A Reconfigurable Process Engine for Flexible Convolutional Neural Network Acceleration

Xiaobai Chen, Shanlin Xiao and Zhiyi Yu
Sun Yat-sen University, Guangzhou 510006, China
E-mail: chenxb29@mail2.sysu.edu.cn; xiaoshlin@mail.sysu.edu.cn; yuzhiyi@mail.sysu.edu.cn

***Abstract*— Convolutional neural network (CNN) is the most powerful artificial intelligence algorithm widely used in computer vision due to its state-of-the-art performance. There are many accelerators proposed for CNN to handle its huge computation and communication cost. In this paper we proposed a reconfigurable process engine which can support different data flows, bit-widths, and parallelism strategies for CNNs. The process engine was implemented on Xilinx ZC706 FPGA board, with high flexibility to support all popular CNNs, and better energy efficiency compared to other state-of-the-art designs.**

I. INTRODUCTION

Since state-of-the-art CNNs [1–4] require more than 600k operations per input pixel and up to hundreds of MB storage for the weights and activations, specific accelerators are required to handle the huge challenges in computation, storage, and communication. There are some accelerators [5-11] proposed for the CNNs to overcome these challenges. Some accelerators obtain high performance and good energy efficiency, but with low flexibility to support only one specific CNN. Other accelerators use huge resources to obtain better flexibility to support various CNNs but with complex architecture and high energy consumption. Since the process engine is the main computation module which consumes the vast majority of the power of the accelerator, a reconfigurable and energy efficient process engine is required to support the acceleration of various CNNs.

As we proposed in [5], a highly efficient CNN acceleration needs the co-design of hardware and software with reconfigurable ISA and process engine. In this paper, we improved and detailed the reconfigure process engine and implemented it on an FPGA. The experimental result shows that it supports all popular CNNs and obtains better energy efficiency than other designs. The main contributions of this paper are:

1. Proposed a CNN-specific reconfigurable process engine, which supports different parallel computations, bit-widths, and data reuses of all popular CNNs.
2. Implemented an accelerator with the proposed process engine on an FPGA, and obtains better energy efficiency than other state-of-the-art designs.

The paper is organized as follows. Section II describes the design principle of the CNN process engine. Section III shows the design details of the process engine. Section IV gives the evaluation and comparison. Finlay is the conclusion.

II. DESIGN PRINCIPLE OF THE CNN PROCESS ENGINE

There are two ways to improve the CNN acceleration efficiency: increasing the parallel computation degrees and decreasing the data transfer cost. Most proposed accelerators use dedicated and complicated computation strategies and data flows to supports a specific CNN, which obtain high performance but are inflexible. Some solutions use complex controller to handle huge reconfigurable hardware resources to fit different CNNs, improving flexibility but with high power consumption. A process engine with great trade-off between flexible and energy efficiency is required.

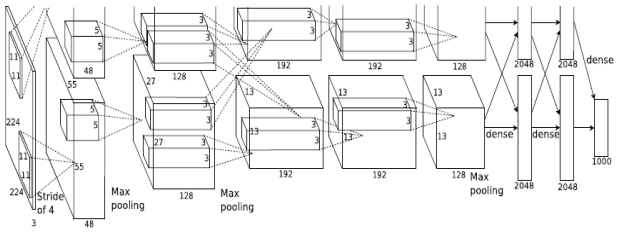

Fig .1 AlexNet framework.

A. Flexibility

Since more than 90% computation cost of CNNs happens in the convolutional layers, a process engine efficiently supporting different convolution operations will obtain ideal flexibility. For different CNNs or different layers in the CNN, the convolutions have different kernel sizes, channel numbers, and feature map sizes. As the AlexNet shown in the Fig 1, in different convolutional layers, the kernel size K can be 11, 5, or 3; the feature map size L can be 224, 55, 27, or 13; and the channel number C can be 3, 48,128, or 193. For other CNNs such as VGG and GoogleNet, the kernel size K can be only 1 and the channel number C can be as many as 1024. In addition, there are different choices of the no-convolutional operations in the convolutional layers, such as the different activation functions (ex. Relu, sigmoid), pooling (ex. max pooling, average pooling). To obtain better flexibility, the process engine needs to support all above-mentioned operations in the convolutional layers.

B. *Energy efficiency*

Due to different types of the convolutional operations and different sizes of the operands in CNNs, the hardware requires various computation strategies and data flows to obtain high energy efficiency. For the computation strategy, there are 3 types of data-level parallelisms for the CNN: input parallelism Pin, convolutional parallelism Pc, and output parallelism Pout. For the data flow of CNN, there are three reuse strategies including input feature reuse, convolution reuse, and weight reuse. Since the inner fault tolerance of the CNN, the weights of the CNN can be compressed into very low bits, such as 4 bits. Low-bit weights reduce both the computation and communication cost. For better energy efficiency, the process engine needs to support these various CNN acceleration strategies and different bit-widths.

III. HARDWARE ARCHITECTURE

A. *Process engine for convolution operations*

The most computational cost block in CNN is the convolution, whose fundamental operation is multiplication and accumulation (MAC). In this design, the convolution process engine consists of multiplier and adder trees to handle the MAC operations.

1) Specific Multiplier element

A specific Multiplier element (ME) is proposed to handle the mass multiplications, as shown in Fig 2. Since the bit-width of the weight in the CNNs can be compressed to 4-16 bits, this ME is designed to support 4-bit, 8-bit, and 16-bit weight multiplications to improve the resource utilization. The ME consists of four multipliers to handle the multiplication with 16-bit and 4-bit operands. For 4-bit weights, four multipliers work independently and the bit shifter is turned off, so there are four multiplications processed in parallel and output four results. For 8-bit weights, two multipliers are combined to work together. One multiplier handles the high 4 bits weight multiplication and the other handles the low 4 bits, then the high bits result is left shifted and added with the low bits result to obtain the 8-bit multiplication result. There are two multiplications processed in parallel and output two results for 8-bit weight situation. For 16-bit weights, four multipliers work together and output only one multiplication result. The ME has a Mux to choses the input data from the exchange data (the adjacent ME) or from the register files to form various ME groups for convolutional computations. The data exchange between the multipliers realizes the convolutional data reuse.

2) Convolution process engine

Fig. 3 is the convolutional process engine (CPE), which contains 2 ME arrays and adder trees. There are 27 MEs in each ME array and the adjacent MEs are connected with each other.

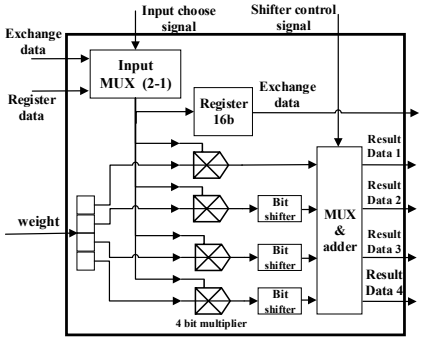

Fig.2. Multiplier element (ME).

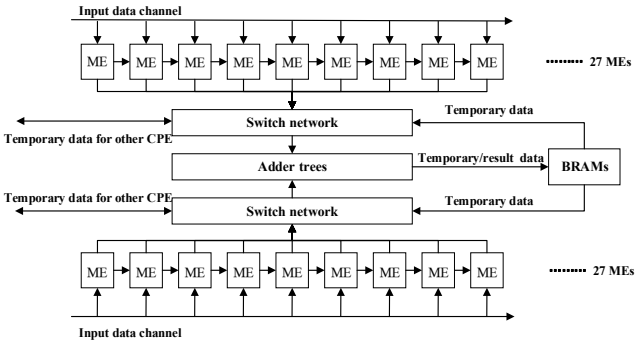

Fig.3. Convolution process engine [5].

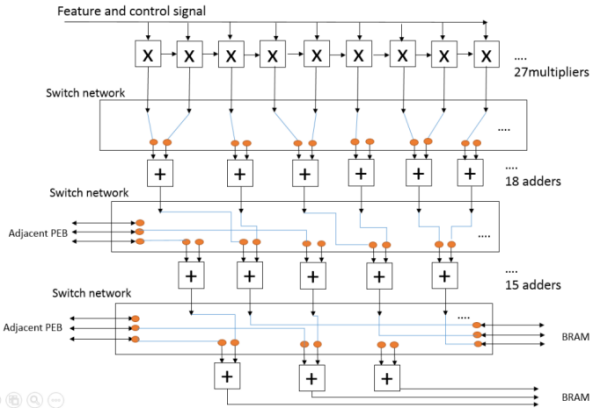

Fig.4. 3 level switch network.

The ME can choose the data from the left ME though the connections between each other or choose the input feature from the register files though the input data channels. The ME and adder trees are connected by a switch network to implement the MAC operations. The temporary/final results of the MAC will be stored into the data buffer BRAMs. The MEs and adder trees can be organized to fit different convolutional parallel computations according to the

convolution kernel size k, the parallel degree parameters, and so on.

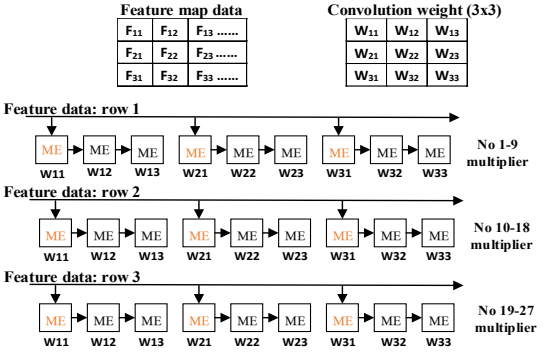

Fig.5. CPE configuration when kernel K= 3. [5].

3) Convolution process engine reconfiguration

The CPE can be reconfigured to support different convolutional parallel computations and various data ruses.

For the convolutional computation, reconfigurable switch networks are used to implement different sizes of convolutions and different parallelisms, as shown in Fig. 4. There are three levels of switch networks in the design. The first level switch network sets the connections between the multipliers and adders, and adds up K products of the multipliers by cutting off the (K + 1)th adder's connection, which is shown in the uppermost switch network in Fig.4. The second level switch network sets the connections to add Pc × K outputs from the adder trees to implement Pc convolutional parallelism, and then adds with the Pin temporary MAC results (Pin is the number of feature channels) from the other ME arrays to implement Pin input parallelism, which is shown in the middle switch network in Fig.4. The final level switch network combines the temporary results from the RAM or other CPEs, and stores the temporary/final result back to the RAM, as shown in the bottom switch network in Fig.4. The Switch network sets the connections in the CPE according to the kernel size K, parallel degree Pc and Pin.

The parallelism parameters determine three types of data reuse. For the convolutional reuse, it obtains the best data reuse when Pc = K^2, and every row of the input feature only needs to be loaded once. It obtains 4 input features reuse for 4-bit weight, since the input feature is shared by four multipliers in every ME. The weight reuse is improved by increasing the input parallelism Pin with the connected ME arrays. The larger Pin also reduces the access frequency of temporary Psum data by Pin times.

Fig.5 is the example of CPE configuration when the convolution kernel K= 3. The first (red one) of every K MEs chooses the operands from register files, and others choose input from the front one and the weight stays stationary. Every row of the input feature is reused by k times so that it only needs to be read once and Pc = K^2. The switch network

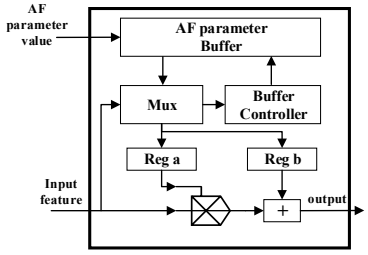

Fig.6. Activation function process engine

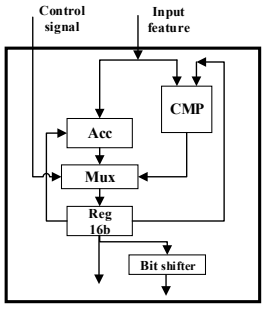

Fig.7. Pooling process engine

connects the output of 9 multipliers with the adders to form 3 adder trees for the convolution output. Here Pin = 1 for single CPE.

B. Process engine for activation function.

The convolution output of the CNN will be processed by the activation function. There are two types of activation functions. One is the simple linear computational model such as the ReLU. The other is the complex non-linear model with relatively large computational cost due to exponential and division operations, such as the sigmoid. For the complex activation function, a linear approximation can be used to reduce the computation complexity with certain accuracy loss. In our design, a linear approximation as the follow formula is proposed, where x is the input feature value, A_i and B_i represent the coefficients for x within the range of $[x_i, x_{i+1}]$. The accuracy loss is negligible since the approximation formula is extremely close to the actual activation function.

$$F(x) = A_i * x + B_i$$

The process engine for the activation function is shown in Fig 6. The coefficient parameters are stored into the parameter buffer at the initial time. For this simple activation function, the A and B are all constant values and there are no any parameter buffer access. For the complex activation function, the Mux and buffer controller need to find the right coefficients from the buffer according to the input feature value, and then store the coefficients into the registers a and b respectively. Then the multiplier and adder compute the result

of activation function. The approximation can reduce more than 80% energy consumption compared to the normal sigmoid computation.

C. *Process engine for pooling*

There are two types of pooling in the CNNs: average pooling and max pooling. The pooling process engine shown in Fig 7 is designed to support both pooling. For the max pooling, the comparer (CMP) compares the input feature and the previous feature stored in the register, and writes back the larger one into the register. For the average pooling, the ACC is used to accumulate the input features, and the sum is shifted by the Bit shifter to realize the average computation. The Mux chooses the input from the ACC or the input from CMP to perform these two types of pooling respectively.

IV. EVALUATION AND COMPARISON

We implemented the process engine and the CNN accelerator, and synthesized, verified, and evaluated the design using Xilinx EDA tool Vivado 2015.4 on the Zynq ZC706 platform. Table I shows the resource utilization of our design.

TABLE I
Resource utilization

Resource	LUT	FF	BRAM	DSP
Utilization	26009	9170	26	801
Percentage	11.88%	1.98%	4%	89.7%

Compare to the work [5], the process engine in this design contains more multipliers, and the utilization of DSP is increased. We also implemented four most popular CNNs: AlexNet[1], VGG[2], Google net[3], and Deep Residual network[4] on the accelerator. The comparisons of performance and energy efficiency are shown in Table II and Table III respectively.

TABLE II.
Performance comparison

Work	Power (w)	AlexNet	GoogleNet	VGG
Work[5]	0.461	201 ms	514 ms	1642 ms
Ours	0.784	48 ms	136 ms	587 ms

TABLE III.
Energy efficiency comparison (frame/J)

work	Power (w)	AlexNet	GoogleNet	VGG
Work[5]	0.461	11.0	4.21	1.32
Ours	0.784	27.8	9.37	2.17

From the comparison, the accelerator obtains almost 4x higher performance and more than 2x better energy efficiency compared to work [5], mainly because of the improvement of process engine. The process engine not only supports different parallel computations and data reuse strategies, but also supports different bit-width computation for different CNN layers and different CNNs, which significant reduce the computation and communication cost.

V. CONCLUSIONS AND ACKNOWLEDGMENT

In this paper we proposed a reconfigure process engine which can support different data flows, bit-widths, and parallelism strategies for CNNs. We implemented the process engine and CNN accelerator on the Xilinx ZC706 FPGA board. Experiment data shows that the accelerator obtains high flexibility, and with 2x better energy efficiency compared to other state-of-the-art designs.

This work was supported in part by grant from national nature science foundation of China (NSFC) under grant No.61674173.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman. (2014). “Very deep convolutional networks for large-scale image recognition.” [Online]. Available: https://arxiv.org/abs/1409.1556

[3] C. Szegedy et al., “Going deeper with convolutions,” in Proc. IEEE Comput. Vis. Pattern Recognit., Jun. 2015, pp. 1–9. 378

[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Comput. Vis. Pattern Recognit., Jun. 2016, pp. 770–778.

[5] Chen, Xiaobai, and Zhiyi Yu. "A Flexible and Energy-Efficient Convolutional Neural Network Acceleration With Dedicated ISA and Accelerator." *IEEE Transactions on Very Large Scale Integration Systems* 26.7 (2018): 1408-1412.

[6] Zhang C, Li P, Sun G, et al. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks[C]// Acm/sigda International Symposium. ACM, 2015:161-170.

[7] Qiu J, Wang J, Yao S, et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network[C]// Acm/sigda International Symposium on Field-Programmable Gate Arrays. ACM, 2016.

[8] Suda N, Chandra V, et al. Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks[C]// Acm/sigda International Symposium on Field-Programmable Gate Arrays. ACM, 2016.

[9] Moons, Bert, and M. Verhelst. "An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS." IEEE Journal of Solid-State Circuits 52.4(2017):903-914.

[10] Yin, Shouyi, et al. "A High Energy Efficient Reconfigurable Hybrid Neural Network Processor for Deep Learning Applications." IEEE Journal of Solid-State Circuits 53.4(2018):968-982.