HeSA: Heterogeneous Systolic Array Architecture for Compact CNNs Hardware Accelerators

Rui Xu
Institute of Microelectronics
National University
of Defense Technology
Changsha, China
nudtxurui@gmail.com

Sheng Ma
Science and Technology on Parallel
and Distributed Processing Laboratory
National University
of Defense Technology
Changsha, China
masheng@nudt.edu.cn

Yaohua Wang
Institute of Microelectronics
National University
of Defense Technology
Changsha, China
yaowangeth@gmail.com

Yang Guo
Institute of Microelectronics
National University
of Defense Technology
Changsha, China
guoyang@nudt.edu.cn

***Abstract*—Compact convolutional neural networks have become a hot research topic. However, we find that the hardware accelerator with systolic arrays processing compact models is extremely performance-inefficient, especially when processing depthwise convolutional layers in the networks.**

To make systolic arrays efficient for compact convolutional neural networks, we propose the heterogeneous systolic array (HeSA) architecture. It introduces heterogeneous processing elements that support multiple modes of dataflow, which can further exploit the reuse data chance of depthwise convolutional layers and without changing the architecture of the naïve systolic array. By increasing the utilization rate of processing elements in the array, HeSA improves the performance, throughput, and energy efficiency compared to the standard baseline. Based on our evaluation with typical workloads, HeSA improves the utilization rate of the computing resource in depthwise convolutional layers by 4.5×-5.5× and acquires 1.5-2.2× total performance speedup compared to the standard systolic array architecture. HeSA also improves the on-chip data reuse chance and saves over 20% of energy consumption. Meanwhile, the area of 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**

I. 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 compute in CNNs. Because convolution is the most expensive part of CNNs, compact model or compact CNNs choose simplified network components to replace standard convolution (SConv). A popular method to alleviate the convolution overhead is depthwise convolution (DWConv) [2].

Depthwise convolution applies a single filter to each input channel, which is a spatial convolution 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 [2]. Compare to SConv, DWConv saves lots of parameters and MACs while maintaining high accuracy. Due to the advantages of DWConv, a series of compact CNNs appear [2]–[4].

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. Sheng Ma is the corresponding author.

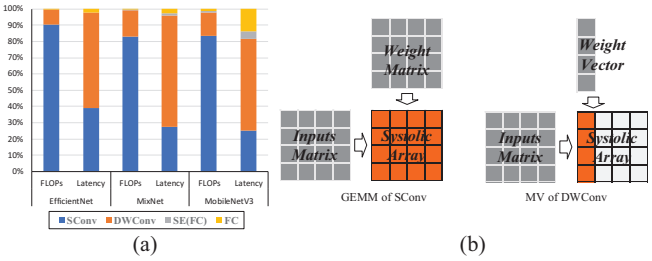

Fig. 1. (a) FLOPs and latency breakdown of different compact CNNs in a 16×16 SA. (b) Different convolution execution on the systolic array. The red PEs are utilized for procession and the white PEs are idle.

DWConv would bring fewer operations and less latency. However, we find the inefficient processing of compact CNNs in the mobile/edge hardware accelerators with systolic arrays (SAs). Fig. 1.a shows the strange result. We count the amount of floating point arithmetics (FLOPs) in three state-of-art compact CNNs, and their latency breakdown in a 16×16 SA. We find that FLOPs of DWConv account for about 10% of the total, but bring over 60% of the latency.

The SA architecture is designed to process and accelerate matrix multiplications (GEMM) and accumulations, which account for more than 95% of the operations in modern CNNs [5]. Usually, typical hardware accelerators adopt the SA core that is a two-dimensional of simple and homogeneous processing elements (PEs). Because the scale of GEMM in SConv layers are both large and multiples of the typical SA sizes, tiling and processing these GEMMs (grey boxes) can fully utilize PEs (red boxes) in the SA [6], as shown in Fig. 1.b. However, the GEMM shrinks to the matrix-vector multiplication (MV) in DWConv layers. Thus, these tiles do not fully occupy the PEs (white boxes), which significantly decreases the performance, as shown in Fig. 1.b.

To solve the inefficiency of calculating DWConv, the data mapping of the SA must be changed [7]. However, the SA is a simple structure with a regular communication pattern and a fixed dataflow. If the mapping method is changed, the

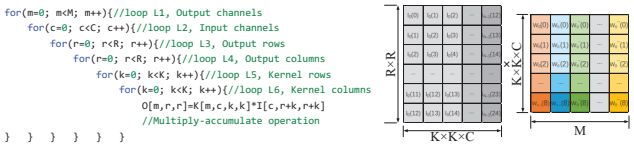

Fig. 2. 6-nested loop of SConv and the GEMM that converted by *im2col* from SConv.

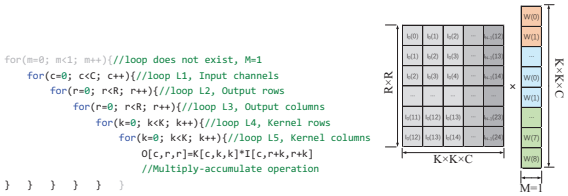

Fig. 3. 5-nested loop of DWConv and the MV that converted by *im2col* from DWConv.

architecture must be modified, which will bring in the increased cost of area, power, and bandwidth. These changes wipe out the advantages of SAs and are not suitable for deployment on resource-limited mobile/edge platforms.

To overcome these challenges, we propose the Heterogeneous Systolic Array (HeSA), which can support multiple dataflows. Different from the previous solution [7] [11], HeSA maintains the original structure of the SA and only makes changes to the PEs in the SA. When processing SConv, HeSA still behaves as a standard SA, maintaining high on-chip reuse and low resource consumption. When dealing with DWConv, HeSA switches dataflow of the SA by changing the data path in heterogeneous PEs. HeSA can increase the performance of DWConv by 4.5×-5.5× without adding additional data paths or increasing external/internal bandwidth. And HeSA can accelerate compact CNNs by 1.5-2.2× and improve 1.1× energy efficiency by compared to a naïve SA design.

II. Background

A. Compact CNNs

CNNs use convolutional (Conv) layers extract different features of the input data [16]. The biggest difference between compact CNNs and traditional CNNs is the implementation of Conv layers.

Standard Convolution (SConv) represents the traditional Conv algorithm. There are multiple channels and kernels to extract different local features of the ifmaps. The operation of SConv can be described as 6-nested loop (regardless of batch) [16], as seen in Fig. 2, where m is the m-th channel of output feature maps (ofmaps), 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. In order to facilitate computing and deployment, SConv is generally converted into GEMM through the *im2col* algorithm. The dimensions of GEMM is large due to the deep and wide structure of modern 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]. The operation of DWConv can be described as 5-nested loop, as shown in Fig. 3, where ofmap is correlated to c and loop m is dismissed (Equivalent to M=1). In addition, using the *im2col* algorithm for DWConv will result in matrix vector multiplication instead of GEMM.

Furthermore, composed by pointwise convolution (PWConv, small SConv) and DWConv, compact CNNs can maintain high accuracy which even surpass the accuracy of dense CNNs [4].

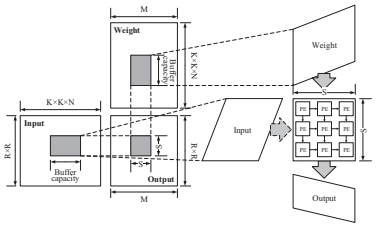

Fig. 4. The systolic array architecture and OS dataflow decide how to map the data on the array.

More and more modern CNNs choose DWConv rather than SConv, like MobileNetV3 [2], MixNet [3], EfficientNet [4], etc.

B. 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 the current cycle in an internal register and then forwards the same data to the neighbor PE in the next cycle [5]. 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 [6]. Note that this data movement and reuse is achieved without the help of control or router units, and only using local register-to-register inter-PE links, without any interconnect logic or global wires. These advantages making it a popular choice for accelerator design. [6]

Dataflow decides the schedule of GEMM tiles in SAs. Output-stationary (OS) is the commonly used ones and achieve high in-core input and output reuse [1]. Fig. 4 also shows the operation of OS dataflow, where both of the two input matrices for a GEMM tile is shifted horizontally and vertically, and the output matrix is fixed on the array.

C. Related work

Systolic arrays have become main trend in hardware CNNs accelerators deployed in both edge devices [7] [8]–[10]and servers used in data-centers [12]. However, we find that these designs only support one fixed dataflow, and all face challenges of inefficient computing and PEs underutilization when processing DWConv of compact CNNs.

Some proposals try to overcome these problems. [7] and [11] try to change dataflow of SAs to improve PEs utilization. However, both of them introduce a lot of storage units, routers, data paths, and control units, increase hardware overhead, and

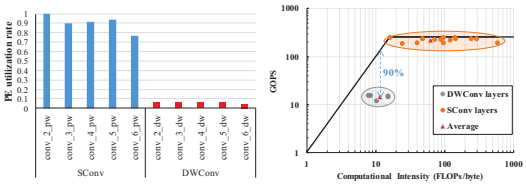

Fig. 5. PE utilization rate of a 16×16 SA processing different Conv layers, and the roofline model for each Conv layer of MobileNetV3.

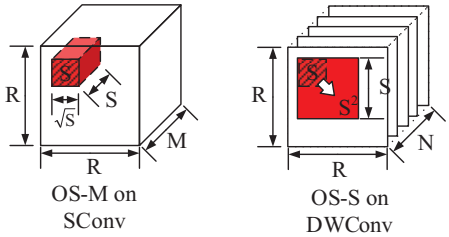

Fig. 6. Data reuse space of OS-M and OS-S dataflow.

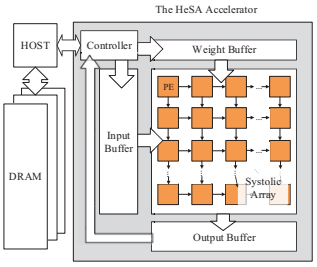

Fig. 7. HeSA architecture.

break the original intention of simple and efficient designs of SAs. [13] try to change the structure of compact CNNs, and make them more suitable for SAs. But it needs to re-train a trained compact CNN and is impractical for edge/mobile devices.

Of course, there are other designs about SAs, such as [14] [15] using column combining to train and compress models, and [5] using structured compression to train CNNs. However, none of these designs consider the situation of compact CNNs.

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

III. Architectural Challenges of Compact CNNs

For further analysis, we record the PEs utilization rate in a 16×16 SA processing MobileNetV3, as shown in Fig. 5. 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.

To find a solution, we also analyze the roofline model of a TPU-like hardware accelerator. We sweep every layer of MobileNetV3, and Fig. 5 also 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 PE utilization of the DWConv layers has room for further improvement without changing the architecture of the SA. In summary, we can find a way to increase the data reuse space of the DWConv layer in the SA and do not need to sacrifice the simplicity of the SA.

A. Discussion of Dataflow

In section II-B, we introduce how OS dataflow schedule and mapping data. It achieves a high output reuse chance. Fig. 6 shows the data ruse space of ofmaps in an $S \times S$ SA processing SConv layers with OS. It focuses on the processing of ofmaps from S channels and S activations (pixels in ofmaps) at a time [16]. We call this OS version OS-M (M for Multi-channels).

However, as we describe in section I, DWConv has no opportunity to reuse data in the output channel dimension. If adopting OS-M directly, the output data reuse space is only S (activations) for the SA. So we use a variant of OS dataflow (OS-S), which processes output data from the same single channel at a time to maximize data reuse opportunities of DWConv [1]. Fig. 6 also shows the ruse space of processing DWConv with OS-S. Ideally, it can increase the reuse chance of output data by S× compared with OS-M.

Whereas the standard SA cannot support OS-S, because OS-S not only need horizontal input data transmission but also need transfer input data vertically [9]. It cannot be achieved in the standard SA which only supports input data propagates in a single direction. But HeSA can support it, and we illustrate that in section IV-A.

IV. Heterogeneous Systolic Array

Fig. 7 shows an overview of 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 OS-M and OS-S dataflow. On-chip buffers provide fmaps 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.

A. Workflow of HeSA

Fig. 8 shows a toy example of HeSA with OS-S, which has 2×2 PEs, and the workload with 2×2 kernel size and 3×3 ifmap size. Four output activations ($O_{0,0}$, $O_{0,1}$, $O_{1,0}$, $O_{1,1}$) are fixed on the PEs respectively. A PE is composed of the weight register (REG1), the input data register(s) (REG2), and the MAC unit which is used to calculate and store output data. The MAC unit begins working when the registers data is ready. The PEs on row0 require additional input data registers (REG3) to cache input data that needs to be propagated vertically to row1.

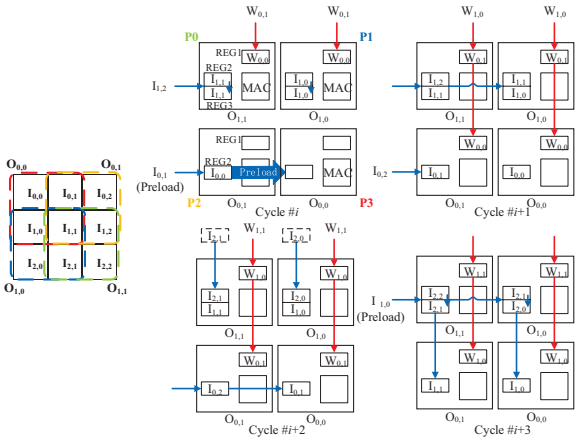

Fig. 8. Workflow of HeSA with OS-S dataflow.

We still use the systolic transmission mode in the standard SA, so the data for PEs at the edge of the array just can come from the neighbor PEs (from up or left) or external storage, while the data for other PEs in the array only come from the neighbor PEs. Due to the systolic transmission mode, PEs on different rows are in different work phases. In Fig. 8, we use the arrows to indicate the source of the data. For conciseness, we only describe the workflow of the four cycles, which can complete the work through repetition.

Cycle #i: OS-S has a data preparation phase that PEs need preload input data. At this time, PEs (PE0 and PE1) on row0 complete preloading, start to receive the weight data, and calculate the partial sum for the output activations. The external storage also prepares the next input and weight data for them. Meantime, PEs on row1 still preload input data. The preloading process takes two cycles, and in this case, one more cycle is needed for PE2 and PE3 to complete preloading.

Cycle #i+1: PE0 and PE1 receive their new input data from external storage and REG2 of PE0 respectively. Their weight data is updated by the external storage. Meanwhile, PE2 and PE3 complete preloading and receive weight data from PE0 and PE1. They start to work.

Cycle #i+2: PE0 and PE1 receive their new input data from external storage. Note that the data enters from the upside. Similar to PE0 and PE1 at cycle #i+1, PE2 and PE3 respectively receive their new input data from external storage and REG1 of PE2. Their new weight data is still from PE0 and PE1.

Cycle #i+3: Similar to cycle #i+2, PE0 and PE1 receive their new input data from external storage and REG2 of PE0. At this time, they complete calculating of output activations, and the external storage prepares new input data from the next channel of ifmaps for preloading. PE2 and PE3 receive their new input data from REG2 of PE0 and PE1. Meantime, REG2 of PE0 and PE1 are updated by REG1 of themselves.

In the toy example presented above, we can see 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 [7] and [11], HeSA still use the systolic

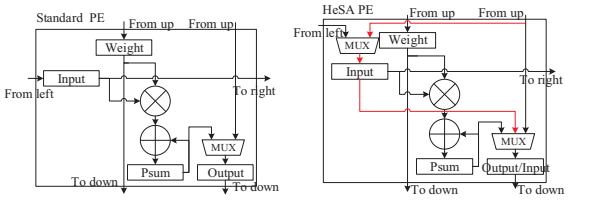

Fig. 9. Structure of PEs in standard SA and HeSA. The red line represents the vertical data path added for ifmaps data in HeSA.

data transformation mode, and it can also be used as a standard SA and support 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. We discuss the solution in section IV-B.

B. Heterogeneous PEs

In section IV-A, we conclude that HeSA PEs should have additional data paths and registers. But We can reuse some components in standard PEs to hide this hardware overhead.

Fig. 9 shows the structure of the standard 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 corresponded buffer.

Because the data paths and registers for output vertical transmission are always idle during the computation, we reuse these components as the input vertical transmission. Fig. 9 also shows the design of PEs in HeSA. We connect the output data path with the input data path (red lines in Fig. 9) and control the data transmission by the multiplexer (MUX). When HeSA uses OS-M dataflow, MUX will close the red path, and the structure of PE in HeSA is the same as the standard PE structure. When HeSA uses OS-S dataflow, MUX will choose the red path under the control of the control unit, and the output register is also used to cache the input data. Besides, the SA with OS-S still needs additional external storage for PEs on row0 (in Fig. 8, Cycle #i+2). For avoiding this overhead, we use the PEs on the top of the array as the set registers (these PEs are no longer used for calculation) for preloading the input data for row0. Although affecting the performance, it saves the cost of storage and is easy to implement in the mobile/edge device.

The design of PEs in HeSA reuse the idle data path and avoid adding additional registers in PEs and additional data paths in SA. At the same time, by using a variety of dataflows, HeSA becomes more flexible and improves the efficiency in computing DWConv.

C. Buffer and controller

In HeSA, on-chip local buffers are double buffered. Double buffering enables the overlap of computation within the PEs

TABLE I
CONFIGURATIONS OF REPRESENTATIVE IMPLEMENTATIONS

	*Standard SA (Gemmini)**	*HeSA*	*Eyeriss**
Process	45 nm	45 nm	45 nm
Frequency	500 Mhz	500 Mhz	500 Mhz
Num of PEs	256	256	256
Local Store/PE	8 B	8 B	8 B
Buffer Size	128 KB	128 KB	128 KB

*Only key components are considered.

with memory access and allows for very simple coarse-grain control of data transfers between buffers and memory.

The control unit has responsible for connection with the host device, data distribution, and arbitration in the standard SA. In HeSA, the control unit can also control the data path in PEs to switch dataflow flexibly. In the compilation stage, we specify which 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.

V. EVALUATION

In our experiments, we use three compact CNNs as workloads – MobileNetV3 [2], MixNet [3] and EfficientNet-lite0 [4]. These models are up-to-date and widely used in applications for mobile or embedded devices. Moreover, due to the characteristics of the mobile/edge platforms, we apply 8-bit quantization for all of the network models and specify that the mini-batch is 1.

We simulate HeSA and standard SA with SCALE-Sim [6], which is a configurable systolic array-based cycle-accurate DNN accelerator simulator [6]. By using this simulator, we can get accurate information on the runtime of the array and PE utilization rate. For area and energy consumption, we implement HeSA in RTL, which is synthesized under a commercial 45 nm library by Synopsys Design Compiler.

We also compare HeSA and other hardware accelerators. The configurations are listed in Tab. I. For a fair comparison, we make these parameters of implementations to be the same. We choose Gemmini [10] for the design of standard SA. Gemmini is an open source TPU-like systolic array generator, and can produce optimal RTL according to parameter requirements [10]. Eyeriss [8] is the most classic CNNs hardware accelerator.

VI. EVALUATION RESULTS

A. PE utilization

We first show how HeSA improves the PE utilization of the array by switching dataflow. Fig. 10 shows the PE utilization rate of different layers of MixNet processed by an 8×8 SA with different strategies. 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 [9], and HeSA is our design.

The results show that for SConv layers (R×R K×K PW), 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

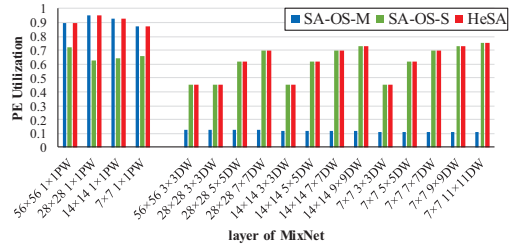

Fig. 10. PE utilization rate of different layers in MixNet with different strategies.

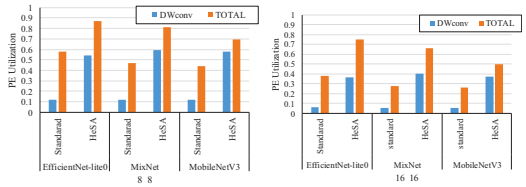

Fig. 11. PE utilization rate of different workloads with different size arrays.

DW), 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%. HeSA always keeps the high PE utilization rate of each layer by switching dataflow according to different scenarios.

For further evaluation, we show the DWConv and total PE utilization rate of different compact CNNs in an 8×8 standard SA and HeSA in Fig. 11. Compared with the low PE utilization rate (only about 11%) of the DWConv layers in standard SA, HeSA can increase the rate to about 60%. Thanks to the huge improvement in the DWConv layers, the total PE utilization rate of three compact CNNs in HeSA is also improved.

In addition, Fig. 11 also shows that the PE utilization rate of standard SA and HeSA are both affected when the SA size is increased to 16×16. The standard SA suffers a huge impact. Its DWConv PE utilization rate is even less than 6%, and the total utilization rate is less than 30%. However, HeSA uses flexible dataflow to keep the DWConv PE utilization rate above 38%, and the total utilization rate above 50% during the test.

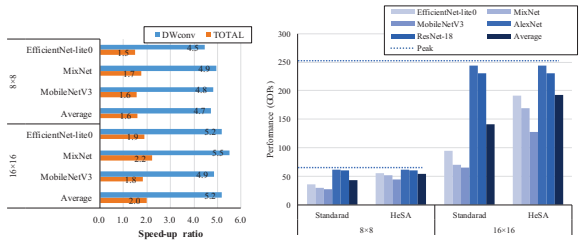

Fig. 12. Speed-up ratio obtained by HeSA compared to the standard SA with different size arrays, and the performance (GOPs) of the standard SA and HeSA.

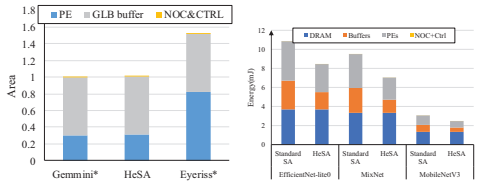

Fig. 13. Area and Energy breakdown of HeSA and representative implementations.

B. Performance

The increase in PE utilization brings about an increase in performance. We sum up the latencies of each layer, and Fig. 12 shows the speed-up ratio obtained by HeSA compared to the standard SA in different array sizes. As the array size is 8×8, HeSA can get an average 4.7× speed-up when processing the DWConv layer compared to SA, and the total performance is 1.6× better. When the array size is 16×16, compared with standard SA, HeSA can achieve an average 5.2× speed-up processing the DWConv layer, and overall performance is improved by 2.0×.

To further show the performance advantages of HeSA, we also evaluate the GOPs of the standard SA and HeSA. In this evaluation, we expand the workload to include two traditional CNNs [16] [17]. Since traditional CNNs do not include DWConv layers, the standard SA and HeSA can almost reach peak performance. However, when processing compact CNNs, the standard SA suffers a significant performance loss, and the larger the size, the greater the loss. HeSA makes up for the performance loss, so its average performance on workload reached 54.5 GOPs (8×8 PEs, 85% peak performance) and 192.6 GOPs (16×16 PEs, 76% peak performance) respectively.

C. Area

We layout the HeSA (16×16) and the total area of it is 1.79 mm^2. We compare the area of HeSA with other accelerators, as shown in Fig. 13. Thanks to the simple architecture, the standard SA has the smallest area. The area of HeSA only increases by 1% compared to the standard SA. Because HeSA only adds some control units, 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. 13 also shows the area breakdown. The PEs in Eyeriss take over half of the total area, which is 2.7× larger than that in standard SA and HeSA.

D. Energy

We illustrate the energy consumption of the standard SA and HeSA in Fig. 13. When processing those three compact CNNs, the energy consumption of HeSA is always less than that of the standard SA (over 20%). Thanks to the improvement of reuse space in DWConv layers, HeSA reduces the number of reads and writes of buffers and registers in PEs. Therefore, about 30% and 40% of energy consumption are saved in PEs and buffers respectively. However, the energy consumption of DRAM is almost unchanged, because DRAM only stores and moves the fmaps 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 too small to be counted.

VII. Conclusion

We design a heterogeneous systolic array architecture (HeSA) for CNNs hardware accelerators, which can switch different dataflow and exploit the reused space of compact CNNs. It greatly mitigates the mismatch between systolic array architecture and depthwise convolutional layers of compact CNNs. Tested by typical workloads, our design provides 4.5-5.5× and 1.5-2.2× 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 HeSA is basically the same as the standard SA. Thanks to the increase of reuse chance, the energy efficiency of HeSA is increased by about 10% over the baseline.

References

[1] Deng, Lei, et al. "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey." Proceedings of the IEEE 108.4 (2020): 485-532.
[2] Howard, Andrew, et al. "Searching for mobilenetv3." Proceedings of the IEEE International Conference on Computer Vision. 2019.
[3] Tan, Mingxing, and Quoc V. Le. "Mixconv: Mixed depthwise convolutional kernels." arXiv preprint arXiv:1907.09595 (2019).
[4] Tan, Mingxing, and Quoc V. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." arXiv preprint arXiv:1905.11946 (2019).
[5] Lym, Sangkug, and Mattan Erez. "FlexSA: Flexible Systolic Array Architecture for Efficient Pruned DNN Model Training." arXiv preprint arXiv:2004.13027 (2020).
[6] Samajdar, Ananda, et al. "Scale-sim: Systolic cnn accelerator simulator." arXiv preprint arXiv:1811.02883 (2018).
[7] Chen, Yu-Hsin, et al. "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices." IEEE Journal on Emerging and Selected Topics in Circuits and Systems 9.2 (2019): 292-308.
[8] Chen, Yu-Hsin, et al. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks." IEEE journal of solid-state circuits 52.1 (2016): 127-138.
[9] Du, Zidong, et al. "ShiDianNao: Shifting vision processing closer to the sensor." Proceedings of the 42nd Annual International Symposium on Computer Architecture. 2015.
[10] Genc, Hasan, et al. "Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures." arXiv preprint arXiv:1911.09925 (2019).
[11] Liu, Bosheng, et al. "Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators." Proceedings of the 24th Asia and South Pacific Design Automation Conference. 2019.
[12] Jouppi, Norm. "Google supercharges machine learning tasks with TPU custom chip." Google Blog, May 18 (2016): 1.
[13] Jha, Nandan Kumar, et al. "DRACO: Co-Optimizing Hardware Utilization, and Performance of DNNs on Systolic Accelerator." 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2020.
[14] Kung, H. T., Bradley Mcdanel, and Sai Qian Zhang. "Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization." arXiv: Learning (2018).
[15] He, Xin, et al. "Sparse-TPU: Adapting Systolic Arrays for Sparse Matrices." International Conference on Supercomputing (ICS'20). 2020.
[16] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
[17] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.