An FPGA-based accelerator for deep neural network with novel reconfigurable architecture

Han Jia1, a), Daming Ren1, and Xuecheng Zou1

Abstract Due to the high parallelism, Data flow architecture is a common solution for deep neural network (DNN) acceleration, however, existing DNN accelerate solutions exhibit limited flexibility to diverse network models. This paper presents a novel reconfigurable architecture as DNN accelerate solution, which consists of circuit blocks all can be reconfigured to adapt to different networks, and maintain high throughput. The proposed architecture shows good transferability to diverse DNN models due to its reconfigurable processing element (PE) array, which can be adjusted to deal with various filter sizes of networks. In the meanwhile, according to proposed data reuse technique based on parameter proportion property of different layers in DNN, a reconfigurable on-chip buffer mechanism is raised. Moreover, the accelerator enhances its performance by exploiting the sparsity property of input feature map. Compared to other state-of-the-art solutions based on FPGA, our architecture achieves high performance, and presents good flexibility in the meantime.

Keywords: deep neural network, accelerate solutions, reconfigurable architecture, data flow

Classification: Integrated circuits (memory, logic, analog, RF, sensor)

1. Introduction

To meet the demands of computer vision applications (e.g., object recognition and classification), numerous deep convolutional neural network (CNN) models has emerged and achieved overwhelming performance [1, 2, 3, 4, 5, 6]. Along with the development of deep learning algorithms, a broad variety of applications in Internet of Things (IoT) rises. However, state-of-the-art DNNs based on deep network models put forward requests for enormous computation and memory access, which lead to great latency and computation resource consumption. Hence, hardware acceleration solutions with real-time processing become an increasingly urgent demand.

DNN acceleration solutions mainly faces two bottlenecks: enormous multiply and accumulate (MAC) operations and great number of parameters. To deal with these problems, researchers have been focused on application and specific integrated circuits (ASIC) [7, 8, 9, 10, 11, 12, 13, 14, 15] and field-programmable gate array (FPGA) [16, 17, 18, 19, 20, 21, 22, 23, 24]. Due to its high parallelism property, data flow architectures has become a key research area [8, 9, 10, 11, 12, 13, 18, 19, 20]. For example, tensor processing unit (TPU) [9] is a typical data flow based architecture, which accelerate the convolutional operations by utilizing systolic matrix multipliers. Compared to contemporary GPU and CPU, TPU achieves approximately 15X–30X speedup. Apart from TPU, Eyeriss [8] exploits reuse strategy of input feature map and weight filters by proposing the row stationary (RS) data flow, and achieves high energy efficiency. Based on above excellent researches, data flow architecture shows great potential in DNN acceleration.

Except for the two bottlenecks mentioned above, adapt to different DNN models [25, 26, 27, 28, 29, 30] while maintaining good performance is another emerging demand for data flow based accelerators. Many researchers have also been dedicated to solve this problem. For instance, based on hardware reconfiguration technique, Venieris [20] proposes a synchronous data flow architecture. By reconfiguring the FPGA, this work can adjust its architecture and hardware resource according to diverse network layers, and it achieves significant speedup compared to other FPGA-based architectures. But it also has drawbacks, that is for different sub-graphs, the accelerator needs to be reconfigured, and thus restrains its performance.

In this paper, a novel reconfigurable architecture is proposed to deal with diverse DNN models while maintaining high performance. All hardware modules of the architecture can be reconfigured to referring to the network framework. To realize high flexibility, reconfigurable PE has been designed to adapt to different filter sizes of various DNN models. To achieve high performance, reconfigurable on chip buffer and corresponding data reuse technique is proposed. What’s more, by utilizing the sparsity property of DNN, an effective utilization method is also introduced. Thus, a high performance accelerate solution with high flexibility is designed. This paper is organized as follows. A novel reconfigurable DNN accelerate solution is presented in section 2. The experimental results on FPGA with different DNN models are demonstrated in section 3. The conclusions are narrated in section 4.

2. Novel reconfigurable DNN accelerator architecture

2.1 Architecture overview

The proposed data flow based reconfigurable architecture is shown in Fig. 1. Unlike existing architectures, by introducing the configuration register, all hardware modules of the accelerator can be reconfigured. Specifically, with configuration instructions stored in DDR, the architecture can...
be reconfigured with configuration register. Then weight and image parameters are transferred to weight and image buffer. The PE array streams calculating results into special function buffer consists of static random-access memory (SRAM) banks in parallel. Besides, special functional layers such as pooling, batch normalization (BN) and activation are included in the special function buffer, and thus minimize data access between on-chip buffer and DDR.

2.2 Reconfigurable PE for convolution
Making the tradeoff between throughput, complexity and flexibility, a spatial 2D PE array is designed, which is the key part of an accelerate solution. Compared to systolic matrix, the spatial 2D structure performs better with more complicated network structures that contains different kernel sizes [8].

As shown in Fig. 2, unlike normal PE design with one multiplier [8, 12], the proposed reconfigurable PE structure contains nine multipliers to realize 1X1 or 3X3 convolution operations, as these are the most dominant operations in existing DNN networks. Nine weight registers and nine image registers are linked to the multipliers. As part of design, image registers are used as shift registers, and transmit image data between PE. Besides, enable signals are linked to multipliers to decide work status of them, to reduce energy consumption. Multipliers outputs are set to zero for those are disabled.

With the design above, one PE can output sum of nine multipliers results, or nine separate results. Thus realize 1X1 or 3X3 convolution operations. Moreover, based on proposed PE array structure, the accelerator can deal with bigger filter sizes such as 5X5 or 7X7. Even 11X11 operation can be conducted by employing 16 PEs. Specific configuration is as shown in Fig. 3. Table I presents the multiplier utilization for different kernel sizes, some extent of waste exists. However, these wastes are tolerable as corresponding kernel sizes are less used.

To lessen on-chip data movement, rows of image data can be reused, thus energy consumption can be greatly reduced. Fig. 4 presents the reuse strategy for a filter size of 3X3 and stride of 1. In the meanwhile, utilization of SRAM banks can be significant reduced by using this strategy.

Except for convolutional layers, fully-connected (FC) layers can be taken as special convolutional layers, with 1X1 input feature map, 1X1 filter, padding of 0 and stride of 1. While for the FC layers which contains enormous weight parameters, another technique is used and will be narrated in the rest of paper.

2.3 Reconfigurable on-chip buffer
On-chip memory resources are limited in embedded implementation. What’s more, data access between on-chip memory and external memory consumes great amount of energy [8]. To deal with these problems, data reuse techniques are usually utilized [12]. For a certain layer, there exists two ideal reuse situations: store all input feature map data on-chip; store all weight parameters on-chip.

Proportion of weight parameter and image data greatly varies among different convolutional layers. As is shown in Fig. 5, dominant parameter changes from image data to weight parameter as layers go deeper.

According to this property, a reconfigurable reuse technique is proposed by select ideal reuse strategy: for shallower layer, mode 2 is used; for deeper layers, model 1 is utilized.
Thus off-chip data access can be significantly lessened. For circumstance that both parameters cannot be loaded at one time, parameter with less proportion will be chosen and split to load on-chip. This inevitably leads to data reload and extra energy consumption.

With the data reuse strategy above, reconfigurable on-chip buffer is designed. A dynamic buffer block is proposed as shown in Fig. 6. The dynamic buffer can be flexibly distributed to image or weight buffer according to demands. For example, when utilizing reuse mode 1, dynamic buffer is distributed to image buffer as input feature map need to be loaded on-chip to the best possibility. Besides of dynamic buffer, special functional operations such as Pooling, Average-pooling [31] and Eltwise [32] are conducted within on-chip buffer. With this method, along with convolutional acceleration, the proposed accelerate solution realizes complete DNN acceleration.

2.4 Optimizing strategy utilizing sparsity

According to Fig. 7, proportion of zero value in input feature map rises as the layers go deeper. This sparse prosperity is caused by the utilization of ReLU activation [25, 31, 32]. Hence, a novel optimization strategy is introduced based on the sparsity.

For the convolutional layers, when input image data of the multiplier is ‘0’, multiplication operation is skipped. Great amount of multiplication can be reduced by using this method, and leads to significant reduction of on-chip energy cost.

As for the FC layers, sparseness also increases in deeper layers. Fig. 8 shows the weight parameter proportion of each layer. Weight parameters occupy a great portion in FC layers, thus leads to long loading time and high energy consumption. So combining with the sparse prosperity, a novel optimization strategy is introduced for the FC layers.

For the convolutional layers, when input image data of the multiplier is ‘0’, multiplication operation is skipped. Great amount of multiplication can be reduced by using this method, and leads to significant reduction of on-chip energy cost.

As for the FC layers, sparseness also increases in deeper layers. Fig. 8 shows the weight parameter proportion of each layer. Weight parameters occupy a great portion in FC layers, thus leads to long loading time and high energy consumption. So combining with the sparse prosperity, a novel optimization strategy is introduced for the FC layers.
Table II Detailed parameters of accelerator.

| Parameter                          | Value |
|-----------------------------------|-------|
| Number of PE Arrays               | 4     |
| Number of PE in each PE array     | 16    |
| Number of image buffer banks for each PE array | 18     |
| Number of dynamic buffer banks for each PE array | 18     |
| Number of Special function buffer banks | 16     |

Table III Detailed parameters of one PE array.

| Kernel Size | Stride | Feature Map Number | Channel Number | Image Rows | SRAM Banks |
|-------------|--------|--------------------|----------------|------------|------------|
| 1x1         | 1      | 8                  | 8              | 2          | 16         |
| 3x3         | 2      | 1                  | 18             | 2          | 10         |
| 5x5         | 1      | 1                  | 8              | 1          | 8          |
| 7x7         | 2      | 1                  | 9              | 2          | 9          |
| 11x11       | 3      | 1                  | 10             | 3          | 10         |

Fig. 9 Overview of accelerator implementation.

in Table II. And Table III introduces detailed parameter for one PE array to cope with different filter sizes.

The accelerate solution is implemented as shown in Fig. 9. An AXI bus based on the chosen FPGA platform is employed for Data exchange among programming logic (PL) and processing system (PS). As the system operates, the address of configuration instruction stored in DDR is sent from PS to the accelerator, and follows with a start signal. Then the accelerator located in PL side reads configurations utilizing DMA engine, and the hardware modules are reconfigured according to configuration instructions. And the accelerator then gets into working status by following the instructions. After the acceleration is complete, PL side sends a finished signal to PS Side. Then the inference result is read from specific DDR address by the PS.

Performance of the proposed reconfigurable DNN accelerator is compared with several existing FPGA-based accelerate solutions [17, 18, 19, 20]. As this paper focus mainly on the design of reconfigurable architecture for DNN models with high flexibility, not just the optimization of specific DNN accelerator, performance density is adopted to acquire relatively fair comparison between different architectures based on different FPGA platforms. Here, throughput density (TD) works as performance density and is expressed as shown in Eq. (2)

\[
TD = \frac{\text{Throughput}}{\text{clock} \times \text{Multiplier Number}}
\]  

The multiplier number is commonly decided by the used DSP, while in some cases one DSP can be taken as two multipliers. And the throughput can be acquired by Eq. (3)

\[
\text{Throughput} = \frac{\text{Total operation number}}{\text{Run time}}
\]  

Compared to other state-of-the-art FPGA-based accelerators, performance of proposed architecture on VGG-16 and ResNet network is shown in Table IV and Table V. For ResNet, FC layers are not taken into consideration.

With the results on different network models, the proposed architecture exhibits high flexibility. Works of Qiu [17] and Venieris [20] shows limited flexibility as to deal with different DNN models, re-designing of the code needs to be done. Also, the proposed accelerator achieves great performance. For VGG-16 network, best performance density is achieved in FC layers for the utilization of sparseness prosperity; taking convolutional layers into consideration, the proposed work also acquires great performance and is only slightly lower than Qiu’s work which has low flexibility. For ResNet architecture, great performance is also achieved, especially when compared to Meloni’s [18] high flexibility accelerator that shares the same research focus point with this paper, the proposed work performs far better on ResNet-18.

4. Conclusion

This paper presents a novel method for DNN model acceleration with high flexibility. To deal with divers DNN models, all hardware modules of the proposed architecture can be reconfigured and forms hardware structure that mostly fits the demand. A novel reconfigurable data reuse strategy is introduced to lessen data access between on-chip buffer and off-chip memory. Moreover, based on this reconfigurable
reuse strategy, reconfigurable on-chip buffer is designed. Experimental results suggest such strategy and design help enhance performance of the accelerator. In addition, by utilizing sparsity of input feature map and weight proportion property of DNN, a comprehensive optimization technique is introduced. Finally, the proposed reconfigurable accelerator solution is implemented on FPGA platform. Through comparison with existing architectures based on FPGA, the proposed work shows great performance, and achieves high generality and low resource consumption in the meantime.

References

[1] X. Jiang, et al.: “Attention scaling for crowd counting,” CVPR (2020) 4705 (DOI: 10.1109/CVPR42600.2020.00476).
[2] X. Jiang, et al.: “Crowd counting and density estimation by Trellis encoder-decoder networks,” CVPR (2019) 6126 (DOI: 10.1109/ CVPR.2019.00629).
[3] K. He, et al.: “Mask R-CNN,” IEEE Trans. Pattern Anal. Mach. Intell. 42 (2020) 386 (DOI: 10.1109/TPAMI.2018.2844175).
[4] O. Russakovsky, et al.: “ImageNet large scale visual recognition challenge,” Int. J. Comput. Vis. 115 (2015) 211 (DOI: 10.1007/s11263- 015-0816-y).
[5] Y. Zhang, et al.: “Single-image crowd counting via multi-column convolutional neural network,” CVPR (2019) 589 (DOI: 10.1109/ CVPR.2019.00370).
[6] C.D. Gurkanayak and N. Arica: “A case study on transfer learning in convolutional neural networks,” 26th Signal Processing and Communications Applications Conference (SIU) (2018) 1 (DOI: 10.1109/ SIU.2018.8404642).
[7] T. Chen, et al.: “DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning,” ASPLOS (2014) 269 (DOI: 10.1145/2541940.2541967).
[8] Y.-H. Chen, et al.: “Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices,” IEEE J. Emerg. Sel. Topics Circuits Syst. 9 (2019) 292 (DOI: 10.1109/JETCAS.2019.2910232).
[9] N.P. Jouppi, et al.: “In-datacenter performance analysis of a tensor processing unit,” ISCA (2017) 1 (DOI: 10.1145/3079856.3080246).
[10] F. Tu, et al.: “Deep convolutional neural network architecture with reconfigurable computation patterns,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 25 (2017) 2220 (DOI: 10.1109/TVLSI.2017.2688340).
[11] L. Du, et al.: “A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things,” IEEE Trans. Circuits Syst. I, Reg. Papers 65 (2018) 198 (DOI: 10.1109/TCSI.2017.2735490).
[12] D. Shin, et al.: “DNPU: an energy-efficient deep-learning processor with heterogeneous multi-core architecture,” IEEE Micro 38 (2018) 85 (DOI: 10.1109/MM.2018.053631145).
[13] W. Choi, et al.: “Low cost convolutional neural network accelerator based on bi-directional filtering and bit-width reduction,” IEEE Access 6 (2018) 14734 (DOI: 10.1109/ACCESS.2018.2816019).
[14] J. Lee, et al.: “SRNPU: an energy-efficient CNN-based super-resolution processor with tile-based selective super-resolution in mobile devices,” IEEE J. Emerg. Sel. Topics Circuits Syst. 10 (2020) 320 (DOI: 10.1109/JETCAS.2020.3014454).
[15] B. Zhang, et al.: “A real-time and hardware-efficient processor for skeleton-based action recognition with lightweight convolutional neural network,” IEEE Trans. Circuits Syst. II, Exp. Briefs 66 (2019) 2052 (DOI: 10.1109/TCSII.2019.2899929).
[16] C. Zhang, et al.: “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” FPGA (2015) 161 (DOI: 10.1145/ 2684746.2689060).
[17] J. Qiu, et al.: “Going deeper with embedded FPGA platform for convolutional neural networks,” FPGA (2016) 26 (DOI: 10.1145/ 2847263.2847265).
[18] P. Meloni, et al.: “NEURA ghe: exploiting CPU-FPGA synergies for efficient and flexible CNN inference acceleration on Zynq SoCs,” ACM Trans. Reconfigurable Technol. Syst. II (2018) 24 (DOI: 10.1145/3284357).