A High Performance Framework for Large-scale 2D Convolution Operation on FPGA

Dawang Zhang
School of Information and Communication Engineering
Beijing University of Posts and Telecommunications
Beijing, China
Email: zdawang@bupt.edu.cn

Zhisong Bie
School of Information and Communication Engineering
Beijing University of Posts and Telecommunications
Beijing, China
Email: zhisongbie@bupt.edu.cn

***Abstract*—Convolutional neural network (CNN), as the focus in the artificial intelligence field, has attracted more and more attention resent years. Various accelerators based on FPGA platform have been proposed because of the high performance and high energy efficiency. However, limited to the on-chip memory resources, only small networks can be accelerated by these accelerators. Due to the fact that 2D convolution operation is the most computing-intensive part, accelerating large-scale 2D convolution operation has become the key to accelerating large-scale CNNs on FPGA platforms. This paper presents a method of rotary data-storage and a design of the new PE (processing unit). Compared to the traditional Z-type PE, it performs non-redundant calculations when the stride of convolution operation is greater than 1, which significantly reduces the calculating time. At the same time, in order to reduce the external memory bandwidth with limited on-chip resources, two optimization architectures and a block-calculation method have been proposed, which can reduce the usage of on-chip memory resources. As a case study, we select some convolution layers of ResNet-34 and compare it with a previous accelerator. Under the condition of same consumption of resources, the result shows that the proposed framework needs only 29.9% time and 51% external memory bandwidth of the previous accelerator.**

Keywords—convolution; acceleration; memory bandwidth; FPGA

I. Introduction

Convolutional Neural Network(CNN) is a well-known deep learning architecture [1] which extends from artificial neural network. Because of the features of weight sharing and local connectivity, the complexity of the model is greatly reduced. This advantage performs better when the input is a multi-dimensional image, being directly used as the input of the network, is capable of avoiding the complicated feature extraction and data reconstruction process in the traditional algorithm. In recent years, CNN has developed rapidly and has been widely used in computer vision [2] and natural language processing [3].

Since FPGA has a large amount of DSP resources and can work in a pipelined style, it becomes a candidate platform for building high-performance deep learning systems [4, 5, 6, 7, 8, 9, 10]. The most computing-intensive part of CNN is 2D convolution operation. Therefore, designing PE architecture for 2D convolution operation to parallel computations is a key to realize the acceleration [11, 12, 13]. The widely used Z-type PE architecture on the FPGA platform were proposed in [11]. Although it simplifies the control of data stream, when the stride of 2D convolution operation is greater than 1, the utilization rate of computing resources drops significantly.

Towards the storage of intermediate results and weights, there are mainly three ways. Firstly, for the small networks, we can store all the intermediate results and weights on chip to reduce the external memory bandwidth [5, 10]. Owing to the limitation of on-chip memory size, we cannot accelerate deep networks by this way. Secondly, all the intermediate results and parameters are stored in the external memory [7]. Deep networks can be accelerated in this way, but memory bandwidth become the bottleneck. Thirdly, we can store the intermediate results on chip and weights on external memory [4, 6, 8, 9]. Normally, we can set input buffer and output buffer, loading data before computation and storing data after computation. It is a trade between resources and bandwidth. But some networks are too big to store the intermediate results on chip, optimizations are needed to reduce the utilization of on-chip memory.

The rest of this paper is organized as follows: In Section 2, the related work and motivation are discussed. Section 3 provides an introduction to 2D convolution operation. In section 4, Z-type PE, rotary data-storage and the new PE design are presented. Section 5 describes the optimizations for the utilization of on-chip memory. A comparison between our work and existing work is made in Section 6. We finally conclude this paper in Section 7.

II. Related Work And Motivation

A. *Related work*

To accelerate 2D convolution operation, majorly two aspects are focused on. One is to design proper parallel structure to increase the throughput. The other is to reduce the external memory bandwidth.

In order to design proper parallel structure, a lot of works have been done such as [4, 6, 11, 12, 13]. There are four types of parallelism adopted: operator-level parallelism (inside the PE), inter-input parallelism (multiple input features are combined to create a single output), intra-output parallelism (multiple independent features are computed simultaneously) and batch parallelism (parallel between different input pictures). All of [4, 6, 11, 12, 13] have used Z-type PE for parallelism within a convolution operation, copying it to increase intra-output parallelism and inter-output parallelism. For further reducing the memory bandwidth, [6] also adopts batch parallelism.

However, [9] proposes that an implementation can be either computation-bounded or memory-bounded, the parallelism cannot be increased unlimited due to the limited memory bandwidth. Memory bandwidth reduction becomes necessary. Reference [5] and [10] store all the intermediate results and weights on chip without external memory. However, with the CNN models become larger and larger, this design become impossible to be carried out. Reference [4, 6, 9] store the intermediate results on chip while the weights on external memory. By fully use data-reuse, the external memory bandwidth can be decreased significantly.

B. *Motivation*

With the rapid development of artificial intelligence, today's CNN models is much larger than early small CNN models. The proposed design to reduce memory bandwidth that store all the intermediate results on chip while the weights on external memory maybe not executable. For example, toward the state-of-art CNN model VGGNet16, the size of the first convolution layer input is $224\times224\times3$ while the output is $224\times224\times64$, quantized with 16 bit, 55.4Mbyte space is needed. However, only very advanced FPGA can meet this requirement.

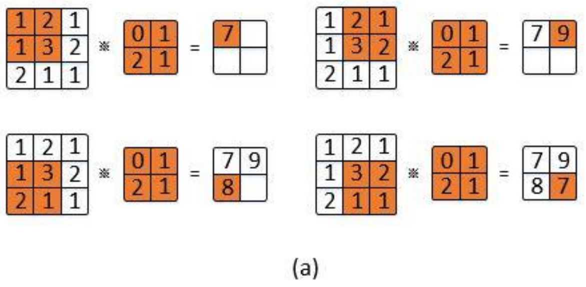
Toward the parallelism, the operator-level parallelism needs to be redesigned. The traditional Z-type PE costs the same time regardless of the value of stride. In case the stride is greater than 1, the Z-type PE needs to wait a few clock before valid computation, which led to longer computing time and waste of computing resources.

Based on the above analysis, our work focus on the optimizations of on-chip memory and new PE design. It should be reached that convolution operation of any size can be accelerated in low-level FPGAs while high memory bandwidth is not needed. For the new PE design, non-redundant calculations should be guaranteed.

III. 2D CONVONLUTION OPERATION

The 2D convolution operation is to convolve the feature maps of the previous layer with a plurality of learnable kernels to form the feature maps of the next layer. It can be divided into three steps. First, operation between one feature map and one channel weight numbers. Second, operation between all feature maps and all channels of a kernel. Third, operation between all feature maps and all kernels.

The first step is shown in Figure 1 (a), it is an operation between a feature map of $N\times N$ and a sliding window of $K\times K$. With the sliding window goes, one intermediate output feature map is generated. The second step is to sum all the intermediate output feature maps resulted by step one to generate one final output feature map. The third step is to repeat step one and step two to generate all final output feature maps, which is shown in Figure1 (b).



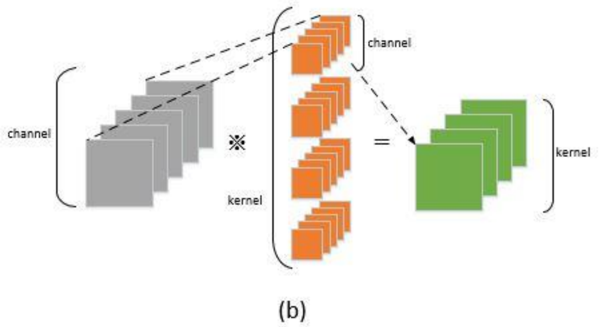


Figure 1: Detail of 2D convolution operation: (a) the operation between a feature map and shift window. (b) the operation between all feature maps and all kernels.

IV. DESIGN OF PE

A. *Z-type PE*

The size of convolutional kernel usually has only several options, such as 3×3, 5×5, and 7×7. Owing to the reduction of calculation and parameter, 3×3 kernel has been widely used, besides, the 3×3 kernel can be used to form kernels of any size. Therefore, the PE is generally designed based on 3×3 kernel.

For a 3×3 convolution window, 9 input data are required for each calculation, which will complex the control system. In fact, because data reuse exists in adjacent convolution windows, it is not necessary to load 9 number every time. If the stride of the 2D convolution operation is 1, for a convolution window, it has 6 same number with the left convolution window, also the upper. Regardless of 4 same number between the left window and the upper window, we need only load 1 number to get all data in current window. Some scholars proposed a PE design based on this idea. Because the direction of the data stream

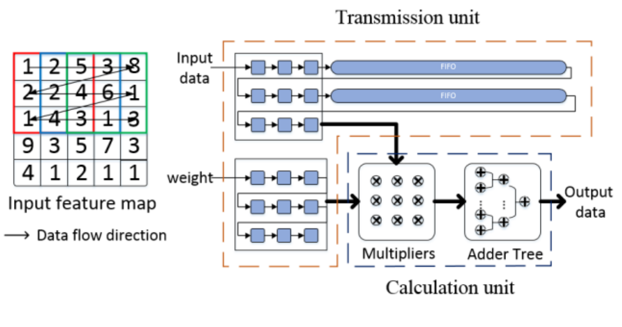


Figure 2: The design of Z-type PE

performs as Z, it is called Z-type PE. Figure 2 shows how Z-type PE works.

There are two parts in Z-type PE: transmission unit and calculation unit.

Transmission unit: this unit is designed to transfer data. It includes 9 weight registers, 9 data registers and 2 FIFOs (First Input First Output). Weight registers store the weight data to be computed. The depth of FIFOs is set to $row_{in}-3$, while row_{in} is the number of pixels in a row of the input feature map. The purpose of this design is to make sure that the numbers in data registers exist in one convolution window. When a new number comes, the data in the data registers and FIFOs move one step, which equals that the convolution window moves right.

Calculation unit: this unit is designed to calculate. It includes 9 multipliers and 8 adders. Firstly, the multiplication operations are done in a cycle, and the addition operations will be done in the next 4 cycles by the adder tree. Because all the circuits can be designed in pipelined style, we can get the result in a cycle.

However, there are two flaws of Z-type PE. First, each Z-type PE costs two BRAMs (Block Random Access Memory, the basic storage unit of FPGA) to implement the FIFOs. When the quantity of Z-type PE increases, the usage of BRAMs for the Z-type PE cannot be ignored. Second, for the 2D convolution operation whose stride is not 1, the amount of overlapping numbers between adjacent convolution windows become less. We need to load more data to perform one calculation, in other word, a few cycles will be wasted for wait before calculation, which lead to longer computing time and waste of computing resources.

Supposed that the size of input feature map is $row_{in}\times row_{in}$, the size of output feature map is $row_{out}\times row_{out}$, the stride is $stride$. The total time (including wait time and computing time) and computing time are:

$$t_{total}=t_{Z-load}=row_{in}^{2} \tag{1}$$

The computing time are:

$$t_{compute}=row_{out}^{2}=\left(\frac{row_{in}}{stride}\right)^{2} \tag{2}$$

So, the utilization rate of computing resources is:

$$\eta=\frac{t_{compute}}{t_{total}}=\frac{1}{stride^{2}} \tag{3}$$

B. Rotary data-storage and new PE design

In order to promote the utilization rate of computing resources, one way is to load 9 numbers each cycle for calculation. A simple way to achieve this is to set 9 input buffers to store the same data, but it costs 9 times storage area. Rotary data-storage method is designed to achieve the goal but without extra storage area. This method uses 9 RAMs, while each RAM stores one number on the 3×3 convolution window. Because we can only read a number from a RAM each cycle, how to make sure the 9 numbers in a convolution window in different RAMs become the difficulty.

The data storage process: the input feature map is divided into three part. The first part is row 0, 3, 6…3n, the second part is row 1, 4, 7…3n+1, the third part is row 2, 5, 8…3n+2. Add 0 to the last of each row to make sure the number of pixels in a row can be divided by 3 with no remainder. This is for further convenience. The first part is stored in 0, 1, 2 RAMs in an alternately manner. The second part is stored in 3, 4, 5 RAMs and the third part is stored in 6, 7, 8 RAMs in the same manner. We call every three RAMs a RAM heap. The storage process is shown in Figure 3.

The reason why this can make sure that the 9 numbers in a convolution window stored in different RAMs is that, the distance between two numbers in the same RAM will be equal or greater than 3 vertically and horizontally. Which means that, in the area of 5x5 centered on number x, there is no other number stored in the same RAM. So, any 3x3 convolution window including number x will not include other number stored in the same RAM.

The data load process: this process is to read 9 numbers based on the coordinate <x, y> of the left-upper number on the convolution window. First, we need to calculate the RAM and the address of the left-upper number, then, we map the location of the left-upper number to get the location of the remain 8 numbers. The way we calculate the RAM of the left-upper number is that, we can calculate the RAM heap according to x, and the RAM shift in the RAM heap according to y. The way we calculate the address of the left-upper number is that, we calculate the address of the first number in the same row according to x, and the address shift according to y. The formulas are shown in Equation 4 and Equation 5.

$$ram=(x\%3)\times3+(y\%3) \tag{4}$$

$$addr=(x\div3)\times3+(y\div3) \tag{5}$$

In FPGA, division and remainder operation are hard to implement. Considering that x and y is normally less than 512 and the divisor is a fixed number 3, we can use the method of LUT (look up table) to replace the operation. The circuit is shown in Figure 4. The idea of LUT is to trade space for time.

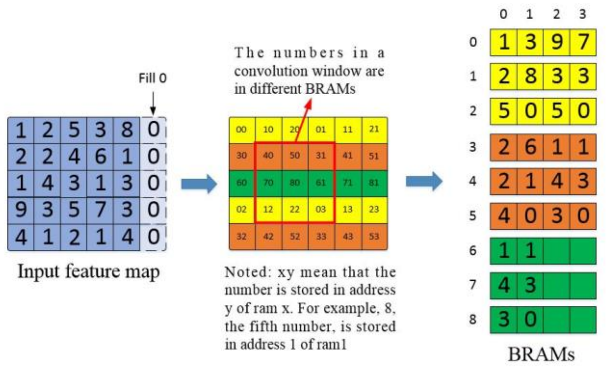

Fill 0
Input feature map
The numbers in a convolution window are in different BRAMs
00 10 20 01 11 21
30 40 50 31 41 51
60 70 80 61 71 81
02 12 22 03 13 23
32 42 52 33 43 53
Noted: xy mean that the number is stored in address y of ram x. For example, 8, the fifth number, is stored in address 1 of ram1
BRAMs
0 1 2 3

Figure 3: The data storage process of rotary data-storage method

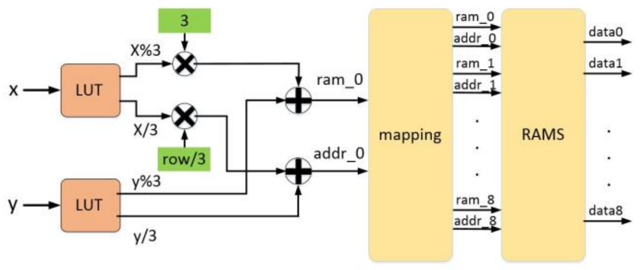


Figure 4: Scheme of the data load process

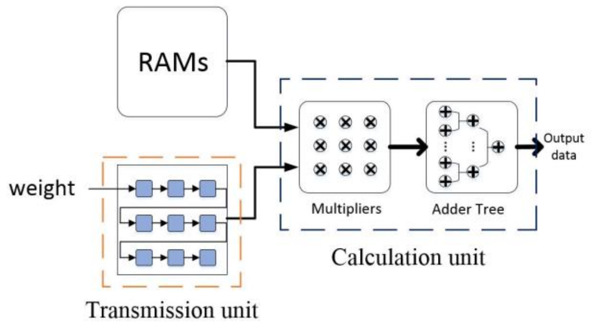


Figure 5: Architecture of the new PE design

We treat the input as the address of a table, the corresponding value become the output. We set the depth of LUT to 512.

Because we already load 9 numbers from input buffer each cycle, 9 weight registers are enough for transmission unit. The new PE design is shown in Figure 5.

V. ON-CHIP MEMORY OPTIMIZATION

It is a common idea that using on-chip memory to reduce external memory access. Normally an input buffer and an output buffer will be created. The input feature maps will be loaded to the input buffer and the immediately results will be stored in the output buffer. After the calculation, the results will be loaded to external memory.

There is a problem that some FPGAs may not have enough on-chip memory resources to store all the data. It becomes necessary to optimize the usage of on-chip memory.

A. *Structures for optimization and parallelism*

There are two structures for optimizing the usage of on-chip memory without increasing external memory bandwidth. They are input-optimization structure and output-optimization structure.

Input-optimization structure, which optimizes the input buffer. One input feature map will be loaded to the input buffer each phase, after calculation, the next input feature map will be loaded. So, only one input feature map will be stored in the input buffer.

Output-optimization structure, which optimizes the output buffer. One output feature map will be calculated each phase, after the calculation, the result will be stored to external memory. So, only one output feature map will be stored in the output buffer. The two structures are shown in Figure 6.

Supposed that the n_{in} is the number of input feature maps, n_{out} is the number of output feature maps. The storage area of input-optimization structure is:

$$storage_{input-optim} = n_{out} \times row_{out}^{2} + row_{in}^{2} \qquad (6)$$

while the storage area of output-optimization structure is:

$$storage_{output-optim} = n_{in} \times row_{in}^{2} + row_{out}^{2} \qquad (7)$$

Because the row_{in} is bigger than row_{out} normally, therefore, $storage_{output-optim}$ is normally greater than $storage_{input-optim}$. Input-optimization structure is adopted in this paper.

there are mainly three types of parallelism which are widely used: operator-level parallelism, inter-input parallelism and intra-output parallelism. In our implementation, all the three types of parallelism are considered. For input-optimization structure, the intra-output parallelism will not cost extra storage area while the inter-input parallelism will increase the size of input buffer. Supposed that the inter-input parallelism is $para_{in}$, the intra-output parallelism is $para_{out}$, the storage area becomes:

$$storage_{input-optim} = n_{out} \times row_{out}^{2} + para_{in} \times row_{in}^{2} \quad (8)$$

Under above parallelism design, the following process is executed for calculation:

1) Load $para_{in}$ input feature maps from the external memory to input buffer, storing in the way of rotary data-storage.

2) Load corresponding parameters of $para_{out}$ kernels from external memory to PEs.

3) Perform calculations, get the intermediate result of $para_{out}$ output feature maps.

4) Repeat 2, 3 until the intermediate result of all output feature maps are get.

5) Repeat 2, 3, 4 until the finally result of all output feature maps are get.

6) Store the output feature maps to external memory.

B. *Block-Calculation*

For large-scale 2D convolution operation, although optimizations have been done for on-chip memory sources. Some FPGAs still don't have enough area to store the intermediate results, block-calculation become necessary. The idea is to split the input feature maps of $row_{in} \times row_{in}$ into some sub-input feature maps of $T_{in} \times T_{in}$ .2D convolution operation is performed on one sub-input feature map each phase, and the sub-output feature maps will be stored to external memory. After all the sub-input feature maps are done, the output feature maps is obtained. The overview of block-calculation method is shown in Figure 7.

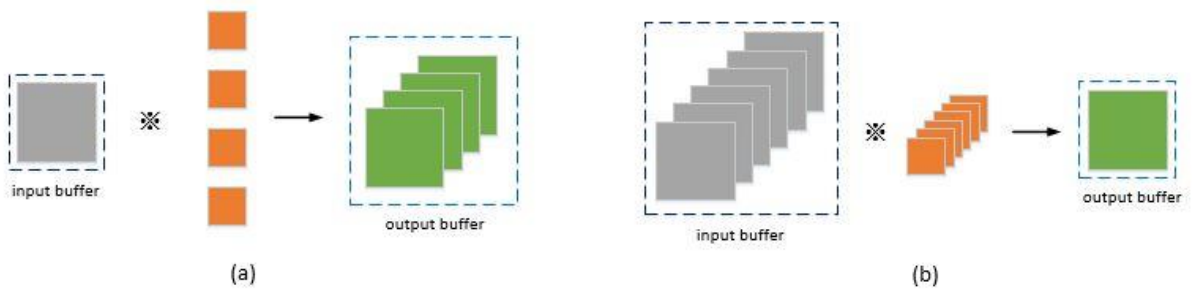


Figure 6: Structures for optimizing the usage of on-chip memory: (a) input-optimization structure. (b) output-optimization structure.

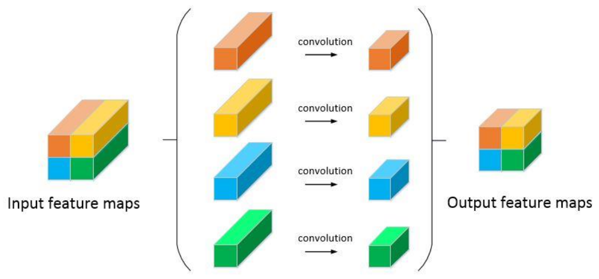


Figure 7: Overview of the block-calculation method

This method allows us to divided a large-scale 2D convolution operation into some small-scale 2D convolution operations. For the output-optimization structure, according to the capacity of output buffer, the size of sub-output feature maps can be calculated, then the division plan is determined.

For input-optimization structure, supposed that the capacity of output buffer is $capacity_{out}$, the maximum size of sub-output feature map is:

$$T_{out} = ceil(\sqrt{capacity_{out} / n_{out}}) \tag{9}$$

The size of sub-input feature map can be calculated by:

$$\begin{aligned} T_{in} &= T_{out} \times stride \\ &= ceil(\sqrt{capacity_{out} / n_{out}}) \times stride \end{aligned} \tag{10}$$

The frequency of external memory access will be calculated following. The number of phase is:

$$N_{phase} = \left(floor(size_{in} / T_{in}) \right)^2 \tag{11}$$

The frequency of loading weight, loading data, storing data are:

$$\begin{aligned} Num_{load-weight} &= N_{phase} \times num_{weight} \\ &= \left(floor(size_{in} / T_{in}) \right)^2 \times num_{weight} \end{aligned} \tag{12}$$

$$Num_{load-data} = N_{phase} \times T_{in}{}^2 \times n_{in} = size_{in}{}^2 \times n_{in} \tag{13}$$

$$Num_{store-data} = N_{phase} \times T_{out}{}^2 \times n_{out} = size_{out}{}^2 \times n_{out} \tag{14}$$

So, for the block-calculation method, the input feature map needs to be loaded once, the weight needs to be loaded N_{phase} times, the output feature map needs to be stored once. Because the number of weights is much smaller than the data, the external memory bandwidth will not increase a lot.

VI. PERFORMANCE ANALYSIS

As mentioned in Section 1, works have been done to develop new solutions for accelerating or optimizing the computation of 2D convolution operation. A comparison is made between our work and reference[4]. In [4], output-optimization structure is adopted, and for further reduction of usage of on-chip memory, input feature map are stored in external memory rather than on-chip memory. Also, Z-type PE is adopted.

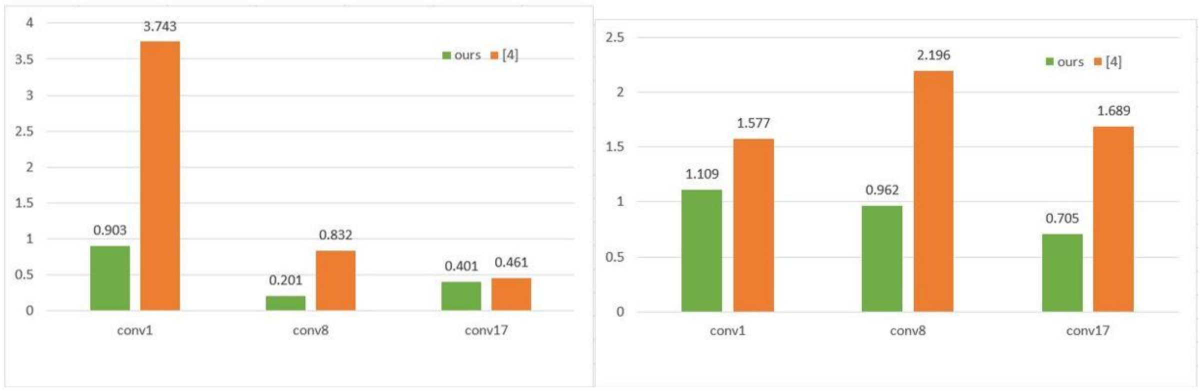
ResNet (Deep Residual Network) plays an import role in the development of deep learning network. We accelerate the first, eighth, and seventeenth convolution layers of ResNet34, to compare the usage of computing time and the external memory access of two solutions. The parameters of the convolution layers are shown in Table 1.

TABLE I. PARAMETERS OF CONVOLUTION LAYERS

	conv1	**conv8**	**conv17**
input size	224×224×3	56×56×64	14×14×256
kernel size	7×7	3×3	3×3
stride	2	2	1
padding	3	1	1
output size	112×112×64	28×28×128	14×14×256

TABLE II. USAGE OF BRAMS

	input buffer	**LUT**	**weight buffer**	**PE-FIFO**	**output buffer**	**total**
[4]	16	0	1	64	16	97
ours	36	2	1	0	64	103



(a) computing time for different layers(MClk) (b) external memory access for different layers(MOP)

Figure 8: Comparison between our work and [4]: (a) the computing time for different convolution layers. (b) the frequency of external memory access for different convolution layers.

We will compare them under the condition of roughly the same resources, including computing resources and on-chip memory resources. Towards the computing resources, the inter-input parallelism degree is set to 2, the intra-output parallelism degree is set to 16, which means that 288 multiplication operations can be done in one cycle. For the on-chip memory resources, we set the output buffer of ours to 64 BRAMs, the detail usage of on-chip memory of two solutions are shown in Table 2.

The costing time of two solutions and the external memory access of two solutions are shown in Figure 8 (a) and (b).

In summary, our work significantly outperforms the previous work. Compared with previous work, only 29.9% time and 51% memory access are needed. It should be noted that, the time reduction is the effect of new PE design and the memory access reduction is the effect of fully usage of on-chip memory.

VII. Conclusion

This paper has focused on two major problems of accelerating 2D convolution operation. First, the waste of computing resources exists in traditional Z-type PE when the stride is greater than 1. Second, the on-chip memory resources maybe not enough to store the intermediate results. For the first problem, we put forward the rotary data-storage method and a new PE design. On-chip memory bandwidth is fully used that nine data are read in single cycle for computing. On the same time, the new PE design decreases the usage of on-chip memory. For the second problem, two structures and a block-calculation method are proposed. The structures cut back about 50% on-chip memory resources. The block-calculation method allows us to divides large-scale convolution operation to small operations according to the resources. The division scheme can be decided base on the on-chip memory resources.

References

[1] Bouvrie J. Notes on Convolutional Neural Networks[J]. Neural Nets, 2006.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in NIPS, 2012, pp. 1097–1105.

[3] Kim Y. Convolutional Neural Networks for Sentence Classification[J]. Eprint Arxiv, 2014.

[4] Qiu J, Wang J, Yao S, et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network[C]// Acm/sigda International Symposium on Field-Programmable Gate Arrays. ACM, 2016:26-35.

[5] Chen Y, Sun N, Temam O, et al. DaDianNao: A Machine-Learning Supercomputer[C]// Ieee/acm International Symposium on Microarchitecture. IEEE, 2015:609-622.

[6] Li H, Fan X, Jiao L, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks[C]// International Conference on Field Programmable Logic and Applications. IEEE, 2016:1-9.

[7] Zhao W, Fu H, Luk W, et al. F-CNN: An FPGA-based framework for training Convolutional Neural Networks[C]// IEEE, International Conference on Application-Specific Systems, Architectures and Processors. IEEE, 2016:107-114.

[8] Venieris S I, Bouganis C S. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs[C]// IEEE, International Symposium on Field-Programmable Custom Computing Machines. IEEE Computer Society, 2016:40-47.

[9] Zhang C, Li P, Sun G, et al. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks[C]// Acm/sigda International Symposium on Field-Programmable Gate Arrays. ACM, 2015:161-170.

[10] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng,Y. Chen, and O. Temam, "Shidiannao: shifting vision processing closer to the sensor," in ISCA. ACM, 2015, pp. 92–104.

[11] Benedetti A, Prati A, Scarabottolo N. Image convolution on FPGAs: the implementation of a multi-FPGA FIFO structure[C]// Euromicro Conference, 1998. Proceedings. IEEE, 1998:123-130 vol.1.

[12] Perri S, Lanuzza M, Corsonello P, et al. A high-performance fully reconfigurable FPGA-based 2D convolution processor[J]. Microprocessors & Microsystems, 2005, 29(8):381-391.

[13] Carlo S D, Gambardella G, Indaco M, et al. An area-efficient 2-D convolution implementation on FPGA for space applications[C]// Design and Test Workshop. IEEE, 2011:88-92.