A Dynamic Mapping Model for General CNN Accelerator Based on FPGA

Xiaoqiang Zhao, Jingfei Jiang, Zhe Han, Jinwei Xu, Zhiqiang Liu

A Dynamic Mapping Model for General CNN Accelerator Based on FPGA*

Xiaoqiang Zhao[1], Jingfei Jiang[1], Zhe Han[1], Jinwei Xu[1], and Zhiqiang Liu[2]

[1] National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha, China
[2] Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, Beijing, China
{zhaoxiaoqiang18,jingfeijiang,hanzhe18,xujinwei13,liuzhiqiang}@nudt.edu.cn

Abstract. As the application scenarios of convolutional neural network(CNN) become more and more complex, the general CNN accelerator based on matrix multiplication has become a new research focus. The existing mapping methods for converting convolution calculation into matrix multiplication need to be improved. This paper proposes a new dynamic mapping model to improve the flexibility and versatility of matrix multiplication. The dynamic mapping model implements two algorithms: dynamic residue processing mapping algorithm(DRPMA) and dilated convolution mapping algorithm(DCMA). The former can dynamically adjust the mapping method according to the number of output channels of the convolution layer, improve the utilization of the multiply-accumulate(MAC) array. The latter extends the efficient support for Dilated CNNs. For demonstration, we implement an accelerator with Verilog on Xilinx VC709 FPGA board and test some typical CNN models. Experimental results show that the general accelerator achieves high performance and energy efficiency.

Keywords: CNN · Matrix multiplication · Dynamic mapping model · FPGA.

1 Introduction

In recent years, CNNs have become one of the most popular models in the artificial intelligence and shown excellent results in many fields including image classification [6, 15], object recognition [3, 13], video analysis [14, 19], voice recognition [4, 1]. With the widespread application of CNNs, FPGA-based CNN accelerators [10, 12, 16, 21, 8, 20, 11, 17, 7, 9, 5, 18, 2]have become a new research focus. However, the application scenarios of CNNs have become more and more complex. Deep CNNs have appeared to improve the inference accuracy, Dilated

* Supported by National Science and Technology Major Projects on Core Electronic Devices,High-End Generic Chips and Basic Software under grant No. 2018ZX01028101 and National Natural Science Foundation of China Key Program No. 61732018.

CNNs have been used in image segmentation, semantic segmentation to enlarger the receptive field. Three-dimensional CNNs have been applied to video analysis. Researchers need to balance the versatility and performance of accelerators to adapt to complex application scenarios. The increase of CNN parameters and intermediate results will exceed the storage capacity on the FPGA chip, making the accelerator design lose the task-level and layer-level parallelism. It is necessary to improve the loop-level and operation-level parallelism to increase accelerator performance. And the application of dilated CNNs requires a more flexible accelerator architecture to support dilated convolution. In order to enable the CNN accelerator to be applied to more scenarios, we need a flexible data buffering scheme. The data buffering scheme should handle networks with different parameters, support more convolution types, and can dynamically adjust the data mapping method according network parameters to improve accelerator performance. In this work, we are motivated to present a new dynamic mapping model based on general matrix multiplication. Our contributions are shown as follows:

1. We propose a new dynamic mapping model, combining the DRPMA and DCMA, which greatly improves the flexibility and versatility of general matrix multiplication.

2. We provide a uniform general accelerator architecture for two-dimensional, three-dimensional and dilated CNNs with dynamic mapping model. The accelerator can dynamically adapt to different computing modes without reconfiguration. The convolutional layer segmentation strategy is introduced to enable the accelerator to handle CNN-base AI applications of large-scale dimensions. It achieves high performance with smaller storage and bandwidth resources and can be ASIC.

3. We implement a RISC-V+CNN heterogeneous system based on the FPGA platform, Experiments show that the utilization of the MAC array is significantly improved, the dilated convolution can be performed efficiently, and achieves an overall throughput of 329.3 GFLOP/s on VGG16 and 354.4GFLOP/s on Resnet18 respectively.

2 Related Work

At present, CNN accelerators based on FPGA are mainly divided into the following four types according to the acceleration methods. The first type is general matrix multiplication CNN accelerators [10, 12, 16]. [10]designed a 2D/3D general reconfigurable convolutional neural network accelerator. [12]designed a maximize resource utilization CNN accelerator.The second type is Fast Fourier Transform(FFT) CNN accelerator [21, 8, 20]. [21] designed a highly parallel 2D FFT CNN accelerator using FFT and Concatenate-and-Pad technique to reduce convolutional redundancy calculations. [8]designed a deep CNN accelerator using embedded FFT. The third type is Winograd CNN accelerator [11, 17, 7].Winograd fast algorithm maps feature to specific domains to reduce the complexity of the algorithm. [11]designed a sparse and effective Winograd CNN Accelera-

tor. [17]designed a Winograd CNN accelerator that adapts to large steps. The fourth type is operator customized CNN accelerator [9, 5, 18]. [9]designed a layer pipeline optimized CNN accelerator. [5]designed a zero weight/activation-aware CNN accelerator. To summarize, the general matrix multiplication accelerator mapping convolutions to matrix multiplications, which has a good versatility. The FFT acceleration method transforms spatial domain convolution operation into frequency domain multiplication operation, which reduces the complexity of the algorithm and is proved to be more effective for the large convolution kernel. The Winograd acceleration method uses the addition operation to replace the multiplication operation through the linear mapping, which reduces the complexity of the algorithm, and is mainly suitable for the convolution stride is 1 and the transform matrices vary with the size of convolution kernels. The customized operator accelerator is optimized according to the algorithm characteristics, which fully exploits the parallelism of algorithms and has high performance.

These four types of CNN accelerators reflect different design ideas, each has its own advantages and complements each other. To summarize, there is still a large space for exploration in the design of accelerators for CNN. Different design concepts make designers adopt different acceleration methods. Aims to quickly respond to the changes in the CNN structures and the iterative speed of artificial intelligence algorithms. We adopt general matrix multiplication method with a new dynamic mapping model, which fully explores the loop levels parallelism and support for dilated convolution.

3 CNN Basics and Matrix Multiplication

This section will introduce the operation characteristics of different convolution types, the method of mapping convolution to matrix multiplication, and analyze their common characteristics and the existing optimization space.

3.1 2D and 3D Convolution

Figure 1a illustrates the process of 2D convolution. The convolution window slides along the column and row directions of the image to extract the spatial information of the image. The input and output feature usually contains multiple channels. The convolution results of each input channel are then accumulated resulting in one channel of the output feature. And the process of calculating other output channels is similar. Figure 1b shows the process of 3D convolution. Compared with the process of 2D convolution, in addition to sliding along the row and column directions, 3D convolution also slides along the temporal direction. In the 3D CNN adds a dimension of L, which represents the convolution depth in the temporal dimension. [10]indicate that 3D convolutions can be computed in the same way as 2D convolutions by combining the accumulation of the channel loop and the temporal loop. We also adopt this method in our implementation. Figure 1c-e illustrate how the convolution window slides across rows of the input feature. The pixels in gray are involved in convolutions along rows

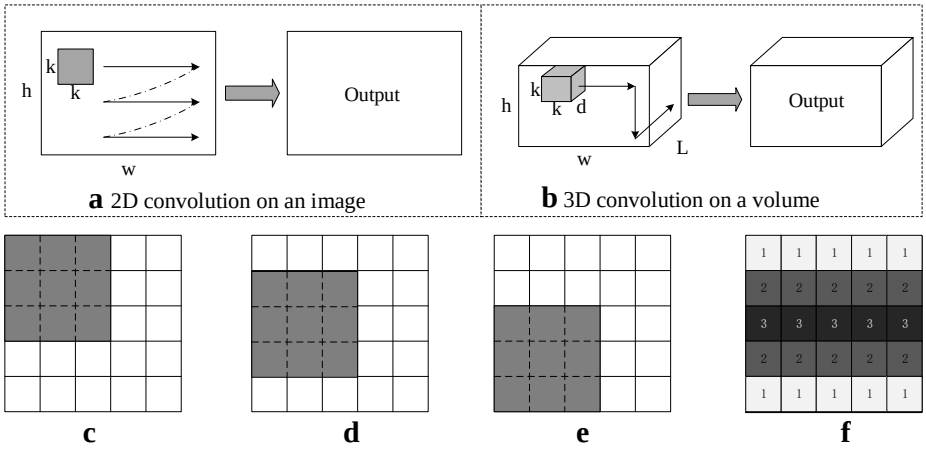

Fig. 1. 2D and 3D convolution operations.

are shown as figure 1f. Accordingly, the first row appears in the input feature 1 time, the second row appears in the input feature 2 times, and the third row appears in the input feature 3 times. The re-usability in the row direction of the sliding window can be used to improve the parallelism of matrix multiplication.

3.2 Dilated Convolution

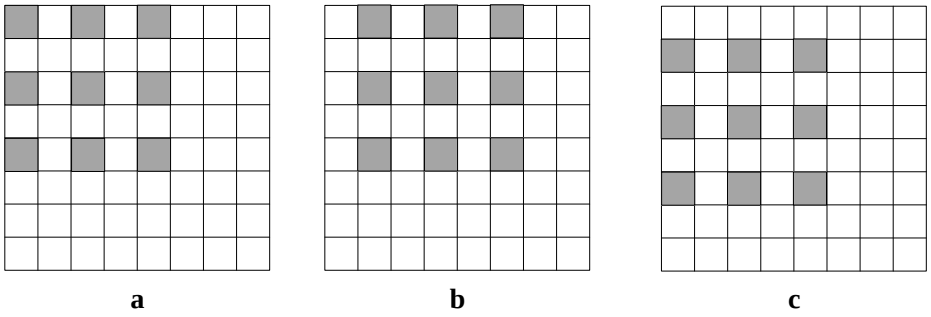

Fig. 2. Dilated convolution operations.

Figure 2 illustrates the process of the dilated convolution. The kernel size is 3x3 and the rate is 2. Figure 2a to figure 2b illustrate how the convolution window slides across column of the input feature. Figure 2a to figure 2c illustrate how the convolution window slides across row of the input feature. Compared to convolution, the dilated convolution changes the features covered by the convolution window. As shown in the shaded part of figure 2, the feature extraction is separated by rate-1 in both the row and column directions. We only need to load the feature according to this pattern, and then dilated convolution and 2D/3D convolution can use the same computing architecture.

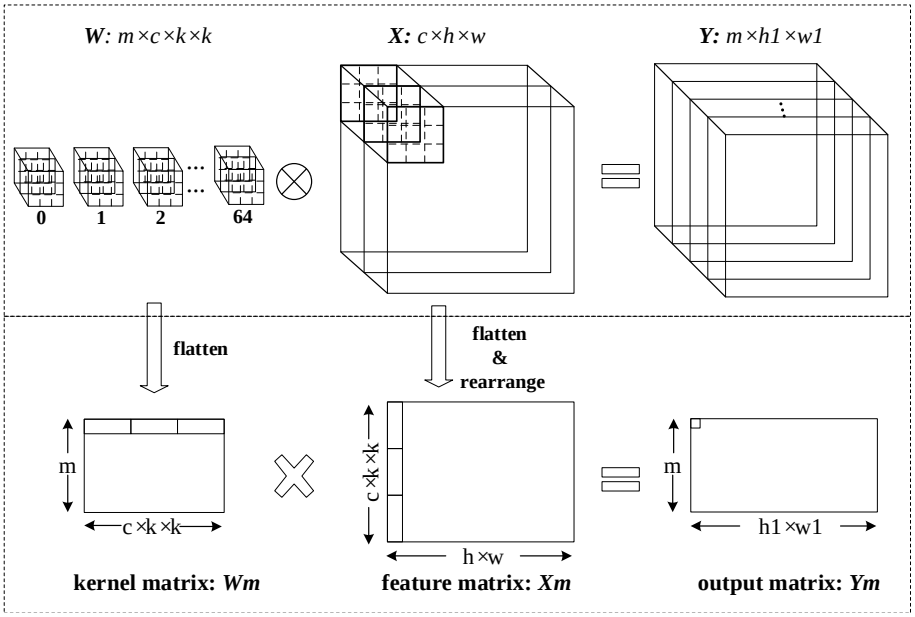

Fig. 3. Mapping convolutions to matrix multiplication operations.

3.3 Mapping convolutions to Matrix Multiplications

Mapping convolution to matrix multiplication is an efficient implementation on FPGA. As illustrated in Figure 3, the kernel of the convolutional layer is W with dimensions of $m \times c \times k \times k$, the input feature of X with dimensions of $c \times h \times w$, and W and X are convolved to obtain the output feature Y with dimensions of $m \times h1 \times w1$. Where m is the size of output channels, c is the size of input channels, k is the size of the convolution kernel, h is the size of input feature height, w is the size of input feature width, $h1$ is the size of output feature height, $w1$ is the size of the output feature width. The matrix multiplication method of convolution operation compresses the weight W into a weight matrix Wm, compresses and reorganizes the feature map X into a feature map matrix Xm. The result of the matrix multiplication is an output matrix Ym with dimensions of $m \times (h1 \times w1)$, which is the flattened format of the output feature. Generally, h is equal to $h1$ and w is equal to $w1$, but dilated convolution combines convolution and downsampling, thus $h1$ and $w1$ will become smaller than h and w. In addition, Xm is almost $(k \times k)$-fold of X because of data replications during mapping. It can be avoided by reusing the overlapped data during the sliding of convolutional windows.

4 Accelerator Architecture Design

As illustrated in figure 4, the accelerator architecture has two-level on-chip caches for the dynamic mapping model. The first level cache is Feature Buffer, Kernel Buffer and Output Buffer. The second level cache is Feature FIFO and Kernel

FIFO. The dynamic mapping model includes three sub-mapping models: Feature Mapping Module, Kernel Mapping Module, and Output Mapping Module. The following of this section will introduce the MAC Array and Buffer setting and the detailed mapping process of dynamic mapping model.

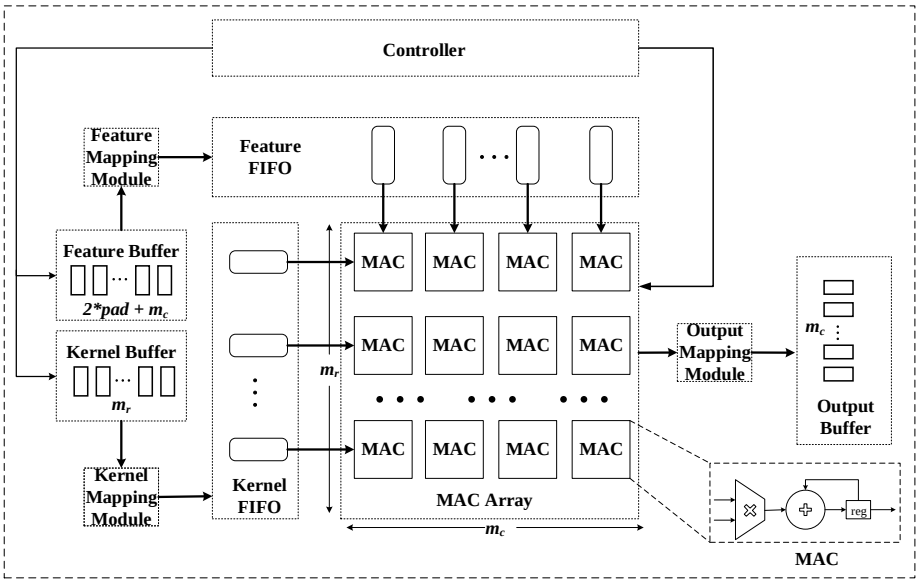

Fig. 4. Accelerator architecture with dynamic mapping model.

4.1 MAC Array and Buffer Setting

The MAC array is computing resource for matrix multiplication. As shown in figure 4, the MAC array includes $m_r \times m_c$ MAC units, where m_r and m_c are the size MAC array row and column respectively. The MAC unit consists of a multiplier, an adder, and a register. The register is used to store intermediate result. The MAC array taps the two loop parallelism of the output channel and the output feature column direction. Each MAC unit is responsible for calculating the matrix multiplication result of the corresponding position. Therefore, the MAC array can calculate $m_r \times m_c$ elements in parallel. When the feature matrix and the kernel matrix is relatively large, the matrix can be divided into blocks. The kernel matrix is divided into $\lceil \frac{m}{m_r} \rceil$ blocks and the feature matrix is divided into $\lceil \frac{h1 \times w1}{m_c} \rceil$ blocks.

Limited by the chip area, the depth of the Kernel Buffer is limited. The Kernel Buffer depth required to load a complete convolutional layer is $d \times c \times k \times k$. When the input channel or the convolution kernel is large, or the temporal dimension parameter of the 3D network is large, the storage space that the layer needs will exceed the depth of the Kernel Buffer. We need to split a large convolution layer into the sum of multiple small convolution layers. Similarly, the depth of the Feature Buffer is also limited. The Feature Buffer depth required to load $k + stride$ rows feature is $d \times c \times (k + stride) \times \lceil \frac{w}{m_c} \rceil$, When the convolutional

layer parameters are too large, it is also necessary to load features in blocks. Although the additional summation operation to be introduced may affect the performance of the accelerator, it ensures the versatility of the accelerator.

4.2 Dynamic Mapping Model

DRPMA Matrix multiplication exploits the two loop level parallelism of output channel and output feature column direction. In CNN, the numbers of output channels of different convolutional layers vary greatly. For example, the output channel numbers of VGG16 are 64, 128, 256, 512. When the MAC array are designed with the dimensions of 256×32, the residue between the output channels and the MAC rows are 64, 128, 0 and 0. We cannot make full use of the MAC array with residue of 64 and 128. Through analysis, we find that the reusability in the row direction of the sliding window can be used to improve the parallelism of matrix multiplication when a residue is generated. As the residue changes dynamically with the number of output channels of the convolutional layer. We name the method of dynamic mapping based on residue as DRPMA. Assuming the residue is R, the parallelism of matrix multiplication can be improved by DRPMA is n, then the parallelism can be defined by the following formula:

$$as \quad \frac{m_r}{R} \geq 2, \quad n = \lfloor \frac{m_r}{R} \rfloor, \ 1 \leq n \leq k \tag{1}$$

$$as \quad 1 \leq \frac{m_r}{R} \leq 2, \quad n = 1 \tag{2}$$

DCMA Dilated convolution extract the feature separated by $rate - 1$ in both the row and column directions that the reusability in row direction has lost, which means $n = 1$. The data reuse of dilated convolution in the column direction is presented every $rate - 1$, It is same as convolution when $rate = 1$. We can load the feature step by $rate - 1$ offset, the reusability of convolution and dilated convolution in the column direction can be utilized through a consistent pattern. we name the mapping method of reuse data in the column direction both for convolution and dilated convolution as DCMA. With the two algorithms of DRPMA and DCMA, We can efficiently map convolution and dilated to matrix multiplication. The following of this section will introduce the details of mapping methods.

As shown in figure 4, the Feature Mapping Module is connected to the Feature Buffer and Feature FIFO, and is used to map the feature in the $2 \times pad + m_c$ Block RAMs to m_c feature FIFOs. In order to save on-chip memory, the Feature Buffer only stores the $k+stride$ rows data of the input feature. When performing matrix multiplication, the Feature Buffer will prefetch the stride rows data from the external chip. The complete mapping process is as algorithm 1.

As shown in figure 4, the Kernel Mapping Module is connected to the Kernel Buffer and the Kernel FIFO, and is used to map the kernel from the m_r kernel Block RAMs to the m_r kernel FIFOs. In order to save on-chip memory, the Kernel Buffer only stores the m_r rows data in the kernel matrix. The complete mapping process is as algorithm 2.

As shown in figure 4, the Output Mapping Module is connected to the MAC array and Output Buffer, and is used to control the way of convolution results to Output Buffer. The Output Buffer adopts the Ping-Pong mechanism. The complete mapping process is as algorithm 3.

Algorithm 1 Generate Feature Matrix

Input: X
Output: Xm
for $i = 1, 2, \ldots, k + stride$
for $j = 1, 2, \ldots, d \times c$
for $t = 1, 2, \ldots, w$
do store in $2 \times pad + m_c$ **feature buffer**
for $i = 1, 2, \ldots, k \times d \times c$
for $j = 0, rate, \ldots, (k-1) \times rate$
do send to m_c **feature fifo**

Algorithm 2 Generate Kernel Matrix

Input: W
Output: Wm
for $i = 1, 2, \ldots, m_r$
for $j = 1, 2, \ldots, k$
for $t = 1, 2, \ldots, d \times c$
for $q = 1, 2, \ldots, k$
do store in the i_{th} **kernel buffer**
if $n == 1$
for $i = 1, 2, \ldots, k$
do store in the i_{th} **kernel fifo**
else
for $j = 1, 2, \ldots, n$
for $t = 1, 2, \ldots, m$
do store in the $(n \times m + t)_{th}$ **kernel fifo**

Algorithm 3 Dynamic output mapping algorithm

Input: Yi, The i_{th} results of them_c columns
Output: Ym
if $n == 1$
for $i = 1, 2, \ldots, m_r$
do store thei_{th} ***Yi*** **in the** m_c **column output buffer**
else
for $j = 1, 2, \ldots, n$
for $i = 1, 2, \ldots, m$
do store the t_{th} ***Yi*** **in the** m_c **column output buffer**
wait $k \times \ d \times c$ **cycles**
do store the $(n \times m + t)_{th}$ ***Yi*** **in the** m_c **column output buffer**

5 Evaluation

5.1 Experimental Setup

We evaluate the effectiveness of the dynamic mapping model by implementing a RISC-V+CNN heterogeneous system prototype on the Xilinx VC709 FPGA platform with 3600 DSP, 1470 BRAM, and two on-chip DDR. We assign 128 rows and 16 columns for the MAC array. Accordingly, the kernel buffer has 128 Block RAMs with depth of 1024, the feature buffer has 26 Block RAMs with depth of 4096, and the Ping-Pong output buffer has 32 Block RAMs with depth of 1024. The kernel FIFO and feature FIFO have 144 Block RAMs with depth of 512. Three typical CNNs: VGG16, C3D, and Resnet18 are tested on heterogeneous system. The CNN parameters and feature are 16-bit floating point. The software versions are implemented with Caffe on Intel Core i7-4790K CPU@ 4.0GHz. The clock frequencies of the RISC-V CPU and CNN are 20MHz and 100MHz respectively.

5.2 Experimental Results

Table 1 reports the hardware resource utilization of our heterogeneous system. The RISC-V and CNN accelerator consume most of the hardware resources. RISC-V uses 290 Block RAMs to support for large-capacity data cache and instruction cache, which can adapt to complex application scenarios. The CNN accelerator consumes 382 Block RAMs for data buffering, and the heterogeneous system uses 676 Block RAMs totally. The MAC array consumes 2048 of the 2080 DSP slices for computing matrix multiplications and the other 32 DSP slices are used for building floating-point arithmetic units.

Table 1. Heterogeneous system resource utilization.

Module	DSP		BRAM		LUT	
RISC-V	32	0.8%	290	19.7%	57078	13%
CNN	2048	56.9%	382	26.0%	232752	54%
Sum	2080	57.7%	676	46.0%	307572	71%

The experiment selects VGG16 to test the effectiveness of the DRPMA. Table 2 presents the evaluation results of three layers in VGG16. The residues of conv1a, conv1b, conv2a are 64, 64, and 0 respectively. The MAC array utilization and throughput are 83.7% and 342.8GFLOP/s on conv1a and 95.3% and 390.3GFLOP/s on conv1b with DRPMA. Without the DRPMA, the MAC array utilization and throughput are 42.5% on conv1a and 48.1% and on conv1b. The MAC array utilization and throughput are the same on conv2a as the R is 0. We can conclude that the DRPMA almost increases 2x utilization of MAC array. Additionally, we can reduce MAC array column width to reduce the memory access bandwidth with DRPMA.

Table 2. DRPMA performance.

Layer	conv1a		conv1b		conv2a	
R	64		64		0	
DRPMA	83.7%	342.8	95.3%	390.3	94.2%	385.8
No	42.5%	174.1	48.1%	197.0	94.2%	385.8

The experiment also selects VGG16 to test the performance of the DCMA. We uses the Hybrid Dilated Convolution method from [1] to set rate group as 1, 2, 5. Table 3 presents the evaluation results. When the *rate* is 1, the throughput remains unchanged. When the *rate* is 2, the MAC array utilization is 89.7%, a decrease of 5%. This is caused by the load feature time increase and the idle state of the MAC array increase. When the *rate* is 5, the utilization of the MAC array is reduced by 23%. The main reason is that the feature Block RAMs we set is 26. And the Block RAMs used for padding is 10, which supports the maximum convolution kernel of the same size output is 11. The dilated convolution with a *rate* of 5 is essentially equivalent to the standard convolution with a convolution kernel of 14. This means that only the first 13 columns of the MAC array can be used. Therefore, the utilization rate of MAC array decreases sharply. In conclusion, the DCMA can efficiently support dilated convolution.

Table 3. DCMA performance.

rate	1	2	5
Utilization(%)	94.2%	89.7%	71.6%
Throughput(GFLOP/S)	385.8	367.4	293.3

Table 4 lists the comparisons between the CPU and heterogeneous system. The heterogeneous system achieves a 5.1x and 5.3x and 24.2x and 32.6x improvement than CPU in VGG16 and Resnet18 in terms of throughput and energy efficiency respectively. After ASIC, the energy efficiency advantage will be more obvious.

Table 4. Evaluation results on the CPU and our accelerator.

CNN Model	VGG16		Resnet18	
	CPU	FPGA	CPU	FPGA
Power(W)	88	13.6	88	13.6
ThroughputGFLOP/S	63.9	329.3	66.5	354.4
Energy Efficiency(GFLOP/s/W)	0.7	24.2	0.8	26.1

Table 5 shows the comparisons with other FPGA platforms. Our heterogeneous system achieves an overall throughput of 329.3 GFLOP/s on VGG16 and 310.2GFLOP/s on Resnet18 respectively. The throughput of the heterogeneous system is better than [16] on VGG16 and lower than [10] on C3D. The usage of DSP slices determines the throughput to a certain extent. [10] high throughput

is based on larger MAC array, high on-chip buffering and high bandwidth. When heterogeneous system runs C3D, the convolutional layer will be split which affects the performance. Our throughput will increase 4x after ASIC which is the best. It needs to be emphasized that our heterogeneous system is a better choice after balancing price, versatility and performance.

Table 5. Comparisons with previous accelerator implementations.

	[16]	[10]	ours	
FPGA	Altera StratixV	Xilinx VX690T	Xilinx VX690T	
Precision	fixed	fixed	float16	
CNN Model	VGG16	C3D	VGG16	C3D
Clock(MHz)	120	120	100	
DSPs	727	3595	2080	
Throughput(GOP/S)	118	667.7	329.3	310.2

6 Conclusions

This paper studies the mapping methods that convert convolution into matrix multiplication, and proposes a new dynamic mapping model, which improves the flexibility and versatility of matrix multiplication. The heterogeneous system prototype based on FPGA platform verifies the performance of the dynamic mapping model. And it can be applied to more complex artificial intelligence scenarios without reconfiguring the FPGA. Future work include demonstrations on dilated Resnet applications and efficient support of transposed convolution.

References

1. Abdel-Hamid, O., Mohamed, A.r., Jiang, H., Deng, L., Penn, G., Yu, D.: Convolutional neural networks for speech recognition. IEEE/ACM Transactions on audio, speech, and language processing **22**(10), 1533–1545 (2014)
2. Azizimazreah, A., Chen, L.: Shortcut mining: Exploiting cross-layer shortcut reuse in dcnn accelerators. In: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 94–105. IEEE (2019)
3. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 580–587 (2014)
4. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal processing magazine **29**(6), 82–97 (2012)
5. Kim, D., Ahn, J., Yoo, S.: A novel zero weight/activation-aware hardware architecture of convolutional neural network. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. pp. 1462–1467. IEEE (2017)

6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
7. Lavin, A., Gray, S.: Fast algorithms for convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4013–4021 (2016)
8. Lin, S., Liu, N., Nazemi, M., Li, H., Ding, C., Wang, Y., Pedram, M.: Fft-based deep learning deployment in embedded systems. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1045–1050. IEEE (2018)
9. Liu, X., Kim, D.H., Wu, C., Chen, O.: Resource and data optimization for hardware implementation of deep neural networks targeting fpga-based edge devices. In: 2018 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP). pp. 1–8. IEEE (2018)
10. Liu, Z., Chow, P., Xu, J., Jiang, J., Dou, Y., Zhou, J.: A uniform architecture design for accelerating 2d and 3d cnns on fpgas. Electronics **8**(1), 65 (2019)
11. Lu, L., Liang, Y.: Spwa: An efficient sparse winograd convolutional neural networks accelerator on fpgas. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2018)
12. Ma, Y., Cao, Y., Vrudhula, S., Seo, J.s.: Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 45–54 (2017)
13. Nair, V., Hinton, G.E.: 3d object recognition with deep belief nets. In: Advances in neural information processing systems. pp. 1339–1347 (2009)
14. Ramanishka, V., Das, A., Park, D.H., Venugopalan, S., Hendricks, L.A., Rohrbach, M., Saenko, K.: Multimodal video description. In: Proceedings of the 24th ACM international conference on Multimedia. pp. 1092–1096 (2016)
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
16. Suda, N., Chandra, V., Dasika, G., Mohanty, A., Ma, Y., Vrudhula, S., Seo, J.s., Cao, Y.: Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 16–25 (2016)
17. Wu, D., Chen, J., Cao, W., Wang, L.: A novel low-communication energy-efficient reconfigurable cnn acceleration architecture. In: 2018 28th International Conference on Field Programmable Logic and Applications (FPL). pp. 64–643. IEEE (2018)
18. Wu, E., Zhang, X., Berman, D., Cho, I.: A high-throughput reconfigurable processing array for neural networks. In: 2017 27th International Conference on Field Programmable Logic and Applications (FPL). pp. 1–4. IEEE (2017)
19. Yang, K., Qiao, P., Li, D., Lv, S., Dou, Y.: Exploring temporal preservation networks for precise temporal action localization. arXiv preprint arXiv:1708.03280 (2017)
20. Zeng, H., Chen, R., Zhang, C., Prasanna, V.: A framework for generating high throughput cnn implementations on fpgas. In: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 117–126 (2018)
21. Zhang, C., Prasanna, V.: Frequency domain acceleration of convolutional neural networks on cpu-fpga shared memory system. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 35–44 (2017)