FPGA-based accelerator for convolution operations

Yunfei Cao[1], Xin Wei[1], Tingting Qiao[1], He Chen[2*]

[1]Beijing Key Laboratory of Embedded Real-Time Information Processing Technology, Beijing, China
[2]Beijing Key Laboratory of Embedded Real-Time Information Processing Technology, Beijing Institute of Technology, 5 South Zhongguancun Street, Haidian District, Beijing, China
[*]Email address: chenhe@bit.edu.cn

***Abstract*—Convolutional neural networks have been widely used in many deep learning applications. Convolutional neural networks have a large number of convolution operations, which poses a huge challenge to real-time performance. In recent years, FPGA implementations of convolutional accelerators have received much attention due to their high performance and energy efficiency. In this paper, we implement an accelerator for convolution operations through the systolic array architecture on Xilinx ZedBoard device. The experimental results show that ours designed accelerators achieving performance density of up to 0.032 Gop/s/DSP.**

***Keywords*—CNN, FPGA, Accelerator, Systolic Array**

I. Introduction

Convolutional Neural Networks [1] (CNNs) are algorithms in deep learning applications that play an important role in image recognition, target detection, and natural language understanding. Since more than 90% of the operations is convolution, the speed of its operation dominates the real-time performance of CNN. The convolution operation is a multidimensional multiply-accumulate (MAC) operation. Taking the popular model AlexNet as an example, one forward propagation in which requires 1.36×10^{13} multiply-accumulate operations and 60 million weights [2]. In practical applications, CNN requires 10^9 to 10^{12} operations per second, which puts massive computational pressure on the processing platform [3]. Although the calculation pressure is large, it also has the potential of huge data reuse and parallel computing.

FPGA (Field-Programmable Gate Array) is an available accelerator platform that delivers outstanding performance when mapping CNN inference. It also has the advantages of customizability and low power consumption. Therefore, the research on the use of FPGA as an implementation platform [4]–[6] for convolution operation optimization has been increasing year by year.

The speed of convolution could be improved through parallel-computing with different dimension. Some existing convolution acceleration designs that use the parallelization of partial dimensions method achieve better computing performance than modern processors. However, these convolution acceleration schemes also exist disadvantages. The structure of different CNN models varies from each other. Some accelerators can only accelerate the fixed network structure, resulting in poor flexibility. The resources of FPGA devices are limited, and they cannot all implement parallel processing, especially for network models with a particularly deep network layer, such as ResNet [7]. In other words, recent convolutional neural network models cannot fully implement parallel computing, or even a single layer.

With the development of CNN's network scale, a suitable architecture is becoming more and more essential for FPGAs to achieve convolution operation acceleration. The systolic array architecture is a suitable structure, which is a two-dimensional grid structure with a number of processing elements (PE) establishing a two-dimensional pipeline. The layout of the systolic array is regular and provides remarkable flexibility while guaranteeing parallelism. Therefore, the researchers attempt to map CNN using a systolic array architecture in recent years [8]–[10].

II. Background

A. Operations in Convolutional Layers of CNNs

The typical CNN structure consists of a series of layers, the most significant of which is the convolution layer in the CNN structure. Convolution, which is used to extract local features of the input data, is the major operation in the convolutional layer. Fig. 1 and Fig. 2 illustrate the process of convolution operation and the pseudo-code of convolution operations, respectively. For simplicity, we use Nif and Nof to indicate the number of input and output channels, and Nix and Niy are the height and width of each input feature map. We use Nkx and Nky to indicate the height and width of each kernel window. Nox and Noy are height and width of each output feature map.

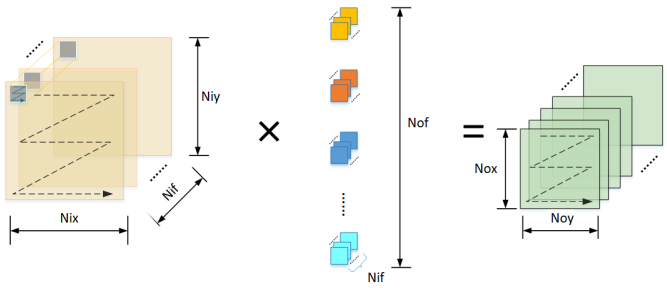

Fig. 1. The process of convolution operation.

978-1-7281-2345-5/19/$31.00 ©2019 IEEE

```
for(no=0;no<Nof;no++)
        for (y=0;y<Noy;y++)
        for (x=0;x<Nox;X++)
                for(ni=0;ni<Nif;ni++)
                        for(ky=0;ky<Nky;ky++)
                        for(kx=0;kx<Nkx;kx++)
                oFM(x,y;no)+=iFM(x+S×kx,y+S×ky;ni)×W(Kx,ky,ni;no);
oFM(x,y,no)+=bias(no);
```

Fig. 2. The pseudo code of convolution operation.

B. Convolution mapping to matrix multiplication

The convolution operation in the CNN can be converted to matrix multiplication by reordering the data.

As shown in Fig 3, the input feature maps with a size of $32 \times 32 \times 3$ are converted to a input feature map matrix with dimensions of $(28 \times 28) \times (5 \times 5 \times 3)$. The five group of kernel weights with a size of $5 \times 5 \times 3$ are transformed to weight matrix with dimensions of $(5 \times 5 \times 3) \times 5$. After rearrangement, the convolutions are mapped to a matrix multiplication.The result of the multiplication of the two matrices, the input feature matrix and the weight matrix, corresponds to the result of the convolution operation. The size of the output feature map matrix is $(28 \times 28) \times 5$. The result of the convolution operation is an output feature map with dimensions of $(28 \times 28) \times 5$. In this paper, we use this strategy to map a convolution operation into a matrix multiplication, which has two advantages: flexibility and continuity.

- Flexibility : The structure of different CNN model may exist large disparity. Some existing designed only for fixed CNN structures are inflexible. A generic design that process vary network structures is needed. Mapping convolution operation to matrix multiplication is a fine strategy to handle a wider range of CNN structures. We only need to consider how to speed up matrix multiplication instead of considering the structure of CNN.
- Continuity : In the Fig. 1, the convolution kernel slides on the input feature map, and each time the data is read, there is a discontinuous address. When the acceleration platform repeatedly reads discontinuous address data, there is a problem of inefficiency. Converting the convolution kernel to a matrix allows the address of each read data to be continuous, which also facilitates subsequent parallelization calculations.

III. ACCELERATOR DESIGN

A. The systolic array architecture for operational array

In this paper, we show a novel systolic array architecture for CNN on FPGA in Fig. 4. As shown in this figure, a number of PE that build a two-dimensional grid structure.

Fig. 5 presents the internal structure of the systolic array structure of PE. As Fig. 5 shown, the processing elements consist of an adder, a multiplier, and a series of registers that act as pipelines or stored data. Each PE has two functions, including shifting data and calculating. At each cycle, the PE can carry the feature map, weight and partial sum to adjacent

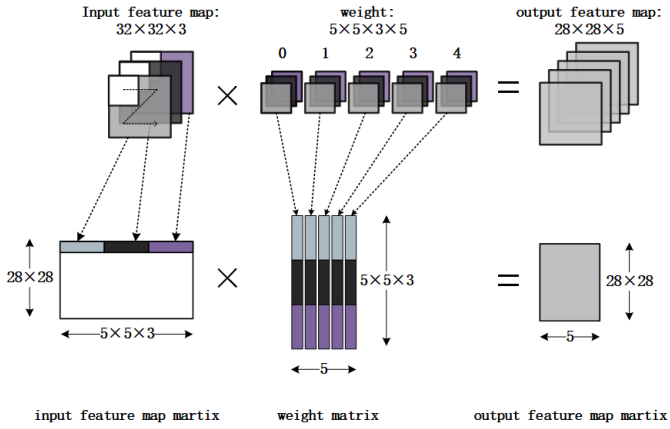

Fig. 3. The process of convolution operation.

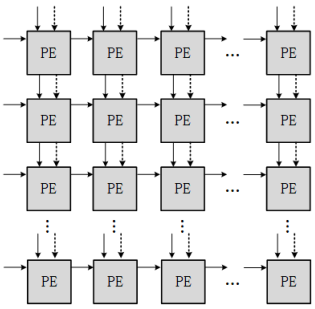

Fig. 4. The overview of the systolic array architecture.

PE. At the same time, the PE will acquire the data passed by the neighbor PE to perform the multiply-add operation.

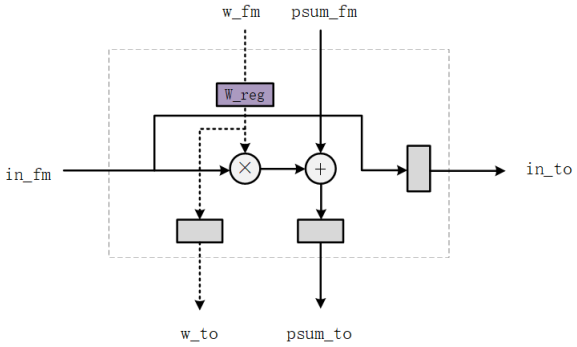

Fig. 5. Structure of processing element.

B. Working mode of PE

In this paper, PE has four working modes: initial mode, calculation mode, weight transfer mode and waiting mode.

- initial mode: When PE is in this mode, the internal registers of PE are cleared as the start of the first computation task.
- weight transfer mode: Weight is transmitted vertically to adjacent PE when PE is in the weight transfer state.

- calculation mode: When PE is in this mode, the feature maps and the partial sum of the previous PE are acquired, and the internal fixed weights of PE are multiplied and added. After PE calculation is completed, partial sum are transmitted vertically, and the feature map data is horizontally transmitted, thereby forming a pipeline structure.
- waiting mode: For the purpose of data output, it is necessary to insert a wait mode in certain clock cycles. When PE is in waiting mode, the data in the registers in the processing unit remains fixed until other modes are activated.

C. Dataflow composition and Matrix mapping

Fig .6 shows the weight data flow on the systolic array during the weight transfer mode.

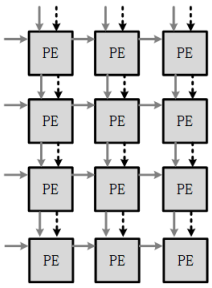

Fig. 6. Weight dataflow.

When the weight data flows along the vertical direction of the systolic array, the data of partial sum and feature map will not enter the systolic array. The weight data is stored in the weight register of the PE after each PE. The columns are independent of each other as the data streams. Fig .7 represents the data stream of input feature map and partial sum. The systolic array can be operated or flowed until the weight data is fixed in the weight register of each PE. The input feature map and partial sum could flow simultaneously, the flow direction of them is perpendicular to each other, the dataflow of partial sum is vertical, and input feature map flows horizontally in the Fig .7.

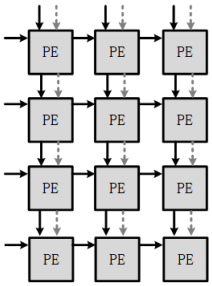

Fig. 7. Dataflow of input feature maps and partial sum.

After the convolution operation is converted to matrix multiplication, the input data needs to meet certain time constraints to ensure the correct output. Combining the dataflow pattern of the systolic array and the operation definition of matrix multiplication in this paper, we can get the mapping method of matrix multiplication and the systolic array. Fig .8 presents an example of matrix multiplication.

Then, we will discuss how matrix multiplication is mapped to the systolic array based on this example.

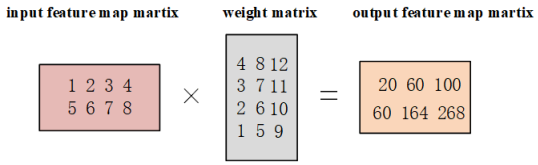

Fig. 8. Example of matrix multiplication.

Fig .9 shows how matrix multiplication maps to a systolic array. Before beginning the operation, the column data of the convolution kernel matrix needs to be stored in the weight register of each PE in the systolic array. During the arithmetic processing, each column of the input feature map sequentially enters each row of the systolic array, and the data input between the rows is sequentially delayed by one cycle. Since the input feature map data is horizontally propagated in the systolic array, the output of each column is delayed by one cycle in turn. In Fig .9, the result of the processing does not begin to output until the sixth clock cycle. This is because the systolic array is a two-dimensional pipeline architecture, so when the processing units of the columns of the systolic array are all operating, the results of the columns start to be generated.

Fig .10 presents the process of how output feature maps generate. Each PE passes the result of the calculation to the PE directly below them. In the pipeline, every partial sum will be accumulated multiple times by PE until the partial sum is converted to output feature maps.

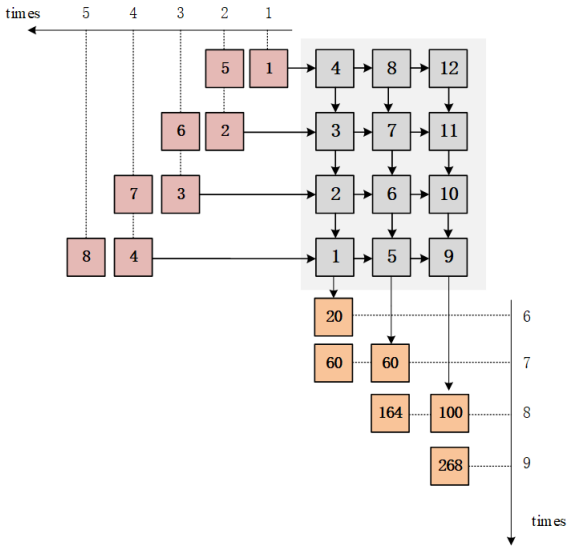

Fig. 9. An example of matrix multiplication mapping to systolic array.

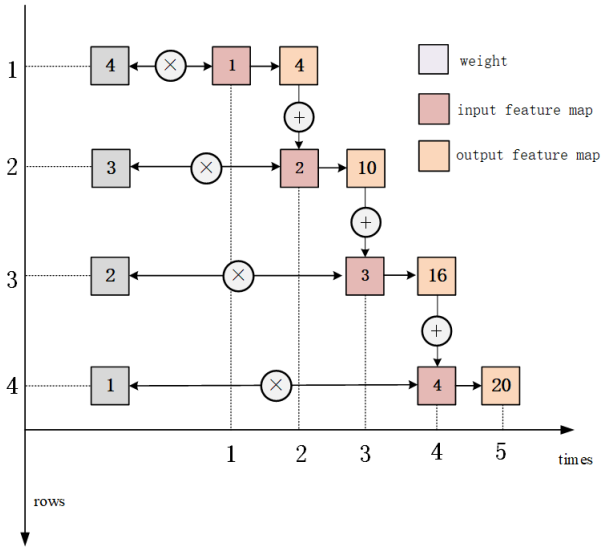

Fig. 10. An example of outputting a feature map data generation process diagram.

D. Buffer module

The systolic array has strict timing constraints on the input dataflow when performing parallelized two-dimensional pipeline calculations, leading to greater bandwidth requirements. In order to achieve the match of the memory bandwidth and the required bandwidth of the systolic array, it is necessary to design a buffer module to reduce the number of repeated readings of data and reduce the bandwidth requirement.

In this paper, the local cache design scheme adopts dual FIFO storage structure, which is connected to both ends of the systolic array and can be used for data reuse. As shown in Fig .11, FIFO A has the ability of data cache. These data are originated from Off-chip memory or FIFO B. PEs could be offered input data by FIFO A. FIFO B is able to collect data generated by the PE, leading to data circulating between the FIFO A, FIFO B, and PEs resulting in the dual FIFO storage structure could exploit the potential of data reuse.

The number of the dual FIFO storage structure is numerous, almost three times the number of rows of PEs $\times$ the number of columns of PEs since feature maps, weights, and partial sum need to be recycled in our design accelerator for convolution operations.

Due to the existence of dual FIFO storage structure, data reuse is thorough resulting in the number of off-chip memory accesses could be reduced obviously.

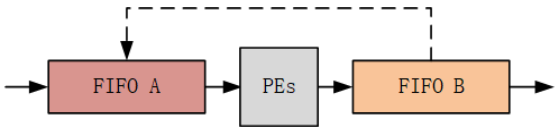

Fig. 11. Connection pattern between systolic array and dual FIFO storage structure.

IV. Implement and Experiment

A. Evaluation setup

We evaluate our design by implementing an accelerator on a Xilinx Zedboard. We select a typical CNN model for the test: LeNet 5. The channel numbers in convolutional layer include 1, 6, and 16. The feature maps sizes include 32×32, 28×28, and 5×5. We assign 8 rows and 4 columns for the systolic array considering the total number of available DSP slices. In the buffer module, the length of each dual FIFO storage structure for partial sum, weight, and input feature map are selected as 2048, 1024 and 1024, respectively. Accelerator design is implemented using VHDL language programming and synthesized with Vivado Design Suite 2017.2.

B. Evaluation result

Table I presents the hardware resource utilization of our accelerator. The systolic array consumes 64 DSP slices for convolution operations. In addition, the accelerator uses 69 BRAM slices to establish a series of dual FIFO storage structure. The utilization results show that our accelerator can consume very few on-chip resources. Table II lists a series of evaluation result, including latency, number of operations, throughput and number of used DSP. Throughput, measuring performance of the processor, is equal to the number of operations divided by the latency in this paper. Performance density is defined as Throughput that one DSP slice executes.

TABLE I
FPGA Resource Utilization

Resource	DSP	BRAM	LUT	FF
Available	220	140	53200	106400
Used	64	69	28861	41828
Utilization (%)	29.09	49.29	54.250	39.31

TABLE II
Summary of Experimental Results

Layer	C1	C3	C5
Latency(us)	191.32	231.24	182.71
Number of operations(op)	401408	486400	96000
Throughput (Gop/s)	2.098	2.086	0.525
DSPs	64	64	16
Performance density (Gop/s/DSPs)	0.0328		

Table III lists the comparisons between CPU and our accelerator. Compared to the CPU, our accelerator achieves a 1.569-fold improvement on convolution layer of LeNet 5 in terms of throughput.

TABLE III
Evaluation Results on the Central Processing Unit(CPU) and Our Accelerators.

Platform	CPU	FPGA
Vendor	Intel i7 8750H	ZedBoard
Throughput(GOP/s)	1.004	1.569

We select several convolution operation accelerator implementations on FPGAs for comparison. As different FPGA platforms are used, we list the performance density in Table IV for fair comparisons. As shown in Table IV, our accelerator is better than [5], [11], [12]. and increased by at least 19.3% in terms of performance density.

TABLE IV
COMPARISONS WITH PREVIOUS ACCELERATOR IMPLEMENTATIONS.

	[5]	**[11]**	**[12]**	ours
Vendor	VX485T	ZU19EG	VX690T	ZedBoard
DSPs	2240	1500	2401	64
Frequency(MHz)	100	200	100	100
Throughput(Gop/s)	61.62	17.96	61.20	2.09
Performance density (Gop/s/DSPs)	0.0275	0.0119	0.0256	0.0328

V. CONCLUSION

In this paper, we have proposed a design of accelerator for convolution operation architecture that can handle convolution operation efficiently. We focus on accelerating the convolutional layers that constitute most of the CNN's computational workload. We adopt systolic array based accelerator architecture. A performance speed-up of 1.569 $\times$ is achieved compared with an Intel i7-8750H CPU, and the performance density is 1.19 $\times$ better than others design.

ACKNOWLEDGMENT

This work was supported by the Chang Jiang Scholars Program under Grant T2012122 and the Hundred Leading Talent Project of Beijing Science and Technology under Grant Z141101001514005.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] Y. Qiao, J. Shen, T. Xiao, Q. Yang, M. Wen, and C. Zhang, "Fpga-accelerated deep convolutional neural networks for high throughput and energy efficiency," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 20, p. e3850, 2017.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[4] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 247–257, 2010.

[5] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, ACM, 2015.

[6] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 40–47, IEEE, 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[8] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 53–60, IEEE, 2009.

[9] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 682–687, 2014.

[10] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision.," in *CVPR Workshops*, pp. 109–116, 2011.

[11] Z. Liu, Y. Dou, J. Jiang, Q. Wang, and P. Chow, "An fpga-based processor for training convolutional neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*, pp. 207–210, IEEE, 2017.

[12] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 22, IEEE Press, 2016.