Hindawi
International Journal of Reconfigurable Computing
Volume 2018, Article ID 1785892, 10 pages
https://doi.org/10.1155/2018/1785892

Research Article

Design of FPGA-Based Accelerator for Convolutional Neural Network under Heterogeneous Computing Framework with OpenCL

Li Luo,[1] Yakun Wu,[1] Fei Qiao,[2] Yi Yang,[2] Qi Wei,[2] Xiaobo Zhou,[1] Yongkai Fan,[3] Shuzheng Xu,[2] Xinjun Liu,[4] and Huazhong Yang[2]

[1]*Department of Electronic Science and Technology, Beijing Jiaotong University, Beijing, China*
[2]*Department of Electronic Engineering, Tsinghua University, Beijing, China*
[3]*China University of Petroleum, Beijing, China*
[4]*Department of Mechanical Engineering, Tsinghua University, Beijing, China*

Correspondence should be addressed to Qi Wei; weiqi@tsinghua.edu.cn

Received 26 January 2018; Revised 5 April 2018; Accepted 28 May 2018; Published 3 July 2018

Academic Editor: Michael Hübner

CPU has insufficient resources to satisfy the efficient computation of the convolution neural network (CNN), especially for embedded applications. Therefore, heterogeneous computing platforms are widely used to accelerate CNN tasks, such as GPU, FPGA, and ASIC. Among these, FPGA can accelerate the computation by mapping the algorithm to the parallel hardware instead of CPU, which cannot fully exploit the parallelism. By fully using the parallelism of the neural network's structure, FPGA can reduce the computing costs and increase the computing speed. However, the development of FPGA requires great design skills. As a heterogeneous development platform, OpenCL has some advantages such as high abstraction level, short development cycle, and strong portability, which can make up for the lack of skilled designers. This paper uses Xilinx SDAccel to realize the parallel acceleration of CNN task, and it also proposes an optimizing strategy of single convolutional layer to accelerate CNN. Simulation results show that the calculation speed could be improved by adopting the proposed optimizing strategy. Compared with the baseline design, the strategy of single convolutional layer could increase the computing speed 14 times. Performance of the whole CNN task could be improved 2 times more than before, and the speed of image classification could attain more than 48 fps.

1. Introduction

Deep learning is currently the most popular field in recent years. It processes complex input data based on the multilayer neural network. Deep learning has a significant effect on the analysis of machine vision [1], video surveillance [2], and other information [3], which solves the problem of pattern recognition in many complex scenes. It is also important for the development of artificial intelligence. As a classical model of the deep learning system, convolutional neural network (CNN) has an enormous advantage in image recognition [4]. Especially in the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) [5], CNN improved the accuracy rate from 70% to 80%, which is far better than the traditional methods. Actually, CNN has broad space for further development, which has been widely used in face recognition [6], audio retrieval, hand posture recognition [7], etc. [8]. It becomes increasingly difficult for CPU to realize complex CNN [9], while using the heterogeneous computing platform is an emerging method to accelerate CNN and the hardware such as GPU [10, 11] or FPGA [12, 13].

Compared to CPUs, the neural network's accelerators based on FPGA are increasingly popular because of their higher efficiency [14]. Over the baseline CPU, FPGA accelerators deliver one to two orders of magnitude speedups [1]. Moreover, the FPGA1024 design delivers almost 50x performance improvement over the baseline CPU (Intel® Xeon E5-2699v3 server) [14]. In addition, due to the structure of CNN,

each layer of calculation is independent of others, and the interlayered structure can be dealt with like a flow structure. Moreover, various modules on FPGA can be executed in parallel [15]. It is very suitable to accelerate CNN by mapping the flow structure on FPGA. However, in order to use FPGA, the development process requires that the hardware description language as well as some knowledge about the internal structure of FPGA must be comprehended, which limits the designers' ability to design to some extent.

OpenCL provides a new method for the acceleration of CNN based on FPGA. It is a unified programming environment [16] and a parallel programming standard for the heterogeneous systems. Based on the traditional C language, the code written in OpenCL is highly portable [17]. Moreover, on account of the existence of the framework about OpenCL to FPGA, designers need neither to synthesize hardware description language nor to understand the hardware circuit structure, which greatly lowers the using threshold [18].

This paper proposed a parallel acceleration strategy of CNN based on FPGA with OpenCL by the use of Xilinx SDAccel. As the convolution layer is the most complicated part of CNN, we first optimize the single convolution layer and then, on this basis, we accelerate complete CNN and explore different optimization strategies.

Throughout this paper, we deal with the following: the general concept of CNN and the traditional method of CNN on CPU in Section 2, the knowledge of heterogeneous computing platform about OpenCL in Section 3, optimization of a single convolution layer in Section 4, the acceleration of complete CNN in Section 5, the experimental environments and the analysis of experimental results in Section 6, and the conclusions in Section 7.

2. CNN Information and Traditional Acceleration

2.1. The Basic Structure of Convolution Neural Network. CNN is an efficient identification method which has received considerable attention in recent years. CNN is a hierarchical model whose basic structure consists of input layers, convolution layers, pooling layers, fully connected layers, and output layers. The main characteristic is that the convolution layer and the pooling layer alternate, and the local feature of the previous layer gets the characteristics of the next layer of convolution by the weight of the convolution. Through the connections of multiple convolution with pooling layers, images gradually change from the surface features to the deep ones, then the features are classified by the fully connected layer and the output layer, and finally the images are divided into categories. Therefore, according to the function of every layer, CNN can be divided into two parts: the feature extractor is composed of the input layer, the convolution layer, and the pooling layer, while the fully connected layer and the output layer constitute the classifier.

CNN was first put forward by a professor from University of Toronto called LeCun [3]. Based on the research of Fukushima, he used BP algorithm to design the first CNN, named LeNet-5. Because LeNet-5 is one of the most classic structure of CNN, the following introductions and

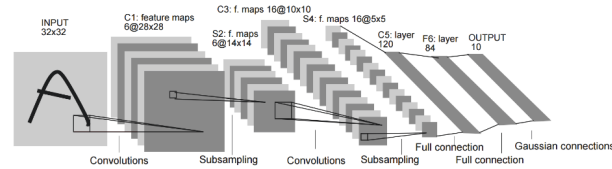


Figure 1: LeNet-5 [3].

experiments are taken with the example of LeNet-5. LeNet-5 is CNN for handwritten numeral recognition; **Figure 1 is the structure of LeNet-5,** which contains three convolution layers, two pooling layers, a fully connected layer, and a Gaussian layer.

2.1.1. The Convolution Layer. In the convolution layer, the input data is convolved with a learning convolution kernel and the result of convolution forms the characteristic pattern of this layer. By the activating function, each characteristic pattern can be combined with the numerical value of previous characteristic patterns. **Its expression is shown in**

$$x_j^l = f\left(\sum_{ieM_j} x_i^{i-1} * k_{ij}^l + b_j^l\right) \tag{1}$$

where 'I' represents the number of layers, 'k' represents the weight of the convolution kernel of l layer, and 'b' represents the bias of the characteristic pattern after the convolution; f(x) is the activation function, like sigmoid and tanh.

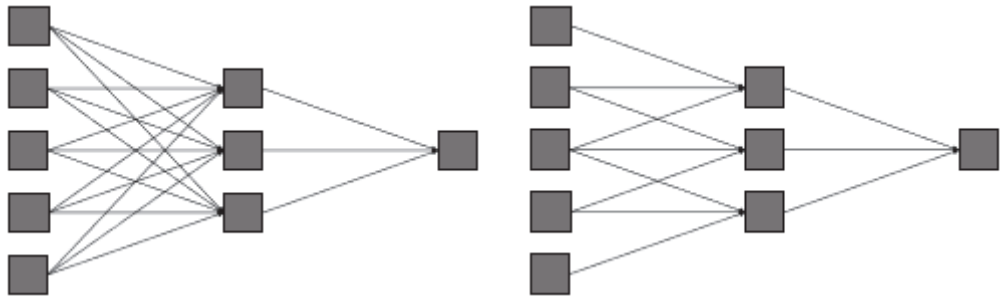
In LeNet-5 model, for example, the input size is 32∗32 in C1 Layer, the convolution kernel's size is 5∗5, and the size of C1 Layer is 28∗28 (28=32-5+1). It can be seen from Figure 1 that C1 Layer has six two-dimensional planes totally, and each plane has 26 (26=5∗5+1) parameters. Therefore, C1 Layer has 156 parameters and 122304 (122304=(5∗5+1)∗6∗28∗28) connections. The same is true for C3 and C5 Layer.

2.1.2. The Pooling Layer. The pooling layer is also called the subsampling layer. It usually connects with the convolution layer. Through partial correlation principle, on the one hand, it can improve the robustness of the system, and on the other hand, it can reduce the calculation of the characteristic pattern. **Its expression is shown in**

$$x_j^l = f\left(\beta_j^l dawn\left(x_j^{l-1}\right) + b_j^i\right). \tag{2}$$

'dawn(x_j^{l-1})' represents the pooling function. Beta represents the weight coefficient of the pooling layer. 'b'represents the bias of the characteristic pattern after the pooling. The main principle is to divide the image into n∗n image blocks, then sum the pixels in each image block, and take the mean or the maximum value. Therefore, the output image is the size of the input image of 1/n. Among them, every characteristic pattern has its own beta and b.

In LeNet-5 model, S2 consists of six characteristic patterns of the size 14 x 14. Each neuron in the characteristic



(a) Full connection layer model (b) Convolutional layer model

Figure 2

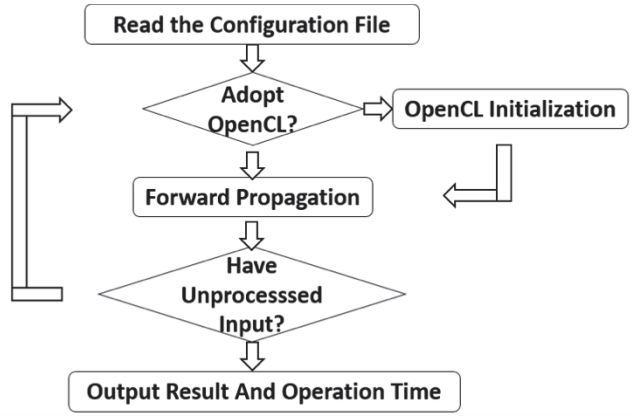


Figure 3: Workflow.

pattern connects with the 2 x 2 neighborhood of the corresponding characteristic pattern in C1 Layer. Because each neuron's sensory field does not overlap, every characteristic pattern in S2 is one quarter of the size of the characteristic pattern in C1. S2 Layer has 12 (12=(1+1)∗6) parameters and 5880 (5880=14∗14∗(4+1)∗6) connections. S4 sampling Layer's analysis is the same.

2.1.3. The Fully Connected Layer. When neurons of this layer are connected with each neuron in the upper layer, it forms a fully connected layer. After multiple feature extraction of multiple convolution layers and pooling layers, CNN uses the fully connected layer to classify the extracted features. **Its expression is shown in**

$$x^l = f\left(u^l\right) \tag{3}$$

$$u^l = w^l x^{l-1} + b^l. \tag{4}$$

In LeNet-5 model, F6 is fully connected with C5, which includes 84 neurons, 10164 parameters, and 10164 connections.

Comparing Figure 2(a) with Figure 2(b), we can see the difference between the convolution layer and the fully connected layer. In the convolution layer, the connections and parameters are reduced by means of local connection and weight sharing. In the fully connected layer, neurons of the layer are connected with all the neurons in the upper layer, and its parameters and connections are numerous, so the fully connected layer generally appears near the output part of CNN.

2.2. The Implementation of CNN by CPU

2.2.1. The Overall Design. The system is implemented by C++ language. The abstract base class is defined; then the convolution layer, the fully connected layer, and the pooling layer are all derived from the abstract base class. CNN is responsible for IO and the configuration of all CNN structures and calls OpenCL's API for the acceleration.

Figure 3 shows the workflow of the entire system. First, read the configuration file of CNN, and then choose the way of calculation according to the need. Finally, process all input data in turn and get the elapsed time.

2.2.2. Test Results. The system is tested on the server platform, and the convolution neural network is LeNet-5. The main parameters of the platform are shown in Table 3. The benchmarks of CNN, which are small convolutional cores and LeNet-5 network, are simple enough. Therefore, for current evaluations, the experimental results have no cumulative effect. The results of multiple experiments are almost constant through multiple running evaluations. **Table 1 lists the single**

Table 1: The single thread calculation time of the LeNet-5 layers on CPU.

Layer	C1	S2	C3	S4	C5	F6	O7	Total
Time (ms)	4.14	0.14	7.86	0.04	1.57	0.16	0.02	13.93

Table 2: Hardware on Xilinx FPGA[25].

OpenCL storage structure	Hardware implementation about Xilinx FPGA
Global memory	The DDR storage
Local memory	The on-chip Block RAM
Private memory	The on-chip registers

thread calculation time of the LeNet-5 layers on server with CPU.

3. The Heterogeneous Computing Platform about OpenCL

OpenCL is an industry-standard framework about heterogeneous platforms made up of CPU, GPU [19], FPGA [20], and other processors. By defining a unified framework, OpenCL significantly increases the portability of programs. At the same time, OpenCL allows developers to make specific optimizations for different hardware if they need. OpenCL framework mainly includes platform models, memory models, programming models, and execution models [21]. Similar to CUDA, OpenCL provides a standard framework for parallel programming and can support programming a variety of hardware platforms, including FPGAs.

Since 2013, Altera and Xilinx started to adopt OpenCL SDKs for their FPGAs [22, 23]. It makes more developers take advantage of the high-performance and low power benefits to design for FPGAs.

OpenCL mainly faces parallel applications and supports two parallel programming models [24]: data parallelism and task parallelism. Data parallelism is implemented with multiple work items in parallel, while task parallelism is a command queue that relies on out-of-order execution. In this paper, the data parallel model is adopted to accelerate the single convolution layer, while CNN adopts the task parallel model to accelerate.

4. The Optimization of the Convolution Layer

In image recognition, the convolution plays a key role. It evolves from two aspects, one is the delay network of sound processing, and the other is the algorithm to extract the feature points on the image processing. For the latter, the convolution is to filter the image and make some eigenvalue extraction. As it can be seen from Table 1, 90% of CNN is consumed in the convolution layer [26]. Therefore, it is necessary to optimize the convolution part. We can think about algorithms, hardware features, and so on.

4.1. Loop Optimization. By using more nested loops, the data's reused effect can be improved to accelerate the computation of the convolution layer. In order to further optimize the convolution layer, when there is no dependence in the loop,

```
#ifdef _xilinx_
_attribute_((opencl_unroll_hint(2)))

#endif
for (int i = 0; i < N; ++i) {

  //DO THE WORK

}
```

Figure 4

```
#ifdef _xilinx_
_attribute_((xcl_pipeline_loop))

#endif
for (int i = 0; i < N; ++i) {

  //DO THE WORK

}
```

Figure 5

we can use loop unrolling to reduce the delay by consuming more hardware resources and parallel execution of loops. **As shown in Figure 4, the loop will be expanded with a base of 2, and the theoretical runtime will be halved.**

However, when the dependence exists in the loop, We can solve this problem by cyclic pipelining. Cyclic pipelining is that the three stages of reading, calculation, and memory are in the working state to improve throughput, through the pipeline operation. **Figure 5 shows how to use cyclic pipelining.**

4.2. On-Chip Memory. In OpenCL storage structure, the memory is divided into the global memory, the local memory, and the private memory. **Table 2 lists this hardware on Xilinx FPGA [25].**

Since the global memory is mainly used for the data transmission between Host and FPGA by adopting the DDR

Table 3: The Main Parameters of The Host.

Feature	Specification
CPU	Intel Xeon E3-1230 V2@3.30GHz
Memory	32GB
OS	CentOS 6.5 Final
OpenCL SDK	Xilinx SDAccel 2015.3(with OpenCL 1.0)

storage, the access delay is longer than before. Meanwhile, a private memory adopts the on-chip registers with minimum delay except for large amount of data of CNN, it will consume large amounts of the resources, and the local memory is a compromise. Its access latency is small while it does not take up a large number of resources. Therefore, before the actual calculation starts, the required data can be read from the global memory to the local memory and then read back after the calculation. It can further shorten the computation time of the convolution layer and reduce the delay. In addition, if the same image convolved, we can convolve all of the convolution kernels together and store these separately.

4.3. Multiple Compute Units. On the FPGA platform, hardware resources are programmable and there is no preconfigured core, which gives users greater freedom. The core is the compute units, which can calculate the work item. For increasing the time parallelism, the Xilinx OpenCL SDK automatically assigns work items to different compute units at runtime. The number of compute units are limited by two factors: one is the total resources on FPGA, and the other is the maximum number of compute units supported by SDAccel. In practice, SDAccel currently supports up to 10 compute units, but they do not make full use of the resource on FPGA. It can be assumed that if SDAccel adds the number of compute units, the time parallelism can be higher.

5. The Optimization of CNN

This part mainly considers the acceleration of CNN from two aspects: the storage and the pipeline.

5.1. The Global Memory. The global memory is implemented by DDR storage to exchange the data between Host and FPGA. However, there are a number of layers in CNN, and the cache between layer and layer is only the intermediate result of calculation, without the need to be visible to the Host. OpenCL specifies that the local memory can only be used within the kernel. Therefore, these caches cannot be implemented in the local memory.

In this case, SDAccel provides an optimization of the global memory. By defining the global memory outside the kernel, SDAccel automatically uses Block RAM on the chip. Therefore, by defining the caches between layers and layers by the global memory, the computation delay will be further reduced. **Figure 6 shows how to use the global memory on the chip.**

5.2. The Pipelining of CNN. In OpenCL, all compute tasks can be represented as an event. The Host submits the compute

```
_global float buffer[100];

_kernel void f1(_ global float* in) {
    //Read from in and write to buffer.
    ...
}
_kernel void f2(_ global float* out) {
    //Read from buffer and write to out.
    ...
}
```

Figure 6

tasks to the task queue in OpenCL, which is assigned to the compute device to complete the calculation. The order of execution of events in a task queue can be sequential and disorderly. In an out-of-order execution queue, a number of dependent events can be set for every event to ensure data dependencies. OpenCL commands that the queue does not run the event until all dependent events are completed. This kind of event mechanism is well suited to realize the pipelining of CNN.

5.2.1. The Dependence Relationship of CNN. The key of the pipelining of CNN is the relationship between events. For N layers of CNN, an input sample can generate (N+2) events, so an event can be represented as a binary group (i, j), representing the jth event of the ith input sample. The arrow symbol (i1, j1)⟶(i2, j2) indicates that the event (i2, j2) depends on the event (i1, j1), so the event (i2, j2) must wait for the event (i1, j1) to be completed.

After the analysis, two dependencies exist in CNN, which are expressed as Theorems 1 and 2.

Theorem 1 ((i, j-1) ⟶ (i, j)). *Event (i, j) depends on event (i, j-1). Obviously for the same input sample, the calculation of the Layer j in CNN depends on the calculation result of the Layer (j-1).*

Theorem 2 ((i-1, j+1) ⟶ (i, j)). *Event (i, j) is dependent on event (i - 1, j + 1). There is only one cache area between Layer j and Layer (j+1), so the calculation of Layer j must wait for the calculation of Layer (j+1) before it outputs the calculation result to the cache between the two.*

Figure 7 shows the dependency among OpenCL events generated by the four input samples in three-layer CNN. Each square represents an event, the horizontal axis represents Event j, and the vertical axis represents Input Sample i.

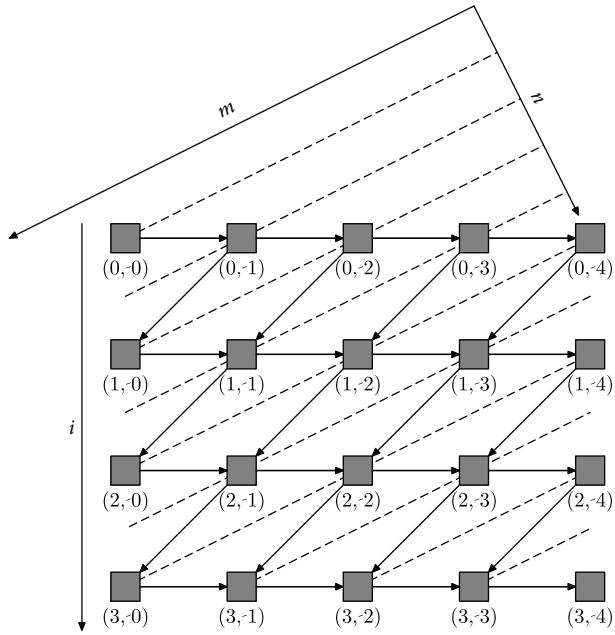


Figure 7

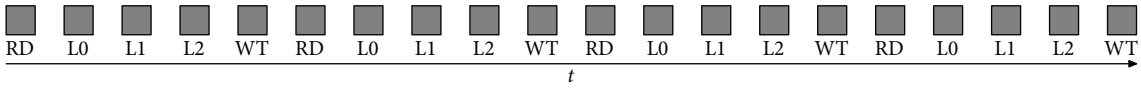


Figure 8

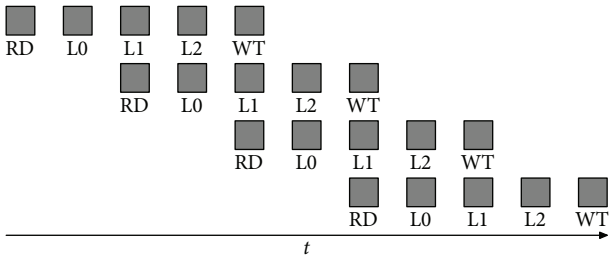


Figure 9

Figure 8 shows a sequence diagram in which CNN is not streamlined. You can see that all events are executed sequentially.

Figure 9 shows a sequence diagram in which CNN is streamlined. Due to the pipelining of each of the two levels, the time required to process an input sample has not changed, but overall throughput has been increased by about N2.

Theorem 3 represents the time t required to deal with an input sample on average.

Theorem 3 ($T = \max(t_0 + t_1, t_1 + t_2 \ldots, t_{j-2} + t_{j-1})$).

5.2.2. The Realization of OpenCL. OpenCL API provides that all dependent events must be continuously distributed in memory when setting dependent events for an event. **It means that all events connected by the dotted line in Figure 7 should be continuously distributed in the memory.** To avoid unnecessary losses, all events need to be remapped to a new two-dimensional coordinate system (m, n). In this frame, all events of the same n coordinates form a one-dimensional array and can be called directly by OpenCL API. To define j as the total events of an input sample, **Theorem 4 represents the mapping relation from coordinates (i, j) to coordinates (m, n) and Figure 10 represents the arrangement of events in (m, n) coordinates after the mapping.**

Theorem 4. *The mapping relation from coordinates (i, j) to coordinates (m, n) is as follows:*

$$m = \min\left(i, \left[\frac{J - 1 - j}{2}\right]\right) \tag{5}$$

$$n = j + 2i. \tag{6}$$

6. Experiment Environment and Result Analysis

6.1. Experimental Environment. The laboratory server is used to set up the experimental environment, in which the Host adopts the CPU and the computing equipment uses FPGA to complete the experiment and measure the data.

6.1.1. Host Configuration. **Table 3 lists the main parameters of the Host.** The CPU is Intel Xeon e3-1230 V2@3.30GHz,

Table 4: Specific parameters of alpha data ADM-PCIE-7V3 FPGA.

Feature	Specification
Memory	Two 8GB DDRS memory speeds up to 1333MT/s
# of Flip Flops	866400
# of LUTs	433200
Device	
# of DSP Slices	3600
# of Block RAMs	1470

Table 5: The optimization results.

Optimization Methods	FFs	LUTs	DSPs	Block RAMs	Time(ms)
Baseline	5360	8793	16	18	50.4
Loop Pipeline	5374	8814	16	18	50.0
Loop Unroll	5533	10214	16	18	27.9

Table 6: Calculating time with different number of compute units.

Compute unit	1	2	3	4	5	6	7	8	9	10
Computing time	27.9	14.5	10.8	7.4	7.3	5.9	5.8	4.2	3.9	3.5

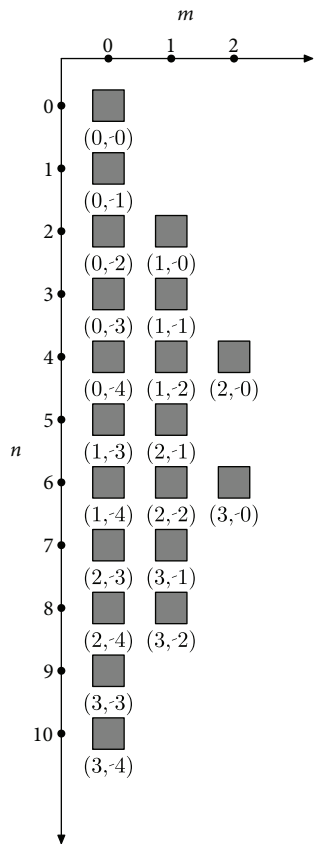


Figure 10

the operating system is CentOS 6.5 Final, and the Xilinx OpenCL SDK is SDAccel 2015.3, which supports OpenCL 1.0.

6.1.2. FPGA Configuration. The computing device uses Alpha Data ADM-PCIE-7V3 FPGA, and its FPGA's core uses Xilinx Virtex-7XC7VX690T-2, which includes dual channel DDR3 memory (1333MT/s). **Table 4 lists its specific parameters.**

6.1.3. CNN Benchmark. CNN for testing uses LeNet-5, introduced in Section 2. As a classical CNN, LeNet-5 can better test the performance of the acceleration on FPGA. At the same time, the whole system stores data by XML format which can automatically generate OpenCL code according to the parameters of CNN.

6.2. Results Analysis

6.2.1. Analysis of Optimization Results about the Convolution Layer. The optimization of the convolution layer is mainly the C3 Layer in LeNet-5. After using a single compute unit of loop tiling and local memory, loop unrolling and cyclic pipelining are used to optimize it. **Table 5 lists the optimization results, and the loop unrolling increases the calculation speed by 45% and consumes more computational resources.**

The optimization is for a single compute unit. If we combine multiple compute units on the FPGA, the computation time is less. **Table 6 shows the value of calculating time with different numbers of computing units.** In current evaluations, since the clock frequency of the FPGA development board is fixed, and the clock cycles of selected benchmark of CNN are also constant. The running time of multiple evaluations is constant as well. In addition, we have no OS interactions and complex memory management, so the running time is not a random value through multiple evaluations. **Figure 11 shows the change trend of the calculation time with the number of compute units.** We can see that the computation time is basically decreasing with the number of

Table 7: The assignments of compute units.

Layer	C1	S2	C3	S4	C5	F6	O7
Number of Compute Units	2	1	3	1	1	1	1

Table 8: The resources of every compute unit on FPGA.

Layer	FFs	LUTs	DSPs	Block RAMs
C1∗2	8331	11762	19	20
S2	6967	10672	15	31
C3∗3	8160	15935	17	18
S4	6777	10071	15	9
C5	7141	11593	16	136
F6	6637	10341	15	10
O7	5087	4488	12	4
SUM(%)	74751(8.63%)	118494(27.32%)	145(4.03%)	284(19.32%)

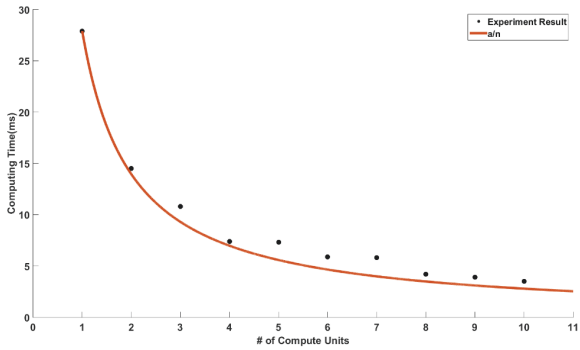


Figure 11: The calculation time with the number of compute units.

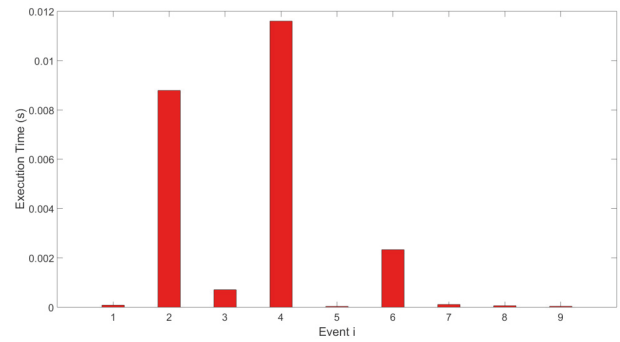


Figure 12: The average running time of every event.

compute units, but the scheduling unit also produces some losses, **so it is not completely consistent with the curve in Figure 11.**

6.2.2. Analysis of Optimization Results of CNN. In sequential execution, all OpenCL kernels calls will be executed sequentially and SDAccel supports up to 10 compute units. The convolution layer takes the most time, so two or three compute units are configured for the convolution layer. **Table 7 lists the assignments of compute units.** However, though the most compute units have been used, the resources of FPGA are still small enough to give full play to the performance of FPGA.

Table 8 lists the resources of FPGA that are consumed by every compute unit. We can see that if more compute units can be supported in parallel, the performance will improve further.

Figure 12 shows the average running time of every event in the program, and we can see that, even after optimization, the two convolutions layers (event 2 and event 4) consume the most time.

Figure 13 shows the time distribution of the first two events without streamline. Each event is represented as a line segment, where left endpoint's coordinate is 'start time', and

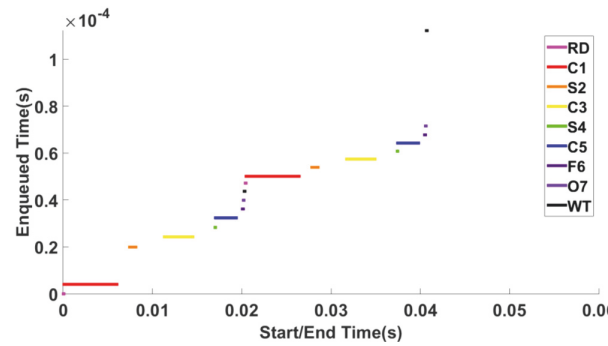


Figure 13: The time distribution without streamline.

right endpoint's coordinate is 'end time'. It can be found in all events being executed sequentially.

Figure 14 shows the time distribution of the events with streamline. It shows that both the second convolution layer of the first input and the first convolution layer of the second input can be calculated on FPGA. The overall throughput on FPGA is also improved, which is represented by the properly working pipeline. Before the pipeline, the throughput of the system is 24.4 FPS, while it reaches 48.5 FPS. Now the throughput has doubled.

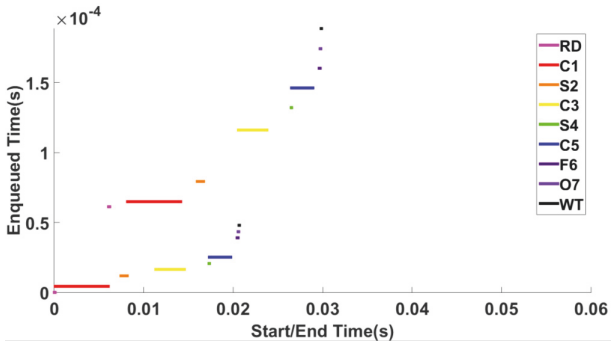


Figure 14: The time distribution with streamline.

7. Conclusion

This paper is devoted to the parallel acceleration of CNN based on FPGA with OpenCL, by using Xilinx SDAccel tools. Taking LeNet-5 as an example, the acceleration of convolution layer and CNN is studied. On the convolution layer, by adopting the loop optimization and memory accessing methods, the computation speed could be increased 14 times of processing convolution layer, while the whole system processing speed would be improved 2 times, where the overall speed reaches 48.5 fps.

Due to their own characteristics, CPU could not fully exploit the parallelism of CNN algorithms and GPU would consume much more power. Alternatively, by fully using the parallelism of the neural network's structure, FPGA can not only reduce the computing costs but also increase the computing speed. Currently, Xilinx SDAccel's tool chain could not support up to 10 compute units in parallel until now, but FPGA has its redundant resources to support more computing units. With the increasing development of the tool chain in the future, the parallel acceleration of CNN based on FPGA with OpenCL has its prospects.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to acknowledge the support from National Natural Science Foundation of China under Grant no. 91648116 and National Key R&D Program of China under Grant no. 2017YFC1500601. The authors also acknowledge the support from Xilinx University Program.

References

[1] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: an FPGA-based processor for Convolutional Networks," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, pp. 32–37, Prague, Czech, September 2009.

[2] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 60, pp. 1097–1105, 2017.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F Pereira, C. J. C. Burges, L. Bottou, and etal., Eds., vol. 60, pp. 1097–1105, 2012.

[6] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5325–5334, Boston, MA, USA, June 2015.

[7] P. Barros, S. Magg, C. Weber, and S. Wermter, "A Multichannel Convolutional Neural Network for Hand Posture Recognition," in *Artificial Neural Networks and Machine Learning – ICANN 2014*, vol. 8681 of *Lecture Notes in Computer Science*, pp. 403–410, Springer International Publishing, Cham, 2014.

[8] T. Yang, *Cellular Neural Networks and Image Processing*, Nova Science Publishers, 2002.

[9] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '10)*, pp. 257–260, IEEE, Paris, France, May-June 2010.

[10] I. Buck, "GPU Computing: Programming a Massively Parallel Processor," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO'07)*, pp. 17-17, San Jose, CA, March 2007.

[11] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of GPU-based convolutional neural networks," in *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP '10)*, pp. 317–324, Pisa, Italy, February 2010.

[12] G. Borgese, S. Vena, P. Pantano, C. Pace, and E. Bilotta, "Simulation, Modeling, and Analysis of Soliton Waves Interaction and Propagation in CNN Transmission Lines for Innovative Data Communication and Processing," *Discrete Dynamics in Nature and Society*, vol. 2015, pp. 1–13, 2015.

[13] 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 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, (FPGA '15)*, pp. 161–170, USA, February 2015.

[14] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proceedings of the 15th International Conference on Field-Programmable Technology, (FPT '16)*, pp. 77–84, December 2016.

[15] A. Fernandez, R. San Martin, E. Farguell, and G. E. Pazienza, "Cellular Neural Networks simulation on a parallel graphics

processing unit," in *Proceedings of the 2008 11th International Workshop on Cellular Neural Networks and Their Applications - CNNA 2008*, pp. 208–212, Santiago de Composteia, Spain, July 2008.

[16] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: a parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, Article ID 5457293, pp. 66–72, 2010.

[17] OpenCL., "The open standard for parallel programming of heterogeneous systems," *The Khronos OpenCL Working Group*, 2011.

[18] H. M. Waidyasooriya, T. Endo, M. Hariyama, and Y. Ohtera, "OpenCL-Based FPGA Accelerator for 3D FDTD with Periodic and Absorbing Boundary Conditions," *International Journal of Reconfigurable Computing*, vol. 2017, 2017.

[19] NVIDIA, "Introduction to GPU Computing with OpenCL," http://developer.download.nvidia.com/CUDA/training.

[20] D. Singh, "Implementing FPGA design with the OpenCL standard," in *Altera Whitepaper*, 2011, Implementing FPGA design with the OpenCL standard, in Altera Whitepaper.

[21] Intel., "Intel, A Development Environment for OpenCL Applications," 2012, http://software.intel.com/en-us/vcsource/tools/opencl-sdk.

[22] T. S. Czajkowski, U. Aydonat, D. Denisenko et al., "From OpenCL to high-performance hardware on FPGAs," in *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL '12)*, pp. 531–534, IEEE, August 2012.

[23] *Environment, Installation and Licensing Guide, Xilinx: sdaccel*, SDAccel Development, Xilinx, 2015.

[24] B. Chapman, G. Jost, and R. v. d. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*, The MIT Press, 2007.

[25] "Xilinx Limited, Melita House, 124 Bridge Rd, Chertsey KT16 8LA, United Kingdom. SDAccel Development Environment: User Guide, 2015.3 ed, 10, 2015".

[26] J. Cong and B. Xiao, "Artificial Neural Networks and Machine Learning," in *Proceedings of the 24th International Conference on Artificial Neural Networks, (ICANN '14)*, vol. 8681, pp. 281–290, Springer International Publishing, Hamburg, Germany.