**A High Performance Framework for large-scale Convolution Operation on FPGA**

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. Because the importance of convolution operation, accelerating large-scale 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 PE, it performs non-redundant calculations when the stride of convolution operation is greater than 1, significantly reduces the time of calculation. At the same time, in order to achieve the goal that reducing 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% memory access times of the previous accelerator.

Keyword: FPGA; convolution; on-chip memory; acceleration

Introduction

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

Because FPGA has a large amount of DSP resources and can work in a pipelined style, it become a candidate platform for building high-performance deep learning systems [4, 5, 6, 7, 8, 9, 10]. The most computational-intensive part of CNN is the convolution operation. Therefore, designing PE architecture for convolution operation to parallel computations is a key to realize the accelerator [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 flow, when the stride of convolution operation is greater than 1, the utilization rate of computing resources dropped significantly.

For the storage of intermediate results and parameters, normally we can do it in three ways. Firstly, for the small networks like LeNet, we can store all the intermediate results and parameters in on-chip memory to reduce the access to external memory [10]. Owing to the limitation of resources, we cannot accelerate deep networks in this way. Secondly, all the intermediate results and parameters are stored in the external memory [5, 7]. Deep networks can be accelerated in this way, but memory bandwidth become the bottleneck that the computing resources cannot be fully used. Thirdly, we can store the intermediate results in on-chip memory and the parameters in external memory [4, 6, 8, 9]. Normally, we can set input buffer and output buffer, loading data before computation and storing data after computation. But some networks are too big to store the intermediate results in on-chip memory, optimizations are needed to reduce the usage of on-chip memory.

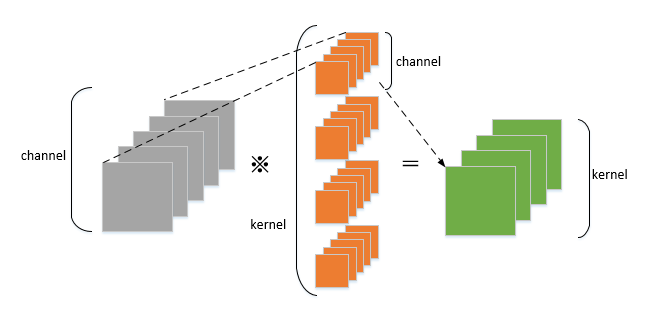
The rest of this paper is organized as follows: Section 2 provides an introduction to 2D convolution operation, traditional PE design and the new PE design. Section 3 describes the optimizations of the usage of on-chip memory. Finally, Section 4 make a comparison between our work and existing work.

**2 Convolution Operation**

The 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 channel weight numbers 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×N and a sliding window of K×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 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. Shown in Figure1 (b). The convolution operation is done.





3 PE Design

A Z-type PE

The size of convolutional kernel usually has only several options, such as 3×3, 5×5, and 7×7. Because of 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 operation, which will complex the control system. In fact, because that data reuse exists in adjacent convolution windows, it is not necessary to load 9 number every time. If the stride of the convolution operation is 1, for a convolution window, it has 6 same number with the left convolution window, also the upper. Removing 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 load data performs as Z, so, we call it as Z-type PE. The following figure 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 , while is the number of pixels in a row of the input feature map. The reason is that it can make sure that the number in data registers exists in one convolution window. When a new number comes, the data in the data registers and FIFOs will flow 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 multiplications are done in a cycle, and the addition 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 convolution results in a cycle.

But there are two major flows of Z-type PE. First, each Z-type PE costs two BRAMs to implement the FIFOs. When the quantity of Z-type PEs increases, the usage of BRAMs for the Z-type PE cannot be ignored. Second, for the 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 operation, in other word, a few cycles will be wasted for wait, which lead to the waste of computing resources.

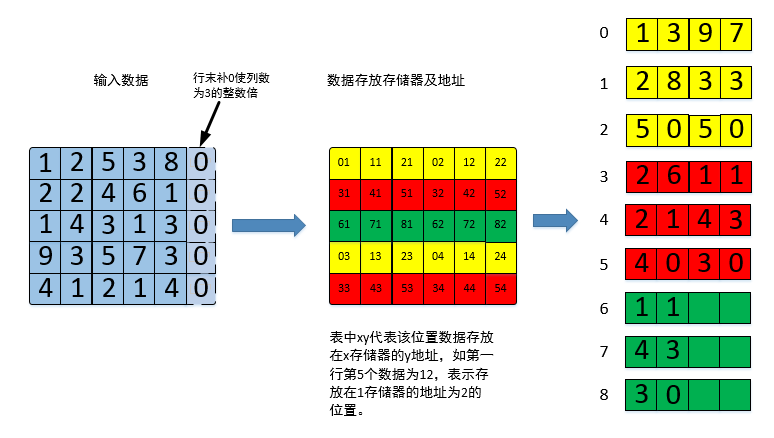
Supposed that the size of input feature map is , the size of output feature map is , the stride is s. the total time (including wait time and computing time) are:

The computing time are:

So, the utilization rate is:

B Rotary data-storage and new PE design

In order to promote the utilization rate, we need 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 with no extra storage area. This method uses 9 RAM, while each RAM store part of the input data. Because we can only read a number from a RAM, how to make sure the 9 numbers in a convolution window in different RAMs become the difficulty. The following figure shows how this method work.

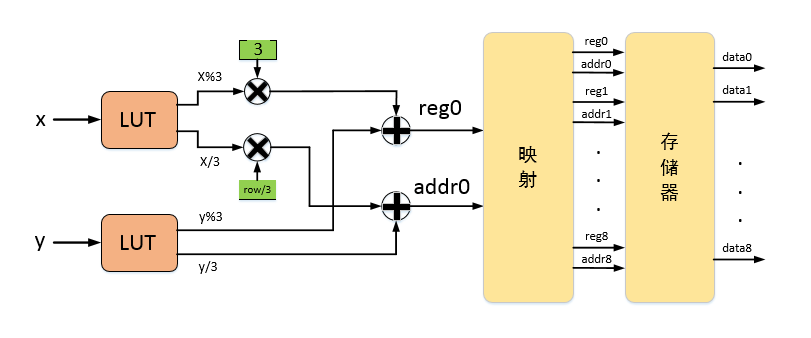


The data storage process: the input feature map is divided into three part. The first part is row 0, 3, 6…3n, the second part is row 1, 4, 7…3n+1, the third part is row 2, 5, 8…3n+2. Add 0 to the last of each row to make sure the number of a row can be divided by 3 with no remainder. The first part is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

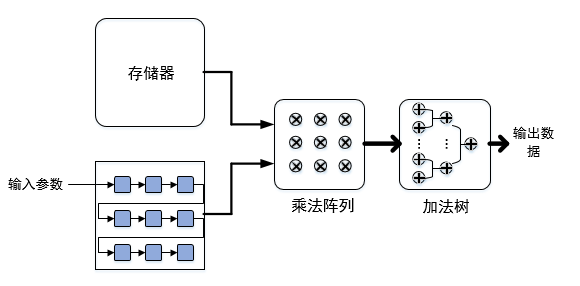
The reason why this can make sure that the 9 numbers in a convolution window in different RAMs is that, the distance between two numbers in the same RAM will be equal or greater than 3 vertically and horizontally. Which means that, in the area of 5x5 centered on number x, there will not exist other number stored in the same RAM. So, any 3x3 convolution window including this number 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. 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 process 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 process we calculate the address of the left-upper number is that, we calculate the address of the first number of the same row according to x, and the address shift according to y. The formula is shown as following. Once we get the RAMs and addresses of the 9 numbers, we can read them from 9 RAMs.

In FPGA, division and remainder operation are hard to implement. Considering that x and y is less than 512 normally and the divisor is a fixed number 3, we can use the method of LUT (look up table) to replace the operation. We set the depth of LUT to 512, so the process becomes:



Because we load 9 numbers each cycle, so Transmission unit is not needed. The new PE includes only calculation unit as following



4 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. First, the input feature maps will be loaded to the input buffer, then, the calculation starts, the immediately results will be stored in the output buffer. After the calculation, the results will be loaded to external memory. The design is shown as figure.

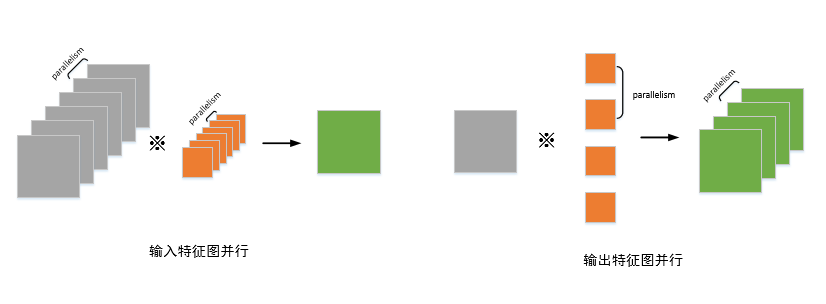
There is a problem that some FPGAs may not have enough on-chip memory resources to store all the data. So, it is necessary to optimize the usage of on-chip memory.

A Structures for optimization and parallelism

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

Output-optimization structure, which optimize 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.

Input-optimization structure, which optimize 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. The two structures are shown as following.



Supposed that the is the number of input feature maps, is the number of output feature maps. The storage area of output-optimization structure is:

while the storage area of input-optimization structure is:

Because the is bigger than normally, so, is normally smaller than . input-optimization structure is adopted in this paper.

For the parallelism, there are mainly three types of parallelism in CNN workloads: operator-level parallelism (inside the PE), inter-input parallelism (multiple input features are combined to create a single output) and inter-output parallelism (multiple independent features are computed simultaneously). In our implementation, all the three types of parallelism are considered. For the output-optimization structure, the inter-output parallelism will not cost extra storage area, but the inter-input parallelism will increase the size of input buffer. The storage area becomes:

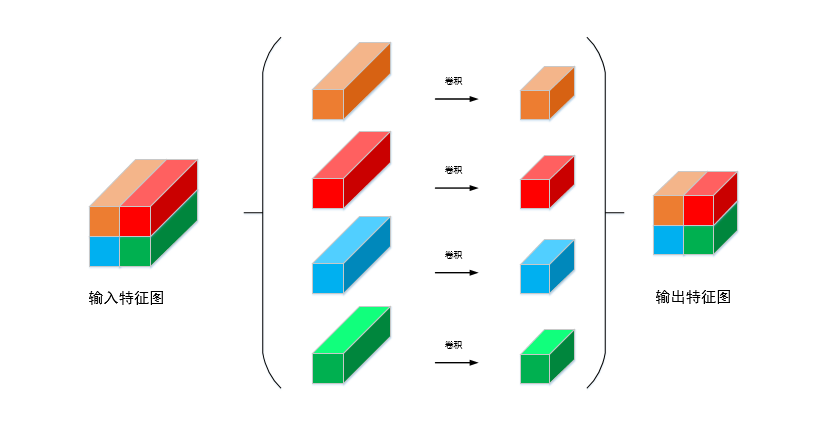
Supposed that the inter-input parallelism is , the inter-output parallelism is , the following process is executed for calculation:

1. Load input feature maps from the external memory to input buffer, stored in the way of rotary data-storage.
2. Load corresponding parameters from kernels to PEs.
3. Perform calculations, get the immediately result of output feature maps.
4. Repeat 2, 3 until the immediately 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 the external memory.

B Block-Calculation

For large-scale convolution operation, some FPGAs may not have enough on-chip memory resources to store the intermediate results, block-calculation become necessary.

The idea is to split the input feature map of × into some sub-input feature maps of ×. Convolution operation is performed on one sub-input feature map each time, and the sub-output feature map will be stored to external memory. After all the sub-input feature maps are done, the output feature map is obtained.



The value of needs to be calculated first. For input-optimization structure, supposed that the capacity of output buffer is , the maximum size of sub-output feature map will be:

So, the size of sub-input feature map is:

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

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

So, for the block-calculation, the input feature map needs to be loaded once, the weight needs to be loaded times, the output feature map needs to be stored once. Normally, the number of parameters is much smaller than the data, the memory bandwidth will not become the bottleneck.

5 Performance Analysis

As mentioned in Section 1, works have been done to develop new solutions for accelerating or optimizing the computation of convolution operation. A comparison is made between our work and reference[x]. In [x], 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 convolution layers are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 层数 | Inputsize | Convsize | Stride | Padding | Out size |
| 1 | 224\*224\*3 | 7 | 2 | 3 | 112\*112\*64 |
| 8 | 56\*56\*64 | 3 | 2 | 1 | 28\*28\*128 |
| 17 | 14\*14\*256 | 3 | 1 | 1 | 14\*14\*256 |

We will compare them under the condition of roughly the same resources, including computing resources and on-chip memory resources. For the computing resources, the parallelism degree of input is set to 2, the parallelism degree of output is set to 16, which means that 288 multiplications 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 as following:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Input-buffer | LUT | Weight-buffer | PE-FIFO | Output-buffer | total |
| Ours | 36 | 2 | 1 | 0 | 64 | 103 |
| [x] | 16 | 0 | 1 | 64 | 16 | 97 |

The costing time of two solutions are shown in figure.

The external memory access of two solutions are shown in figure.

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

7 Conclusion

This paper has focused on two major problems of accelerating 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 decrease the usage of on-chip memory. For the second problem, two structures and a block-calculation method are proposed. The structures can achieve the same external memory access within about 50% on-chip memory resources. The block-calculation method can implement convolution operation of any scale, and the cost is the repeated reading of weight parameters, which needs few external-memory access.

Conference

[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.