Efficient Fast Convolution Architectures for Convolutional Neural Network

Weihong Xu[1,2], Zhongfeng Wang[3], Xiaohu You[2], and Chuan Zhang[1,2,*]
[1]Lab of Efficient Architectures for Digital-communication and Signal-processing (LEADS)
[2]National Mobile Communications Research Laboratory, Southeast University, Nanjing, China
[3]School of Electronic Science and Engineering, Nanjing University, Nanjing, China
Email: [2]{wh.xu, xhyu, chzhang}@seu.edu.cn, [3]zfwang@nju.edu.cn

***Abstract*—Due to the world-wide interests on artificial intelligence, many acceleration architectures for convolutional neural network (CNN) have been proposed recently. However, few of them focus on reducing convolution computation strength. In this paper, we first present fast convolution algorithm and its matrix form. Then based on the fast convolution algorithm, a fully parallel architecture with high throughput is proposed. To further increase efficiency and reduce computation redundancy, output data reuse scheme corresponding to CNN is also considered by introducing affordable adders and buffers. The hardware implementation and complexity comparison are conducted among different convolution architectures. Implementation results on Zynq XC7Z045 platform demonstrate the effectiveness of proposed fast convolution architectures in the reduction of complexity. Compared to conventional 2-D convolver, our 3-parallel fast convolution filter reduces 28% hardware resources and improves throughput by 17%. After deploying data reuse scheme, our fast convolution architecture is $10.56\times$ faster.**

***Index Terms*—Fast convolution, parallel processing, data reuse, hardware reconfigurability, convolutional neural network (CNN).**

I. Introduction

Convolutional neural network (CNN), as a type of feed-forward artificial neural network for machine learning, has shown great potential in image recognition, video analysis, drug discovery, and other related fields. Research has shown that CNN can achieve a fairly high accuracy in tasks like object recognition when the network is deep enough. One amazing application of CNN is ResNet [1], a 152-layer net setting a new record in terms of error rate and depth. On the other hand, deeper convolutional neural network poses not only heavier workload but also greater challenge in complexity and energy efficiency in its implementation platform. Although CNN can be realized with pure software, with the drastically increased network size, the computational capabilities of traditional PCs or servers are far from enough to run a CNN with fast speed and high efficiency. Hence, hardware acceleration of CNN has become increasing popular.

A lot of excellent works have been done to achieve hardware acceleration of CNN with high efficiency. The first solution is to employ GPUs. Many papers have implemented the training process and calculation of CNN in GPU [2]. First, it is easy to deploy software into GPU platform. Second, GPU offers high speed and throughput. However, GPU-based CNN has limitations on hardware cost and power consumption, which make the implementation still expensive.

Recently, research have proved the great advantages of customized hardware option (FPGA and ASIC), which can reduce hardware complexity and power consumption. Among existing literatures, Sankaradas presented a massively parallel coprocessor [3] by adopting reconfigurable 2-D convolvers [4] as the operation units. [5] optimized the on-chip memories by using tiling for data locality, since the external memory bandwidth is limited. Zhang [6] proposed a method aiming at acceleration across different layers with the help of roofline model. Paper [7] focused on the local data reuse of filters and feature map pixels, which maintains high level parallelism and minimizes data movement inside the network.

Existing FPGA-based customized architectures speed up CNN mainly in two ways. One way is to optimize memory access and data movement to keep a good balance between the computation throughput and memory bandwidth. The other way is to accelerate convolution computation with parallel technique. Admittedly, high parallelism for sure improves the system performance, but will also introduce prohibitive hardware cost. It has been pointed out that convolution computation takes up more than 90% of the computation complexity and determines the processing time of CNN [8]. However, very few works consider reducing the complexity in convolutional layers. Therefore, this paper devotes itself in proposing design methodology for efficient, parallel, low-cost, and reconfigurable convolution kernels.

In this paper, background of CNN and fast convolution is given in Section II. Then we propose efficient architectures and optimization corresponding to CNN in Section III, which reduces the computation strength and redundancy based on fast convolution algorithm. FPGA implementation and comparison are given in Section IV. Section V concludes the entire paper.

II. Preliminaries

Some basics of convolutional neural network (CNN) as well as fast convolution algorithm are introduced in this section.

A. The Basics of Convolutional Neural Network

A typical convolutional neural network (CNN) is composed of several types of computation layers: convolutional layer, pooling layer, and fully connected layer. Convolutional layer is an essential part of CNN, which extracts hidden features by sliding filters through input feature maps. In general, this operation is a 2-D convolution for each input feature map $\mathbf{I}$ and the output feature map $\mathbf{O}$ can be computed as:

$$\mathbf{O}[x][y] = \sum_{x'=0}^{X-1} \sum_{y'=0}^{Y-1} \mathbf{I}[x-x'][y-y']\mathbf{W}[x'][y'], \tag{1}$$

where X and Y denote the spatial dimensions of the convolutional filter. And $\mathbf{W}$ is the convolution filter.

B. Fast Convolution Algorithm

Fast convolution algorithm is proposed to reduce strong operations (such as multiplications) in convolution at the expense of weaker operations (such as additions) [9]. In fact for a CNN, fast convolution algorithm can also improve the system speed, because deeper and fine-grain pipelines can be introduced conveniently.

Assume an $L \times L$ linear convolution, its fast convolution algorithm [10] can be expressed in the following matrix form:

$$\mathbf{Y}_L = \mathbf{P}_L^T \mathbf{H}_L^T \mathbf{Q}_L^T \mathbf{X}_L, \tag{2}$$

where $\mathbf{Q}_L^T$ and $\mathbf{P}_L^T$ represents transposed pre-processing matrix and post-processing matrix, respectively. $\mathbf{H}_L^T$ is a diagonal matrix, whose entries on the diagonal can be calculated by $[h_0, h_1, ..., h_{L-1}] \times \mathbf{P}_L^T$ and h_k denotes the k-th filter weight of a row in a filter. $\mathbf{X}_L$ and $\mathbf{Y}_L$ are two column vectors containing parallel input and parallel output signal. We will give more details of Eq. (2) in Section III-A.

III. Proposed Efficient Fast Convolution Architectures

In this section, efficient hardware architectures of the aforementioned fast convolution algorithms are proposed in detail.

A. Architecture of Elementary Fast Convolution Filter

3×3 is the most commonly used filter size in current CNNs [2, 11]. The most direct and simple method for computing $k \times k$ convolution is to implement k^2 multiplications and $k^2 - 1$ additions [4]. However, this solution doesn't reduce computation complexity of convolution and causes exponential increase in hardware cost with filter size growing, which severely restricts the system throughput. Thus, it will make great sense to design optimization schemes for such filter.

Fortunately, fast convolution algorithm (FCA) [9] offers an excellent strategy to speed up CNN by sharing some intermediate terms of convolution calculation. In Eq. (2), an L-parallel short convolution representation consists of $\mathbf{P}_L^T$, $\mathbf{H}_L^T$, $\mathbf{Q}_L^T$, and $\mathbf{X}_L$. Without loss of generality, the 3-parallel filter is employed as an example. Other sized fast convolution filter can be easily derived in a similar fashion [9].

According to [10], a feasible 3-parallel filter has been given. For this case, Eq. (2) becomes:

$$\mathbf{Y}_3 = \mathbf{P}_3^T \mathbf{H}_3^T \mathbf{Q}_3^T \mathbf{X}_3, \tag{3}$$

where

$$\begin{aligned}
&\mathbf{H}_3^T = \mathbf{diag}\left[h_0, h_1, h_2, h_0+h_1, h_0+h_2, h_1+h_2\right], \\
&\mathbf{Y}_3 = \begin{bmatrix} Y_2 \\ Y_1 \\ Y_0 \end{bmatrix}, \quad \mathbf{P}_3^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \\
&\mathbf{X}_3 = \begin{bmatrix} X_2 \\ X_1 \\ X_0 \\ z^{-3}X_2 \\ z^{-3}X_1 \end{bmatrix}, \quad \mathbf{Q}_3^T = \begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.
\end{aligned} \tag{4}$$

It should be noted that $\mathbf{H}_3^T$ is a 6×6 diagonal matrix. Three data in a row of a feature map are fed into $\mathbf{X}_3$ at a time and the first two data are latched by two D flip-flops. From $\mathbf{Y}_3$, three convolution results are generated in parallel.

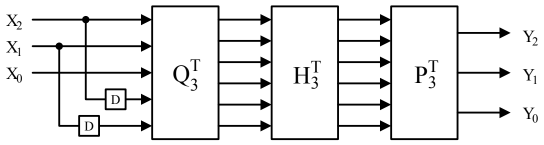

Fig. 1. Overall architecture of the 3-parallel fast convolution filter.

Fig. 1 illustrates the overall architecture implementing Eq. (3). The $\mathbf{H}_3^T$ module is made up of 6 multipliers. The main diagonal entries of matrix $\mathbf{H}_3^T$ are loaded to each multiplier, respectively. $\mathbf{P}_3^T$ requires 6 adders. And $\mathbf{Q}_3^T$ requires 6 adders. Thus, the proposed 3-parallel fast convolution filter requires 6 multipliers and 12 adders in total.

B. Fully Parallel Architecture

Different from conventional 2-D filter [4], the method mentioned in Section III-A just calculates the convolution in a specific row of input feature maps per clock cycle, which well suits the recent 1×3 or 3×1 sized kernel [12]. However, overwhelming majority of existing CNNs use 3×3 filter. The 2-D fast convolution can be performed through multiplexing single fast convolution filter or applying parallel processing. To meet the high throughput requirement of CNN, a fully parallel architecture is proposed here. For the purpose of modularity, it is divided into two parts: fast convolution kernel and adder array (see Fig. 2). These two modules are connected by a 9-to-9 crossbar. The details are described below.

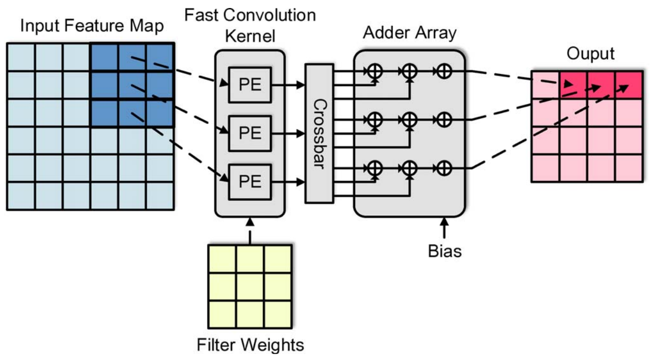

Fig. 2. Fully parallel architecture for 3×3 fast convolution.

Fast Convolution Kernel: This part generally processes k^2 pixels from input and receives k^2 filter weights in a $k \times k$ convolution. Hence, it consists of k processing elements (PEs), each containing one k-parallel fast convolution filter. Considering a 3×3 convolution, there are 3 PEs inside fast convolution kernel. 3×3 pixels in continuous 3 rows of a feature map are fed into fast convolution kernel simultaneously. As we have explained in Eq. (2), a single PE accepts 3 coefficients of a specific filter row. Therefore, a 3×3 sized filter are preloaded to PEs, respectively.

Adder Array: Every clock cycle, k^2 partial sums are generated from fast convolution kernel. To calculate 2-D convolution and add up bias, we need an adder array after fast convolution kernel to calculate the k^2 partial sums to obtain k pixels of output feature map. To this end, k^2 adders should be employed within adder matrix for a $k \times k$ convolution. More specifically, we implement a 3×3 adder array for 3×3 fast convolution. The adders in first two columns sum up 9 results from fast convolution kernel and the last column is used to add bias.

C. Data Reuse Consideration

Theoretically, the fully parallel architecture achieves $3\times$ throughput rate than the elementary filter. Nevertheless, the potential of fast convolution architecture can be excavated through output data reuse. Convolution stride of CNN is normally smaller than the filter size, which causes data overlap when the filter window slides. The redundancy of stride operation can be consequently exploited to improve performance.

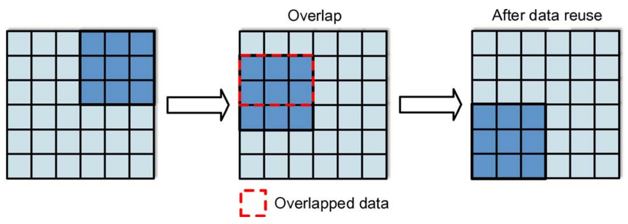

Fig. 3. Example of data reuse in one-stride 3×3 convolution.

A $k \times k$ filter should shift to next row after finishing convolving with k rows of input feature map. In this case, the last $(k-1) \times k$ pixels in current filter window overlap and lead to repeated calculation. Therefore, the overlapped partial sums can be reused for calculation when it is required. In our design, the partial sums of last $(k-1)$ rows of input feature map that the filter are calculating should be cached in $(k-1)$ FIFOs (first-in first-out), respectively. The i-th $(i = 1, 2, ..., k-1)$ FIFO stores the $(i+1)$-th row of output feature map. Suppose an $N \times N$ input feature map with stride S, the depth of each FIFO is $\lceil \frac{N-k}{S} + 1 \rceil$. An example of data reuse is given in Fig. 3. Before adopting data reuse strategy, it costs $\lceil \frac{6}{3} \rceil \times \lceil \frac{6-3}{1} + 1 \rceil = 8$ clock cycles to complete calculation. Only $\lceil \frac{6}{3} \rceil \times \lceil \frac{6}{3} \rceil = 4$ clock cycles are required with data reuse. The advantages of data reuse scheme will be analyzed in Section IV-A.

After taking data reuse into consideration, the architecture is as shown in Fig. 4. Two additional adder arrays are equipped

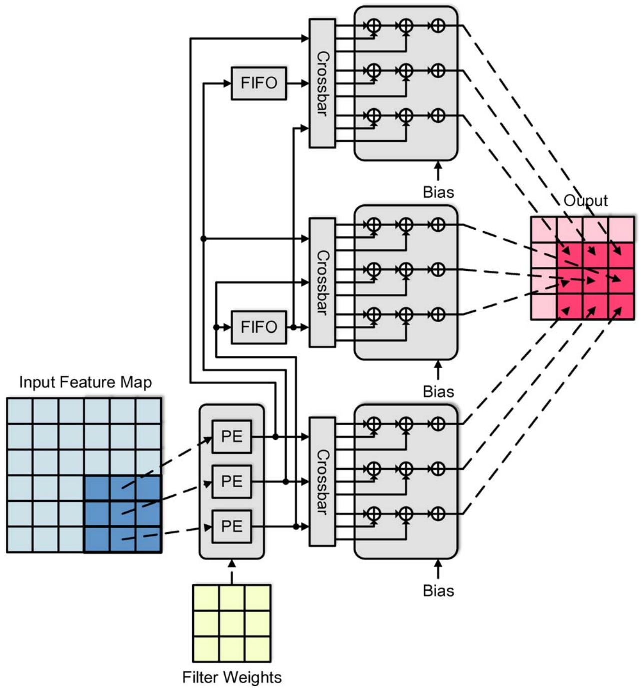

Fig. 4. Hardware architecture for 3×3 convolution with data reuse scheme.

to sum up the buffered reuse data and output data. As a result, the system throughput is about $3\times$ than the fully parallel architecture at the cost of adders and FIFOs. The advantage is attractive because the proposed data reuse strategy increases throughput by duplicating inexpensive adders and buffers in FPGA rather than costly convolvers and bulky multipliers.

IV. Hardware Implementation and Comparison

In this section, FPGA implementations of different convolution structures are provided based on 3×3 convolution. The complexity and timing comparison is also given.

A. Complexity Analysis and Comparison

Multiplication is computation expensive compared to addition. To evaluate the hardware complexity with other architectures in a fair manner, carry-save adder and a modified Wallace multiplier [13] are employed in our analysis, where a 32-bit quantization scheme is considered. In this case, a total of 124 full adders are required to implement a 32-bit carry-save adder. A 32-bit Wallace multiplier contains 907 full adders and 23 half adders in total.

The number of multiplications, additions, total hardware cost (in term of full adder #) and normalized complexity are listed in Table I. Compared with the conventional 2-D convolver, our fast convolution architectures save large amount of hardware resources. The normalized complexity per output is $1.28\times$ to $3.45\times$ lower than direct convolution method.

For an $N \times N$ input feature map with stride S and $k \times k$ filters, the efficiency of data reuse strategy becomes more obvious as the filter size and image size increase. The clock

TABLE I
ESTIMATED HARDWARE COMPLEXITY AND COMPARISON (IN NUMBER OF REQUIRED FULL ADDERS)

Convolution structure	Mul.	Add.	Full adders	Normalized complexity
3×3 convolver [4]	9	9	$9,486$	1
Fast convolution filter	6	15	$7,440$	0.78
Fully parallel architecture	18	45	$22,320$	0.78
Data reuse scheme	18	63	$24,552$	0.29

cycles required to convolve this map can be numerically represented as following equation:

$$\#_{\text{clock}} = \left\lceil \frac{N}{k} \right\rceil \times \left\lceil \frac{N-k}{S} + 1 \right\rceil \tag{5}$$

The processing clock cycles after applying data reuse scheme becomes:

$$\#_{\text{clock}} = \left\lceil \frac{N}{k} \right\rceil \times \left\lceil \frac{N}{k} \right\rceil \tag{6}$$

Fig. 5 illustrates the efficiency of data reuse scheme in terms of clock cycles required for processing each input map.

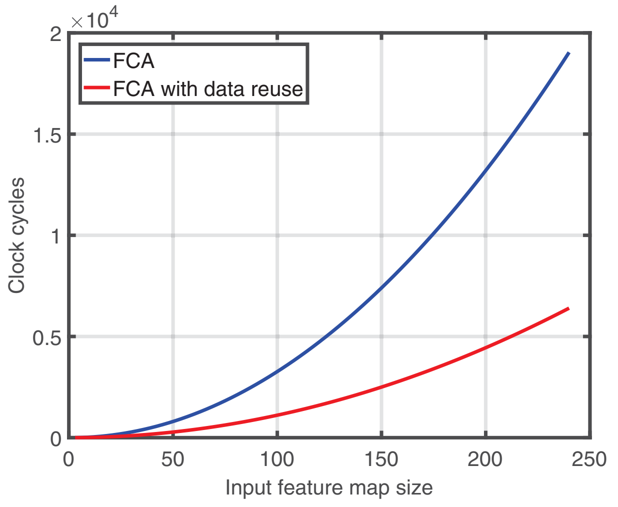

Fig. 5. Speedup of 3×3 convolution data reuse scheme versus input size.

B. FPGA Implementations

In order to verify the advantages of proposed architecture, we implement our 16-bit fast convolution architectures and regular 2-D convolver [4] on Xilinx Zynq XC7Z045.

TABLE II
HARDWARE AND TIMING COMPARISON

Design	DSP	LUT	FF	Critical path (ns)	Throughput (GOP/s)
3×3 conv	9 (1%)	144 (0%)	16 (0%)	4.181	4.30
Fast conv	6 (0%)	240 (0%)	192 (0%)	3.567	5.05
Fully para	18 (2%)	420 (0%)	576 (0%)	3.567	15.14
Data reuse	18 (2%)	684 (0%)	672 (0%)	3.567	45.42

Table II summarizes corresponding hardware parameters, critical path and throughput. It is indicated that our proposed fast convolution architectures reduce hardware cost on DSP at the expense of increasing LUTs and registers. Since the number of LUTs and FFs in FPGA is far more than DSPs, more kernels which we propose can be implemented with the same hardware. Fast convolution filter, the counterpart of 2-D convolver, gains $1.17\times$ throughput with less hardware.

It is noted that the critical path of our designs remains constant and is shorter than that of regular 2-D convolver [4]. Therefore, under different structures, $3.52\times$ and $10.56\times$ performance benefits are realized in term of throughput.

V. CONCLUSION

In this paper, we propose efficient convolution architectures based on fast convolution algorithm. And optimization is also introduced. Performance analysis have shown the advantages in computation complexity and throughput. The future work will focus on deeper optimization of overall CNN.

ACKNOWLEDGEMENT

This work is supported in part by NSFC under grant 61501116, Jiangsu Provincial NSF under grant BK20140636, Huawei HIRP Flagship under grant YB201504, the Fundamental Research Funds for the Central Universities, State Key Laboratory of ASIC & System under grant 2016KF007, ICRI for MNC, and the Project Sponsored by the SRF for the Returned Overseas Chinese Scholars of MoE.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *arXiv:1512.03385 [cs]*, 2015.
[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv:1409.1556*, 2014.
[3] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, “A massively parallel coprocessor for convolutional neural networks,” in *Proc. IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2009, pp. 53–60.
[4] B. Bosi, G. Bois, and Y. Savaria, “Reconfigurable pipelined 2-D convolvers for fast digital signal processing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 3, pp. 299–308, 1999.
[5] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, “Memory-centric accelerator design for convolutional neural networks,” in *Proc. IEEE International Conference on Computer Design (ICCD)*, 2013, pp. 13–19.
[6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2015, pp. 161–170.
[7] Y. H. Chen, T. Krishna, J. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, Jan. 2016, pp. 262–263.
[8] H. Nakahara and T. Sasao, “A deep convolutional neural network based on nested residue number system,” in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–6.
[9] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons Inc, 1999.
[10] C. Cheng and K. K. Parhi, “Hardware efficient fast parallel FIR filter structures based on iterated short convolution,” *IEEE Transactions on Circuits and Systems I: Reg Papers*, vol. 51, no. 8, pp. 1492–1500, 2004.
[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *NIPS*, 2012, pp. 1097–1105.
[12] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the impact of residual connections on learning,” *arXiv:1602.07261 [cs]*, 2016.
[13] R. S. Waters and E. E. Swartzlander, “A reduced complexity Wallace multiplier reduction,” *IEEE Transactions on Computers*, vol. 59, no. 8, pp. 1134–1137, 2010.