Journal of Circuits, Systems, and Computers
Vol. 31, No. 16 (2022) 2250289 (32 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218126622502899

RNA: A Flexible and Efficient Accelerator Based on Dynamically Reconfigurable Computing for Multiple Convolutional Neural Networks[*]

Chen Yang[†], Jia Hou[‡], Yizhou Wang[§], Haibo Zhang[¶], Xiaoli Wang[||] and Li Geng[**]

School of Microelectronics, Xi'an Jiaotong University, No. 28, Xianning West Road, Beilin District, Xi'an, Shaanxi 710049, P. R. China
[†]*chyang00@xjtu.edu.cn*
[‡]*hjwst@stu.xjtu.edu.cn*
[§]*wangyizhou@stu.xjtu.edu.cn*
[¶]*haibozhang@stu.xjtu.ed.cn*
[||]*xlwang@xjtu.edu.cn*
[**]*gengli@xjtu.edu.cn*

Received 11 November 2020
Accepted 28 May 2022
Published 7 July 2022

The increasingly complicated and versatile convolutional neural networks (CNNs) models bring challenges to hardware acceleration in terms of performance, energy efficiency and flexibility. This paper proposes a reconfigurable neural accelerator (RNA) for flexible and efficient CNN acceleration. To provide hardware flexibility, RNA employs dynamically reconfigurable computing framework to rapidly configure data path between processing elements (PE) at run-time, as well as an interlaced data access mechanism for multi-bank RAM. To achieve high energy efficiency, three optimization mechanisms, including image row broadcasting dataflow (IRBD), tile-by-tile computing (TTC), and zero detection technology (ZDT), are dedicatedly designed for RNA to exploit data reuse and decrease memory bandwidth requirement, which is the key to improving performance and saving power consumption. To save hardware overhead, an online dynamic adaptive data truncation (DADT) mechanism is designed to compensate accuracy loss of multiplication results so that the computational precision in RNA can be reduced from 16-bit to 8-bit, which contributes to reducing the area of data path. The RNA architecture is implemented on Xilinx XC7Z100 FPGA and works at 250 MHz. Experimental results show that the performance of running LeNet, AlexNet and VGG are 500 GOPS, 598 GOPS and 660 GOPS, respectively. Compared to previous FPGA-based designs, RNA achieves $1.5\times - 4.3\times$ performance speedup and $7.6\times - 8.4\times$ improvements on energy efficiency.

Keywords: CNN; reconfigurable computing; image row broadcasting dataflow; tile-by-tile computing; zero detection technology; multi-bank RAM; dynamically adaptive data truncation.

[*]This paper was recommended by Regional Editor Tongquan Wei.
[†]Corresponding author.

1. Introduction

Convolutional neural network (CNN) is a mathematical model inspired in the function of the brain, which can effectively deal with clustering and classification problems. CNN has shown significant superiority in the field of modern artificial intelligence (AI) applications, including semantic segmentation,[1] medical diagnosis,[2] stock market prediction[3] and pattern recognition.[4] Substantially, the CNN algorithm is constructed by convolutional layers (CLs) along with full connection layers (FCLs) for feature extraction and classification. Figure 1 demonstrates the layers, operations and parameters of several typical CNNs.[5–10] Current CNN models are usually composed of tens to hundreds of convolutional layers with different filter sizes. In a single inference pass, they require hundreds of megabytes of weights and billions of operations, which bring a great challenge on throughput and energy efficiency to the underlying hardware design.

To accelerate the inference process of CNN, graphics processing unit (GPU) is a usual choice due to its extremely high performance.[11,12] However, typically over one hundred Watt power consumption prevents GPU from deployment in embedded scenarios, such as portable hardware, wearable systems, and Internet of Things (IoT) devices. Application-specific integrated circuits (ASICs) have been viewed as an energy-efficient way for CNN acceleration.[15–20] However, limited by exorbitant fabrication cost and excessive development period, ASIC designs lack flexibility for the rapid evolution of CNNs. Increasing research attention is focused on field-programmable gate arrays (FPGA) based accelerators due to the possibility of a trade-off between power consumption and configurability.[21–32] However, the statically reconfigurable FPGA-based accelerators need to power down the system and recompile the architecture when changing hardware function, just providing limited flexibility.

Dynamically reconfigurable computation[33] represents another class of emerging computing architecture that has shown considerable advantages in energy efficiency and flexibility. It has two distinct characteristics different from other computation structures. One is the customization capability after post-chip, that is, it still changes functions of computation as required even if the tape-out is carried out. Another one

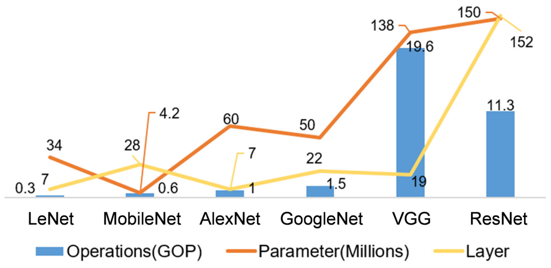

Fig. 1. Variety and complexity of CNNs.

is that it can realize the spatial mapping from algorithm to calculation engine to great extent. The two characteristics are superior to the traditional ASICs and general-purpose processors driven by instruction stream.[33] The starting point of its design is to consider the advantages of general-purpose processors and dedicated circuits, making a reasonable compromise between performance, power consumption and flexibility. Arising in response to the development of semiconductor technology, it emphasizes resource reuse and flexibility. Reconfigurable computation provides a processing elements (PEs) array that can be dynamically programmed. It can switch computational shapes by reconfiguring data paths and interconnections between PEs.[34] Therefore, reconfigurable computing fabrics can balance the performance of ASIC with the flexibility of the processor, thereby achieving substantial advantages in both energy efficiency (i.e., performance/power) and flexibility for domain-specific applications, such as speech recognition,[35] bioinformatics,[36] videos and image processing.[37–39]

In this paper, a dynamically reconfigurable neural accelerator (RNA) is designed to provide high performance, energy efficiency and dynamical flexibility. This accelerator employs a heterogeneous PEs array with high parallelism, which can be dynamically configured to flexibly perform different computing shapes and achieve high throughput. Data optimization mechanisms presented in RNA can improve degree of data reuse to cut down data transmission. RNA uses dynamic adaptive data truncation mechanism to save hardware resource and maintain a high inference precision. Overall, the main contributions of the proposed RNA accelerator are as follows:

(1) Dynamically reconfigurable computing framework is designed to meet the diversity and complexity of CNNs arithmetic. A heterogeneous array of 484 PEs (including 154 special PEs and 330 normal PEs) that can be configured with high computational parallelism for throughput at run-time is constructed to perform different computing shapes. Alongside PEs array, an interlaced data access mechanism for multi-bank RAM is proposed, which can avoid complex data schedules in a single global memory and combine the feature of PEs array to facilitate data transmission.
(2) Three data optimization mechanisms, including image row broadcasting dataflow (IRBD), tile-by-tile computing (TTC), and zero detection technology (ZDT) are proposed, resulting in a high level of data reusability and throughput. IRBD broadcasts the image data and distributes the weigh data onto the PEs array. TTC considers the need for computing and access to make the full use of data. ZDT substantially decreases the computing number by eliminating the computing for zero-value. Such mechanisms can effectively reuse data and reduce data movements.
(3) An online dynamic adaptive data truncation (DADT) mechanism is employed to improve the computing accuracy. This mechanism is powerful support for the use

of 8-bit fixed point operand rather than 16-bit, which is beneficial for saving hardware resource and improving performance as well as maintaining a high inference precision.

The rest of the paper is organized as follows. Section 2 introduces the related works about CNN accelerators and the motivation of this paper. Section 3 describes the details of the proposed RNA architecture. Section 4 presents the implementation results of RNA and comparisons with other designs. Section 5 summarizes this paper.

2. Related Work and Motivation

2.1. *Related work*

Previous works have proposed a variety of CNN accelerators with different platforms. GPU is suitable for parallel computing of a large amount of data, and has the advantages of high bandwidth, high frequency, and high parallelism. With its powerful floating-point computing power and highly parallel computing architecture, GPU is the platform of choice for neural network training. Neural network accelerators based on GPU[11,12] achieve extremely high performance, which can work at more than 1 GHz clock frequency and reach up to thousands of giga operations per second (GOPS). With its powerful parallel computing capabilities and advanced fabricated technology, GPU can efficiently be dedicated to a large number of parallel floating-point calculations and matrix vector operations in CNN calculations. However, the power consumption of GPU is as high as hundreds of Watts, so that the typical energy efficiency of GPU-based accelerators can only achieve 4.41 GOPS/W[24] and 7.13 GOPS/W.[23] Therefore, high power consumption of GPU is an apparent obstacle for a wide range of applications.

In order to pursue better performance with high energy efficiency, some ASIC-based accelerators are designed, including TPU,[13,58] NPU[14] and DianNao.[16,55–57] According to the algorithm structure of CNN, the corresponding customized circuit architecture is dedicatedly designed, and the acceleration operation of the neural network algorithm can be completed efficiently. TPU[13] employs systolic array architecture to perform convolution computing, which can keep all systolic PEs busy and simplify the structure of PE to achieve high performance and low power. Subsequently, a second-generation chip TPU2[58] is designed for neural network training. It can support floating-point operations and split the 64K array into two 128×128 arrays to increase PE utilization, as well as a higher HBM (High-bandwidth Memory) storage structure to increase internal memory bandwidth. NPU[14] presents an energy-efficient butterfly-structure dual-core accelerator having 1,024 multiply–accumulate (MAC) units and exploits a three-fold parallelism in computing CLs and FCLs. References 16 and 55– 57 are machine learning accelerators called the DianNao series. The operation core of Refs. 16, 55 and 56 is Neural Functional Units (NFU). The logic resources of NFU are divided into multiplication units, addition trees and activation units. In many representative neural network layers, the DaDianNao[55] accelerator can achieve an acceleration ratio of 450.65 times compared with GPU, while the average energy consumption is reduced by 150.31×. PuDianNao[57] supports seven machine learning algorithms, of which in addition to machine learning units (MLU), there is also an arithmetic and logic unit (ALU) for dealing with general operations and other operations that MLU cannot handle. Aiming at variable weight bits used in different CNN models, Stripes,[60] Loom,[61] UNPU[62] and Bit Pragmatic (PRA)[59] are the class of ASIC-based CNN accelerators that perform layer-wise mixed-precision inference using bit-serial MACs. Among these works, experiments showed that Stripes achieved a 1.3× throughput increase over bit-parallel DaDianNao with VGG-19. ASIC desi gn can customize the data path of CNN accelerator according to the structure and operation type of the neural network. Hence, it has the advantages of high performance, low power consumption and small area. However, when ASIC flow is used for neural network acceleration, the design cycle and the cost of development are very high. At the same time, customized circuits still have no advantages in flexibility, making it difficult to accelerate diverse structures of rapidly evolving neural network.

The FPGA-based acceleration is considered an attractive solution to balance the performance and timeliness of deployment. Caffeine[29] proposes a uniformed convolutional matrix-multiplication representation for CLs and FCLs and maximizes the underlying FPGA computing and bandwidth resource utilization. ESE[30] proposed by DeePhi Tech compresses the model size and partitions the compressed model to multiple PEs for parallel processing. To improve data reuse and reduce data movement, Refs. 31 and 32 focus on the dataflow across convolutional layers. Ref. 31 fuses the processing of multiple CNN layers by modifying the order in which the input data are brought on chip, enabling caching of intermediate data between evaluations of adjacent CNN layers. Reference 32 proposes a layer conscious memory management framework, which exploits the layer diversity and the disjoint lifespan information of memory buffers to efficiently utilize the on-chip memory, so as to improve the performance of the layers bounded by memory. These CNN accelerators are done either by providing a domain-specific processor that is programmable or by partial reconfiguration, which shows weak flexibility. To further achieve the balance of performance and power as well as possessing more flexibility, dynamically reconfigurable accelerators are designed. Eyeriss[40,54] and Thinker[18,41] are representative reconfigurable accelerators with high performance, and they can adapt a wide range of neural network models. Eyeriss applies data compression and data gating to reduce data movement and maximizes the local data reuse. Thinker supports bit-width adaptive computing to meet various bit-widths of neural layers and employs on-demand array partitioning and reconfiguration for processing different models in parallel. Due to advantages of dynamically reconfigurable computing, the research on reconfigurable accelerators will get more attention.

2.2. *Motivation*

CNNs have complex structural characteristics. An example of typical CNN model called AlexNet is given to analyze the characteristics of CNN. AlexNet[5] is composed of five convolutional layers, three pooling layers and three full connection layers. Specifically, convolutional layers extract features of the input image and pooling layers following convolutional layers subsample the feature maps. Full connection layers are acting as a classifier in the last output of the result. CNN includes mass data like input images and weights and performs a large number of MAC operations. Especially, convolutional layers have various kernel sizes and convolution strides for different levels of feature extraction. The differences in computing shapes among the layers lead to different demands for computation and storage.

Deep learning has been a research hotspot in the recent years and new neural models emerge in an endless stream. Emerging CNN models become increasingly complicated and versatile. The complexity is reflected in the number of layers and convolutional kernel size. In general, one hardware design only corresponds to a certain algorithm to implement a class of applications. With the advent of the 5G and IoT (Internet of Things) era, it makes hardware necessary to perform diverse neural network models to cope with different scenarios. For example, AlexNet for image recognition and LeNet for digital handwriting recognition can be implemented in one hardware device, which brings great convenience for customers.

According to the above statement, there are two key limitations for hardware to accelerate CNNs. On the one hand, the differences in computing shapes among layers make it difficult to map the entire neural network onto a single hardware. Taking common CNN models as example, LeNet[6] contains five layers, including three types of network structures: convolution layer with 5×5 kernel size, pooling layer and nonlinear layer. AlexNet[5] involves eight layers, where the network type is similar to that of LeNet, but including three kernel sizes (i.e., 11×11, 5×5 and 3×3). VGG[47] consists of 3×3 convolution kernel and 2×2 pooling kernel, and the depth of layers in VGG is also deeper than LeNet and AlexNet. For some emerging CNN models, such as MobileNet,[8] Yolov3-tiny[48] and SqueezeNet,[49] they also show some diversity. Therefore, it can be seen that the shapes and parameters of different neural network models are diverse. On the other hand, the structural difference among CNNs makes it difficult to implement multiple neural networks onto a single hardware. CNN accelerators require a trade-off between flexibility and energy efficiency. In general, ASIC designs can achieve higher power efficiency but less flexibility,[50] so that only certain CNN model can be efficiently performed on ASIC-based accelerators. In order to better match different types of CNN models, we need a more flexible hardware architecture, such as dynamically reconfigurable architecture.[15,40,41,51] Due to these two limitation factors, a strong requirement arises for AI systems to employ a common hardware structure with high performance, low power and strong flexibility. The hardware structure based on dynamically reconfigurable computation is a good choice to meet such demand.

Aiming at the aforementioned two limitations, this paper is to build a reconfigurable accelerator so that various computing shapes and CNN structures can be flexibly and efficiently mapped on the single hardware. Compared to existing reconfigurable accelerators, the proposed RNA accelerator not only employs dynamically reconfigurable computing framework but is also equipped with several novel improvements, including a run-time reconfigurable data path on PEs array for different computing shapes, an interlaced data access mechanism for a multi-bank RAM, an image row broadcasting dataflow (i.e., IRBD dataflow), a novel tile-by-tile computing mechanism for fully data reuse and memory bandwidth reduction (i.e., TTC strategy), and an online dynamic adaptive data truncation mechanism for computational precision compensation (i.e., DADT mechanism). Firstly, a dynamically reconfigurable PEs array is designed to perform different computing shapes with high parallelism. Instead of typical high-capacity global memory in existing reconfigurable accelerators,[52,53] a multi-bank RAM with interlaced data access is proposed to meet the high-parallel requirement of PEs array, thereby facilitating data transfer between PEs and RAM during inference. Next, existing reconfigurable accelerators commonly use a layer-by-layer strategy to perform the computation of CNN models.[5,6,48] Instead, the TTC strategy proposed in this paper allows the computation of convolution layer in parallel with the data reading for FC layer, thereby avoiding the storage of intermediate data and improving throughput. Besides, existing reconfigurable accelerators, such as Eyeriss,[40] only complete the image data of one channel at the same time on a PE. As a contrast, the IRBD dataflow used in this paper separates the image data and filter weights in different ways flowing to PEs, which can increase data reuse and reduce data movements to lower the power and latency. Finally, this paper employs the DADT mechanism to save hardware resource and to maintain higher accuracy by retaining the most significant bits of intermediate data, which is superior to existing fixed point quantization[16] and range distribution truncation[17,40] used in existing reconfigurable accelerators.

Figure 2 shows the generic framework of dynamically reconfigurable architecture. The framework consists of a reconfigurable controller, a reconfigurable PEs array, a data memory, a configuration context memory. The reconfigurable controller is responsible for scheduling the overall execution sequence of context and the full mapping of data. As the key component in the framework, the PEs array can execute operations of various computing shapes. This function depends on the interconnection network to organize data paths according to configuration contexts, thereby building the required operational circuit. Data memory and configuration context memory are memory units to store data and configuration context, respectively. At run-time, the reconfigurable controller schedules contexts into PEs array and then interconnection network adjusts data path to refactor PEs array, thereby

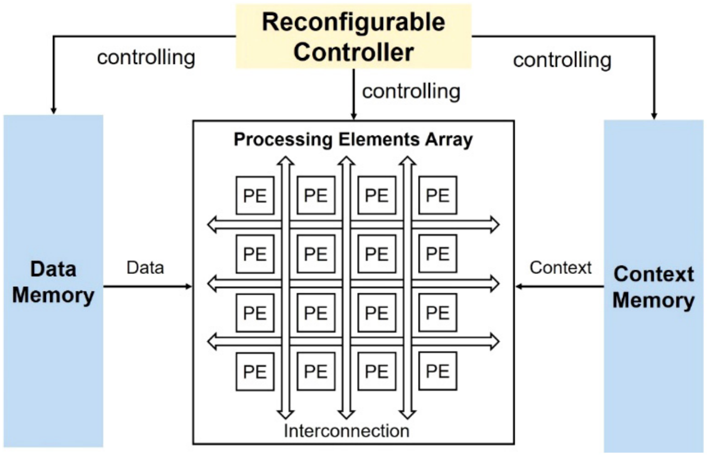

Fig. 2. Framework of dynamically reconfigurable architecture.

constructing the corresponding operational circuit. With the requirement of computing shape, data are transferred from data memory to the operational circuit.

As described above, reconfigurable architecture includes a PEs array that can be configured to perform operation of diverse computing shapes, which satisfies a variety of computing shapes and models for CNNs. In addition, reconfigurable computing can re-allocate hardware resources by the reconfiguration of PEs array and achieve high throughput by operational circuits in parallel. Therefore, CNN accelerators based on reconfigurable computing can be applied to accelerate inference processing of different CNNs with high performance, low power and strong flexibility.

3. Reconfigurable Neural Accelerator

In this section, the architecture of RNA is presented, which can implement hardware acceleration operations of diverse CNNs. First, the designs of normal PE (NPE) and special PE (SPE) are constructed in arithmetic unit, performing convolutional computing and activation function. Next, some optimization mechanisms are introduced in detail, which are beneficial for more flexibility, high energy efficiency and high precision.

3.1. *Overview of architecture*

As shown in Fig. 3, RNA consists of a reconfigurable array of 484 PEs organized as a 22×22 rectangle, 22 11-bit $\times 256$ filter memories, a ping-pong 2.56 KB image memory, 22 704B multi-bank RAMs as output buffer, FSM and configuration module.

The PEs array is the key component of RNA to perform the computations of convolution operation and activation function, which can be configured as required

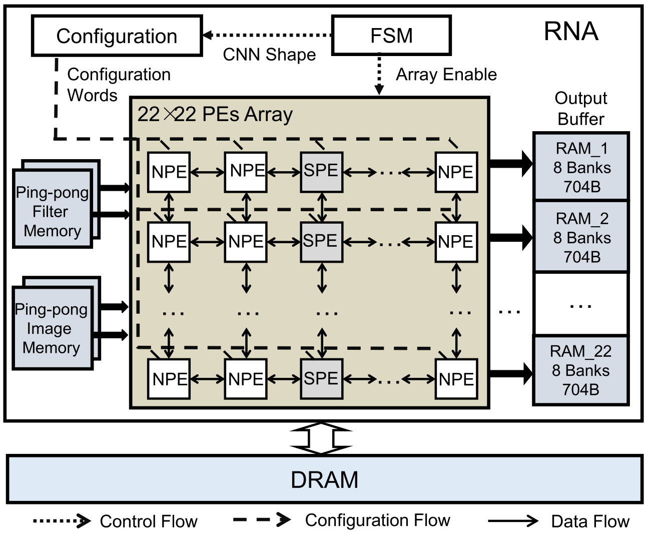

Fig. 3. Architecture of RNA.

for computing shapes. Filter memory and image memory are placed between PEs array and DRAM to supply the input data (image data and filter weights) for PEs array. Multi-bank RAMs store the output data of PEs array. For pooling layer, the input data need to be subsampled so that the model can compress feature of image and reduce parameter and operation of the subsequent layer. In RNA, max pooling is employed to achieve such function. That is to say, the maximum one of four data derived from one SPE at four output times is selected as the final output to store into multi-bank RAMs, when encountering the pooling layer. The overall execution is controlled by the finite-state machine (FSM), which loads weights and configuration context through flag signal (i.e., CNN shape). Configuration module saves the configuration words on CNN models. At run-time, for a certain CNN model, FSM firstly controls the system to switch the corresponding state. It defines the CNN model and enables PEs array to compute. Next, based on the CNN shape, configuration module sends the configuration words that denote the context on data path to configure computing shape of PEs array. Then, the PEs array is configured into a certain computing shape according to the configuration words to process the image data and filter weights. Finally, the output data are stored in the multi-bank RAMs.

3.2. *PEs and interconnection*

In this work, aiming to the computation characteristic of CNN, PEs array possessing two functions is designed. One of both is the MAC operation called NPE, and another is the MAC operation with activation function called SPE. Notably, SPE can make the transition to NPE on demand under no-activation case.

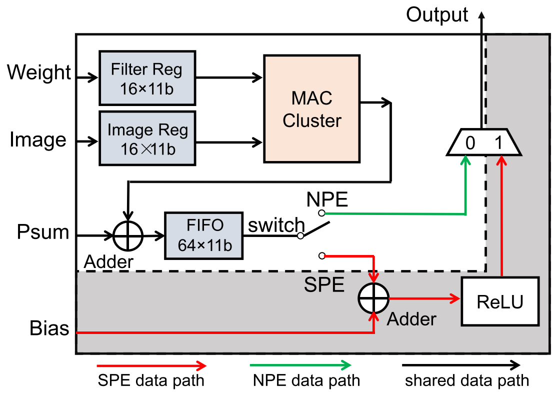

Fig. 4. Structure of NPE and SPE.

Hardware structures of NPE and SPE are depicted in Fig. 4. NPE is shown in the top-left dotted line. Each NPE consists of two shift registers, a MAC cluster, an adder and a FIFO. 16 × 11b shift registers referred to as filter reg and image reg store weights and image data, respectively. MAC cluster is built to perform the row convolution operation. As shown in Fig. 5, it contains three multipliers, two adders and one accumulator. Three multipliers can realize computing of three pixels in parallel. Two adders form addition tree to accumulate the multiplication results. Such design is due to the fact that the kernel size of 3 × 3 is the main computing shape in CNN models, which benefits PE utilization. The accumulator in MAC cluster can supply other computing shapes (i.e., 5 × 5 and 11 × 11). For different computing shapes, the work manner of the MAC cluster is as follows. At the convolution kernel size of 3 × 3, it consumes only one cycle and gains the result after addition tree computation without accumulator operation. While at 5 × 5 kernel, it takes two cycles to complete one-row convolution operation. It needs to execute the addition with the results from the first cycle and the second cycle by accumulator. Similarly, it consumes four cycles to realize the operation for 11 × 11 kernel. For the adder in NPE, it sums the output of the current MAC cluster and Psum from the last PE. FIFO, an 11-bit width and 64-depth storage unit, is used to temporarily store the intermediate data of the convolution operation. In contrast to NPE, SPE has extra 11-bit adder and an active function module for activation operation (i.e., ReLU). Here, the adder sums the results of bias and row convolution. The data paths

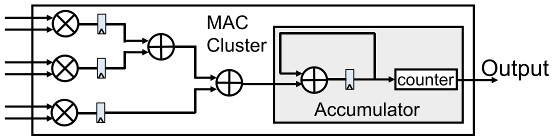

Fig. 5. Structure of MAC cluster.

of NPE and SPE are marked with green and red lines, respectively, in Fig. 4. When ReLU operation is needed, it switches to red line of SPE, while it connects with green line of NPE without activation operation.

The reconfigurable technology puts emphasis on reconfigurable PEs array. It not only meets the function for computation of CNN, but also possesses high computing efficiency and flexibility. The focus on CNN is convolution operation, and it is noted that the output of operation cannot do without activation function. Therefore, as introduced above, the design for PEs array is heterogeneous. If the PEs array consists of homogeneous NPEs, it cannot finish the CNN operation without an extra activation module. If the PEs array is constituted of homogeneous SPEs, it will introduce a useless activation function module, resulting in the wasting of hardware resource. Thus, this makes it necessary for heterogeneous design. In our design, PEs array is placed as 22×22. The consideration for 22×22 is due to computing shapes (i.e., kernel sizes). This setting can maximize the utilization of PE. For the kernel size of 11×11, it can make use of all the PE. Also, for kernel sizes of 3×3 and 5×5, it only wastes one and two PEs, respectively. For a row in PEs array, it contains 15 NPEs and 7 SPEs. Here, seven SPEs achieve maximum parallelism, that is, when performing the kernel size of 3×3, it can compute seven kernels in parallel.

The deigned heterogeneous PEs array can perform different computing shapes by controlling the multiplexers to change data paths (i.e., interconnections between PEs). The selection of multiplexers depends on the configuration contexts representing computing shapes. As described in the above paragraph, the realization for row convolution operation by a PE unit has been explained. On this basis, the whole convolution operation is introduced as follows.

By changing interconnections between PEs, the PEs array can form different processing subsets. Taking part of the interconnections on PEs array as an example, it is shown in Fig. 6. When the convolution kernel size is 3×3, each subset contains three PE units, such as PE1, PE2 and SPE1. When the kernel size is 5×5, each subset contains five PE units, e.g., PE1, PE2, SPE1, PE3 and SPE2, especially, SPE1 is refactored to NPE without activation function. When the kernel size is 11×11, each subset contains 11 PE units, such as the group of PE1, PE2, SPE1, PE3, PE4, SPE2, PE5, PE6, SPE3, PE7 and SPE4. In fact, the number of PE units required to compute one kernel of convolution is equal to the number of rows of this kernel. Table 1 lists the configuration words for Fig. 6 at different kernel sizes.

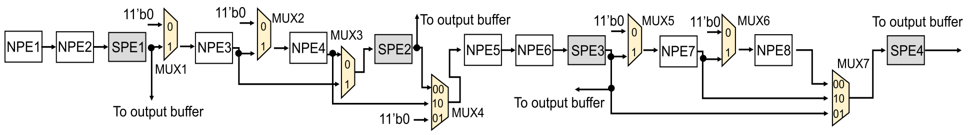

Fig. 6. Configuration for part of PEs array.

Table 1. Configuration words in different shapes.

Kernel size	mux1	mux2	mux3	mux4	mux5	mux6	mux7
3×3	0	1	0	10	0	1	00
5×5	1	0	1	01	0	1	10
11×11	1	1	0	00	1	0	01

Table 2. The distribution of configuration words.

Configuration	Bits	Description
Connection between PEs	[26:0]	[0:14]: Each bit controls a multiplexer to configure the connection of NPE. [15:26]: Each two bits controls a multiplexer to configure the connection of SPE.
Function of SPEs	[40:27]	Each two bits determines a SPE function.
Function of PE	[44:41]	Each bit controls the computing shape under the current state.

This example given in Table 1 is only the configuration of the data path in part. Table 2 lists the whole configuration words for PEs array. The length of configuration words is 45-bit. Firstly, 27 bits control the connection of PEs array. Each bit from 0 to 14 controls a multiplexer to configure the connection of NPE. Every two bits from 15 to 26 can handle a multiplexer to configure the connection of SPE. In addition, 14 bits control the function of SPE. Every two bits configure SPE to determine whether to use the activation function or not. Finally, four bits select the computing shapes to perform different operations.

3.3. *Data scheduling for multi-bank RAMs*

It is necessary to store the output data of the convolution operation from the PEs array. Meanwhile, it takes into consideration how to read the storage data conveniently for use of the next calculation or next layer. A high-capacity global memory can be used to store and schedule the intermediate data in the system. However, such method has not considered combining with the characteristic of CNN arithmetic. Therefore, it needs to design an extra access pattern of data scheduled for multiple channels in a single memory. With the increase of parallelism, the difficulty of data scheduling will increase greatly. The memory storage structures in the existing CNN accelerators are specially designed for convolution computation. Most of them employ spatial architecture for parallel computing to improve throughput and energy efficiency. In this paper, an interlaced data access mechanism for multi-bank RAM is proposed. It can meet writing scheduling of output data at different computing patterns and facilitate reading schedule for the subsequent data, thereby improving utilization of resources and realizing high performance of computation.

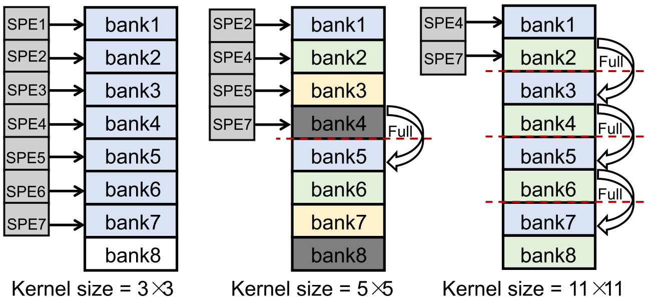

Fig. 7. Data storage distribution in multi-bank RAMs.

Figure 7 shows the mechanism of storage distribution of data. Each row of PEs array corresponds to one memory, which is partitioned to eight banks (i.e., bank1–bank8). Each bank is the 11-bit width and 64 depth. When writing data into banks, the storage path can be configured according to computing shape. Taking AlexNet as an example, it includes three kernel sizes: 3×3, 5×5, and 11×11. Together with the configuration of PEs array for computing shape, the storage path can be changed simultaneously. At 3×3 kernel, the parallelism of each row in PEs array is up to 7, so that each row obtains results of seven output channels from SPE1–SPE7 simultaneously. Bank1–bank7 are assigned for these channels, respectively. In this case, SPE1–SPE7 connects in order with bank1–bank7 via multiplexer and the seven output data are written into bank1–bank7, respectively. At 5×5 kernel, the parallelism of each row in PEs array is 4, representing that results of four output channels are calculated from SPE2, SPE4, SPE5, and SPE7, respectively. The eight banks are partitioned into two sets. The output data of four channels are stored into the first set (i.e., bank1 –bank4) if the set is not full, or into the other set (i.e., bank5 –bank8) in case that the first set is full. At 11×11 kernel, the parallelism is only 2, indicating that the results of two output channels come from SPE4 and SPE7. The eight banks are partitioned to four sets (i.e., bank1–bank2, bank3–bank4, bank5–bank6, bank7–bank8). That is, the results of two output channels can be mapped onto four sets. These four sets can be sequentially used to expend the capacity. Similarly, the data are first stored into the first set, and then into other sets in succession if former set is full.

Figure 8 illustrates the structure of connection between multi-bank RAM and SPEs, which is derived from the aforementioned mechanism of storage distribution. The connection can be switched via multiplexer. The outputs from SPE1, SPE3, and SPE6 are solely stored into bank1, bank3, and bank6, respectively. The reason is that SPE1, SPE3 and SPE6 generate the convolution results for 3×3 kernel only, while the three SPEs are configured to turn into NPEs when kernel size is 5×5 or 11×11. SPE2 and SPE5 are in the same situation. They do not generate the convolution results for 11×11 kernel. For 3×3 kernel, the outputs of SPE2 and SPE5 will connect with bank2 and bank5, respectively. Also, when kernel size is 5×5, the

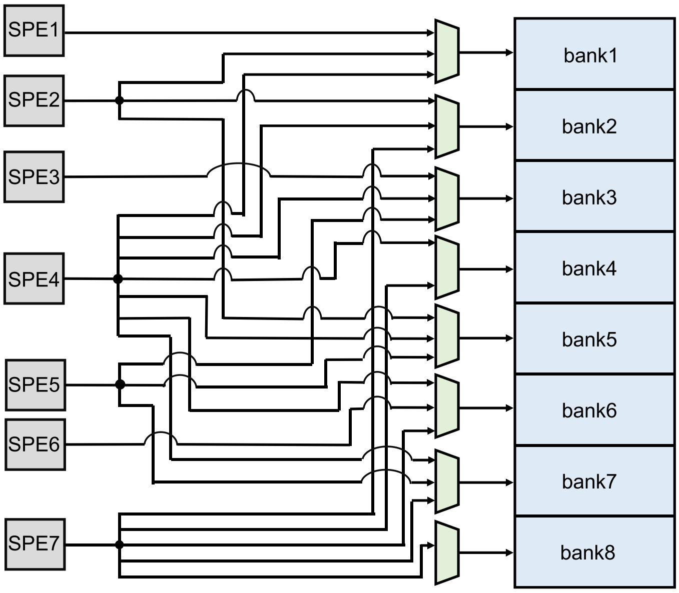

Fig. 8. Structure of connection between multi-bank RAM and SPEs.

outputs of SPE2 may be assigned to bank1 and bank5, and the outputs of SPE5 may be assigned to bank3 and bank7. Since SPE4 and SPE7 always act as the output processing elements, they have more mapping destinations. For instance, the output of SPE4 may be stored into bank4 when kernel size is 3×3, into bank2 and bank6 for 5×5 kernel, and into bank1, bank3, bank5, bank7 under the condition of 11×11 kernel.

The computation data of SPEs are from different input channels. Meanwhile, each bank in one set stores the data belonging to the same output channel. In allusion to 3×3 kernel, the interlaced writing and reading mechanism for multi-bank RAM mechanism is represented in detail. Figures 9 and 10 show the writing process and the

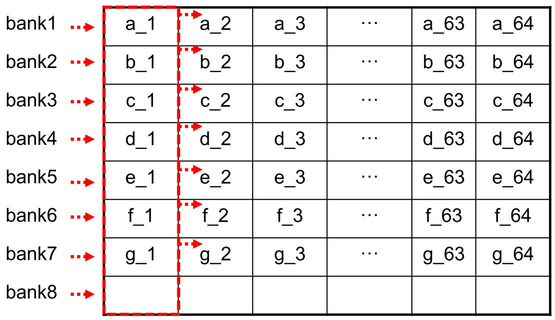
bank1
bank2
bank3
bank4
bank5
bank6
bank7
bank8
a_1 a_2 a_3 ··· a_63 a_64
b_1 b_2 b_3 ··· b_63 b_64
c_1 c_2 c_3 ··· c_63 c_64
d_1 d_2 d_3 ··· d_63 d_64
e_1 e_2 e_3 ··· e_63 e_64
f_1 f_2 f_3 ··· f_63 f_64
g_1 g_2 g_3 ··· g_63 g_64

Fig. 9. Writing in parallel to multi-bank RAM.

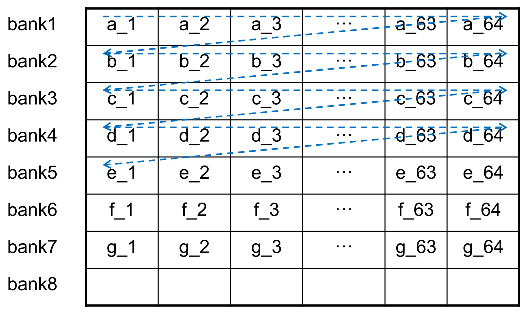
bank1
bank2
bank3
bank4
bank5
bank6
bank7
bank8
a_1 a_2 a_3 ··· a_63 a_64
b_1 b_2 b_3 ··· b_63 b_64
c_1 c_2 c_3 ··· c_63 c_64
d_1 d_2 d_3 ··· d_63 d_64
e_1 e_2 e_3 ··· e_63 e_64
f_1 f_2 f_3 ··· f_63 f_64
g_1 g_2 g_3 ··· g_63 g_64

Fig. 10. Reading bank-by-bank from multi-bank RAM.

reading process, respectively. During writing, the convolution data generated by the SPEs are stored into the multi-bank column-by-column in parallel. That is, the data from SPE1–SPE7 are stored into bank1–bank7, respectively. However, the reading process is different from the writing process. Since the data of different output channels are stored into multi-bank, the readout should be performed channel by channel. That is, the data are sequentially read from bank1 to bank7. As shown in the blue dotted line of Fig. 10, it reads data from a_1 to a_64, meaning the finishing readout of bank1. Then it continues to read the bank2 until all are read out. In this way, it executes repeatedly until bank7 is empty. The proposed multi-bank access pattern can directly write data of multi-channel in parallel without complex storage control. It can effectively improve the speed of writing process and regularly distribute the data. In addition, the access pattern can expediently read data as required for the use of the next operation. Hence, this access pattern is more efficient for convolution operation in high-degree parallel.

3.4. *Image row broadcast dataflow*

Aiming at the RNA hardware architecture, the mapping method for CNN dataflow is dedicatedly designed in this paper, called image row broadcasting dataflow (IRBD). IRBD is a mechanism of data mapping onto the reconfigurable PEs array. It supports multiple neural networks for acceleration. Taking 3×3 kernel as an example, the operation steps of IRBD are shown in Fig. 11.

- **Step 1.** Distributing filter weights onto the PEs array. As shown in Fig. 11, the kernel size is 3×3, and every PE set processes one-row data from kernel. To realize the convolution operation, three PE units (including two NPEs and one SPE) by interconnection constitute a PE set, which is corresponded to the kernel size. The PEs array can be configured to process multiple convolution operations in parallel. Therefore, the weights stored in the filter memory are distributed and fixed to the PEs array during the convolution process, as demonstrated in black lines of Fig. 11.

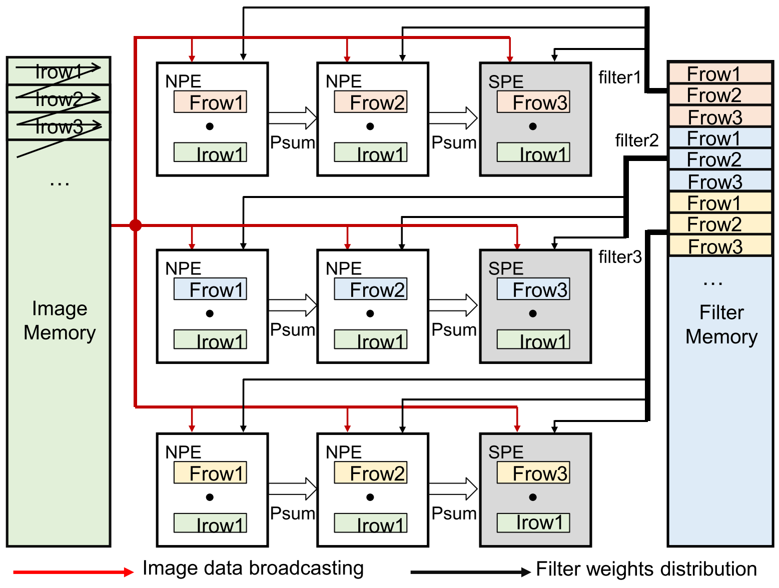

Fig. 11. Image row broadcasting dataflow (IRBD).

- **Step 2.** Broadcasting the image data onto the PEs array. The image data are stored in image memory. After filter weights distribution, the image data are mapped onto PEs via broadcasting row by row to perform MAC operations with weights, as demonstrated in red lines of Fig. 11.
- **Step 3.** Storing the intermediate result of the convolution into the FIFO in the NPE. The result, expressed as Psum, will be used as the input for the following NPE or SPE. The ReLU function module activates the result from the FIFO in the SPE to generate the convolution value.

In this way, there are two main advantages for convolution acceleration. On the one hand, on the basis of dynamically reconfigurable technology, RNA accelerator adopts IRBD as the computation mapping mechanism for CNN dataflow, which separates the image data and filter weights in different ways flowing to PEs, avoiding the complex sequence control. On the other hand, due to the kernel sliding on the image, the filter weights will be reused repeatedly to compute with all the image data. Although systolic array,[13] an acceleration mechanism for matrix multiplication, can also reuse filter weights to compute in parallel, the processing states need to proceed in well-designed lock steps, which impose restrictions on flexibility for supporting various convolution shapes. To sum up, IRBD can improve the reusability of data and avoid repeated reading of filter weights, as well as maintain more flexibility, thereby lowering the power and promoting the performance.

3.5. *Tile-by-tile computing*

For CNNs, convolution layer is computation-intensive due to a large number of MAC operations, and FC layer is memory-intensive due to a large number of weight-storage. In general, typical CNN accelerators use a layer-by-layer strategy, which makes convolutional layer with computation-intensive and FC layer with memory-intensive perform separately. This strategy is more in line with the architecture of the neural network model. However, it is implied that the needs for computing and memory accessing cannot be combined to make the maximum use of hardware resources for the improvements of inference. Though the FC layer consumes only a tiny portion of the whole inference time, its large memory bound results in long latency for data access. Hence, the strategy for computing convolutional layers and FC layers at the same time is beneficial for accelerating inference. In this work, the tile-by-tile computing strategy is proposed instead of the layer-by-layer computing as a computing flow to simultaneously process convolutional layers and FC layers, thereby improving throughput and reducing inference time. In addition, this strategy can mitigate the usage of hardware resources, which is derived by the reduction of accessing data.

The key idea for the tile-by-tile computing strategy is that it allows the computation for convolution layer in parallel with the data reading for the FC layer. The process of tile-by-tile computing is described as follows. Firstly, the input image is divided into different tiles. Secondly, for the present tile, it executes the convolution for the weight and the image tile until the FC layer is achieved. It means finishing the convolution operation of a tile. Then, the next tile is performed in the same way, meanwhile, the weights of FC layer are read. In other words, it is synchronous for full connection operation of the current tile and convolution operation of the next tile. Finally, it generates the result after completing the operation of all the tiles. Taking AlexNet as an example, Table 3 lists the partition of the first tile for computation. The coordinates denote the corresponding pixel of the image diagonal in the top-left tile.

Figure 12 shows the TTC workflow in this example. TTC allows the convolutional layer to perform in parallel with FC layer's data reading, so that two processes

Table 3. Computation partition of the first tile for AlexNet.

Layer	Area
FC	(1,1), (2,2)
CONV5	(1,1), (5,5)
CONV4	(1,1), (6,6)
CONV3	(1,1), (7,7)
CONV2	(1,1), (17,17)
CONV1	(1,1), (39,39)
Original image	(1,1), (163,163)

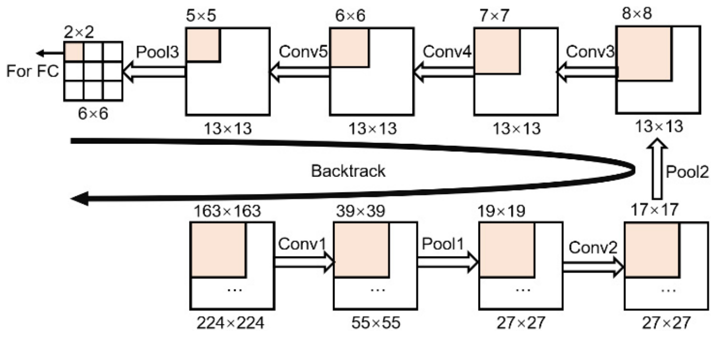

Fig. 12. Example of the tile-by-tile computing mode.

overlap concurrently to accelerate the CNNs. After Pool3 in Fig. 12, the size of the image is 6×6, which is divided averagely into nine 2×2 tiles. For the first 2×2 tile, it can get the original input image data by backtracking the structure of AlexNet. It can be seen that the 2×2 tile in the FC layer is derived from the 163×163 tile of the input layer. According to the tile-by-tile computing, it firstly performs the convolution operation on this tile until the final convolutional layer is finished. Next, the data reading of the FC layer for this tile starts. At the time, it can compute the next tile. This means that this simultaneity between the computing of convolution layer and the data reading of FC layer speeds up the processing of CNN. In this example, with the layer-by-layer computing, it needs to read the 6×6 data. While using tile-by-tile computing, the amount of data is 2×2. Hence, TTC strategy saves eight-ninths bandwidth compared to typical layer-by-layer computing. Specifically, tile-by-tile computing strategy reduces the storage of data over the layer-by-layer computing by the greatest amount in CONV1 (49.7%), CONV2 (60.4%), CONV3 (71.0%), CONV4 (78.7%), CONV5 (85.2%), FC (88.8%), which indicates the reduction of hardware resources.

Figure 13 shows the execution flow of performing TTC on PEs array. As described above, the input image of convolutional layer 1 is firstly divided into different tiles (i.e., *Tile 1–Tile N* as shown on the left in Fig. 13). The x-axis in Fig. 13 represents the stage time. In each stage (i.e., *Stage 1–Stage N*), the PEs array performs the convolutional computation of the corresponding tile throughout all convolutional layers. The y-axis in Fig. 13 expresses the layer time. In deferent layer times, the PEs array is configured to a dedicated data path under the control of configuration words, which depend on convolutional kernels of different layers (i.e., kernels of *Layer 1–Layer M*). When performing the execution of TTC, the PEs array starts with *Stage 1* and performs *Tile 1* throughout all convolutional layers along with y-axis. Then moving to the next stage (i.e., *Stage 2*) along x-axis, the PEs array performs the next tile (i.e., *Tile 2*). Such a process is repeated for every stage one by one, until the last tile (i.e., *Tile N*) is executed. At each layer time of stage time, the whole PEs array

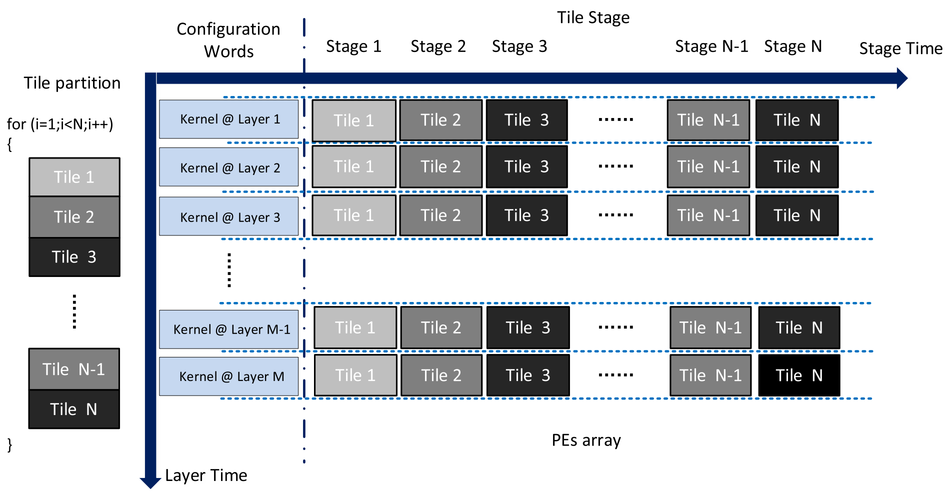

Fig. 13. The computing flow of TTC execution on PEs array.

concentrates on the computation of one tile. The function of PE and interconnections between PEs for performing the current tile is configured by configuration words for the kernel of the current layer.

3.6. *Zero detection technology*

For CNNs, ReLU activation function introduces a large number of zero-value by rectifying all the negative results to zero-value. While the number depends on the input data of CNNs, it tends to increase with deep layers. In Ref. 42, it gathers statistics that the percentage of zero elements in each layer of AlexNet is 57.53% on average, especially, it accounts for 77.6% in the CONV5. In VGG, this proportion is even up to 88.5%. For the FC layer, its output comes from the results of multiply-accumulate for data of the last convolutional layer and the corresponding weights. Based on the feature of zero, it is unnecessary for data to multiple with weights, leaving out the computing. Therefore, For the FC layer, the weights corresponding to the pixel with image data of zero can be regarded as unnecessary access. In particular, the FC layer is memory-intensive. It is crucial for the performance improvement of acceleration to optimize the computation on zero-value. In this perspective, ZDT can effectively skip the reading for unnecessary weights to reduce the accessing from memory.

Figure 14 shows the framework of ZDT. When implementing the CNNs onto hardware without the zero detector, the designed PEs array issues the reading demands named rd_en_1 and rd_en_2 enabling reading operation from memories, then the input data and weights are read into PEs array for computation. A zero detector is employed into the system, working in conjunction with the PEs array. It

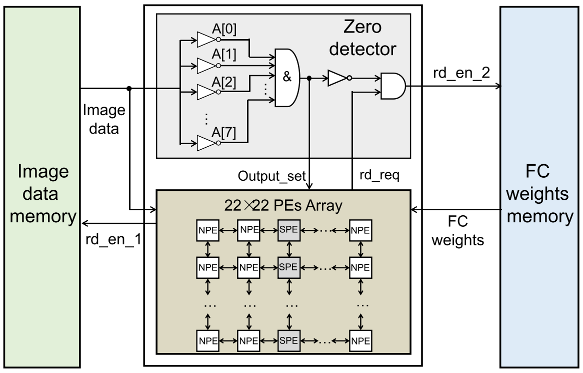

Fig. 14. Framework of zero detection technology.

consists of nine inverters, one 8-input AND gate and one 2-input AND gate. The zero detector determines whether the image data value is 0 or not. If the value is not 0, the output of 8-input AND gate is 0, that is, the output_set is 0. At this time, the rd_req from PEs array is 1, which enables rd_en_2 to read the FC weights from memory to PEs array for computation. If the value is 0, the 8-bit data become 8'b1111_1111 via eight inverters. That is, the input of 8-input AND gate is 8'b1111_1111, which means that the output is 1. The output_set signal generated by the 8-input AND gate is 1. It makes the output of 2-input AND gate be 0, that is, rd_en_2 is 0. This means that it is unnecessary to read the FC weights in memory. At the first FC layer of AlexNet, the number of zero data accounts for about 80% of the weight data.[40] Therefore, ZDT can reduce the reading of weight data by about 80%, which means that the time for data access can be reduced and the performance can be improved.

3.7. *Dynamic adaptive data truncation*

For the CNNs acceleration, it involves the problem with the fix-point interception of intermediate data. For example, the maximum bit width for the multiplication of two 8-bit inputs is 16. The result from multiplication of previous layer is used as the input of the next layer. Obviously, this will lead to increase in the data width. The sequential increase is unpractical for the hardware design. Therefore, it is desirable that the bit width of data is constant. Eyeriss[40] and DNPU[17] detect the range distribution of data for each layer and then determine them. Assume that constant bit width represents all the data for each layer. The large loss of accuracy is inevitable for some data. Diannao[16] uses a truncated multiplier to truncate data directly, which can save many hardware resources. However, the accuracy for the truncated multiplier is still not high, and it is necessary to design a custom circuit specifically. Besides, it is impossible to utilize the DSP hardware resources on the FPGA.

In this work, 8-bit fixed point operand is employed instead of 16-bit fixed point and floating-point operand. For the 8-bit fixed point operand, the dynamic adaptive data truncation is proposed to address the data width mismatch and reduce the resource consumption, as well as compensate the accuracy loss. This mechanism keeps the high-bit and more decimal part possible by data flag and dynamical detection. Figure 15 shows the workflow of DADT. It can be realized according to three steps. The first step is the detection and storing of decimal point place. By employing 8-bit width, it only needs 3-bit to denote the decimal place. For example, the 3-bit is 3'b001, it signifies that the 8-bit number consists of 7-bit integer front and 1-bit decimal last. The second step is the computation of integer and decimal. For 8-bit number, it is multiplied by using the DSP, while for 3-bit number, it is added by adder. It generates 16-bit number via DSP and 4-bit number via adder, as shown in Fig. 15. The final step is the truncation of multiplication value and the modulation of a decimal value.

The process illustrated in the red box of Fig. 15 is detailed in Fig. 16. We define that in the 16-bit number there exists N-bit zero from the highest bit to the first non-zero bit and the place of decimal point after adding is M. At truncation, the N-bit

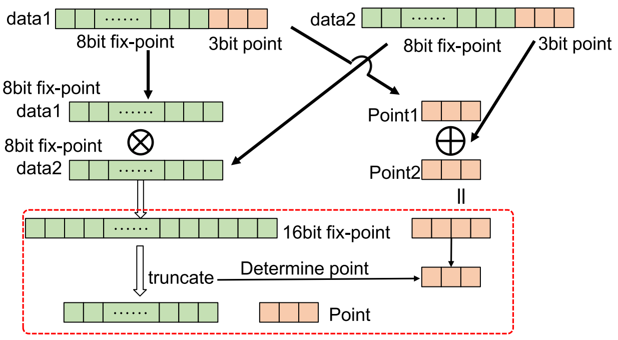

Fig. 15. Workflow of dynamic adaptive data truncation.

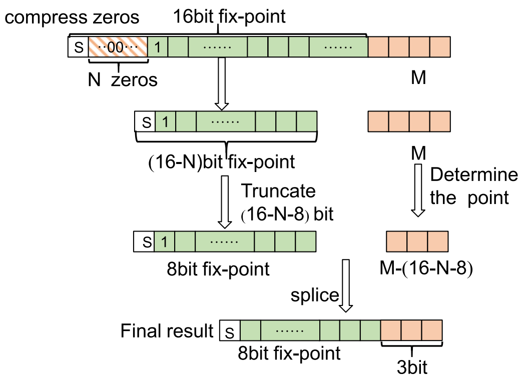

Fig. 16. Truncation mechanism.

Table 4. Relative error comparison with truncated multiplier.

Integer part	Relative error of truncate multiplier	Relative error of DADT
6'b1xxxxx	$2^{-8} < e < 2^{-7}$	$2^{-8} < e < 2^{-7}$
6'b01xxxx	$2^{-7} < e < 2^{-6}$	$2^{-8} < e < 2^{-7}$
6'b001xxx	$2^{-6} < e < 2^{-5}$	$2^{-8} < e < 2^{-7}$
6'b0001xx	$2^{-5} < e < 2^{-4}$	$2^{-8} < e < 2^{-7}$
6'b00001x	$2^{-4} < e < 2^{-3}$	$2^{-8} < e < 2^{-7}$
6'b000001	$2^{-3} < e < 2^{-2}$	$2^{-8} < e < 2^{-7}$

zeros will be discarded to form a (16-N)-bit number. Then, it extracts 16-bit number from (16-N)-bit to (16-N-7)-bit. That is, (8-N)-bit is also discarded and the remained 8-bit is valid bit. At modulation, the actual place of decimal point should be M-(8-N). This value is represented as a 3-bit binary number. The remaining 8-bit number and generated 3-bit number will constitute 11-bit number for follow-up use. In this way, it not only unifies the bit width of data, but also retains the maximum precision.

The key idea behind the DADT mechanism is to retain the most significant bit (MSB) to the greatest extent with satisfying the bit width. Inevitably, the data error exists in the proposed truncation mechanism. This error appears on the truncated portion, that is, (8-N)-bit. The highest bit after truncation is definitely 1 and placed at m-bit in 16-bit. The highest bit of the truncated data is (m-7) in 16-bit. We evaluate the impact with relative error, which is derived from Eq. (1).

$$R\_e = \frac{A\_e}{A\_c}\,. \tag{1}$$

In Eq. (1), R_e is the relative error, A_e is the absolute error, and A_c is the actual value. For the proposed truncation technology, the R_e ranges from 2^{-8} to 2^{-7} with no relationship to actual data because high-bit zeros are cut out. While for the traditional truncated multiplier like Ref. 16, the R_e becomes bigger with the increase of number of high-bit zeros. The reason is that A_e is the low 8-bit but the A_c becomes small with more high-bit zero. For example, the integer part is 6-bit, and both relative errors are listed in Table 4. The comparison indicates that the relative error of DADT mechanism is within a certain range while that of the traditional method is related to the data value itself.

4. Experimental Evaluation

In this section, the experimental setup for the evaluation of RNA is described firstly. Then, the results of implementation with 8-bit fixed point operand on FPGA are given. Finally, the performance comparison of RNA to other designs is listed and discussed.

4.1. *Experimental setup*

The proposed RNA is implemented on a Xilinx Zynq-1000 equipped with a dual ARM core and a Kintex-7 XC7Z100 FPGA. FPGA contains enough DSP and LUT to realize convolution operations of PEs array, and enough memory to meet storage of data scheduling. Figure 17 shows the experimental system for RNA. The ARM core as host end can control the RNA. It configures registers to achieve different CNNs models and layer types and exchanges data with memories via AHB (Advanced High Performance) Bus. The controller contains FSM and configuration. The registers in the controller are configured via Slaver interface, which stores the configuration words of CNN model, such as parameters (layer numbers) of CNN model and properties (kernel size, stride, etc.) of each layer. The other three interfaces are used for data transmission (image data and filter weights). DDR is an off-chip memory to store image data. At first, the ARM core reads the image data from DDR through rwMaster interface. Next, the configuration of the first tile is transferred into the registers via Slaver interface, and simultaneously, the image data are read into the image buffer via rMaster interface. Then, the accelerator starts to run and keeps it up tile-by-tile. The output of PEs array is stored in output buffer. Finally, when the last tile is executed, RNA writes the result data to ARM core via wMaster interface.

4.2. *Implementation results*

Three CNN models (i.e., LeNet, VGG, and AlexNet) are tested to classify different datasets on the evaluation system. CIFAR-10 is used for AlexNet and VGG, and MNIST is used for LeNet. The inference accuracy and hardware resource are compared on 8-bit fixed point operand and 16-bit fixed point operand of RNA, as listed in Table 5. Experimental results of the three CNN models show that compared with 16-bit fixed point, the 8-bit fixed point operand can economize the hardware resource and power consumption along with a comparable accuracy on the ImageNet dataset. Figures 18 and 19 clearly manifest the superiority of the 8-bit fixed point operand

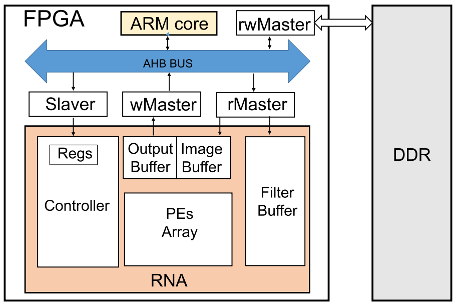

Fig. 17. Experimental platform.

Table 5. Comparison of different bits fixed point of RNA.

	Items	8-bit fixed point	16-bit fixed point
Inference accuracy	AlexNet	73.45%	73.50%
	LeNet	98.62%	98.62%
	VGG	80.33	80.40%
Hardware resource	LUT	86.878 K	132.00 K
	Register	118.88 K	201.06 K
	BRAM	0.84 Mb	1.56 Mb
	Frequency	250 MHz	200 MHz
	Power	5.50 W	6.63 W

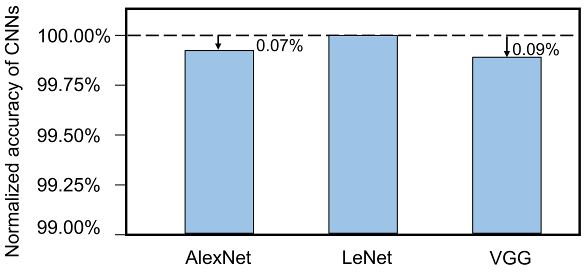

Fig. 18. Normalized comparison of inference accuracy between 8-bit versus 16-bit fixed point operands.

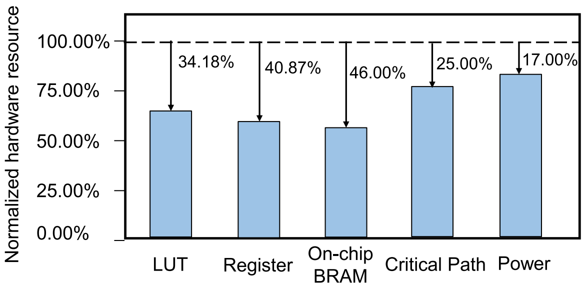

Fig. 19. Normalized comparison of hardware resource between 8-bit verus16-bit fixed point operands.

when normalized to that of 16-bit fixed point operand. Using 8-bit fixed point operand in RNA can save 34.18% LUT, 40.87% register, 46% on-chip BRAM, and 17% power consumption at the loss of 0.07–0.09% inference accuracy reduction. In addition, the running frequency of 8-bit fixed point operand is higher than that of 16-bit fixed point operand, increasing by 25%. The reason is that the lower bit width of data operation shortens the critical path in logic synthesis, thereby improving the clock frequency. Therefore, the proposed RNA is implemented using 8-bit fixed point operand.

Table 6. Breakdown of hardware resource for implementation.

Items	LUT	Register	BRAM	DSP
NPE	165	227	1.375 Kb	3
SPE	167	238	1.375 Kb	3
PEs array	82,722	115,852	665.5 Kb	1,452
Weight-storage	0	0	60.5 Kb	0
Image-storage	0	0	20.48 Kb	0
Intermediate-storage	0	0	121.00 Kb	0
Others	4,156	3,029	0	0
Total on use	86,878	118,881	0.84 Mb	1,452
Total on FPGA	277,400	554,800	13.27 Mb	2,020
Utilizations	31.32%	21.43%	6.33%	71.88%

The RNA accelerator was implemented on Xilinx Kintex-7 XC7Z100 FPGA. Table 6 lists the breakdown of hardware resource for the accelerator. Implementation result shows that RNA consumes more than half of the total DSP resource on FPGA, as well as 31.32% LUT, 21.43% Register, and 6.33% on-chip BRAM of the total hardware resource.

Table 7 lists the performance of RNA implementation. It shows that the RNA supports three different CNN models when running at 250 MHz clock frequency under 5.50 W power consumption. The PEs array contains 484 PEs, whose utilization rate reaches 95.5% for kernel size 3×3, 90.1% for kernel size 5×5, and 100% for kernel size 11×11, respectively. The peak throughputs of RNA are as follows: AlexNet at 598 GOPS, LeNet at 500 GOPS and VGG at 660 GOPS. Accordingly, the energy efficiencies for these three CNN models are 108.7 GOPS/W, 90.9 GOPS/W and 120.0 GOPS/W, respectively. In addition, the inference latencies of three

Table 7. The implementation results of RNA.

Platform	Kintex-7 XC7Z100	
Frequency	250 MHz	
Power	5.50 W	
Precision	8-bit fixed point operand	
CNN model	AlexNet, LeNet, VGG	
Kernel size	3×3, 5×5, 11×11	
Number of PEs	484	
PE utilization	3×3 kernel size 95.5%	
	5×5 kernel size	90.1%
	11×11 kernel size	100%
Peak throughput	AlexNet:	598 GOPS
	LeNet:	500 GOPS
	VGG:	660 GOPS
Frame rate	AlexNet:	88.7 fps
	LeNet:	93,999.7 fps
	VGG:	16.2 fps
Energy efficiency	AlexNet:	108.7 GOPS/W
	LeNet:	90.9 GOPS/W
	VGG:	120.0 GOPS/W

CNN models (i.e., AlexNet, LeNet, and VGG) performed on RNA are 11.27 ms, 0.0106 ms, and 61.64 ms, respectively. That is to say, RNA can process 88.7, 93,999.7 and 16.2 frames per second (fps) for these three CNN models. It should be noted that the first convolutional layers of AlexNet and VGG both include three input channels, which are collectively regarded as one frame for these two CNN models. It is worth mentioning that the frame rates are beneficial from TTC computing mode, so that the memory bandwidth required by intermediate data among convolutional layers is completely eliminated.

4.3. *Performance comparison*

Tables 8 and 9 compare implementation results of the proposed RNA accelerator with other recent designs, including GPU, ASIC designs, and FPGA-based designs.

Firstly, the performance and energy efficiency are evaluated. The comparison indicates that the proposed RNA has shown significant superiority over the other FPGA designs in terms of performance and GPU in terms of energy efficiency. In addition, due to complicated fine-grained interconnection and redundant logic on FPGA, the energy efficiency of the proposed RNA is inferior to that of ASIC designs, but the performance approximates even exceeds some ASIC designs. Aiming at the competitors of FPGA-based designs, RNA can improve the performance by 4.3× compared to Ref. 23 for AlexNet, by 1.5× compared to Ref.46 for AlexNet, by 1.7× compared to Ref. 43 and by 1.8× compared to Ref. 12 for VGG, respectively. Such performance improvements on RNA are partially due to DADT mechanism, so that 8-bit fixed point operation can be used to shorten critical path as well as maintain a high inference accuracy. Therefore, compared with other FPGA-based designs, RNA

Table 8. Comparisons with GPU and ASIC designs.

Work	CVPR 2016 Ref. 11	TNNLS 2019 Ref. 45	JSSC 2017 Ref. 40	JSSC 2018 Ref. 41	TCAS-I 2018 Ref. 44	RNA
Platform	GPU NV Tian X	ASIC 28 nm	ASIC 65 nm	ASIC 65 nm	ASIC 65 nm	FPGA Kintex-7
CNN model	VGG	VGG	AlexNet	AlexNet/VGG/RNN	AlexNet	AlexNet/LeNet/VGG
Freq. (MHz)	1,126	500	200	200	500	250
Precision	Float 16	Fixed16	Fixed 16	Fixed 8/16	Fixed 16	Fixed 8
Perf. (GOPS)	7,060.0	420.0	40.0	409.6	152	598/500/660
Power	250 W	155.0	278 mW	290 mW	350 mW	5.50 W
EE$^{\$}$ (GOPS/W)	28.24	2,715.0	143.88	1,270	434	108.7/90.9/120
Logic cell*	NA	NA	1,852 K	2,950 K	1,300 K	86 K
DSP	NA	NA	NA	NA	NA	1,452
On-chip Memory	NA	1,088 Kb	864 Kb	2,784 Kb	NA	983 Kb

Notes: $^{\$}$EE is the abbreviation for energy efficiency.
*Both of Xilinx and Intel FPGAs' logic cell count in term of LUT, while the logic cell of ASIC designs is calculated by gate count.

Table 9. Comparisons with FPGA-based designs.

Work	FPGA 2018 Ref. 43	FCCM 2017 Ref. 12	FPGA 2016 Ref. 23	ACCESS 2020 Ref. 46	RNA
Platform	FPGA Stratix-V	FPGA Stratix-V	FPGA Zynq	FPGA Arria 10	FPGA Kintex-7
CNN model	VGG	VGG	AlexNet	AlexNet/VGG	AlexNet/LeNet/VGG
Freq. (MHz)	150	150	150	200	250
Precision	Fixed 8	Fixed 16	Fixed 16	Fixed 8–16	Fixed 8
Perf. (GOPS)	383.7	364.36	136.97	394.6/736.9	598/500/660
Power	∼27 W	25 W	9.63 W	27.2 W	5.50 W
EE$ (GOPS/W)	14.2	14.6	14.22	14.51/27.09	108.7/90.9/120
Logic cell*	320 K	42 K	218 K	360 K	86 K
DSP	3,282	1,036	780	410	1,452
On-chip Memory	41,269 Kb	NA	NA	27,320 Kb	983 Kb

Notes: $EE is the abbreviation for energy efficiency.
*Both Xilinx and Intel FPGAs' logic cell count in term of LUT, while the logic cell of ASIC designs is calculated by gate count.

can run at a higher clock frequency, achieving about 1.7× speedup. Besides, the proposed two data reuse (i.e., IRBD and TTC) mechanisms are beneficial for reducing data access delay and improving throughput. This improvement is on the basis of maintaining more than 7× energy efficiency enhancement compared to other FPGA-based designs.[12,23,43,46] As for ASIC-based designs Refs. 40, 41, 44, and 45, RNA outperforms them in terms of performance. For AlexNet, RNA can achieve the performance improvements by 14.9× over Ref. 40, by 1.5× over Ref. 41, and by 3.9× over Ref. 44, respectively. For VGG, RNA can improve the performance by 1.6× over Refs. 41 and 45. The higher performance obtained on RNA is because the design adopts lower bit width of data path to decrease the length of critical path, reconfigurable PEs array calculates in parallel with IRBD to reuse PE and data, and storage pattern employs multi-bank interlaced writing and reading to improve the speed of transmission. Compared to GPU,[11] RNA can provide energy efficiency improvements by about 4×. The reason is that RNA is a dedicatedly designed architecture aiming at CNN characteristic, along with utilizing three data optimization mechanisms (e.g., IRBD, TTC, and ZDT) to substantially reduce data movements between memories, thereby improving performance and saving power consumption.

Secondly, utilization of resource is evaluated. The RNA is implemented at less cost of hardware resource than other FPGA-based or ASIC-based designs while maintaining high performance and high energy efficiency. Compared with recent FPGA designs,[23,43,46] the logic cell of RNA decreases by 76.11%, 73.13% and 60.55%, respectively. Such redundancy on hardware overhead of RNA is because the 8-bit fixed point operand used for resource saving, as shown in Table 5. Even though the use for the logic cell and DSP of RNA is 40.15–104.76% more than that of Ref. 12, the performance of RNA has achieved much more improvements (i.e., 1.8×) in return.

As for storage overhead, 97.61% on-chip RAM saving on RNA is achieved, when compared to FPGA design in Ref. 43. The comparison result of on-chip RAM with the recent ASIC design indicates that RNA utilized only 35.30% on-chip RAM to that of Ref. 41. Such memory capacity saving on RNA is mainly due to the 8-bit fixed point operand adapted in the proposed accelerator, which directly makes the capacity of data storage reduce by 46% compared with 16-bit fixed point version as shown in Fig. 19. The other reason for such improvement attributes to the proposed multi-bank RAM with interlaced writing and reading mechanism, which makes full use of on-chip memory to avoid storage capacity wastage, thereby substantially saving the memory resource.

Finally, configurability is evaluated. For FPGA designs,[12,23] they generally perform one CNN model. If they are required to support other CNN models, they have to redesign and recompile their hardware architectures and generate new FPGA bit streams for processing new CNNs. In contrast to typical statically reconfigurable FPGA designs,[12,23,43,46] RNA employs dynamically reconfigurable computing architecture to rapidly configure data path between PEs at run-time, thereby achieving fast switching among different kernel sizes and CNN models. RNA can form a new data path in the heterogeneous PEs array for performing different operations by switching the configurable context during several clock cycles. Moreover, RNA has the ability of great expandability for other more kernel sizes (such as 4×4, 6×6, and 7×7), only simply rerouting the interconnection between PEs is required.

5. Conclusion

This paper proposes a dynamically reconfigurable accelerator RNA, which employs dynamically reconfigurable computing framework to provide hardware flexibility for diverse CNN models. To improve performance and save power consumption, three data optimization mechanisms, including IRBD, TTC, and ZDT, are especially designed for RNA to improve data reuse and reduce data movements between memories. Equipped with DADT mechanism, the computational precision in RNA is reduced from 16-bit to 8-bit, which contributes to substantially save hardware overheads and improve performance. RNA is implemented on the Xilinx Kintex-7 XC7Z100 and runs at 250 MHz. Experimental results show that RNA achieves a peak performance of 660 GOPS and a maximum energy efficiency of 120 GOPS/W. In future work, we intend to pursue the optimization of accelerator for performance and energy efficiency. Additionally, we also seek to design with more flexibility and less resource to support more CNN models.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62176206, and in part by the Aeronautical Science Foundation of

China under Grant 2020Z066070001, and in part by Key-Area Research and Development Program of Guangdong Province under Grant 2019B010154002.

References

1. R. Girshick, J. Donahue, T. Darrell and J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Columbus, Ohio, USA, 2014, pp. 580–587.
2. T. Okamoto, M. Odagawa, T. Koide, S. Tanaka and H. Mieno, Feature extraction of colorectal endoscopic images for computer-aided diagnosis with CNN, *Proc. Int. Symp. Devices, Circuits and Systems (ISDCS)*, Higashi-Hiroshima, Japan, 2019, pp. 1–4.
3. J. Eapen, D. Bein and A. Verma, Novel deep learning model with CNN and bi-directional LSTM for improved stock market index prediction, *Proc. IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2019, pp. 0264–0270.
4. R. He, X. Wu, Z. Sun and T. Tan, Wasserstein CNN: Learning invariant features for NIR-VIS face recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* **41** (2019) 1761–1773.
5. A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Proc. 25th Int. Conf. Neural Information Processing Systems (NIPS)*, Harrahs and Harveys, Lake Tahoe, USA, 2012, pp. 1097–1105.
6. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE*, 86 (1998) 2278–2324.
7. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv:1409.1556, 2014.
8. 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:1704.04861, 2017.
9. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 1–9.
10. K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778.
11. A. Lavin and S. Gray, Fast algorithms for convolutional neural networks, *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 4013–4021.
12. Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang and J. Cong, FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates, *Proc. IEEE 25th Annual Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, 2017, pp. 152–159.
13. N. P. Jouppi *et al.*, In-datacenter performance analysis of a tensor processing unit, *Proc. ACM/IEEE 44th Annual Int. Symp. Computer Architecture (ISCA)*, Toronto, ON, Canada, 2017, pp. 1–12.
14. J. Song, Y. Cho, J. S. Park, J. W. Jang and I. Kang, An 11.5 TOPS/W 1024-MAC butterfly structure dualcore sparsity aware neural processing unit in 8 nm flagship mobile SoC, *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2019, pp. 130–131.

15. J. Lee, C. Kim, S. Kang, D. Shin and H. J. Yoo, UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision, *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 218–219.
16. T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen and O. Temam, Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, *Proc. ACM Int. Conf. Architectural Support for Programming Languages Operating Systems (ASPLOS)*, Salt Lake City, Utah, USA, 2014, pp. 269–284.
17. D. Shin, J. Lee, J. Lee and H. J. Yoo, DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks, *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2017, pp. 240–241.
18. S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu and S. Wei, A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural network processor for deep learning applications, *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Kyoto, Japan, 2017, pp. C26–C27.
19. S. Wang, D. Zhou, X. Han and T. Yoshimura, Chain-NN: An energy-efficient 1D chain architecture for accelerating deep convolutional neural networks, *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, Lausanne, Switzerland, 2017, pp. 1032–1037.
20. G. Desoli, N. Chawla, T. Boesch, S. P. Singh and N. Aggarwal, A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28 nm for intelligent embedded systems, *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2017, pp. 238–239.
21. Y. Choi, D. Bae, J. Sim, S. Choi, M. Kim and L. S. Kim, Energy-efficient design of processing element for convolutional neural network, *IEEE Trans. Circuits Syst. II Express Briefs*, **64** (2017) 1332–1336.
22. Y. Ma, Y. Cao, S. Vrudhula and J. Seo, Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks, *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, 2017, pp. 45–54.
23. J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang and H. Yang, Going deeper with embedded FPGA platform for convolutional neural network, *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, 2016, pp. 26–35.
24. H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou and L. Wang, A high performance FPGA-based accelerator for large-scale convolutional neural networks, *Proc. 26th Int. Conf. Field Programmable Logic and Applications (FPL)*, Lausanne, Switzerland, 2016, pp. 1–9.
25. J. Wang, J. Lin and Z. Wang, Efficient hardware architectures for deep convolutional neural network, *IEEE Trans. Circuits Syst. I Regul. Pap.* **65** (2018) 1941–1953.
26. Y. Ma, Y. Cao, S. Vrudhula and J. S. Seo, Optimizing the convolution operation to accelerate deep neural networks on FPGA, *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **26** (2018) 1354–1367.
27. A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, An architecture to accelerate convolution in deep neural networks, *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **65** (2018) 1349–1362.
28. L. Lu, Y. Liang, Q. Xiao and S. Yan, Evaluating fast algorithms for convolutional neural networks on FPGAs, *Proc. IEEE 25th Annual Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, 2017, pp. 101–108.
29. C. Zhang, Z. Fang, P. Zhou, P. Pan and J. Cong, Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks, *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Austin, TX, USA, 2016, pp. 1–8.

30. S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang and W. J. Dally, ESE: Efficient speech recognition engine with spare LSTM on FPGA, *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, 2017, pp. 75–84.
31. M. Alwani, H. Chen, M. Ferdman and P. Milder, Fused-layer CNN accelerators, *Proc. 49th Annual IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Taipei, Taiwan, 2016, pp. 1–12.
32. X. Wei, Y. Liang and J. Cong, Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management, *Proc. ACM/IEEE Design Automation Conf. (DAC)*, Las Vegas, NV, USA, 2019, pp. 1–6.
33. R. Tessier, K. Pocek and A. Dehon, Reconfigurable computing architectures, *Proc. IEEE* **103** (2015) 332–354.
34. L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin and S. Wei, A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications, *ACM Comput. Surv.* **52** (2019) 1–39.
35. J. Lee, S. Park, I. Hong and H. J. Yoo, An energy-efficient speech-extraction processor for robust user speech recognition in mobile head-mounted display systems, *IEEE Trans. Circuits Syst. II Express Briefs* **64** (2017) 457–461.
36. X. Jiang, X. Liu, L. Xu, P. Zhang and N. Sun, A reconfigurable accelerator for Smith–Waterman algorithm, *IEEE Trans. Circuits Syst. II Express Briefs* **54** (2007) 1077–1081.
37. M. Fons, F. Fons and E. Cantó, Fingerprint image processing acceleration through run-time reconfigurable hardware, *IEEE Trans. Circuits Syst. II Express Briefs* **57** (2010) 991–995.
38. K. Wang, J. Chen, W. Cao, Y. Wang, L. Wang and J. Tong, A reconfigurable multi-transform VLSI architecture supporting video codec design, *IEEE Trans. Circuits Syst. II Express Briefs*, **58** (2011) 432–436.
39. S. Chen, VLSI implementation of a low-cost high-quality image scaling processor, *IEEE Trans. Circuits Syst. II Express Briefs*, **60** (2013) 31–35.
40. Y. Chen, T. Krishna, J. S. Emer and V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, *IEEE J. Solid-State Circuits* **52** (2017) 127–138.
41. S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu and S. Wei, A high energy efficient reconfigurable hybrid neural network processor for deep learning applications, *IEEE J. Solid-State Circuits* **53** (2018) 968–982.
42. S. Han, J. Pool, J. Tran and W. J. Dally, Learning both weights and connections for efficient neural networks, *Proc. Int. Conf. Neural Information Processing Systems (NIPS)*, Montreal, Canada, 2015, pp. 1135–1143.
43. R. Zhao, X. Niu and W. Luk, Automatic optimising CNN with depthwise separable convolution on FPGA, *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, 2018, p. 285.
44. L. Du, Y. Du, Y. Li, J. Su, Y. Kuan, C. Liu and M. F. Chang, A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things, *IEEE Trans. Circuits Syst. I Regul. Pap.* **65** (2018) 198–208.
45. A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. Liu and T. Delbruck, NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps, *IEEE Trans. Neural Netw. Learn. Syst.* **30** (2019) 644–656.

46. S. Li, Y. Luo, K. Sun, N. Yadav and K. K. Choi, A novel FPGA accelerator design for real-time and ultra-low power deep convolutional neural networks compared with Titan X GPU, *IEEE Access* **8** (2020) 105455–105471.
47. M. F. Haque, H. Lim and D. Kang, Object detection based on VGG with ResNet network. *Proc. Int. Conf. Electronics, Information, and Communication (ICEIC)*, Auckland, New Zealand, 2019, pp. 1–3.
48. L. Li and Y. Liang, Deep learning target vehicle detection method based on YOLOv3-tiny, *Proc. IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conf. (IMCEC)*, Chongqing, China, 2021, pp. 1575–1579.
49. Z. Wentao, G. Lan and Z. Zhisong, Garbage classification and recognition based on SqueezeNet, *Proc. World Conf. Mechanical Engineering and Intelligent Manufacturing (WCMEIM)*, Shanghai, China, 2020, pp. 122–125.
50. W. Chen, H. Wu, S. Wei, A. He and H. Chen, An asynchronous energy-efficient CNN accelerator with reconfigurable architecture, *Proc. IEEE Asian Solid-State Circuits Conf. (ASSCC)*, Tainan, Taiwan, 2018, pp. 51–54.
51. D. Shin, J. Lee, J. Lee, J. Lee and H. J. Yoo, DNPU: An energy-efficient deep-learning processor with heterogeneous multi-core architecture, *IEEE Micro* **38** (2018) 85–93.
52. S. Hsiao and P. Wu, Design tradeoff of internal memory size and memory access energy in deep neural network hardware accelerators, *Proc. IEEE Global Conf. Consumer Electronics (GCCE)*, Nara, Japan, 2018, pp. 735–736.
53. C. B. Wu, C. S. Wang and Y. K. Hsiao, Reconfigurable hardware architecture design and implementation for AI deep learning accelerator, *Proc. IEEE Global Conf. Consumer Electronics (GCCE)*, Kobe, Japan, 2020, pp. 154–155.
54. Y. H. Chen, J. Emer and V. Sze, Eyeriss v2: A flexible and high-performance accelerator for emerging deep neural networks, arXiv:1807.07928v1 (2018).
55. Y. J. Chen, T. Luo, S. L. Liu, S. J. Zhang, L. Q. He, J. Wang, L. Li, T. S. Chen, Z. W. Xu, N. H. Sun and O. Temam, Dadiannao: A machine-learning supercomputer, *Proc. ACM/IEEE Int. Symp. Microarchitecture (MICRO)*, Cambridge, UK, 2014, pp. 609–622.
56. Z. D. Du, R. Fasthuber, T. S. Chen, P. Lenne, L. Li, T. Luo, X. B. Feng, Y. J. Chen and O. Temam, Shidiannao: Shifting vision processing closer to the sensor, *Proc. ACM/IEEE Int. Symp. Computer Architecture (ISCA)*, Portland, OR, USA, 2015, pp. 92–104.
57. D. F. Liu, T. S. Chen, S. L. Liu, J. H. Zhou, S. Y. Zhou, O. Teman, X. B. Feng, X. H. Zhou and Y. J. Chen, PuDianNao: A polyvalent machine learning accelerator, *Proc. ACM Int. Conf. Architectural Support for Programming Languages Operating Systems (ASPLOS)*, Istanbul, Turkey, 2015, pp. 369–381.
58. C. Ying, S. Kumar, D. Chen, T. Wang and Y. Cheng, Image classification at supercomputer scale, arXiv:1811.06992v1 (2018).
59. J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov and A. Moshovos, Bit-Pragmatic deep neural network computing, *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Boston, MA, USA, 2017, pp. 382–394.
60. P. Judd, J. Albericio and A. Moshovos, Stripes: Bit-serial deep neural network computing, *IEEE Comput. Archit. Lett.* **16** (2017) pp. 80–83.
61. S. Sharify, A. D. Lascorz, K. Siu, P. Judd and A. Moshovos, Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks, *Proc. ACM/ESDA/IEEE Design Automation Conf. (DAC)*, San Francisco, CA, USA, 2018, pp. 1–6.
62. J. Lee, C. Kim, S. Kang, D. Shin, S. Kim and H. J. Yoo, UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision, *IEEE J. Solid-State Circuits* **54** (2019) 173–185.