A FPGA-based Hardware Accelerator for Multiple Convolutional Neural Networks

Yuchen Yao[1], Qinghua Duan[2], Zhiqian Zhang[3], Jiabao Gao[4], Jian Wang[5], Meng Yang[6], Xinxuan Tao*, Jinmei Lai*

[1]State Key Laboratory of ASIC&System, Fudan University, Shanghai 201203, China

[2]Chengdu Sino Microelectronics Technology Co., Ltd

* Email: xxtao7718@163.com , jmlai@fudan.edu.cn

Abstract

Convolution Neural Network (CNN) has been widely used in many computer vision tasks. Due to the rapid growth of CNN, the accelerator that only supports single network could not meet the requirement of application. Based on the work of ZynqNet, which is a dedicated CNN accelerator, in this paper, we propose a FPGA-based CNN accelerator which supports the acceleration of multiple networks, and present an automatic mapping flow in which users only need to provide network description files and test image to accelerate a specified network. And we adopt a dynamic fixed-point quantization strategy to reduce resource consumption. Experimental results shows the performance density and power efficiency of our design can reach 0.054GOPS/DSP and 5.24GOPS/W respectively when accelerating SqueezeNet.

1. Introduction

Convolutional Neural Network (CNN) is one of the state-of-art deep learning architectures. Since AlexNet [1] won the 2012 ImageNet competition with its outstanding recognition capability, more and more improved CNN architectures have emerged and been widely used in many computer vision tasks. And recently, various accelerators have been proposed to deal with the increasing computation complexity of CNN. Among them, FPGA-based accelerators have become a promising candidate due to its reconfiguration ability and low-power consumption.

Nowadays, there are many studies on FPGA-based accelerators. In [2], ZynqNet, an efficient light-weight CNN topology, is proposed and its dedicated FPGA accelerator is implemented. Also, an automatic flow to extract parameters of convolutional layers is used. And it seems that using high level synthesis (HLS) tool to design an accelerator has gradually became a tend because of its improvement to the productivity[2][3].

However, most FPGA accelerators is designed for a specific network, which is hard to meet the diversity of CNN. Based on the work in [2], in this paper, we propose a FPGA-based accelerator which is able to accelerate multiple networks, and present an automatic mapping flow supporting the exaction of most CNN layers and the quantization to a dynamic fixed-point format. The automatic mapping flow exacts the information of each layer in the network to be accelerate, according to the network description file, and output a configuration file. Based on the configuration, our accelerator which supports most CNN operations executes computation, i.e. accelerates the specified network. The main contributions of this work are as follows.

1) An accelerator which can support the acceleration of multiple networks is introduced. The operations that it can achieve are convolution and pooling. And the optimizing strategies used in HLS makes high performance density and power efficiency possible.

2) A automatic mapping flow based on [2] is perfected and used, which can exact the information of convolutional, pooling and fully-connected layers. And with this mapping flow and our accelerator, users only need to provide network description files and test image data to accelerate a network.

3) A dynamic fixed-point quantization strategy is adopted, reducing the utilization of resources significantly.

2. Automatic mapping flow

The overall mapping flow is showed in Figure 1. Each step in this flow is explained below.

2.1 Input files

In this flow, users only need to provide CNN model files, that is, .prototxt file and .caffemodel file, and test image data. The model files are described under the popular caffe framework, which allows users to directly use the files for training CNN in the cloud or GPU and spares users the trouble of writing extra input files.

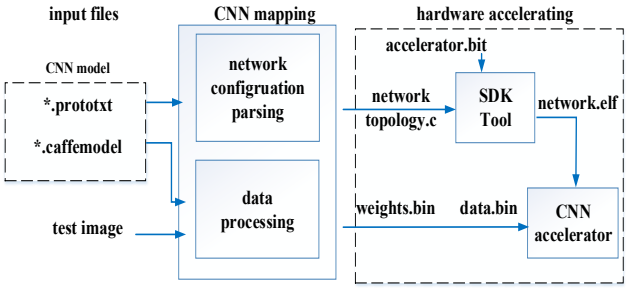

Figure 2. Overall mapping flow

2.2 CNN mapping

Network configuration parsing: In the operation of

978-1-5386-4441-6/18/$31.00 ©2018 IEEE

network parsing, the mapping flow extracts the information from the .prototxt file and output a .c file, which contains the configuration information of convolutional, pooling and fully-connected layers.

Data processing: Data processing includes data transformation, rearrangement and quantization. Because the data format of the CNN parameters contained in .caffemodel file is not supported by FPGA, data should be transformed to .bin format. Besides, the arrangement of the data is different from that required in the hardware implementation. To improve the utilization of bandwidth, the data needs to be rearranged. In addition, it is very resource-consuming to use float point numbers which is used in GPU/CPU during CNN training phase. So there is a great need to transfer float point number into fixed point format. And given the big difference between the value of layer output and that of parameters, using the same fixed-point format to represent these two will result in a waste of the bit width, so we use a dynamic fixed-point quantization strategy. As illustrated in Figure 2, the principle of this strategy is that the parameters and the layer output are approximated by different fixed-point representations.

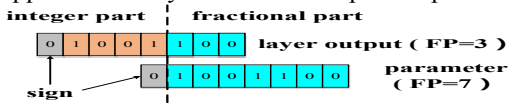

Figure 2. Example of dynamic quantization

And we use the equation (1) to determine the number of intger bit in the dynamic fixed point number. The purpose of this formula is to select enough integer bit to avoid overflow.

$$IP = \mathrm{ceil}(\log_2(\max x + 1)) \quad (1)$$

2.3 Hardware accelerating

For the FPGA platform used in this design, all information is stored in the SD card. To accelerate CNN on the hardware, two kinds of files is needed to be download to the SD card, which is .elf file and .bin files. The .elf file is composed of bitstream file which contains information about the hardware and some instructions which converted from .c file.

It is worth mentioning that the bitstream file need not be modified for the acceleration of different networks, because the CNN accelerator itself supports the acceleration of multiple networks. However, since the extraction information of different networks is different, the acceleration of each network requires a corresponding elf file.

3. CNN accelerator

3.1 Overall architecture

As illustrated in Figure 3, the system consists of two parts: processing system(PS) and programmable logic (PL).On the PS side, CPU is responsible for sending parameter configuration and start signal to PL side for each layer. The PL side is the CNN accelerator, which consists of the memory controller, on-chip buffers and processing elements (PEs). The memory controller contains a number of registers for storing configuration information for different layers, which is one of the keys to implement different networks with the same accelerator. PE comprises two unit, MAC and pool unit. MAC is used to support multiple convolution operations and pool unit can achieve commonly used pooling operations, making this design can achieve the acceleration of multiple networks.

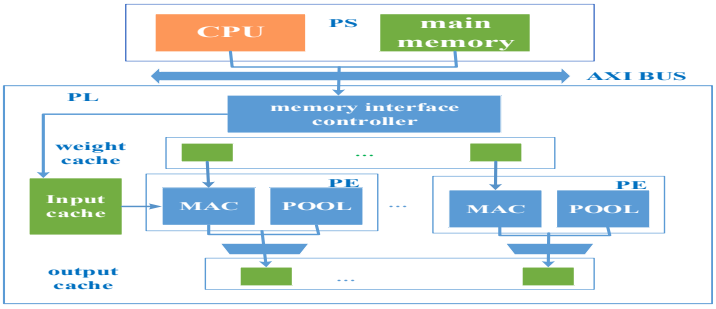

Figure 3. Syetem achitecture

3.2 MAC unit

We use the MAC unit which supports 3*3 convolutional operation as the basic computational unit. As deptied in Figure 4, the DSPs in the MAC can perform 3*3 multiplication operations in parallel. And the purpose of using input and weights buffers is to provide 3*3 data per cycle. This design can support 7*7 and 1*1 convolution as well. The 7*7 convolution can be achieved by assigning computaion tasks to 5 basic computational units and using an adder tree to integrate the results. For 1*1 convolution, it is achieved by setting the extra 8 inputs in the basic computational unit to zero.

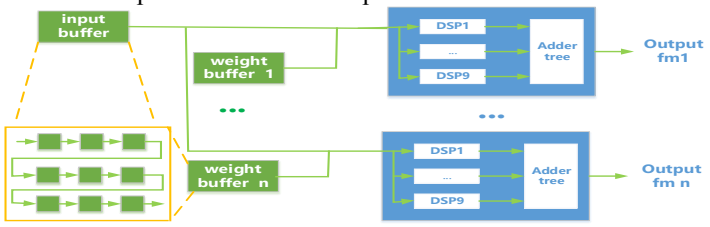

Figure 4. The structure of MAC

3.3 Pool unit

This accelerator supports two kinds of pooling oepration: average pooling and max pooling. Since the averge pooling can be achieved mainly by using MAC, this section mainly introduces the implementation of the max pooling.

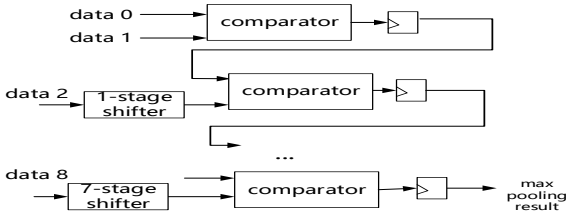

Figure 5. The structure of pool unit

We design the pool unit which supports 3*3 max-pooling as the basic pool unit. As showed in Figure 5, pool unit consists of 9 comparators and some registers, forming a pipeline structure.The pipleline strucutre allows pooling operation to be performed in parallel, which can improve the operation efficiency. For 2*2 max-pooling operation, it is achieved by setting the extra 5 input in the basic pool unit to zero.

4. Experimental results

The accelerator is designed by using HLS and is implemented on Xilinx Zynq-7000 SoC. Table 1 shows the comparison of the resource utilization of using float point and dynamic fixed-point number. By using fixed-point quantization, the consumption of all kinds of resources can be reduced. It is worth mentioning that not all platforms have as much resources as this SoC. Thus, we try to use fewer resources to make sure our design can be applied to most embedded platform.

Table 1. Comparison of resource utilization

	BRAM	DSP	FF	LUT
Available resource	1510	2020	554800	277400
Utilization (float point)	70%	43%	28%	112%
Utilization (16bit fixed point)	36%	18%	11%	42%

To evaluate our system, we have input CNN model files about ZynqNet [2] and SqueezeNet [5], and achieved acceleration of these networks on our design. Figure 6 is an input test image, which is a golden retriever and its corresponding label in ImageNet dataset is 207. The top-5 recognition probabilities generated by ZynqNet and SqueezeNet implemented on our system are showed in Figure7, which are both correct.

Figure 6. Test image

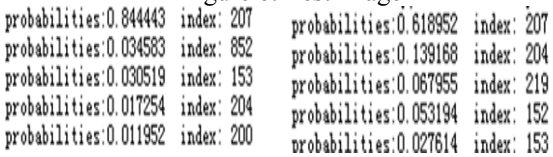

Figure 7. Recognition results of ZynqNet (left) and SqueezeNet (right)

In Table 2, we compare our accelerator with other desgins. As can be seen, the throughput is positively correlated with the number of DSP. As a consequence, the fewer resources consumed by our design would inevitalbly make the throughput lower than [4]. Nevertheless, compared with other works, this design consumes less power. And the performance density and power effiency can reach 0.054 GOPS/DSP and 5.24 GOPS/W, which is much higer than other desgins. In addition, since this accelerator can support multiple networks, for samll networks, such as Zynqnet, there will be functional redundacny. And that's why the performance of Zynqnet is lower than that of SqueezNet.

Table 2. Comparison with other works

	[4]	[3]	[2]	This work	
Platform	Virtex5 SX240t	Virtex7 VX 485t	Zynq XC 7Z045	Zynq-XC7Z100	
Quantization Strategy	16 bit fix	32bit float	32bit float	16bit fixed	
CNN size (GOP)	0.52	1.33	1.06	1.06	3.88
Throughput (GOPS)	16	61.62	6.71	6.63	19.81
Numbers of DSP in use	-	2240	739	364	364
Performance Density (GOPS/DSP)	-	0.027	0.009	0.018	0.054
Power(W)	14	18.61	12.00	3.78	3.78
Power effiency (GOPS/W)	1.14	3.31	0.56	1.75	5.24

*The penultimate column is the results of ZynqNet.
*The last column is the results of SqueezeNet.

5. Conclusion

This paper proposes a FPGA-based accelerator for the acceleration of multiple CNNs and presents an automatic mapping flow in which users only need to provide corresponding files to accelerate the network, which is based on the work of ZynqNet. In addition, we adopt a dynamic fix-point quantization strategy to reduce the resource consumption. This accelerator have achieved the implementation of ZynqNet and SqueezeNet, and the performance density and power efficiency can reach 0.054 GOPS/DSP and 5.24 GOPS/W,which is higher than most previous work.

References (follow the examples please)

[1] Krizhevsky A, Sutskever I, Hinton G E, Advances in neural information processing systems, pp.1097-1105 (2012).

[2] Gschwend, D, vol. Master ETH-Zurich: Swiss Federal Institute of Technology Zurich (2016).

[3] Zhang C, Li P, Sun G, et al., Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays,pp. 161-170(2015).

[4] Chakradhar S, Sankaradas M, Jakkula V, et al.,ACM SIGARCH Computer Architecture News, pp.247-257 (2010).

[5] Iandola F N, Han S, Moskewicz M W, et al, arXiv preprint arXiv:1602.07360 (2016).