies of computing small-scale convolution. Besides, we redesign the structure of PEs so that the systolic array can flexibly change the dataflow to calculate depthwise convolution more efficiently. Experimental results show that, compared with the typical systolic array, CMSA can increase the utilization rate by up to 1.6 times when computing small-scale convolution. When computing depthwise convolution, CMSA can increase the utilization rate by up to 14.8 times. Meanwhile, CMSA consumes similar area and energy with the traditional systolic.

REFERENCES

[1] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS'14),* (Salt Lake City, UT, March 1–5 2014). Rajeev Balasubramonian, Al Davis, and Sarita V. Adve (Eds.). ACM, 269–284. https://doi.org/10.1145/2541940.2541967

[2] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid State Circuits* 52, 1 (2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[3] Yu-Hsin Chen, Tien-Ju Yang, Joel S. Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Sel. Topics Circuits Syst.* 9, 2 (2019), 292–308. https://doi.org/10.1109/JETCAS.2019.2910232

[4] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17), (*Honolulu, HI, July 21-26, 2017). IEEE Computer Society, 1800–1807. https://doi.org/10.1109/CVPR.2017.195

[5] S. Das, A. Roy, K. K. Chandrasekharan, A. Deshwal, and S. Lee. 2020. A systolic dataflow based accelerator for CNNs. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS45731.2020.9180403

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'09), 20-25 June 2009,* (Miami, FL, June 20–25 2009). IEEE Computer Society, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[7] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *42nd Annual International Symposium on Computer Architecture* (Portland, OR, June 13-17, 2015), Deborah T. Marr and David H. Albonesi (Eds.). ACM, 92–104. https://doi.org/10.1145/2749469.2750389

[8] Hasan Genc, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Wright, Colin Schmidt, Jerry Zhao, Albert J. Ou, Max Banister, Yakun Sophia Shao, Borivoje Nikolic, Ion Stoica, and Krste Asanovic. 2019. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *CoRR* abs/1911.09925 (2019). arXiv:1911.09925 http://arxiv.org/abs/1911.09925.

[9] Dibakar Gope, Jesse G. Beu, and Matthew Mattina. 2020. High throughput matrix-matrix multiplication between asymmetric bit-width operands. *CoRR* abs/2008.00638 (2020). arXiv:2008.00638 https://arxiv.org/abs/2008.00638.

[10] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *32nd International Conference on Machine Learning (ICML'15),* (Lille, France, July 6–11, 2015) *(JMLR Workshop and Conference Proceedings)*, Francis R. Bach and David M. Blei (Eds.), Vol. 37. JMLR.org, 1737–1746. http://proceedings.mlr.press/v37/gupta15.html.

[11] Xing Hao, Guigang Zhang, and Shang Ma. 2016. Deep learning. *International Journal of Semantic Computing* 10, 03 (2016), 417–439.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)* (Las Vegas, NV, June 27-30, 2016*)*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

[13] Xin He, Subhankar Pal, Aporva Amarnath, Siying Feng, Dong-Hyeon Park, Austin Rovinski, Haojie Ye, Kuan-Yu Chen, Ronald G. Dreslinski, and Trevor N. Mudge. 2020. Sparse-TPU: Adapting systolic arrays for sparse matrices. In *2020 International Conference on Supercomputing (ICS'20)* (Barcelona Spain, June 2020), Eduard Ayguadé, Wen-mei W. Hwu, Rosa M. Badia, and H. Peter Hofstee (Eds.). ACM, 19:1–19:12. https://dl.acm.org/doi/10.1145/3392717.3392751.

[14] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861.

[15] Nandan Kumar Jha, Shreyas Ravishankar, Sparsh Mittal, Arvind Kaushik, Dipan Mandal, and Mahesh Chandra. 2020. DRACO: Co-Optimizing hardware utilization, and performance of DNNs on systolic accelerator. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'20),* (Limassol, Cyprus, July 6-8, 2020). IEEE, 574–579. https://doi.org/10.1109/ISVLSI49217.2020.00088

[16] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, 2017. In-datacenter performance analysis of a tensor processing unit. In *44th Annual International Symposium on Computer Architecture (ISCA'17)* (Toronto, ON, Canada, June 24-28, 2017). ACM, 1–12. https://doi.org/10.1145/3079856.3080246

[17] A. Krizhevsky and G. Hinton. 2009. Learning multiple layers of features from tiny images. *Handbook of Systemic Autoimmune Diseases* 1, 4 (2009).

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90. https://doi.org/10.1145/3065386

[19] H. T. Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. In *24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)* (Providence, RI, April 13-17, 2019), Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, 821–834. https://doi.org/10.1145/3297858.3304028

[20] H. T. Kung, Bradley McDanel, and Sai Qian Zhang. 2020. Term Revealing: Furthering quantization at run time on quantized DNNs. *CoRR* abs/2007.06389 (2020). arXiv:2007.06389 https://arxiv.org/abs/2007.06389.

[21] H. T. Kung, Bradley McDanel, Sai Qian Zhang, Xin Dong, and Chih-Chiang Chen. 2019. Maestro: A memory-on-logic architecture for coordinated parallel use of many systolic arrays. In *30th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'19)* (New York, NY, July 15-17, 2019). IEEE, 42–50. https://doi.org/10.1109/ASAP.2019.00-31

[22] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding reuse, performance, and hardware cost of DNN Dataflow: A data-centric approach. In *52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'19)* (Columbus, OH, October 12-16, 2019). ACM, 754–768. https://doi.org/10.1145/3352460.3358252

[23] Hyoukjun Kwon, Michael Pellauer, and Tushar Krishna. 2018. MAESTRO: An open-source infrastructure for modeling dataflows within deep learning accelerators. *CoRR* abs/1805.02566 (2018). arXiv:1805.02566 http://arxiv.org/abs/1805.02566.

[24] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. In *23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)* (Williamsburg, VA, March 24-28, 2018), Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar (Eds.). ACM, 461–475. https://doi.org/10.1145/3173162.3173176

[25] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 4 (1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

[26] Zhi Gang Liu, Paul N. Whatmough, and Matthew Mattina. 2020. Systolic tensor array: An efficient structured-sparse GEMM accelerator for mobile CNN inference. *IEEE Comput. Archit. Lett.* 19, 1 (2020), 34–37. https://doi.org/10.1109/LCA.2020.2979965

[27] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA'17)* (Austin, TX, February 4-8, 2017). IEEE Computer Society, 553–564. https://doi.org/10.1109/HPCA.2017.29

[28] Sangkug Lym, Armand Behroozi, Wei Wen, Ge Li, Yongkee Kwon, and Mattan Erez. 2019. Mini-batch Serialization: CNN training with inter-layer data reuse. In *Machine Learning and Systems 2019 (MLSys 2019)* (Stanford, CA, USA, March 31–April 2, 2019), Ameet Talwalkar, Virginia Smith, and Matei Zaharia (Eds.). mlsys.org. https://proceedings.mlsys.org/book/261.pdf.

[29] Sangkug Lym and Mattan Erez. 2020. FlexSA: Flexible systolic array architecture for efficient pruned DNN model training. *CoRR* abs/2004.13027 (2020). arXiv:2004.13027 https://arxiv.org/abs/2004.13027.

[30] Mostafa Mahmoud, Isak Edo Vivancos, Ali Hadi Zadeh, Omar Mohamed Awad, Gennady Pekhimenko, Jorge Albericio, and Andreas Moshovos. 2020. TensorDash: Exploiting sparsity to accelerate deep neural network training and inference. *CoRR* abs/2009.00748 (2020). arXiv:2009.00748 https://arxiv.org/abs/2009.00748.

[31] Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman P. Jouppi, and David A. Patterson. 2020. Google's Training Chips Revealed: TPUv2 and TPUv3. In *IEEE Hot Chips 32 Symposium (HCS'20),* (Palo Alto, CA, August 16-18, 2020). IEEE, 1–70. https://doi.org/10.1109/HCS49909.2020.9220735

[32] Phi-Hung Pham, Darko Jelaca, Clément Farabet, Berin Martini, Yann LeCun, and Eugenio Culurciello. 2012. NeuFlow: Dataflow vision processing system-on-a-chip. In *55th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'12)* (Boise, ID, August 5-8, 2012). IEEE, 1044–1047. https://doi.org/10.1109/MWSCAS.2012.6292202

[33] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'20)* (San Diego, CA, February 22-26, 2020). IEEE, 58–70. https://doi.org/10.1109/HPCA47549.2020.00015

[34] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'20),* (Boston, MA, August 23-25, 2020). IEEE, 58–68. https://doi.org/10.1109/ISPASS48437.2020.00016

[35] Ananda Samajdar, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN accelerator. *CoRR* abs/1811.02883 (2018). arXiv:1811.02883 http://arxiv.org/abs/1811.02883.

[36] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Ross Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel S. Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'19)* (Columbus, OH, October 12-16, 2019). ACM, 14–27. https://doi.org/10.1145/3352460.3358302

[37] Sayeh Sharify, Alberto Delmas Lascorz, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Dylan Malone Stuart, Zissis Poulos, and Andreas Moshovos. 2019. Laconic deep learning inference acceleration. In *46th International Symposium on Computer Architecture (ISCA'19)* (Phoenix, AZ, June 22-26, 2019) Srilatha Bobbie Manne, Hillery C. Hunter, and Erik R. Altman (Eds.). ACM, 304–317. https://doi.org/10.1145/3307650.3322255

[38] Gil Shomron, Tal Horowitz, and Uri C. Weiser. 2019. SMT-SA: Simultaneous multithreading in systolic arrays. *IEEE Comput. Archit. Lett.* 18, 2 (2019), 99–102. https://doi.org/10.1109/LCA.2019.2924007

[39] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR'15)* (San Diego, CA, May 7-9, 2015)*, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556.

[40] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *36th International Conference on Machine Learning (ICML'19),* (Long Beach, CA, June 9-15, 2019) *(Proceedings of Machine Learning Research),* Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 6105–6114. http://proceedings.mlr.press/v97/tan19a.html.

[41] Yu Wang, Gu-Yeon Wei, and David Brooks. 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. *CoRR* abs/1907.10701 (2019). arXiv:1907.10701 http://arxiv.org/abs/1907.10701.

[42] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *54th Annual Design Automation Conference (DAC'17)* (Austin, TX, June 18-22, 2017). ACM, 29:1–29:6. https://doi.org/10.1145/3061639.3062207

[43] Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qiaoyi Liu, Steven Bell, Jeff Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark Horowitz. 2018. DNN dataflow choice is overrated. *CoRR* abs/1809.04070 (2018). arXiv:1809.04070 http://arxiv.org/abs/1809.04070.

Received November 2020; revised February 2021; accepted April 2021