An Energy-Efficient Systolic Pipeline Architecture for Binary Convolutional Neural Network

Baicheng Liu, Song Chen*, Yi Kang, and Feng Wu
School of Microelectronics, University of Science and Technology of China, Hefei 230027, China
*Email: songch@ustc.edu.cn

***Abstract*—Memory and computation cost limit the applications of Convolutional Neural Networks (CNN) on mobile devices. Binary CNN (BCNN) is a quantized neural network, which can reduce memory requirement and achieve multiplication-free computation. This paper focuses on BCNN. First, we proposed a hardware friendly CNN model to decrease the inference accuracy loss by eliminating the decimal part of each floating number. Second, we presented a fully pipelined on-chip BCNN architecture. The architecture has systolic data flow and an inter-layer pipeline, which ensures weights reuse and high throughputs. The results show that we achieve an inference accuracy of 99.04% for the MNIST dataset on the Pytorch platform and 98.91% in the hardware architecture, which means the inference accuracy loss is only 0.13%; the inference accuracy loss without this model is 0.41%. Besides, this architecture can achieve 23.08k: qfps and 353.5 GOP/s/W at 120MHz with small resource use while processing the BCNN.**

I. Introduction

CNN has been successfully used in image recognition [1], object detection [2], etc. However, significant demands of computation and memory are challenges to the widespread deployment of CNNs. To overcome these problems, Zhao et al. [3] built CNNs with structured and dense matrices, which can reduce independent parameters and decrease computation cost from $O(n^2)$ to $O(nlog(n))$. Glorot et al. [4] used the pruning method to prune unimportant weights. These methods still focused on full precision data. Quantization, where a few bits are used to represent floating numbers, is widely used to reduce the memory and computation demand. Li et al. [5] constrained all weights at -1, 0 and $+1$ (ternary weights), and thus each weight can be represented using 2 bits in the hardware architecture instead of 32 bits. Courbariaux et al. [6] proposed a training algorithm of BCNN, where the weights and activations of BCNN are -1 or $+1$, but the input images in the first layer and some intermediate data such as bias and other parameters in BCNN are still floating numbers.

Traditional Central Processing Units (CPUs) cannot provide sufficient parallelism in a single inference pass, and Graphic Processing Units (GPUs) consume too much power. Fortunately, FPGA/ASIC based accelerators balanced performance and power consumption well. Zhang et al. [7] presented a floating-point accelerator for AlexNet [1]. However, because floating-point operations require too many resources, the most state-of-the-art FPGA based CNN accelerators use fixed-point units. Liang et al. [8] adopted 16-bit fixed-point units in their work. Some works focused on exploring the benefits of using 1 bit for computations [9] [10], using high-level synthesis tools to design their BCNN accelerators. Jouppi et al. [11] used systolic data flow to accelerate full precision CNNs, but there is no doubt that the inter-layer pipeline will bring more parallelism.

In this work, we present a high-throughput and energy-efficient systolic pipeline architecture for BCNN. The architecture was implemented on Xilinx Zynq-7000 SoC ZC706.

The main contributions of this work are as follows.

- First, we propose to replace a large number of floating numbers in inference pass with integer values by directly removing the decimal part of model parameters, which can decrease inference accuracy loss compared with fixed-point decimal approximation in the calculation. For a BCNN model trained with the MNIST dataset, which has an accuracy of 99.04%, the inference accuracy of hardware architecture is 98.91%, where the accuracy loss decreases from 0.41% to 0.13%.
- Second, we present an energy-efficient architecture for binary convolutional neural network, which combines systolic data flow and inter-layer pipeline. The four-stage inter-layer pipeline ensures high throughput while the systolic data flow reuses weights and inputs in the convolutional layers. Besides, we used the 937 PE_R for row calculations and 195 PE_C for column calculations, which can provide sufficient parallelism.

The rest of this paper is organized as follows: Section II presents the basics of CNN and our hardware friendly BCNN model. Section III gives the details of BCNN architecture. Section IV compares the experimental results with related work. Section IV concludes this paper.

II. BCNN for Hardware Implementation

A. CNN Basics

A CNN model typically includes convolution (CONV), pooling (POOL), fully-connected (FC), and activation layer. CONV is used for extracting local features from input feature maps. POOL is the downsampling layer, and it can compress the feature maps into a smaller scale. Max-pooling (MP) is one type of POOL that picks the maximum value in a square region. FC is the fully-connected layer, where the $3-D$ input feature maps will be compressed into a $1-D$ feature map for linear transformation operation. The activation function, such as *ReLU*, *Sigmoid*, or *Tanh*, is used for increasing the nonlinear classification ability of a neural network. Because

978-1-7281-0735-6/19/$31.00 ©2019 IEEE

the input feature distribution of each layer can influence the convergence speed during training, Ioffe et al. [12] proposed Batch Normalization (BN) to speed up the training. BN can be shown as follows.

$$y = \gamma \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}} + \beta \tag{1}$$

B. Hardware Friendly BCNN Model

This work proposes a hardware friendly BCNN model. First, we binarize the input images because the binary image can match the *XNOR* operation in BCNN. Second, we convert the BN and *Sign* calculations into comparison and *XNOR* operations, with a simplified formula while there is almost no inference accuracy loss. Third, we remove the BN in the last layer with a small penalty of inference accuracy loss to achieve multiplication-free operations. Figure 2 shows the framework of the proposed BCNN for hardware implementation.

As mentioned in [6], the weights in BCNN are $+1$ or -1 and the value of features maps in each layer except for the last layer are also $+1$ or -1. To realize multiplication-free neural networks, S. Liang et al. [9] and Y. Umuroglu et al. [10] used the *PopCount*, which counts the number of 1 in a kernel, and $XNOR(\odot)$ operations. The convolution operation with the *PopCount* and *XNOR* operations in BCNN is as follows.

$$Y = 2 \times PopCount(IN \odot W) - L, \tag{2}$$

where IN are input feature maps, W is the filter, L is the kernel size, and Y is the convolution result. Figure 1 shows an example of BCNN convolution, where -1 is converted to 0, and the multiplication is replaced by *XNOR*.

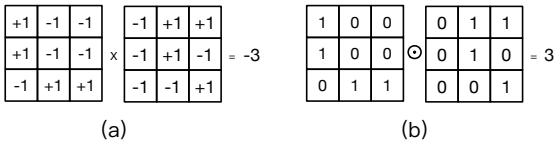

Figure 1. BCNN Convolution Computing.

In the proposed hardware friendly BCNN model, the binarized images can avoid using floating numbers in images while the simplified calculations involve only integral values in the inference stage, while the inference accuracy is kept unchanged. The formula is derived as follows.

$$Y = \frac{\gamma(2 \times \sum_{k=1}^{K^2} in \odot w - L + bias - \mu)}{\sqrt{\sigma^2+\varepsilon}} + \beta. \tag{3}$$

Formula 3 is the deformation of BN operation and it can be split and simplified as follows.

$$\begin{aligned} Y &= k_f \times (X - h_f), \\ k_f &= \frac{2 \times \gamma}{\sqrt{\sigma^2+\varepsilon}}, \\ h_f &= (L + \mu - bias - \beta \times \frac{\sqrt{\sigma^2+\varepsilon}}{\gamma}) \times 0.5, \\ X &= \sum_{k=1}^{k=K^2} in \odot w, \end{aligned} \tag{4}$$

where k_f and h_f are floating numbers, and X is an integer. Combining the *Sign* activation as follows:

$$O = \begin{cases} +1, & if \quad Y \geq 0, \\ -1, & if \quad Y < 0, \end{cases} \tag{5}$$

we modify Formula 4 as follows.

$$\begin{aligned} O &= k_b \odot (X > h_i), \\ k_b &= binary(k_f), \\ h_i &= int(h_f). \end{aligned} \tag{6}$$

In Formula 6, we use $>$ operation instead of $\geq$ operation to eliminate directly the decimal part of h_f. As a consequence, h_f is an integral value, and the k_b is a binary value.

Besides, we remove the last BN layer for the hardware implementation, which causes a small penalty (0.13%) of inference accuracy loss.

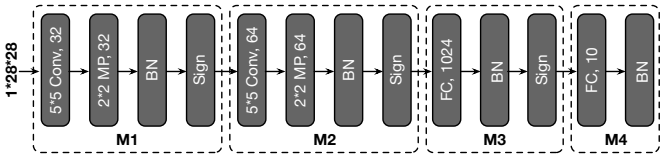

Figure 2. Excution BCNN Model.

As shown in Figure 2, M1 contains CONV1, MP1, BN1, and a Sign layer. M2 contains CONV2, MP2, BN2, and a Sign layer. M3 contains FC1, BN3, and a Sign layer. M4 only contains FC2 and BN4.

III. Hardware Architecture for BCNN

A. The Overall Architecture and Inter-Layer Pipeline

The whole architecture includes a four-stage inter-layer pipeline, where the four stages respectively correspond to the four parts M1, M2, M3, and M4 in the BCNN model. The architecture generates an output every N cycles, as shown in Figure 3.

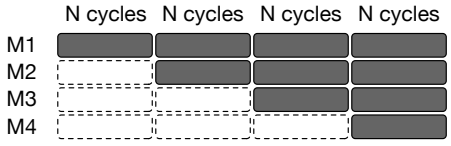

Figure 3. Inter-Layer Pipeline.

Figure 4 shows an overview of the pipelined hardware architecture. M1, M2, M3, and M4 all consist of PE_ARRAY, BC, MP, and PPB. All filters and K_H parameters would be processed off-line and stored in BRAM. The PE_ARRAY is

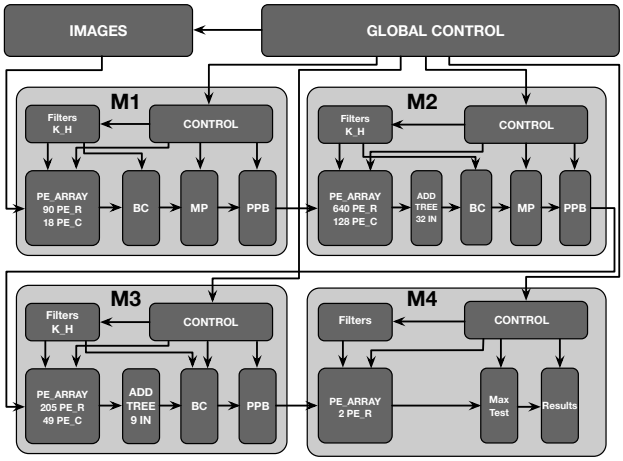

Figure 4. BCNN Architecture Outline.

for *XNOR* and *addition* operations. The BC calculates BN and *Sign* activation. The MP completes Max-Pool downsampling and picks up the maximum value in a 2×2 region. The PPB has two identical memory blocks for storing intermediate data. Furthermore, the 32-input adder-tree in M2 and the 9-input adder-tree in M3 accumulate inputs from different channels.

B. PE Architecture Details of Each layer

The PE_ARRAY of each layer consists of several PE_Rs and PE_Cs. The PE_R is for kernel row calculations while the PE_C accumulates the results from PE_R.

In CONV1, the PE_ARRAY has 90 PE_R and 18 PE_C. It can be time-multiplexed to form one 18×6 systolic array or three 6×6 systolic arrays in two phases. The architecture of the 6×6 systolic array is shown in Figure 5(a). It has 5 weight inputs and 10 feature map inputs. Because there exists weight-sharing and input-sharing in CONV1, the weights would be transmitted horizontally and each diagonal PE_R has the same feature map inputs in the 6×6 systolic array.

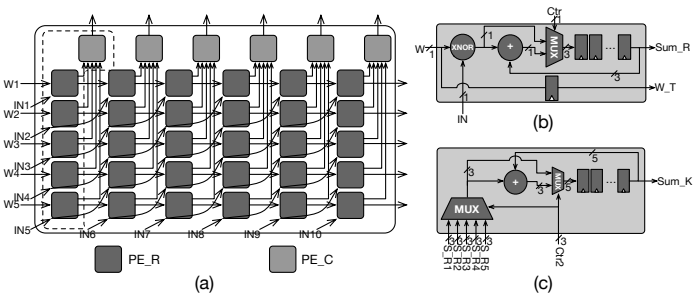

Figure 5. (a):A 6×6 systolic PE Array, (b):PE_R, (c):PE_C.

In CONV2, the PE_ARRAY has 640 PE_R and 128 PE_C, which forms 32 6×4 systolic PE arrays and each of them corresponds to one input channel from M1. Besides, the 6×4 systolic PE arrays in CONV2 also have weight-sharing and input-sharing. Furthermore, each result of one 6×4 systolic PE array in CONV2 corresponds to one input of the 32-inputs adder-tree.

In FC1, there is no weight-sharing. We achieve the calculation with the three-level systolic PE array, which is shown in Figure 6. Each three-level systolic PE array can complete 125-25-5-1 accumulation. FC1 needs 8 three-level systolic PE arrays and one 1×6 systolic PE array. And the results from these 9 PE arrays are accumulated in the 9-input adder-tree. In FC2, the calculation is simple, and only two PE_Rs are enough.

Figure 5(b) shows the architecture of the PE_R, where the W and IN are pins for receiving weights and feature maps, respectively. The W_T delivers weights to the adjacent right PE_R. The Sum_R delivers results to the corresponding PE_C. And the *Ctr* is a control signal for resetting the accumulation registers. Figure 5(c) shows the architecture of PE_C, where the *Ctr*2 is a control signal for selecting inputs from different PE_Rs, while the Sum_K delivers the results to the modules.

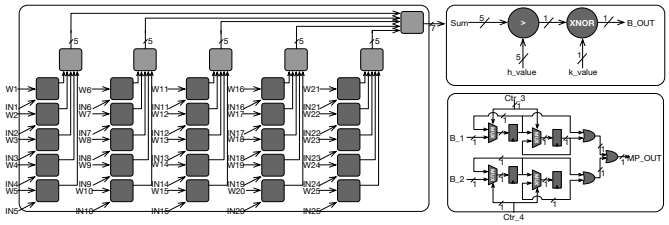

Figure 6. Left:three-level systolic PE Array for FC1, Upper-right:binarizing circuit (BC), Bottom-Right:Max-Pooling circuit.

The three-level systolic PE array has three computing levels, which is shown on the left side of Figure 6. The first level completes *XNOR* operations and accumulation of 125 groups of data inputs and weights in 5 cycles, using 25 PE_Rs, whose outputs are accumulated at the second level in 5 cycles. In third computing level, one PE_C is used to accumulate five results generated in the second level.

The architecture of MP and BC are respectively shown in Figure 6 Upper-Right and Bottom-Right. Because the order of max-pooling and binarizing does not affect the results, the MP can be moved to the behind of the BC in the hardware circuit for saving registers.

C. Parallelism and Data Reuse in This BCNN Architeture

Besides the inter-layer level pipeline structure, the parallelism and data reuse in the presented architecture are as follows.

- **PE level parallelism**. Each layer has more than one PE and all PEs work in parallel and form systolic arrays.
- **Input reuse**. Inputs are reused in CONV1 and CONV2 respectively because diagonal PE_R in these two layers has the same inputs.

- **Weight reuse**. Weights in CONV1 and CONV2 are transmitted horizontally. Therefore, each PE_R reuses the previous weight from the adjacent left PE_R.

IV. Experimental Results

The BCNN model in this paper is trained on the Pytorch platform under the CentOS system with one piece of Nvidia Tesla K40, and the BCNN architecture is described with Verilog HDL and implemented with Xilinx Zynq-7000 SoC ZC706 with Vivado 2018.2.

Table 1. Experimental Results Comparision

	[9]	[10]	[13]	Ours
Platform	Stratix-V 5SGSD8	ZC706	Kintex-7 160T	ZC706
Accuracy	98.24%	98.40%	98.29%	98.91%
Parameters(Mb)	9.55	2.78	5.56	1.06
Precision	1	1	2	1
On Chip Power	–	0.8	1.84	0.63
Feequency(MHz)	150	200	200	120
BRAM(18Kb)	2210	114.5	–	85
LUT/ALM	182301 ALM	5636 LUT	–	28932 LUT
FPS	294k	12.2k	255.1k	23.08k
FPS/BRAM	133.0	106	–	272
GOP/s	5904.4	71.0	448.7	222.7
GOP/s/W	–	88.8	243.9	353.5

Table 1 shows comparisons between this work and the related works. Liang et al. [9] and Umuroglu et al. [10] focused on the BCNN while Alemdar et al. [13] focused on the ternary neural network. The dataset used in Table 1 is MNIST. Experimental results show that, compared with others, we got the highest inference accuracy at 98.91% in the hardware circuit. The FPS in our BCNN architecture is only 23.08k, but we use much fewer resources, such as BRAM. Therefore, our FPS/BRAM is 272 and higher than [9] [10]. Moreover, the GOP/s is not very high in our architecture, but the on-chip power is only 0.63W. Consequently, the architecture in this work is more energy-efficient compared with those in [10] [13]. Also, compared with traditional CNNs, the BCNN in this paper can save storage at least 15 times. It can completely achieve multiplication-free computation and high energy efficiency with a small penalty of accuracy loss.

To demonstrate the effectiveness of the BCNN model, we also trained a CNN with the same framework as BCNN. The activation function of CNN is *Tanh*, and the inference accuracy is 99.45%. The inference accuracy of the CNN with *Tanh* function is 0.36% higher than 99.09% of the BCNN, where the input image pixels in the first layer and BN parameters are floating numbers. When we used the 12-bit fixed-point numbers for the input images and BN parameters, the inference accuracy of the BCNN model is only 98.68%, which means the inference accuracy loss is 0.41%. Fortunately, with the hardware friendly BCNN model in Section II, we got an inference accuracy of 99.04% on the Pytorch platform and 98.91% in the BCNN hardware circuit, which means the inference accuracy loss decreases from 0.41% to 0.13%.

V. Conclusion

In this paper, we proposed a method to convert a trained BCNN model with floating parameters to a BCNN model having only integral parameters, while the inference accuracy loss decreases from 0.41% to 0.13%. Moreover, we proposed a fully on-chip systolic pipeline BCNN architecture for the converted BCNN model, which combines inter-layer pipeline and systolic data flow. The architecture can achieve 23.08k fps at 120MHz and energy efficiency of 353.5 GOP/s/W.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (NSFC) under grant Nos. 61874102 and 61732020, and the Fundamental Research Funds for the Central Universities under grant No. WK2100000005.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[3] L. Zhao, S. Liao, Y. Wang, Z. Li, J. Tang, and B. Yuan, "Theoretical properties for neural networks with weight matrices of low displacement rank," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 4082–4090.

[4] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[5] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[7] 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*, pp. 161–170.

[8] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 152–159.

[9] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[10] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74.

[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[13] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient ai applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2547–2554.