An FPGA-Based CNN Efficient Storage Processor

IOP Conf. Series: Materials Science and Engineering **569** (2019) 032013 doi:10.1088/1757-899X/569/3/032013

An FPGA-Based CNN Efficient Storage Processor

Tong Zhao[1], Lufeng Qiao[1*] and Qinghua Chen[1]

[1]Institute of Communication Engineering, Army Engineering University of PLA, Nanjing Jiangsu 210007, China.

[*]Corresponding author's e-mail: 13357837783@189.cn

Abstract. At present, deep learning algorithms such as neural networks are widely used in various aspects of artificial intelligence. The computational performance of the CPU is low and the power consumption of the GPU is large. Therefore, this paper studies the implementation of CNN on the FPGA and proposes an effective storage management scheme, which greatly reduces the bandwidth requirement of the operation. The Winograd algorithm is used in the operation to reduce the computational complexity of the convolution, so that the performance of the FPGA is optimized. This design implements Alexnet on the Virtex7 xc7vx690t with 1.32 TFlop/s performance and an average Energy efficiency of 43.5 GOP/s/W.

1. Introduction

At present, the implementation of deep neural networks is mostly based on general-purpose computers. Not only the size of the computer is large, but also the speed of the computer's input and output interfaces limits the overall speed, the breadth and depth of the application. As reported in[1], the search time can take several days even with hundreds of GPUs. The designer of the deep neural network must have a good understanding of the working hardware and software environment of the network to apply it to the actual system, which limits the development of deep neural networks.

Therefore, if the controller can be implemented in the form of FPGA, the hardware size can be greatly reduced, and the execution have fast speed and high flexibility[2-5]. Moreover, the deep neural network will increase the user's interest due to the small in size and low power consumption of the FPGA hardware, especially in terms of learning accuracy, which is not too different due to hardware. In addition, due to the reprogrammability of FPGAs, algorithm updates can be easily implemented. But these existing design flows dedicated for CNNs are not suitable for such complicated structures [6–10, 13]. As for the scheduling on FPGA, a few of works exist in the literature[11,12,13], so the design of processor's architecture and storage management is vital on FPGA.

2. The basic architecture of processor

Due to the increasing size of the neural network and the deeper layers and the limit by the storage resources on the FPGA, the entire network must be completed by means of an external memory chip. Therefore, this design uses a combination of external DDR memory chips and operational FPGAs. The entire architecture is shown in figure 1. the architecture of the entire hardware platform consists of an operational FPGA chip and a memory chip DDR. The internal FPGA consists of io_control module and processing element (PE).

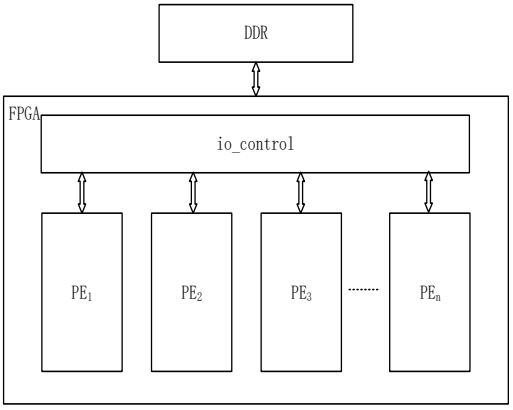

Figure 1. The basic architecture of the CNN accelerator.

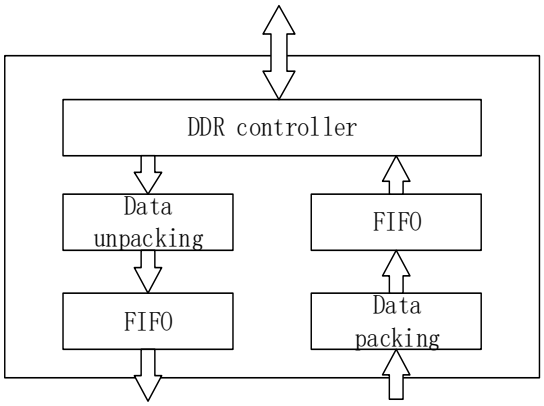

Figure 2. The architecture of the io_control module

2.1. Io_control module

The master control is mainly responsible for managing the input feature map, the weight of the network, and the intermediate results generated, and is responsible for data conversion and communication between the PE and external DDR. Its structure is shown in figure 2. As can be seen, the master control module includes a DDR controller, a data interface, a data parsing/packaging module, and a FIFO that stores intermediate results.

The DDR controller uses communication with the FPGA and the external DDR, which contains an IP core running the DDR interface protocol. After satisfying a burst condition, the internal data of the FPGA is transferred to the DDR or the data which in the DDR is transmitted to the FPGA.

2.2. PE module

For the most common network at present, the most used is the 3x3 convolution kernel, so the architecture design for PE is shown in figure 3 of which the convolution kernel is 3x3. It can be seen from the above figure that the PE is mainly composed of the memory array controller, the RAM array, and the multiply-add operation matrix. The storage array controller is responsible for the data in the RAM array, control of row and column number conversion and manage the RAM array read and write operations; RAM matrix stores corresponding eigenvalues and weights; multiply-add operation matrix is responsible for calculating the output eigenvalues and weights, then the final result has been obtained. The purpose is to adjust the distribution of PE in the whole design according to the structural characteristics of different networks, so that the performance is optimal.

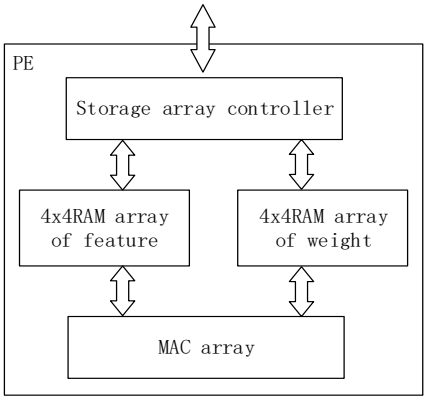

Figure 3. The architecture of PE module.

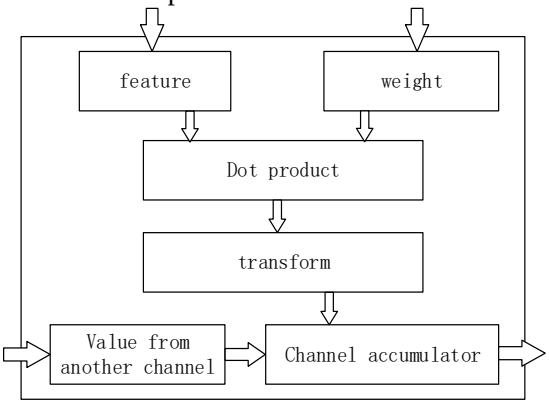

Figure 4. The architecture of compute array module

2.3. Compute array

For the 3x3 convolution of stride is 1, we use the Winograd algorithm[14] to speed up the entire calculation. The basic principle of the Winograd algorithm is to use addition instead of multiplication to reduce the amount of computation. For example, F(2×2, 3×3) use in this design requires 4×4 = 16 multiplication operations, while the standard algorithm requires 2×2×3×3 = 36 multiplications. The Winograd algorithm will make the computational complexity reduce by 36/16 = 2.25 times.

For $F(2 \times 2,3 \times 3)$, the output is

$$Y = A^T\left[[GgG^T] \odot [B^T dB]\right]A \quad (1)$$

Among the remaining parameters, g is a 3x3 filter and d is a 4x4 eigenvalue matrix. Each feature map is divided into 4x4 sub-feature matrices, and there are two eigenvalue overlaps between two adjacent sub-feature matrices.

For the characteristics of the Winograd algorithm, let $U = GgG^T$, $V = B^T dB$, then the formula becomes

$$Y = A^T[U \odot V]A \quad (2)$$

This design can convert the convolution kernel and the eigenvalues in advance before the operation starts, and then directly perform calculations inside the FPGA, which can reduce a lot of repetitive work. The structure of this module is shown in figure 4. As can be seen, the eigenvalues and weight values input to the PE are all values that are converted in advance in the software, and then the multiplication result is obtained by the dot product module, and then the converter obtains the value of the current channel. In the end, the values of the rest channels are accumulated, the final result can be obtained.

3. Storage management

Implementation of multiple computing resources in hardware is to speed up the inference process of CNN, the limitation of storage bandwidth is often the bottleneck of handling CNN[15]. For example, for a convolutional layer, a large number of multiply-and-accumulate operations inevitably result in a large amount of memory read and write, because each multiply-and-accumulate operation requires at least 2 memory read operations and 1 memory write operation. Due to limitations of FPGA internal storage resources, most network parameters and intermediate results must be stored on the external DDR, so it will seriously affect the throughput and energy consumption, even more than the energy consumption of the multiply-accumulate operation itself.

3.1. Feature and weight value storage

In response to the limitation of storage bandwidth, this design proposes a scheme that can reduce the bandwidth requirement for external storage. For the most widely used 3x3 convolution kernel, the following design is implemented. For a nxn feature map, the storage scheme as shown in figure 5.

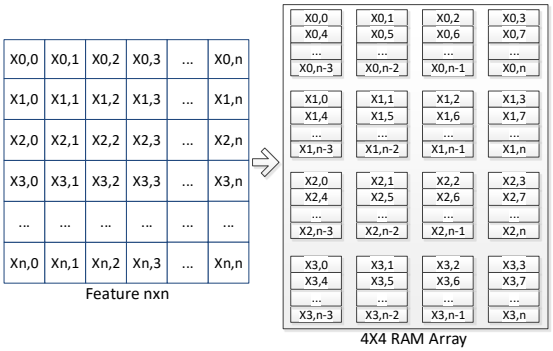

Figure 5. The feature map in 4x4 RAM array.

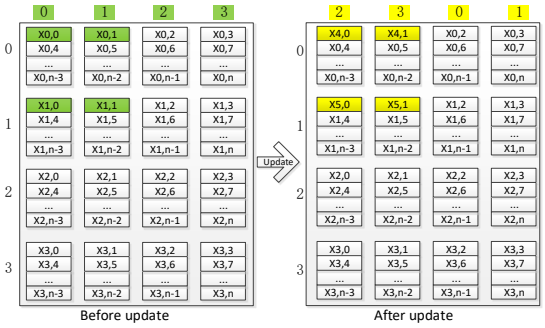

Figure 6. The feature value update and the column number change.

As shown in figure 5, the first four rows of the feature map are stored in a 4x4 RAM array before the entire operation begins. In the RAM array, each row of RAM only stores feature map which are in the same row. This allows you to get the convolved results in one shot without the need for additional operations. For the CNN parameters, the same format is also used for storage. Thus, the weight value corresponds to the feature value, and the calculation result can be obtained in one shot.

3.2. RAM array design

For each group of RAMs, the four RAMs of each row store the feature values of the same row, that is, the entire four rows of the feature map need to be stored in the slice separately, and then updated from the fifth row and the sixth row. When convolving on the same row, only the column number changes, and the row number remains unchanged. Only after the entire column ends, all the values can be updated which before the row number changes.

When the weight value which stored in the RAM array runs to the last one, the feature map is updated. Before this, the entire graph of the feature map is transferred to the on-chip FIFO for storage in the form of DMA burst, and then The FIFO extracts a feature value at a time to update the feature values in the RAM array. The specific process is shown in figure 6. It can be seen that after updating the feature value, the column number in the RAM array change, and the read pointer in the updated RAM is incremented by one which can facilitate the operation of the RAM array. For the interface, you only need to operate the RAM with the column number 0, 1, 2, 3 at a time, you can read the corresponding operand.

Each of the row and column numbers in the RAM array has a corresponding actual operation, and the corresponding specific values are shown in the following table1.

Table 1. The row and column numbers and the operation number.

Row/Column Number of RAM Array	Operation Number（Binary）
0	00011011
1	01101100
2	10110001
3	11000110

As can be seen from the operation number shown in table1. The row/column number is converted from 0, 1, 2, 3 to 2, 3, 1, 0 when the RAM update. We just ring shift 4bit left of the actual operation number which can achieve the corresponding conversion.

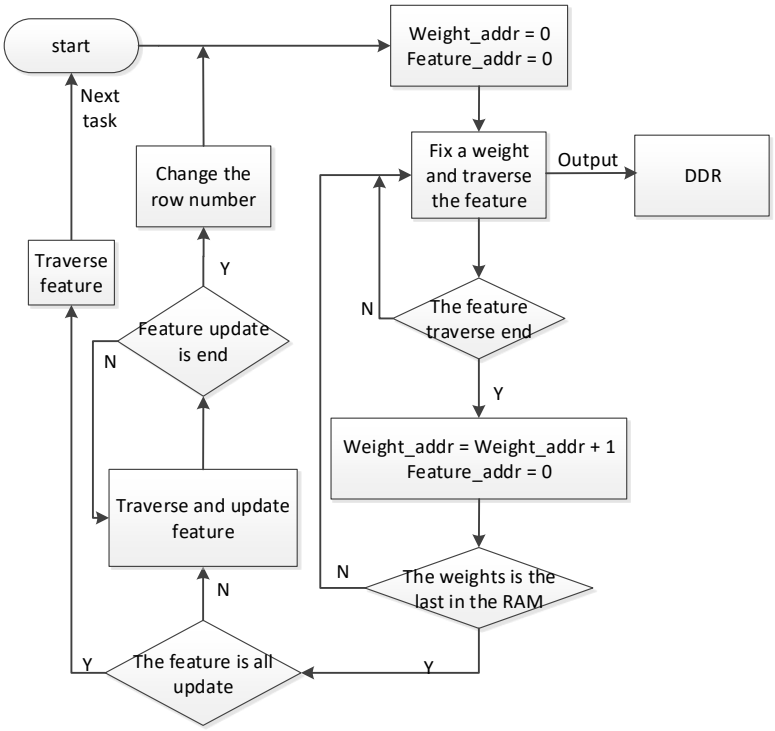

Figure 7. The control flow of storage management.

3.3. Storage management design

The design process of the storage management module is shown in figure 7. It can be seen that in the present design, a weight value is fixed after the operation starts, and then the feature map is traversed. In this way, the output of the part belongs to the same feature map, which is convenient for storing to DDR. When the RAM array of the current storage weight runs to the last value, updating the RAM array storing the feature map can ensure uninterrupted flow and can maximize the parallelization of the design.

4. Result analysis

This design is implemented on the Xilinx Virtex7 xc7vx690t platform. In this design, we use a pipeline structure, each layer is designed separately, and then unified control which can greatly improve the speed. In the design of the pipeline, the calculation amount of each layer should be kept substantially equal. Otherwise, the bottleneck of the pipeline will be generated which can reducing the efficiency of the entire pipeline.

This design takes Alexnet as an example to implement the proposed algorithm and compare it with other FPGA implementations. The results are shown in table 2.

Table 2. The performance comparison of Alexnet on different platforms.

Work	Wei et al.[16]	Avdonat et al.[17]	Liqiang Liu et al.[18]	Xushen Han et al.[19]	Our Implementation
FPGA platform	AWSF1	GX1150	ZCU102	Virtex xc7v2000	Virtex7 xc7vx690t
Precision	32bit float	32bit float	16bit fixed	32bit float	16bit float
Frequence (MHz)	230	303	200	189	170
Performance (TFlop/s)	1.88	1.38	1.28	1.24	1.35
Energy efficiency (GOP/s/W)	27.5	40.2	36.2	38.1	43.2

It can be seen from table 2 that the storage management and operation framework proposed by the scheme can achieve relatively high performance, and the energy efficiency of the design is the highest among the five design schemes listed in this table.

Table 3. The performance comparison between out implementation and GPU.

Device	TitanX	Virtex7 xc7vx690t
Precision	32bit float	16bit float
Performance(Tflop/s)	4.17	1.35
Energy efficiency (GOP/s/W)	16.9	43.2

We also compare the experimental results of the FPGA platform with the implementation results of the GPU. We use NVIDIA TitanX GPUs to implement Alexnet under the Caffe framework[20]. In the implementation of the GPU, the design is accelerated by the Winograd algorithm, and the results are shown in table 3. It can be seen that although the performance of the GPU is higher than our design when implement the same network, the energy consumption performance of the design is 2.5 times that of the GPU.

5. Conclusion

This paper studies the FPGA hardware platform based on CNN, which designs an efficient hardware platform architecture and proposes a storage management scheme that can effectively reduce the storage bandwidth. On this basis, the Winograd algorithm is used to reduce the amount of computation, and the processing modules of the convolution kernels of 3x3 is designed, so that the hardware platform can accelerate CNN more efficiently. This design implements Alexnet on the Virtex7 xc7vx690t with a performance of 1.35TFlop/s and the average energy efficiency of 43.2 GOP/s/W. The result is better than several of current popular centralized design.

References

[1] Barret ,Z, et al.(2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.
[2] Chen, Y.H., et al.(2017) Eyeriss: An energy efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127138.
[3] Venieris, S.I., et al.(2017) FPGA convnet: Automated mapping of convolutional neural networks on FPGAs. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2017, pp. 291292.
[4] Li, H., et al. (2016) A high performance FPGA-based accelerator for large-scale convolutional neural networks. Field Programmable Logic and Applications (FPL), 2016 26th International Conference on, 2016, pp. 19.
[5] Ma, Y., et al. (2017) Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks. Proceedings of the 2017 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), 2017, pp. 4554.
[6] Chung, E., et al. (2018) Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. IEEE Micro 38, 2 (2018), 8–20.
[7] Fowers, J., et al. (2018) A configurable cloud-scale DNN processor for real-time AI. In Proc. of ISCA. IEEE Press, 1–14.
[8] Jiang, W.W., et al. (2018) Heterogeneous FPGA-based Cost-Optimal Design for Timing-Constrained CNNs. IEEE TCAD.
[9] Shen, Y.M., et al. (2017) Maximizing CNN Accelerator Efficiency Through Resource Partitioning. In Proc. of ISCA. 535–547.
[10] Wei, X.C., et al. (2018) TGPA: tile-grained pipeline architecture for low latency CNN inference. In Proc. ICCAD. ACM, 58.
[11] Zhang, C., et al. (2015) Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proc. of FPGA. ACM, 161–170.

[12] Zhang, C., et al. (2016) Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster. In Proc. of ISLPED. 326–331.
[13] Zhang, X.F., et al. (2018) DNN Builder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In *Proc. ICCAD*. ACM, 56.
[14] Lavin, A., et al.(2015) Fast algorithms for convolutional neural networks. arXiv preprint arXiv:1509.09308, 2015.
[15] Nguyen et al. (2017) Double MAC: doubling the performance of convolutional neural net-works on modern fpgas. Design, Automation and Test in Europe Conference and Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017, pages 890–893, 2017.
[16] Wei, X.C., et al. (2017) Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs. In DAC 2017. ACM, 29.
[17] Aydonat, U., et al. (2017) An OpenCL (TM) Deep Learning Accelerator on Arria 10. CoRR abs/1701.03534.
[18] Lu, L.Q., et al. (2017) Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs. FCCM. IEEE, 978-1-5386-4037-1/17.
[19] Han, X.S., et al. (2017) CNN-MERP: An FPGA-Based Memory-Efficient Reconfigurable Processor for Forward and Backward Propagation of Convolutional Neural Networks. arXiv preprint arXiv:1703. 07348.
[20] Jia, Y.Q., et al. (2014) Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv: 1408. 5093.