Heterogeneous Systolic Array Architecture for Compact CNNs Hardware Accelerators

Rui Xu, Sheng Ma, Yaohua Wang, Yang Guo, Dongsheng Li, and Yuran Qiao

Abstract—Compact convolutional neural networks have become a hot research topic. However, we find that the systolic array accelerators are extremely inefficient in dealing with compact models, especially when processing depthwise convolutional layers in the neural networks. To make systolic arrays more efficient for compact convolutional neural networks, we propose the heterogeneous systolic array (HeSA) architecture. It introduces heterogeneous processing elements that support multiple dataflows, which can further exploit the reuse data chance of depthwise convolutional layers and without changing the structure of the naive systolic array. By increasing the utilization rate of processing elements in the array, the HeSA improves the performance, throughput, and energy efficiency compared to the standard baseline. In addition, we design the flexible buffer structure for the HeSA. Through configuring it, the HeSA can allocate bandwidth flexibly to maintaining high performance and low communication cost. Based on our evaluation with typical workloads, the HeSA improves the utilization rate of the computing resource in depthwise convolutional layers by 4.5× - 11.2× and acquires 1.6 - 3.1× total performance speedup compared to the standard systolic array architecture. In the large-scale array design, the HeSA can reduce the data traffic by 40% while maintaining the same performance as the scaling-out method. By improving the on-chip data reuse opportunities and reducing data traffic, the HeSA saves over 20% in energy consumption. Meanwhile, the area of the HeSA is basically unchanged compared to the baseline due to its simple design.

Index Terms—hardware accelerator, architecture, convolutional neural network, depthwise separable convolution, systolic array

1 INTRODUCTION

The compact model is one of the popular convolutional neural networks (CNNs) compression approaches [1], which aim to reduce the cost of memory and computation in CNNs. Because the convolution is the most expensive part of CNNs, compact models or compact CNNs choose other simplified network components to replace the standard convolution (SConv). A popular choice is the depthwise convolution (DWConv) [1], [2], [3].

Depthwise convolution applies a single filter to each input channel, which means one spatial convolution is performed independently over each channel of an input feature map (ifmap) [1]. The fundamental hypothesis behind DWConv is that spatial and channel correlations can be sufficient decoupled and separately realized. Compared to SConv, DWConv saves lots of parameters and Multiply-Accumulate Operations (MACs) while maintaining high accuracy. Due to the advantages of DWConv, the compact CNNs using DWConv have become a hot research topic and a series of compact CNNs models have also been widely used [4], [5], [6], [7], [8], [9].

DWConv would bring fewer operations and less latency in the mobile/edge hardware accelerators. However, we find that the popular systolic array (SA) design is very inefficient when computing compact CNNs. [10], [11], [12], [13]. Fig. 1 shows the strange result. We count the amount of floating point arithmetics (FLOPs) in three state-of-art compact CNNs and record their latency breakdown in a 16×16 SA. We find that the FLOPs of DWConv in the model account for about 10% of the total, but lead over 60% of the latency. This result is related to the architecture and dataflow of the SA.

The SA architecture is composed of two-dimensional simple and homogeneous processing elements (PEs). By the dataflow mapping data to the array, the SA can process and accelerate matrix multiplications (GEMM) and accumulations, which account for more than 95% of the operations in the SConv layers of the traditional CNNs [14]. Usually, the scale of GEMM in SConv layers is large and multiples of the array size. Tiling and processing these GEMMs (grey boxes) can fully utilize PEs (red boxes) in the SA [15], as shown in Fig. 2a. However, the GEMM shrinks to the matrix-vector
multiplication (MV) in the DWConv layers. Thus, the tiles of MV lead to many idle PEs (white boxes), which significantly decreases the performance, as shown in Fig. 2.b.

Moreover, when the SA processes DWConv layers, the larger the size of the SA, the lower the PE utilization rate. Fig. 2c shows a typical situation. The performance of the SA is not only affected by the MV, but also limited by the size of the model. This will result in extremely low utilization. Therefore, the scalability of the SA will also affect the final performance of the hardware accelerator.

To solve the inefficiency of calculating DWConv, the dataflow of the SA must be changed [16]. However, the architecture of the SA is too simple. It only supports one fixed dataflow. Some related work [16], [17] try to change the mapping method and modify the architecture. But they all bring in too much hardware overhead. These changes wipe out the advantages of the SA design and are not suitable for deployment on resource-limited mobile/edge platforms.

To overcome these challenges, we propose the heterogeneous SA (HeSA), which can support multiple dataflows. Different from the previous solution [16], [17], the HeSA maintains the original structure of the SA and only makes changes to the PEs in the SA. When processing SConv, the HeSA still behaves as a traditional SA, maintaining the high on-chip reuse opportunities and low resource consumption. When dealing with DWConv, the HeSA switches the dataflow and data mapping of the SA by changing the data path in the heterogeneous PEs. The HeSA can increase the performance of DWConv by 4.5× - 11.2× without adding additional data paths or increasing external/internal bandwidth. And the HeSA can accelerate compact CNNs by 1.6 - 3.1× and improve 1.1× energy efficiency compared to a naive SA design.

At the same time, we delve into the scalability of the SA. The large-scale PE array often has a very low PE utilization rate when processing the compact CNNs. For this challenge, we choose to use a flexible buffer structure for the communication between the PE array and the external storage. Through flexible routing connection, our design not only guarantees a high PE utilization rate but also reduces the data traffic. Compared with the traditional scaling-up solution, the performance of the array is improved by nearly 2×. Compared with the radical scaling-out method, the data traffic is reduced by 40%. Part of this research was presented at DATE 2021 [18].

The main contributions of the paper are summarized as follows:

- We propose a new dataflow for the SA, which can process DWConv more efficiently. It increases the data reuse opportunities of DWConv in the SA design and can map more data to the PE array.
- We design the heterogeneous systolic array architecture to support multiple dataflows. Flexible switching of dataflows can make better use of data reuse opportunities in CNNs. In this way, accelerators can better process and accelerate compact CNNs.
- We design the flexible buffer structure which is responsible for the communication between the array and the external buffer. It ensures that when the array size is increased, the performance can be better improved, and the data traffic can be reduced as much as possible.
- Our experimental results demonstrate the validity of the HeSA design. We show the advantages of our design compared to the traditional SA in terms of the PE utilization rate, scalability, performance, area, energy consumption, etc.

2 BACKGROUND
2.1 Compact CNNs
CNNs use the convolutional layers to extract different features of the input data [19], [20]. The biggest difference between compact CNNs and traditional CNNs is the implementation of convolutional layers.

Algorithm 1 The 6-nested loop of SConv

Input: \(T[C, R + K, R + K]\) and \(W[M, C, K, K]\)

Output: \(O[M, R, R]\)

1: \(\text{for } m = 0 \text{ to } M \text{ do}\)
2: \(\text{for } c = 0 \text{ to } C \text{ do}\)
3: \(\text{for } r = 0 \text{ to } R \text{ do}\)
4: \(\text{for } r = 0 \text{ to } R \text{ do}\)
5: \(\text{for } k = 0 \text{ to } K \text{ do}\)
6: \(\text{for } k = 0 \text{ to } K \text{ do}\)
7: \(O[m, r, r]' = W[m, c, k, k] * T[c, r + k, r + k]\)

Standard Convolution (SConv) represents the standard convolution algorithm. There are multiple channels and filters (kernels or weights) to extract different local features of the ifmaps. The operation of SConv can be described as a 6-nested loop (regardless of batch size) [21], as seen in Algorithm 1, where \(m\) is the \(m\)-th channel of output feature maps (ofmaps, some papers [22], [23] also call \(m\) number of filters), \(c\) is the \(c\)-th channel of ifmaps, \(r\) is the height and width of ofmaps (height is equal to width), and \(k\) is the height and width of kernels. To facilitate computing and deployment, SConv is converted into GEMM through the \text{im2col} algorithm. The dimensions of GEMM are large due to the deep and wide structure of the traditional CNNs.

Depthwise Convolution (DWConv) replaces SConv in compact CNNs. Unlike SConv, where entire channels in a kernel are convolved with all ifmaps and produce one ofmap, DWConv only allows one filter-channel to convolve with only one ifmap to produce one ofmap [1]. This means
the number of filters is only one, and ifmaps data cannot be reused between the filters. The operation of DWConv can be described as a 5-nested loop, as shown in Algorithm 2, where the ofmaps channel is correlated to $c$ and loop $m$ is disappeared (Equivalent to $M=1$). In addition, using the im2col algorithm for DWConv (Fig. 3b) results in the matrix-vector multiplication (MV) instead of GEMM.

$$O[c, R, R] = O[c, r, r] + I[c, r, k, k] \times W[c, k, k]$$

Furthermore, composed by pointwise convolution (PWConv, small-scale SConv) and DWConv, compact CNNs can maintain high accuracy which even surpass the accuracy of dense CNNs [5], like MobileNetV3 [24], MixNet [4], EfficientNet [5], etc.

### 2.2 Systolic Arrays

Fig. 4 shows the architecture of the SA. They comprise several PEs (usually MAC units), and each PE stores the incoming data in an internal register and then forwards the same data to the neighbor PE [14]. These store and forward behaviors result in significant savings in external storage and read bandwidth, and can very effectively exploit reuse opportunities provided by GEMM [15]. Note that this data movement and reuse is achieved without the help of control or router units. These advantages make it a popular choice for accelerator design [15].

**Dataflows** decide the schedule of GEMM tiles in the SA. The Output-stationary (OS) dataflow is the commonly used one and achieve high ifmaps and weight data reuse opportunities [1]. Fig. 4 also shows the operation of the OS dataflow, where the two input matrices for a GEMM tile are shifted horizontally and vertically respectively, and the output matrix is fixed on the array.

### 2.3 Scalability

Scalability is important in the SA design. Intuitively, the more PEs the SA contains, the higher its performance will be. However, hardware scaling must also consider various issues such as the PE utilization rate, data traffic, area, and energy consumption. Therefore, it is a challenge for the SA to improve performance by increasing the array size.

Scaling-up [14], [15] is a common method. It is very simple and only enlarge the PE array size. Google’s TPU series uses this method to design the large-scale SA [25]. However, it is hard to map compact CNNs and small-scale workloads to the large-size SA. The bigger the array, the more idle PEs caused by compact CNNs. So the PE utilization rate of the SA always decreases when the array size becomes larger.

Scaling-out [14], [15], [16] can handle these problems. This method chooses to have more small arrays rather than making the arrays bigger. So, it can maintain a higher PE utilization rate. Some work [11], [12], [14] choose this method to design their accelerators. But scaling-out needs more hardware overhead and increases the data traffic.

### 2.4 Related work

Systolic arrays have become the main trend in hardware CNNs accelerators deployed in both edge devices [11], [12], [16], [26], [27] and servers used in data-centers [25], [28], [29], [30]. Google designed a deep learning accelerator TPU [25], [28] based on a SA and achieved success. This makes the two-dimensional SA a hot research field. NVIDIA also integrated the SA into the GPU architecture and proposed the Tensor Core architecture [29], [30] to accelerate deep learning. Coupled with low-precision calculations and network sparsity, Tensor Core architecture further improves the performance of the GPU and makes it the most popular deep learning accelerator. In academia, Pham et al. [10] deploy the Weight Stationary dataflow in the PE array and use the systolic mode to reuse data. However, because the array size is limited to the size of the kernels, its scalability is poor. Du et al. [11] use the OS dataflow to make the PE array more scalable. Its architecture is similar to a SA. By reusing data between PEs, their design achieves high performance. But it is not optimized for depthwise separable convolution. Lu et al. [31] and Chen et al. [26] propose a new dataflow 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 [15]. However, we find that these designs are relatively inefficient when computing Compact CNNs.

Some proposals try to overcome these problems. Chen et al. [16] and Liu et al. [17] try to change the dataflow of the SA to improve the PE utilization rate. However, both of them introduce a lot of storage units, routers, data paths, and control units, increase the hardware overhead, and break the original intention of simple and efficient designs of the SA. Jha et al. [13] try to change the structure of compact CNNs, and make them more suitable for the SA. But it needs to re-train a trained compact CNN and is impractical for edge/mobile devices.

Of course, there are other designs for the SA. For example, Kung et al. [32] and He et al. [33] use column combining to train and compress models, and Lym et al. [14] using structured compression to train CNNs. However, none of these designs consider the situation of compact CNNs.

In this paper, we consider the design of edge/mobile platforms with a small SA, because mobile devices are more likely to face the challenge of compact CNNs and their hardware resources are limited.

3 ARCHITECTURAL CHALLENGES FROM THE COMPACT CNNS

The architecture and dataflow of the traditional SA lead to the inefficiency of calculating DWConv. However, through experiments and analysis, we find that the structure of the SA still has room for optimization in compact CNNs.

![Fig. 5. (a) PE utilization rate of a 16×16 SA processing different Conv layers, and (b) the roofline model for each Conv layer of MobileNetV3.](image)

**3.1 Analysis**

For further analysis, we record the PE utilization rate of a 16 × 16 SA processing MobileNetV3, as shown in Fig. 5a. These experimental results are all obtained through a configurable systolic array-based cycle-accurate DNN accelerator simulator [15]. The PE utilization rate of most of the SConv layers exceeds 90% due to the high efficiency for processing GEMM of the SA. However, the average PE utilization rate of DWConv is only about 6% and even only 3% at the worst. It increases the calculation delay and reduces the performance of the accelerator.

To find a solution, we analyze the roofline model of the 16 × 16 SA. We sweep every layer of MobileNetV3, and Fig. 5b shows the roofline data points for the models. Most SConv layers are in the region of compute-bound and near the roofline, which means they make full use of the SA. DWConv layers are in the region of memory-bound, which means the reused space of data is not enough to activate every PE in the array.

More importantly, our roofline analysis shows that the performance of DWConv layers only accounts for 10% of the theoretical performance. It means that the current dataflow cannot make full use of the computing resources of the SA and the PE utilization rate of the DWConv layers has room for further improvement.

![Fig. 6. The ofmaps data space of the OS-M dataflow processing (a) SConv and (b) DWConv, and (c) the OS-S dataflow processing DWConv. The OS-M dataflow also decide how to map ofmaps and reuse ifmaps/weights data on the SA when processing (d) SConv and (e) DWConv, and (f) how the OS-S dataflow processing DWConv to map the ofmaps data. The arrows indicate that the data can be reused in this direction](image)

**3.2 Discussion of dataflows**

In section 2.2, we introduce how the OS dataflow schedules and maps data. Fig. 6a shows the data space of ofmaps when the S × S SA processes SConv layers with OS. It processes the ofmaps from S channels and S activations (pixels in ofmaps) at one time [21]. Meantime, it can achieve the high data reuse opportunities of ifmaps and weights (Fig. 6d). We call this OS version OS-M (M for Multi-channels).

However, as we describe in section 2.1, the DWConv has no opportunity to reuse data in the ofmaps channel dimension (between multiple filters). If adopting OS-M directly, the ofmaps data space is only S (activations) (Fig. 6b) for the SA. Fig. 6e also shows the ifmaps data cannot be shared between filters, resulting in a large number of idle PEs when the SA with the OS-M dataflow is processing DWConv.
So, we use a variant of the OS dataflow (OS-S), which processes ofmaps data from the same single channel at a time to maximize data reuse opportunities of DWConv [1] (S for Single-Channel). Fig. 6c shows the ofmaps data space of processing DWConv with the OS-S dataflow. Ideally, it can increase the reuse opportunities of output data by \( S \times \) compared with the OS-M dataflow.

To map the single-channel ofmaps data to the SA, not only needs horizontal ifmaps data transmission but also need to transfer it vertically [11]. Because computing one ofmap data requires ifmaps data from multiple rows and columns, this allows ifmaps data to be reused between multiple rows and columns of PEs (including horizontal and vertical direction). If the ofmaps data is directly mapped to the PE array according to its position, one PE needs the ifmaps data from the PEs of the next row (Fig. 6f). It cannot be achieved in the traditional SA which only supports input data propagating in a single direction. But the HeSA can support it, and we illustrate that in Section 4.1.

4 HETEROGENEOUS SYSTOLIC ARRAY

Fig. 7 shows the overview of the HeSA architecture. It is composed of a SA, on-chip buffers, a control unit, and the connection to the host device. The SA uses heterogeneous PEs to support the OS-M and OS-S dataflow. On-chip buffers provide ifmaps and weight data for the SA. The control unit communicates with the host device, moves data, and controls the work state of the SA. We describe these structures below.

4.1 Operation process of the HeSA

Since we adopt different dataflows in the SA, the traditional operating procedure must be adjusted and changed accordingly. The operation process and data mapping of the OS-M dataflow remain unchanged, so we do not discuss and explain in detail.

However, the data mapping of the OS-S dataflow is different from the OS-M dataflow. So we make many changes and adjustments to the SA operating procedure to adapt to the OS-S dataflow. The new operation process has two main features. First, the ifmaps data needs to be transmitted horizontally and vertically between PEs, as described in section 3.2. Second, the SA with the OS-S dataflow requires a preloading process to complete the mapping of the ifmaps.

To illustrate this process more clearly, we take a SA that applies the OS-S dataflow to compute a simple convolution as an example. Fig. 8a shows the convolution, which contains a \( 3 \times 3 \) ifmap, a \( 2 \times 2 \) kernel and a \( 2 \times 2 \) ofmap. The OS-S dataflow needs to map the ofmap data to the PE array. However, the ofmaps data are not directly mapped to the PEs according to their positions. They need to be rotated by 180° and then mapped, as shown in Fig. 8b. The advantage of this adjustment is that ifmaps can be reused and spread from top to bottom, rather than from bottom to top (Fig. 6f). In this way, there is no need to design a separate upward data path for ifmaps data.

After the data mapping is completed, the operation process of the PE array enters the preloading/computing phase. In order to allow the SA to support for the OS-S dataflow, we also modify the PE structure in the array. A traditional PE is composed of the weight register (REG1 in Fig. 8b), the input data register(s) (REG2 in Fig. 8b), and the MAC unit which is used to calculate and store output data in the traditional SA. However, the PEs require additional input data registers to cache ifmaps data that needs to be propagated vertically to other rows in the OS-S dataflow. So we add an extra register (REG3 in Fig. 8b) for PEs. It is noted that the PEs in the last row of the array do not need REG3.

Fig. 9 shows how the \( 2 \times 2 \) SA with the OS-S dataflow computes the convolution. For conciseness, we only describe the operation process of the six cycles. The accelerator can complete the work by repeating the process. In addition, we use arrows to indicate where the data will flow to at the next cycle.

Cycle #1: Data preparation. The external buffer prepares the input data. Note that the data is skewed (the data in the next buffer row has one more bubble than the previous row).

Cycle #i+1: Preloading. The PEs of the first row in the array start preloading the input data. At the same time, the external storage prepares the weight data for the next cycle (the number of cycles required for preloading is \( array_{width} - 1 \)). It should be noted that the weight data is the same for each column of the PEs.

Cycle #i+2: Calculation starts. \( PE_{00} \) and \( PE_{01} \) finish preloading data and they start the calculation. The PEs in the second row is still in the preloading phase because of the presence of skew in the input feature data. In addition, \( PE_{00} \) and \( PE_{01} \) also need to store input data of this cycle into the REG3 registers for data reuse for the next row of PEs.

Cycle #i+3: Calculation continues. At this cycle, \( PE_{00} \)
and PE$_{01}$ complete the calculation of the first row of input feature data and need the next row of input feature map data. So their input path switches to the storage above the array for the new row of input feature data. Meantime, PE$_{10}$ and PE$_{11}$ receive the data and start the calculation.

**Cycle #i+4:** Calculation continues. The buffer on the left side of the array continues to provide the input feature map data for PE$_{00}$ and PE$_{01}$. PE$_{10}$ and PE$_{11}$ switch to the next row of input feature map data so that the input data is provided by REG3 in the first row of PEs. At the same time, the data in REG3 is updated. (In this case, this update has no effect, but is necessary for larger kernel sizes.)

**Cycle #i+5:** Calculation is finished. PE$_{00}$ and PE$_{01}$ complete their calculation of this channel of feature maps and weights, so the external buffer starts preparing the data from the next channel of input feature maps and weights. It corresponds to entering a new preloading phase (Cycle #i'). The second row of PEs is still calculating and needs one more cycle to complete the calculation.

By pipeline and loop these phases, the SA with OS-S can finally load and calculate the entire workload.

In the toy example presented above, we can see the HeSA using heterogeneous PEs (the structure of PEs is different in different rows) to achieve multi-directional input data transformation to support OS-S. Unlike Chen et al. [16] and Liu et al. [17], the HeSA still uses the systolic data transformation mode, and it can also be used as a standard SA and support the OS-M dataflow. However, this toy example introduces some components, like REG3, vertical input data paths, and additional external storage for input data from the upside. These components increase the hardware overhead and bring challenges to the hardware accelerator design of edge/mobile devices.

### 4.2 Heterogeneous PEs

In section 4.1, we conclude that the HeSA PEs should have additional data paths and registers. But we can reuse some components in traditional PEs to hide this hardware overhead.

Fig. 10a shows the structure of the traditional PE. It reads the weight (input) data from up (left) and transfers the weight (input) data to down (right). The PE also accumulates the product of the input and weight data, and the Psum register stores the partial sum temporarily. After the PE computation has been finished, output data is passed into the output register and is shifted across vertical PEs to the corresponding buffer.

Since the data paths and registers for output vertical transmission are always idle during the computation, we reuse these components for the input vertical transmission.
Fig. 11. (a) The SA with the OS-S dataflow needs an extra storage unit for the top PEs, which brings the hardware overhead. (b) The HeSA design uses the top PE to replace the storage unit, which saves the hardware cost.

Fig. 10b shows the design of PEs in the HeSA. We connect the output data path with the input data path (red lines in Fig. 10b) and control the data transmission by the multiplexer (MUX, the control unit manipulates it). When the HeSA uses the OS-M dataflow, MUX will close the red path, and the structure of PE in the HeSA is the same as the traditional PE. When the HeSA uses the OS-S dataflow, the MUX will choose the red path, and the output register is also used to cache the input data. In addition, the SA with the OS-S still needs an extra storage unit (like the register set) with additional control logic and data paths for the top PEs (like PEs in the row0 of Fig. 9e, Cycle #i+4). It brings the additional hardware overhead to the SA architecture, as shown in Fig. 11a. To avoid this overhead in the HeSA design, we use the PEs on the top of the array as the register set (these PEs are no longer used for calculation) for preloading the input data for the next row PEs, as shown in Fig. 11b. Although affecting the performance, it saves the hardware cost of the storage and does not need to modify the SA architecture. Meantime, the performance penalty of this design is acceptable.

4.3 Buffer and controller

In the HeSA, on-chip local buffers adopt double buffering strategy. Double buffering enables the overlap of computation of the PEs with memory access and allows for very simple coarse-grain control of data transfers between buffers and memory.

The control unit is responsible for connection with the host device, data distribution, and arbitration in the standard SA. In the HeSA, the control unit can also control the data path in PEs to switch dataflows flexibly. In the compilation stage, we specify which the dataflow is used by the current layer of the network. Since we only add one MUX unit for each PE, there is only one more bit of control signal, and the overhead is negligible.

5 Scalability

Because the workload of the compact CNNs is usually small, it actually brings challenges to the scalability of the SA.
5.1 Scaling-up and scaling-out

There are two major ways to enlarge the design scale of the systolic array. One is scaling-up, as shown in Fig. 12a. It is a very simple method, and just need to add more PEs to the array. The advantage of scaling-up is the need for less bandwidth. When the array is scaled up by a factor of $N$, the bandwidth required by the array is only increased by $\sqrt{N}$. This reduces the data traffic between storage and computing arrays. In fact, it is equivalent to selecting a large-scale array for the accelerator and has been widely used in the industry [25].

However, there are problems with the scaling-up method. As the array becomes larger, the PE utilization rate of the SA decreases once the data size is small or the workload parallelism is limited, as shown in Fig. 2c. It is very unfriendly to the accelerator, especially for the inference task, which needs low latency. So maintaining a high PE utilization rate and performance is more important.

Another way of scaling is the scaling-out method, as shown in Fig. 12b. It uses many small arrays to replace a large-scale array in the design and is more suitable for compact CNNs. From another perspective, each PE array can be regarded as a separate computing node. So the design is more flexible in the process of data mapping. Compared with the scaling-up, the scaling-out can maintain a higher PE utilization rate and better performance on the less parallelism workload.

But the scaling-out design also has problems. Each array contains separate buffers, and such distributed storage means additional data read and write overheads (such as data replication and synchronization). In addition, the scaling-out design needs more bandwidth compared to scale-up. When the array is scaled by a factor of $N$, the bandwidth required by the array is also increased by $N$ times. What’s more, the accelerator with multiple arrays requires the host or control unit to allocate computation and data to the arrays reasonably, which is also a challenge.

5.2 Flexible buffer structure

In our design, We try to combine the advantages of both scaling-up and scaling-out while avoiding their shortcomings. So we design the flexible buffer structure (FBS) which uses the crossbar to connect multiple small arrays and buffers in the HeSA, and Fig. 13 shows the HeSA using the FBS design. It is very similar to the scaling-out design, but we optimize the buffer design and add the crossbar unit.

The crossbar unit has three connection modes, which are one-to-one unicast (Fig. 14a), one-to-two multicast (Fig. 14b), and one-to-all broadcast (Fig. 14c). In this way, we can make a combination of multiple arrays so that these arrays share one buffer, achieve unified storage space, and reduce the data traffic. Compared with the scaling-up design, it is more flexible, because by adjusting the routing scheme, a variety of array combinations can be simply implemented to adapt to more task scenarios. Compared with the scaling-out design, our design can make use of the data reuse and parallelism as much as possible to reduce buffer read and write costs.

Because it only needs to support three connection modes, the structure of the crossbar unit is very simple. Fig. 15 shows the crossbar structure. Users only need to configure the crossbar to switch the communication mode. For example, if we turn on the red path in Fig. 15, we realize the 1-4 broadcast.

To further illustrate the optimization, we take the $8 \times 8$ array scaling to the $16 \times 16$ array as an example. The scaling-out design uses four $8 \times 8$ small arrays, and the scaling-up design uses a $16 \times 16$ array. But the flexible buffer structure in the HeSA can realize the multiple allocations of the PE arrays, as shown in Fig. 16. It can achieve not only the array sizes of the scaling-up (Fig. 16) and scaling-out (f in Fig. 16) design, but also the PE array of other sizes. This is equivalent to the flexible switching between a large-scale array and multiple small-scale arrays according to the condition of the workload. Users can achieve this by properly configuring the crossbar in the flexible buffer structure, which is also shown in Fig. 16.

The biggest advantage of the FBS is that it realizes the adjustment and allocation of bandwidth through configuration. Fig. 17 shows the comparison of the normalized maximum bandwidth of the three scaling methods. Scaling-out has the largest maximum bandwidth for flexibility. Scaling-up has a small maximum bandwidth. Since FBS is configurable, it has the most flexible bandwidth options, ranging from the largest to the smallest bandwidth. It can even independently configure the bandwidth of the ifmaps and weights data according to the task requirements. This ensures our design can achieve better performance/energy efficiency than scaling-up and scaling-out.
We design two scenarios, the scaling-out and the flexible buffer structure (FBS). We can get accurate information on the runtime of the array accurate DNN accelerator simulator. By using this simulator, Sim [15], which is a configurable systolic array-based cycle-accurate simulator, to evaluate the power consumption of the HeSA design. We also compare the HeSA and other hardware accelerators. The configurations are listed in Table 1. To have a fair comparison, we make these parameters of implementations to be the same. We choose Gemmini [12] for the RTL design of the standard SA. Gemmini is an open source TPU-like systolic array generator, and can produce optimal RTL designs to the same. We choose Gemmini [12] for the RTL design. We also compare the HeSA and other hardware accelerators. The configurations are listed in Table 1. For a fair comparison, we make these parameters of implementations to be the same. We choose Gemmini [12] for the RTL design of the standard SA. Gemmini is an open source TPU-like systolic array generator, and can produce optimal RTL designs. For the energy consumption, we use dataflows. (Fig. 18) the PE utilization rate of different layers in MixNet with different dataflows. Fig. 18 shows the PE utilization rate of different layers of MixNet processed by an 8 × 8 SA with different dataflows. SA-OS-M represents a standard SA that only uses the OS-M dataflow. SA-OS-S represents a variant SA that only uses the OS-S dataflow [11], and the HeSA is our design.

The results show that for SConv layers (R × R K × K PW in Fig. 18), the average PE utilization rate in SA-OS-M is about 90%, while the PE utilization rate in SA-OS-S is relatively low, most of which are only 70%. However for DWConv layers (R × R K × K DW in Fig. 18), the PE utilization rate in SA-OS-M is only about 11%, while the PE utilization rate in SA-OS-S is maintained at more than 45%, and the maximum even reaches 75%. The HeSA always keeps the high PE utilization rate of each layer by switching dataflows according to different scenarios.

For further evaluation, we show the DWConv and total PE utilization rate of different compact CNNs in an 8 × 8
The PE utilization rate of the (a) 8 × 8, (b) 16 × 16 and (c) 32 × 32 standard SA and HeSA.

Fig. 19. The PE utilization rate of the (a) 8 × 8, (b) 16 × 16 and (b) 32 × 32 standard SA and HeSA.

7.2 Performance

The increase in PE utilization brings about an increase in performance. We sum up the latencies of each layer, and Fig. 21 shows the speed-up ratio obtained by the HeSA compared to the standard SA. The HeSA can get an average 4.5× - 11.2× speed-up when processing the DWConv layer compared to the standard SA, and the total performance is 1.6× - 3.1× better.

To further show the advantages of the HeSA, we also evaluate the GOPs of the standard SA and HeSA. When processing compact CNNs, the standard SA suffers a significant performance loss, and their performance only reaches 30.9 GOPs (8 × 8 PEs, 48% peak performance), 76.3 GOPs (16 × 16 PEs, 29.8% peak performance), and 170.9 GOPs (32 × 32 PEs, 16.7% peak performance) respectively. The larger the array size, the greater the loss. The HeSA makes up for the performance loss, so its average performance on workload reaches 50.3 GOPs (8 × 8 PEs), 197.5 GOPs (16 × 16 PEs), and 525.3 GOPs (32 × 32 PEs) respectively.

7.3 Area

We layout the HeSA with the FBS design (16 × 16) and the total area of it is 1.84 mm². We compare the area of HeSA with other accelerators, as shown in Fig. 22. Thanks to the simple architecture, the standard SA has the smallest area. The area of HeSA only increases by 3% compared to the standard SA. Because the HeSA only adds some control units and data paths, and there are no extra storage or computing requirements. Eyeriss has the largest area. Although adopting a systolic structure, Eyeriss requires huge storage in each PE. Fig. 22 also shows the area breakdown. The PEs in Eyeriss take over half of the total area, which is 2.7× larger than that in the standard SA and HeSA.
and weight data of the network models, so it is difficult to benefit from the optimization. In addition, the energy consumption of the on-chip network and control units is so small.

8 Conclusion

We design a heterogeneous SA architecture (HeSA) for CNNs hardware accelerators, which can switch different dataflows and exploit the reused space of compact CNNs. It greatly mitigates the mismatch between SA architecture and depthwise convolutional layers of compact CNNs. Tested by typical workloads, our design provides 4.5× - 11.2× and 1.6× - 3.1× performance speed-up over the baseline in processing DWConv layers and total models. Besides, our design maintains the simple and efficient design principle without changing any structure of the array. Therefore, the area of the HeSA is basically the same as the standard SA. Thanks to the increase of reuse opportunities, the energy efficiency of the HeSA is increased by about 10% over the baseline.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work is supported in part by the NSFC (61802420, 62025208, 62172430), the NSF of Hunan Province (2021JJ10052), and the STIP of Hunan Province (2019RS2027).

References

[1] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model compression and hardware acceleration for neural networks: A comprehensive survey,” Proc. IEEE, vol. 108, no. 4, pp. 485–532, 2020. [Online]. Available: https://doi.org/10.1109/JPROC.2020.2976475
[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” arXiv preprint arXiv:1704.04861, 2017.
[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.
[4] M. Tan and Q. V. Le, “Mixconv: Mixed depthwise convolutional kernels,” in 30th British Machine Vision Conference 2019, Cardiff UK, September 9-12, 2019. BMVA Press, 2019, p. 74.
[5] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in International Conference on Machine Learning. PMLR, 2019, pp. 6105–6114.
[6] R. Huang, J. Pedoem, and C. Chen, “Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers,” in 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018, pp. 2503–2510.
[7] M. J. Shaifee, B. Chywl, F. Li, and A. Wong, “Fast yolo: A fast you only look once system for real-time embedded object detection in video,” Journal of Computational Vision and Imaging Systems, vol. 3, no. 1, 2017.
[8] K. K. Singh and A. Singh, “Diagnosis of covid-19 from chest x-ray images using wavelets-based depthwise convolution network,” Big Data Mining and Analytics, vol. 4, no. 2, pp. 84–93, 2021.
[9] D. Haase and M. Amthor, “Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets,” in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020. Computer Vision Foundation / IEEE, 2020, pp. 14 588–14 597.
[10] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, “Neulfow: Dataflow vision processing system-on-a-chip,” in 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, 2012, pp. 1044–1047.
[11] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, “Shidianna: Shifting vision processing closer to the sensor,” in Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015, pp. 92–104.

[12] H. Genc, A. Haj-Ali, V. Iyer, A. Amid, H. Mao, J. Wright, C. Schmidt, J. Zhao, A. Ou, M. Banister et al., “Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures,” arXiv preprint arXiv:1911.09825, 2019.

[13] N. K. Jha, S. Ravishankar, S. Mittal, A. Kaushik, D. Mandal, and M. Chandra, “Draco: Co-optimizing hardware utilization, and performance of dnns on systolic accelerator,” in 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2020, pp. 574–579.

[14] S. Lym and M. Ozer, “Flexsa: Flexible systolic array architecture for efficient pruned dnn model training,” arXiv preprint arXiv:2004.13027, 2020.

[15] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, “A systematic methodology for characterizing scalability of dnn accelerators using scale-sim,” in 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2020, pp. 58–68.

[16] 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 Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 2, pp. 292–308, 2019.

[17] B. Liu, X. Chen, Y. Wang, Y. Han, J. Li, H. Xu, and X. Li, “Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators,” in Proceedings of the 24th Asia and South Pacific Design Automation Conference, 2019, pp. 733–738.

[18] R. Xu, S. Ma, Y. Wang, and Y. Guo, “Hesa: Heterogeneous systolic array architecture for compact cnns hardware accelerators,” in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2021, pp. 657–662.

[19] R. Xin, J. Zhang, and Y. Shao, “Complex network classification with convolutional neural network,” Tsinghua Science and technology, vol. 25, no. 4, pp. 447–457, 2020.

[20] G. Zheng, G. Han, and N. Q. Soomro, “An inception module cnn classifiers fusion method on pulmonary nodule diagnosis by signs,” Tsinghua Science and Technology, vol. 25, no. 3, pp. 368–383, 2019.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” Advances in neural information processing systems, vol. 25, pp. 1097–1105, 2012.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in Proceedings of the 22nd ACM international conference on Multimedia, 2014, pp. 675–678.

[23] X. Fan, M. Dai, C. Liu, F. Wu, X. Yan, Y. Feng, Y. Feng, and B. Su, “Effect of image noise on the classification of skin lesions using deep convolutional neural networks,” Tsinghua Science and Technology, vol. 25, no. 3, pp. 325–334, 2019.

[24] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan et al., “Searching for mobilenetv3,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324.

[25] N. Jouppi, “Google supercharges machine learning tasks with tpu custom chip,” Google Blog, May, vol. 18, no. 1, 2016.

[26] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” IEEE journal of solid-state circuits, vol. 52, no. 1, pp. 127–138, 2016.

[27] X. Fei, Y. Zhang, and W. Zheng, “Xb-sim: A simulation framework for modeling and exploration of reram-based cnn acceleration design,” Tsinghua Science and Technology, vol. 26, no. 3, pp. 322–334, 2020.

[28] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers et al., “In-datacenter performance analysis of a tensor processing unit,” in Proceedings of the 44th annual international symposium on computer architecture, 2017, pp. 1–12.

[29] S. Markidis, S. W. Der Chien, E. Laure, J. B. Pong, and J. S. Vetter, “Nvidia tensor core programmability, performance & precision,” in 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2018, pp. 522–531.

[30] NVIDIA, “Nvidia a100 tensor core gpu architecture,” https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, 2020, [Online; accessed 02-Nov-2021].

[31] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, “Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks,” in 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2017, pp. 553–564.

[32] H. Kung, B. McDanel, and S. Q. Zhang, “Pooling sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization,” in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 821–834.

[33] X. He, S. Pal, A. Anamath, S. Feng, D. H. Park, A. Rovinski, H. Ye, Y. Chen, R. Dreslinski, and T. Mudge, “Sparse-tpu: Adapting systolic arrays for sparse matrices,” in Proceedings of the 34th ACM International Conference on Supercomputing, 2020, pp. 1–12.

[34] R. Xu, S. Ma, Y. Wang, and Y. Guo, “Cmsa: Configurable multi-directional systolic array for convolutional neural networks,” in 2020 IEEE 38th International Conference on Computer Design (ICCD), 2020, pp. 494–497.

[35] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, “Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures,” in 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA). IEEE, 2014, pp. 97–108.

Rui Xu is currently a PhD candidate with the the College of Computer Science, National University of Defense Technology, Changsha, China. His research interest is in computer architecture and machine learning.

Sheng Ma received the B.S. and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT) in 2007 and 2012, respectively. He visited the University of Toronto from Sept. 2010 to Sept. 2012 as a visiting Ph.D. student. He is currently an Associate Professor at the NUDT. From Dec. 2012 to Dec. 2017, he was an Assistant Professor at the NUDT. His research interests include microprocessor architecture and on-chip networks. He is a recipient of Young Talent Development Program of the China Computer Federation, 2016.

Yaohua Wang received the B.S. and Ph.D. degrees from the College of Computer Science, National University of Defense Technology, Changsha, China. He is currently an Associate Professor with the College of Computer Science, National University of Defense Technology. His research interest is in computer architecture, machine learning and security. His work spans and stretches the boundaries of computer architecture. He is especially excited about novel, fundamentally efficient computation, and memory/storage paradigms, applied to emerging machine learning applications.

Yang Guo is currently a professor with the NUDT.

Dongsheng Li received his Ph.D. degree from National University of Defense Technology in 2005. He is currently a Full Professor with the National Laboratory for Parallel and Distributed Processing, National University of Defense Technology. His research interests include distributed computing, cloud computing, computer network, and large-scale data management. He was awarded the prize of the National Excellent Doctoral Dissertation of China by Ministry of Education of China in 2008.

Yuran Qiao is currently an assistant professor with the NUDT.