F-C3D: FPGA-based 3-Dimensional Convolutional Neural Network

Hongxiang Fan*, Xinyu Niu†, Qiang liu*, Wayne Luk†
*School of Microelectronics, Tianjin University, China
†Dept. of Computing, School of Engineering, Imperial College London, UK
Email: {AUSTIN_fan, qiangliu}@tju.edu.cn*, {nx210, w.luk}@doc.ic.ac.uk†

***Abstract*—In recent years, 3-dimension convolutional neural networks (3D CNNs) have been widely used for video analysis, 3-dimension geometric data and medical image diagnosis. While conventional CNNs are computationally intensive, 3D CNNs push the computational requirements into another level, since each computation depends on multiple image frames. This paper describes a novel hardware architecture for a 3D convolutional neural network, and design strategies to resolve memory usage and bandwidth limitations. The proposed architecture F-C3D is implemented on zc706 at 172MHz, showing 231 times speed up compared with software implementation on 1 GHz ARM CPU, 7.4 times speed up on 3.07 GHz Intel CPU and nearly 10 times lower power consumption than GPU.**

I. Introduction

As deep learning has demonstrated its exciting potential, FPGA-based accelerator of deep learning algorithms such as 2D CNNs has become the hot topic in FPGA filed. In recent years, various FPGA-based implementations of 2D CNNs [1][2] and related optimization strategies [3] have been proposed. However, 2D CNNs are not directly suitable for 3-dimension data analysis since it can only extract information from two dimensions.

Recently, a novel 3D CNN algorithm with the ability in extracting the third dimension information has been proposed in [4]. Subsequently, various 3D CNNs have been implemented in CPUs and GPUs respectively, which were used for human action recognition in video [5], real-time object recognition in volumetric data [6] and disease detection in medical images [7].

The main difference between 3D CNNs and 2D CNNs lies in convolution layer. A comparison between 2D convolution and 3D convolution is illustrated in Figure 1. 3D convolution layer computes features from contiguous frames, which thereby can capture motion information along the temporal dimension. The pseudo code of 3D convolution is given in Figure 2. Table I summaries all the parameters used in this paper. As shown in the pseudo code, comparing with 2D convolution, the innermost loop i is the extra loop to compute the third dimension information, which means that the computation of 3D convolution is S times more than 2D convolution.

There are two main challenges to implement 3D CNNs on FPGAs:

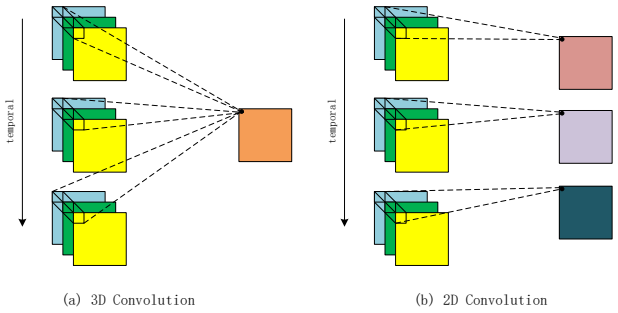

Fig. 1: Comparison of 3D convolution and 2D convolution

TABLE I: Parameters in 3D CNNs

Parameter	Description
H	the height of input feature map
W	the width of input feature map
K_c	the kernel size of 3D convolution
K_p	the kernel size of 3D pooling
S	the stride of 3D convolution kernel
N_c	the number of channels
N_f	the number of filters
N_l	the number of frames

(1)With more nested loops, 3D CNN algorithms are more complicated than 2D CNN algorithms. Thus, it is difficult to determine the execution sequence of different loops in order to improve data locality, which could effectively remove the off-chip data transfer bottleneck.

(2)With the large number of layers and computation, 3D CNN requires more on-chip memory and computational

```
for ( channels = 0 ; channels < Nc; channels ++ ) {
    for ( filters = 0 ; filters < Nf; filters ++ ) {
        for ( frames = 0 ; frames < Nl; frames++ ) {
            for( i = frames; i < S; i ++ ){
                        Output_fm[filters][channels][frames]+=
                        Coefficient[filters][channels]*
                        Input_fm[channels][i]
}   }   }   }
```

Fig. 2: Psuedo code of 3D convolution

resorces. It is far more difficult to implement each layer of 3D CNN separately on one FPGA chip than 2D CNN[2].

This paper presents an FPGA-based implementation of a 3D CNN called C3D [5]. To our best knowledge, this is the first attempt to implement 3D CNN on FPGA. The implementation is applied to video human action recognition for demonstration. The main contributions of our work are:

- A novel hardware architecture called F-C3D, which is reconfigurable for different 3D convolution layers in 3D CNNs.
- Design strategies to resolve the design difficulties in hardware implementation with limited resource and bandwidth, which makes our design portable to different FPGAs.

This paper is organized as follows. Section II presents hardware architecture of 3D convolution and related design strategies. Section III presents FPGA-based 3D CNN system and experiment results. The conclusion and future work are given in Section IV.

II. Hardware Architecture and Design Strategy

In this section, we first present the hardware architecture of F-C3D. Based on the architecture, several design strategies for F-C3D are proposed to resolve memory and bandwidth limitations.

A. Hardware Architecture

A classical CNN mainly includes convolution, fully connected (fc) and pooling layers. In 2D CNNs, a previous study [8] showed that convolution operations occupy over 90% of the computation time. The same phenomenon also appears in 3D CNNs. Hence, in this paper, we mainly focus on designing optimized hardware architecture for 3D convolution. For fc layer, we adopt the architecture proposed for 2D CNNs [3]. In F-C3D, only one 3D convolution hardware module and one fc hardware module are implemented on the FPGA. Both modules are reconfigured to realize different layers.

Figure 3 illustrates the architecture of the 3D convolution module when K_c = 3. The architecture contains three 2D convolution modules, several buffers and a 3D pooling module. Each 2D convolution module is composed of K_c^2 multipliers and a pipelined adder tree with $\log(K_c)$ levels.

Three types of buffers are designed to make the 3D convolution a streaming architecture. The line buffer receives 1 pixel and outputs K_c pixels in parallel. Then the K_c pixels stream into the frame buffer and the matrix buffer respectively. Since 3D convolution kernel needs to accumulate K_c contiguous output frames together, $K_c - 1$ frame buffers are added in the design to cache $K_c - 1$ contiguous input frames. For example, in Figure 3, when input frame i is processed in kernel 1.3, the cached input frame $i-1$ and $i-2$ will also stream from the frame buffers to kernel 1.1 and kernel 1.2 respectively. Therefore, three contiguous output frames are generated from three 2D kernels and accumulated together at the same time.

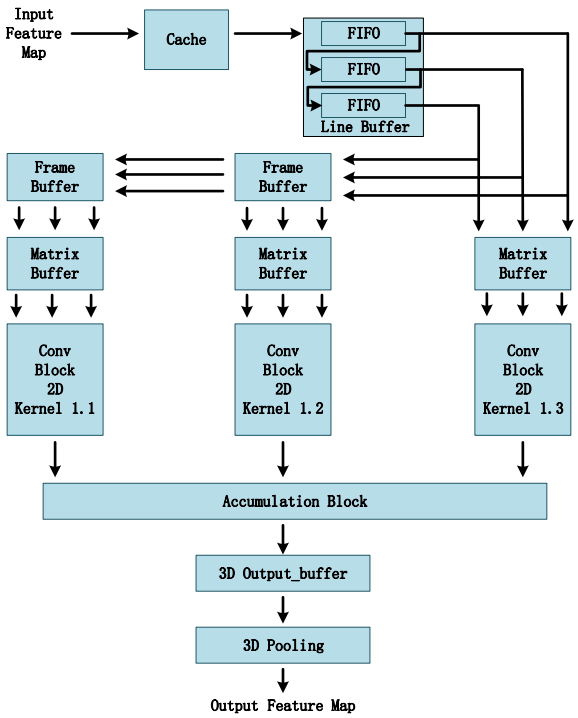

Fig. 3: Hardware architecture for 3D covolution layer

Then matrix buffer takes 3 pixels as input and outputs K_c * K_c pixels at one time. Therefore, each 2D kernel can perform convolution for every K_c * K_c input vector.

B. Design Strategy

The hardware architecture of 3D CNNs needs K_c 2D convolutional kernels and large amount of on-chip memories to buffer input data of different frames. As a result, F-C3D is more resource-sensitive.

1) Input Cache and 3D Blocking: Due to the limited bandwidth between on-chip memory and off-chip memory, we propose input cache strategy to reuse the input data. As shown in Figure 2. Each input feature map with N_c channels needs to be calculted N_f times. With the input cache strategy, we cache the input feature map on chip until the calculation of the N_f filters finishes.

However, one output feature map needs $S * N_c$ input feature maps. Due to limited on-chip memory, it is hard to cache all the input data needed for one output feature map on chip. Hence, we propose 3D blocking to resolve this issue.

There are two methods to implement blocking operation in 3D CNNs, frame blocking and pixel blocking. Frame blocking divides input data among frames and keeps each frame as original size. Pixel blocking divides each frame into squares with equal size.

As shown in Figure 4, both methods have overhead in input data transfer. In frame blocking, if the first block contains $1st$ to

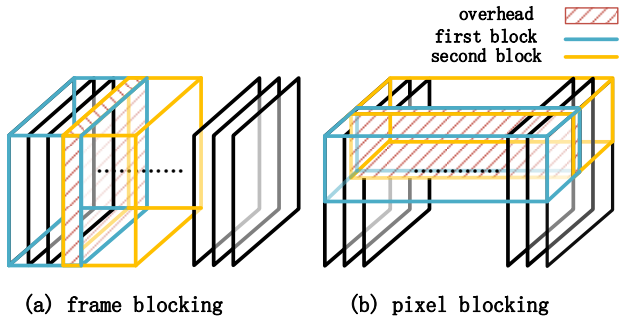

Fig. 4: Frame blocking and pixel blocking

Nth frames, the second block should include $(N-K_c+1)th$ to $(2N-K_c+1)th$ frames. Hence, the overhead is K_c-1 frames. Assume that the cache size is C, each block contains $C/(N_c*H*W)$ frames. The overhead of each block is

$$(K_c-1)*N_c*H*W/C \tag{1}$$

As shown in Figure 4 (b), the second block contains the pixels from the first block in pixel blocking. Assume the kernel stride is 1, each frame has $2*(K_c-1)$ overhead pixels. Given each frame contains $C/(N_l*N_c)$ pixels, the side length is $\sqrt{C/(N_c*N_l)}$, the overhead for each block would be

$$2*(K_c-1)/\sqrt{C/(N_c*N_l)} \tag{2}$$

dividing (1) by (2) leads to

$$(H*W*\sqrt{N_c})/(2*\sqrt{N_l*C}) \tag{3}$$

This equation means the overhead ratio of two methods. If the ratio is large than 1, the pixel blocking would have less overhead than frame blocking. Thus, we need to take parameters H, W, C, N_c and N_l into consideration and then choose different strategies according to different FPGA chips.

Since C varies from different FPGA boards, we set zc706 as our target board. In F-C3D, we take 3D CNNs' network setting into (3) and find that the value is greatly large than 1. Therefore, we choose pixel blocking as our blocking strategy. Then, block size for each layer can be determined according to (1) and layer's parameter. Table II presents the blocking strategy of F-C3D. For some layers, since all the input data can be cached on chip, it is no need to use blocking strategy.

TABLE II: Implementation Detail of C3D on zc706

layer	block strategy	block size
conv1a	pixel blocking	$56*56$
conv2a	pixel blocking	$16*16$
conv3a	pixel blocking	$16*16$
conv3b	pixel blocking	$8*8$
conv4a	no	original size
conv4b	pixel blocking	$8*8$
conv5a	no	original size
conv5b	no	original size

2) Filter and Pixel Parallelism: Apart from optimization strategies of improving input data locality, effective parallelism strategy is also critical for our hardware architecture.

Figure 5 illustrates two methods of parallelization when the degree of parallelism is 2. In filter parallelism, two 3D kernels with different coefficients take the same pixel as input to compute one result for two filters respectively. In pixel parallelism, the coefficients for both kernels are the same. Each 3D kernel receives different pixels to generate results for one filter.

Although two methods produce the same number of outputs at one time, there are two disadvantages in pixel parallelism. Assume the degree of parallelism is P.

- Since P 3D kernels need P pixels at the same time, the bandwidth between all buffers needs to increase P times.
- Because matrix buffer receives input data line by line, when P is large than the number of pixels in one line, zero-pad needs to be added into input feature map to fit the control logic in matrix buffer, which will cause overhead.

To simplify the control logic and minimize the overhead, we use filter parallelism as our parallelism strategy.

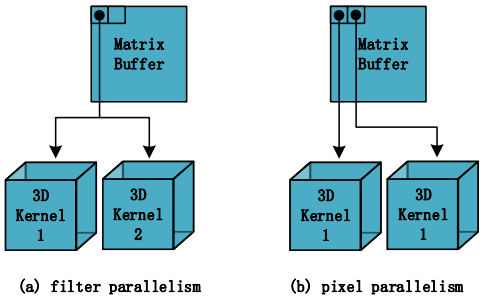

Fig. 5: Two parallelism methods

III. Implementation Details and Experiment Result

In this section, we first present implementation details of the whole system. Then, the experiment result is presented to show the performance of F-C3D design.

A. Implementation Details

As illustrated in Figure 6, the whole system fits in one Zynq chip along with DDR3 external memory. We implement one convolution and one fc module on Processing Logic (PL) and use Processing System (PS) to reconfigure the settings of 3D convolution and fc through AXI4 and APB bus. In this way, various layers with different parameters can be implemented. The communication within PL is based on APB bus.There are two DMA controllers in the system, which are used to transfer data and coefficients from DDR to PL with the bandwidth 128 bits/clock cycle.

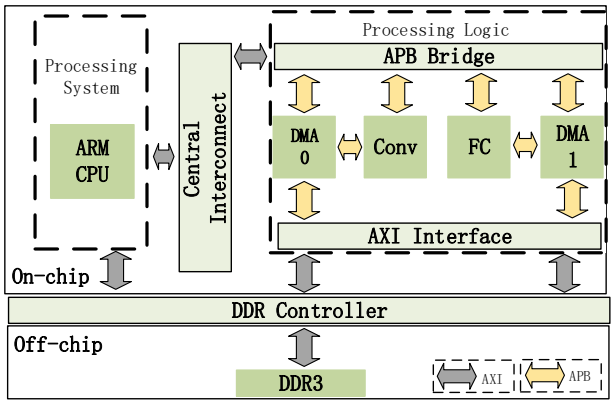

Fig. 6: System overview

For each layer, we first use PS to set parameters and then transfer input feature maps and coefficient from DDR3 to PL through DMA. The output feature maps will also be read out from PL to DDR3 using DMA. Ping-pong buffers are utilized in each output buffer in 3D convolution and fc modules to increase the bandwidth.

B. Experiment Result

We quantitatively make a comparison between our FPGA based design with ARM, Intel CPU and GPU implementations. F-C3D is implemented on Zynq zc706 with Vivado (v2016.2) and the resource utilization is presented in Table III.

TABLE III: Resource Utilization on Zynq zc706

	LUT	FF	DSP48	BRAM
Available	218600	437200	900	545
Utilization	90281	114983	810	472
Used Percentage	41.2%	26.3%	90.0%	86.6%

The UCF101 dataset [9] is used in the experiment and the dataset includes 13320 videos of 101 human action categories. To keep the accuracy of prediction, each video has 10 batches and each batch contains 16 frame clips [5]. The precision in experiment is 16 bits fixed point with 8 integer bits, 7 fraction bits and 1 signed bit. We compare our work with C3D tool [5], which is running on ARM v7_A, Intel (R) Core I7-950 (3.07 GHz) CPU and GTX 860M GPU. In ARM and Intel CPU, GCC 4.8.5 is used to compile C3D tool with $-Ofast$ compile flag. Table IV shows the performance of four implementations in terms of platform information, processing time per video, power and energy consumption per video(10 batches). In our design, the time of processing one batch (contains 16 frames) is 542.5 ms, which is nearly 231 times faster than C3D tool running on ARM CPU and 7.4 times faster than CPU. Although the processing speed of C3D tool running on GPU is 2.9 times higher than our design, it consumes 9.6 times more energy than FPGA implementation. To recognize one action, our design consumes 52.6J energy.

TABLE IV: Comparison F-C3D With CPU And GPU Implementation

	C3D CPU	C3D ARM	C3D GPU	FPGA
Platform	Intel (R) Core I7-950	ARM v7_A	GTX 860M	Zynq zc706
Num. of cores	8 (4 used)	2 (2 used)	–	–
Complier	GNU GCC	GNU GCC	CUDA (v7.0)	Vivado (2016.2)
Compile Flags	$-O$fast	$-O$fast	$-O$fast	–
Frequency	3.07 GHz	Up to 1 GHz	1020 MHz	172 MHz
Precision	32bits	32bits	32bits	16bits
Technology	45 nm	28 nm	16 nm	28 nm
Processing Time / video (ms)	39800	1253700	1840	5425
Power (W)	168	0.85	276	9.7
Energy / video (J)	6686.8	1066	507	52.6

Compared to the C3D software implementation, the action recognition error of F-C3D is 0.01%.

IV. Conclusion

This work proposes a novel hardware architecture, design strategies for 3D CNN. To the best of our knowledge, this is the first work to implement 3D CNN on FPGA. Through optimisation techniques such as blocking and on-chip cache, our design achieves 231 times speed up compared with ARM implementation, and 10 times energy efficiency improvement compared with designs running in GTX 860M GPU, with negligable 3D CNN model accuracy loss.

References

[1] 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 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[2] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 2016, pp. 1–9.

[3] R. Zhao, X. Niu, Y. Wu, W. Luk, and Q. Liu, "Optimizing CNN-based object detection algorithms on embedded FPGA platforms," 2017.

[4] 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.

[5] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4489–4497.

[6] D. Maturana and S. Scherer, "Voxnet: A 3D convolutional neural network for real-time object recognition," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 922–928.

[7] Q. Dou, H. Chen, L. Yu, L. Zhao, J. Qin, D. Wang, V. C. Mok, L. Shi, and P.-A. Heng, "Automatic detection of cerebral microbleeds from MR images via 3D convolutional neural networks," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1182–1195, 2016.

[8] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International Conference on Artificial Neural Networks*. Springer, 2014, pp. 281–290.

[9] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in neural information processing systems*, 2014, pp. 487–495.