Binary Convolutional Neural Network on RRAM

Tianqi Tang, Lixue Xia, Boxun Li, Yu Wang, Huazhong Yang
Dept. of E.E., Tsinghua National Laboratory for Information Science and Technology (TNList),
Tsinghua University, Beijing, China
e-mail: yu-wang@mail.tsinghua.edu.cn

***Abstract*—Recent progress in the machine learning field makes low bit-level Convolutional Neural Networks (CNNs), even CNNs with binary weights and binary neurons, achieve satisfying recognition accuracy on ImageNet dataset. Binary CNNs (BCNNs) make it possible for introducing low bit-level RRAM devices and low bit-level ADC/DAC interfaces in RRAM-based Computing System (RCS) design, which leads to faster read-and-write operations and better energy efficiency than before. However, some design challenges still exist: (1) how to make matrix splitting when one crossbar is not large enough to hold all parameters of one layer; (2) how to design the pipeline to accelerate the whole CNN forward process.**

In this paper, an RRAM crossbar-based accelerator is proposed for BCNN forward process. Moreover, the special design for BCNN is well discussed, especially the matrix splitting problem and the pipeline implementation. In our experiment, BCNNs on RRAM show much smaller accuracy loss than multi-bit CNNs for LeNet on MNIST when considering device variation. For AlexNet on ImageNet, the RRAM-based BCNN accelerator saves 58.2% energy consumption and 56.8% area compared with multi-bit CNN structure.

I. Introduction

Convolutional Neural Networks (CNNs) have achieved great performance in various recognition tasks, including image classification [1], video tracking [2] and natural language processing [3]. At the same time, larger computational intensity and higher bandwidth are required than traditional non-CNN models [4]. The emerging RRAM-based Computing System (RCS) has been considered as a promising solution for future CNN accelerators [5]–[8], where the RRAM-based crossbar can not only store the weight parameters of CNN models but also be used as the matrix-vector multiplier. In this way, the energy cost for data transfer is reduced and less bandwidth is required. Moreover, thanks to the crossbar-level parallelism, it also reduces the running time complexity from $O(n^2)$ to $O(1)$.

However, the current resistance precision of RRAM device is limited [9], and the impact of writing variation and reliability problem increases with the bit-level amounts of RRAM device [10]. On the other hand, the interfaces between the analog RRAM-based crossbars and the digital peripheral units take up most of the area and power consumption, which makes the whole RRAM-based computing system not so efficient as expected [5]. Therefore, the high precision of data and weights in state-of-the-art CNNs becomes the main challenge for RRAM-based implementation.

Recently, researchers in the field of machine learning have demonstrated that Binary CNNs (BCNNs) achieve satisfying recognition accuracy on ImageNet dataset [11], [12]. BCNNs use binary weights and data when processing the forward propagation. It provides a promising solution to break the high precision limits in current RRAM-based CNN accelerator design. Faster read-and-write operations and better energy efficiency can be achieved by exploiting the binary characteristics of BCNN.

Some challenges still exist in the RRAM-based BCNN accelerator design when the network scale increases. First, the length of crossbar column is not large enough to hold all the weight parameters of one Convolution (Conv) kernel in large BCNNs like VGG [1]. Therefore, the operation of matrix splitting is inevitable, and the high-cost interfaces are still required for the intermediate data in splitting. Second, the size of intermediate data between layers increases rapidly with the network scale and introduces large overhead.

In this paper, an RRAM crossbar-based BCNN accelerator is proposed. The contributions of this paper include:

- In our BCNN accelerator design, the matrix splitting problem is well discussed when mapping weight parameters to RRAM. Thanks to the line buffer introduced for intermediate data buffering, a pipeline strategy is proposed for system efficiency.
- The robustness under device variation of BCNN on RRAM is demonstrated. For LeNet on MNIST, binary CNN achieves 0.75% on 3bit RRAM devices in the case of device variation.
- Experimental results show that BCNN saves 58.2% of energy and 56.8% of area consumption are saved when using BCNN for AlexNet on ImageNet.

The rest of this paper is organized as follows: Section II introduces the related background and the motivation of our work; Section III proposes the RRAM-based BCNN accelerator design, especially the pipeline design; Section IV uses the case studies of LeNet on MNIST, and AlexNet on ImageNet to analyze recognition accuracy, area and energy efficiency; and Section V shows the conclusion.

II. Preliminaries and Motivation

A. CNN

A typical CNN consists of a number of different kinds of layers that run sequentially, i.e. the output of the previous layer is the input of the next layer. The input/output of one layer is named "feature map" while the parameters of one layer are called "weights". In a standard CNN structure, cascaded Convolutional (Conv) layers (optionally followed by Neuron layers, Max Pooling layers, Normalization layers) are followed by one or more Fully-Connected Layers [1].

Conv Layer can be expressed as in Eq. 1:

$$f_{\text{out}}(x,y,z)=\sum_{i=0}^{h-1}\sum_{j=0}^{w-1}\sum_{k=0}^{C_{\text{in}}-1} f_{\text{in}}(x+i,y+j,k)\cdot c_z(i,j,k) \quad (1)$$

where i, j, and k are the spatial coordinate of three-dimensional (3-D) matrix input feature map $\mathbf{F}_{\text{in}}$ with the size of $H_{\text{in}}\times W_{\text{in}}\times C_{\text{in}}$; x, y, and z are the coordinate of output feature map $\mathbf{F}_{\text{out}}$ with the size of $H_{\text{out}}\times W_{\text{out}}\times C_{\text{out}}$; $\mathbf{C}_z$ is the z^{th} Conv kernel with the size of $h\times w\times C_{\text{in}}$; and there are C_{out} kernels in one Conv Layer. In this way, all the Conv kernel parameters in one layer form a 4-D blob with the size of $(h,w,C_{\text{in}},C_{\text{out}})$. Sliding stride s is used for jumping some pixels and reduce the computation amount, while zero padding is introduced when the convolution is processed at the edge of feature maps and the pixels are not enough for one whole Conv kernel.

Neuron Layer is attached after the Conv Layer which makes a nonlinear one-by-one mapping ($y=\mathrm{f}(x)$). Binary Neurons are used

This work was supported by 973 Project 2013CB329000, National Natural Science Foundation of China (No. 61622403, 61373026, 61261160501), Brain Inspired Computing Research, Tsinghua University. And we gratefully thank Dr. Xudong Fei from Huawei Co. for the discussion.

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

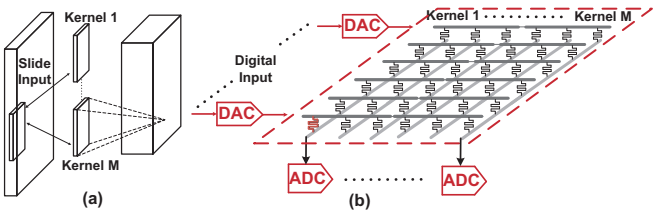

Fig. 1: Structure of the RRAM-based Crossbar.

in our BCNN system as proposed in BinaryNet [11]. The forward function can be expressed in Eq. 2:

$$y = \begin{cases} 1, & x > 0 \\ -1, & x \leq 0 \end{cases} \tag{2}$$

Max Pooling Layer is cascaded after the non-linear neurons. It picks the largest element among the neighboring area of input feature map in order to reduce the data amount and keep the local invariance.

Fully-Connected (FC) Layer can be expressed as in Eq. 3:

$$f_{\text{out}}(y) = \sum_{x=0}^{C_{\text{in}}-1} f_{\text{in}}(x) \cdot c(x, y) \tag{3}$$

where x is the index of the 1-D input feature map vector $\mathbf{F}_{\text{in}}$ with the length of C_{in}; y is the index of output feature map vector $\mathbf{F}_{\text{out}}$ with the length of C_{out}; and the 2-D weight matrix $\mathbf{C}$ is in the size of $C_{\text{in}} \times C_{\text{out}}$.

Batch Normalization (BN) Layer [13], which solves the problem of internal covariate shift, has been introduced in most state-of-art network models, especially the binary network models, e.g. BinaryNet [11], XNOR-Net [12]. The operation of BN Layer can be abstracted into a linear one-to-one mapping. The parameters are well-trained in the training process.

B. RRAM Device, Crossbar Array and Crossbar Interface

An RRAM device is a passive two-port element with multiple resistance states, and multiple devices can be used to build the crossbar structure. When the "matrix" is represented by the conductivity of the RRAM devices and the "vector" is represented by the input voltage signals, the RRAM crossbar is able to perform as the analog matrix-vector multiplier (MVM). Specifically, the relationship between the input and output signals can be expressed as in Eq. 4 [6]:

$$i_{\text{out}}(k) = \sum_{j=0}^{N-1} g(k, j) \cdot v_{\text{in}}(j) \tag{4}$$

where $\mathbf{V}_{\text{in}}$ is the vector of input voltage (denoted by $j = 0, 1, ..., N-1$), $\mathbf{I}_{\text{out}}$ is the the vector output current (denoted by $k = 0, 1, ..., M-1$), and $\mathbf{G}$ is the conductivity matrix of the RRAM device. Taking advantage of the natural "multiplication and merging" function of the crossbar structure, the RRAM crossbar can implement the Conv kernels and FC Layers in analog mode with high speed, small area, and low power [6].

For FC Layers, the weight matrices are directly mapped to the RRAM crossbars [5]. While for Conv Layers, one Conv kernel is mapped to one RRAM column, and different columns in one crossbar correspond to different Conv kernels [6], as shown in Fig. 1.

C. Motivation

Compared with the well-trained network which uses floating-point weight parameters and feature maps on CPU/GPU platforms, the RRAM devices and the crossbar interfaces can only support limited bit levels.

1) Limited Bit Levels of RRAM Devices: To the best of our knowledge, only 7-bit weights [9] are currently available for the single RRAM device. However, state-of-the-art fixed-point CNNs require 8 or 16 bit precision weights [4], [14]. As a result, multiple RRAM devices have to be used for representing one number [7], [8] and large energy overhead are introduced. Moreover, the multi-bit devices suffer from more variation and reliability problems than single-bit devices [10], which decreases the recognition accuracy of computing accelerator. Therefore, the precision of RRAM resistance levels limits both the energy efficiency and the accuracy of RRAM-based computational system.

2) Limited Bit Levels of Crossbar Interfaces: Since the crossbars work in the analog mode, interfaces are needed for the transformation between the digital signals in the nearby computing units and the analog signals in the crossbar-based MVM. There are two kinds of interfaces in RRAM-based computational system. On the one hand, the interfaces between RRAM crossbar and CPU, i.e. the input interface in the first layer and the output interface in the final layer, are required. On the other hand, the interfaces between RRAM crossbars in different layers are required in CNNs. This is because CNNs are not full-connected networks. Therefore, each RRAM crossbar need to process multiple cycles with different inputs, and the temporary results of each cycle need to be buffered until all the neighboring results are obtained. The detailed function will be illustrated in Section III. An intuitive choice is to use DAC/ADCs as the interfaces, but huge overheads are introduced by high-precision ADC/DACs. Li [5] pointed out that 8-bit ADC/DACs contribute to more than 85% of the area and power consumption of the whole RCS.

Therefore, it will contribute a lot to energy efficiency if achieving a well-trained network model with low bit-level weight parameters and feature maps, especially the binary ones.

D. Challenges of RRAM-based BCNN

Some recent papers have already shown that completely binary CNNs (BCNNs) are achievable if the 1-bit quantization is processed in training. Courbariaux [11] proposes a sampling method which trains the binary network together with the floating-point network; while Rastegari [12] proposes the BinaryWeight by minimizing the binary quantization loss while training. Moreover, the weights and the feature maps are also binarized.

Based on these results, **in this paper, we propose an RRAM crossbar-based BCNN accelerator, achieving higher energy efficiency compared with multi-bit CNNs.** However, when the network scale increases, two main challenges limit the energy efficiency of the accelerator.

1) Splitting Interface Overhead: Splitting is required when the size of the Conv kernels is larger than the length of crossbar column. State-of-the-art RRAM crossbars only achieve the column length of 512 [6]. Crossbar of such size is not able to hold some large Conv kernels, e.g. Conv kernel with the size 4608 ($= 3\times3\times512$) in VGG16 model. Therefore, the high-cost interfaces are still required because the intermediate data in splitting need high precision. PRIME [7] and ISAAC [8] discussed matrix splitting method for full-precision CNNs by using high-precision ADC/DACs, so the energy efficiency is still limited. Considering that BCNN provides the potential for low-precision crossbar interfaces, a BCNN-specific low-precision splitting structure is in demand.

2) Buffer Overhead: Since RRAM crossbar uses multiple inputs in the same cycle, the processed data between layers can only be buffered by registers instead of RAMs. Therefore, thousands of registers and corresponding multiplexers are required for large networks. ISAAC [8] gives a rough design for pipelining the Conv

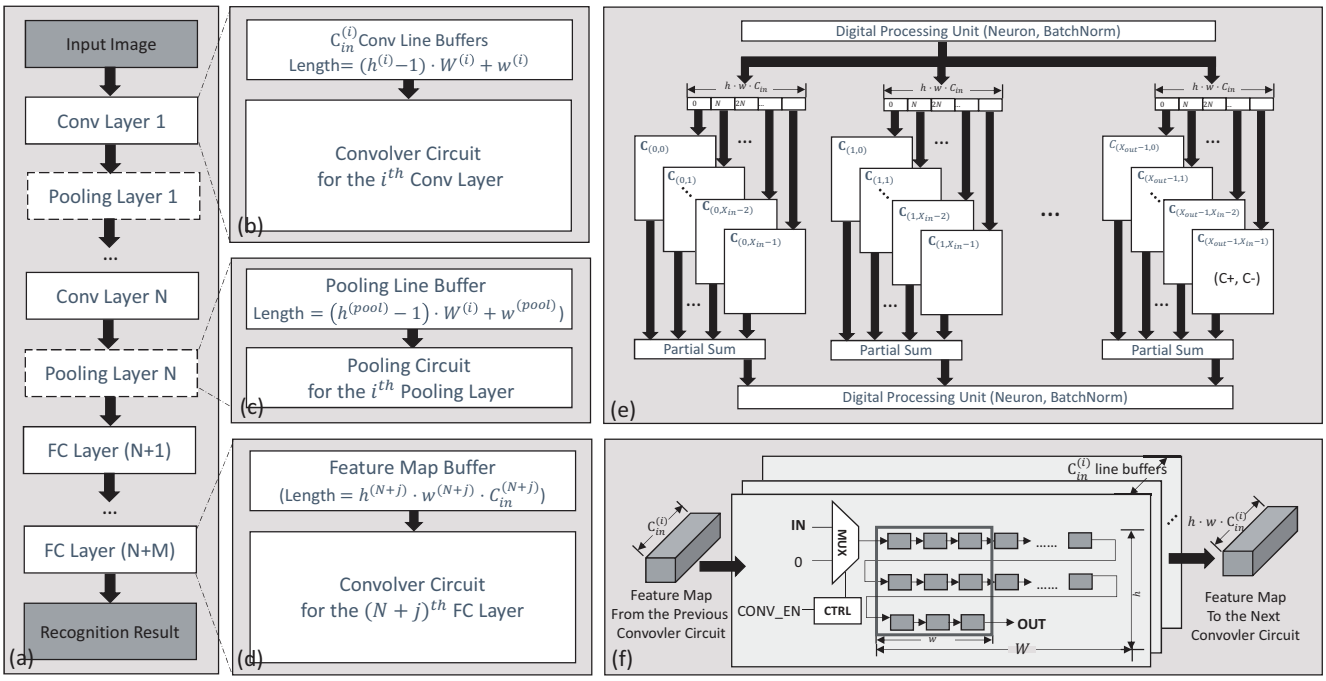

Fig. 2: (a) Overall Structure of the RRAM-based BCNN Accelerator: "N Conv Layers + M FC Layers" with the Input Image and the Output Recognition Result, and each Conv Layer Optionally Followed by the Pooling Layer; (b)-(d) The Dataflow of the "Conv Layer", "Pooling Layer", and "FC Layer"; (e) The Convolver Circuit for one Conv/FC Layer on RRAM-based Platform; (f) The Conv Line Buffers.

operation of different layers where weight duplication is introduced for balance the load of the pipeline, but a throughout discussion on pipeline implementation is still in lack. Since only a few registers are used in each cycle, there exists the parallel potential between layers to reduce the buffer size while boosting the processing speed. As a result, a pipeline design between layers is necessary for both CNN and BCNN accelerators.

III. RRAM-based BCNN Accelerator Design

As shown in Fig. 2 (a), the whole accelerator is made up of a series of **Conv Layer**s cascaded by a series of **FC Layer**s. The **Pooling Layer** module optionally follows the Conv Layer. The data paths for the Conv, Pooling, and FC Layer are respectively shown in Fig. 2 (b)-(d). Each layer consists of its own **Input Buffer**, and the **Computing Circuit**.

- *Computing Circuit*: For Conv and FC Layers, the **Convolver Circuit** is made up of the RRAM crossbar-based MVMs, as shown in Fig. 2 (e). Some digital peripheral units, including circuits for neurons and batch normalization, are also placed in front of or at back of the crossbar groups. For Pooling Layers, the computing circuit can be easily implemented as the multi-input "OR" gate in the BCNN design.
- *Input Buffer*: For Conv and Pooling Layers, the operation of sliding window exists. In this way, the structure of **Line Buffer** (LB) is introduced for intermediate data buffering and fetching, as shown in Fig. 2 (f). For FC Layers, the regular buffers are used since nearby layers are fully connected.

In this section, we first discuss the design of the Convovler circuit and matrix splitting in III-A; then discuss the design of intermediate data buffering and the implementation of pipeline in III-B.

A. Convolver Circuit: The Problem of Matrix Splitting

Considering both the large energy cost of writing operation and the endurance limit of RRAM device [15], reusing RRAM crossbar by repeatedly R&W operation is not available. Since the high area density is an important advantage of RRAM, all the Conv kernels can be mapped onto the crossbars in the Convolver Circuit of the corresponding layer. In this way, each output channel is able to get one output element in one processing cycle if enough data have been fed into this layer's Line Buffers by the former Conv Layer. However, matrix splitting is necessary for large Conv kernels, as discussed in Section II-D, which is the same for large FC matrices.

1) Column Splitting: If the crossbar column count (M) is smaller than the Conv kernel count (C_{out}, the same with the output channel count) of this layer, then C_{out} Conv kernels are split into $\text{X}_{\text{out}}^{(\text{Conv})}$ groups of RRAM crossbars, as shown in Eq. 5. Copies of the input feature maps with the size of one Conv kernel ($h \cdot w \cdot C_{\text{in}}$) are sent to each groups of crossbars.

$$\text{X}_{\text{in}}^{(\text{Conv})} = \lceil \frac{h \cdot w \cdot C_{\text{in}}}{N} \rceil, \ \text{X}_{\text{out}}^{(\text{Conv})} = \lceil \frac{C_{\text{out}}}{M} \rceil \tag{5}$$

2) Row Splitting: If the cross-point count (N) in one RRAM column is smaller than the Conv kernel size ($h \cdot w \cdot C_{\text{in}}$), then the elements of one Conv kernel are split into $\text{X}_{\text{in}}^{(\text{Conv})}$ groups of RRAM crossbars, also as shown in Eq. 5. Moreover, the input feature map is also split into $\text{X}_{\text{in}}^{(\text{Conv})}$ groups and the partial sum is achieved from each group of crossbars. An adder tree needs to be cascaded after the crossbar groups in order to merge the $\text{X}_{\text{in}}^{(\text{Conv})}$ partial sums.

The intermediate data before adder tree still use high precision. However, since the cascaded digital functions, i.e. non-linear function and BN, are monotone increasing functions, the 1-bit quantization can be merged with these functions by changing the threshold and output data range. Therefore, the result after addition can also be only 1 bit, which provide the potential for using lower-precision intermediate data for addition. Based on this observation, we reduce the ADC precision into 4 bit, which can save large amount of overhead especially when the splitting amount is large.

3) Signal Splitting: The resistance of RRAM device is positive, i.e. it is unable to represent negative values. In this way, it is necessary to

map one weight matrix onto a crossbar pair: one crossbar for positive weights $(+1)$, the other for negative ones (-1).

B. Line Buffer & Pipeline Implementation

The sliding window exists in the Conv Layers. Data dependency analysis shows that the convolver circuit can awake (A) from sleep (S) once the input data of the Conv kernel size is achieved. In this way, the structure of the Line Buffer is introduced for the following reasons: First, much fewer registers are used for data buffering since it is unnecessary to buffer the whole input feature maps; second, with Line Buffer introduced in every Conv/Pooling Layer, a pipeline can be implemented, which makes the forward process much faster than computing the Conv Layers in the one-by-one mode. And it is the same for the Pooling Layers.

As the Pooling Layer is optionally followed by the Conv Layer, there exist "Conv-Conv" and "Conv-Pooling-Conv" two modes for nearby layer relationship. Here, we use the dataflow behavior our experiment of CIFAR-10 on VGG11 as case study to show the line-buffer-based pipeline implementation.

1) Conv-Conv: For the Conv Layers in VGG11, the "kernel size" is set as 3×3; the "stride" is set as 1, and zero padding is introduced in order to keep the input and output feature map as the same size. When the feature map feeds in following the row-major order, the Line Buffer of each channel only needs $(h-1)\cdot(W+p)+w$ registers. In the initial periods of a layer, zero padding in the length of $(W+p)$ and the first row of $x^{(k)}_{1,:}$ are sent sequentially into the k^{th} Conv Layer's Line Buffer before T_0. And in these cycles, the Convolver Circuit of the k^{th} is in the sleep (S) mode. Finally at the T_0 cycle, $x^{(k)}_{2,1}$ is sent into the Line Buffer, as the dataflow shown in Fig. 3. In the next cycle, the input Line Buffer shown in Fig. 2(f) is fulfilled by data, and therefore k^{th} Conv Layer starts at time T_1. Additionally, at the end of the layer's computation, $(W+p)$ cycles are needed for computing the last row just like the initial cycles.

A main challenge for the Conv-Conv pipeline design is the sleep control for the "line feed" problem. When the computation of a row is accomplished, the input data need to be changed from the end of current line to the front of the next line, which means at least $(w-1)$ data (usually we have $w>3$) in the next layer need to be prepared. However, for the line-buffer-based pipeline design, the input field shown in Fig. 2(f) is invalid during the preparing cycles, e.g. $(x^{(k)}_{i-3,1}, 0, x^{(k)}_{i-2,W}; x^{(k)}_{i-2,1}, 0, x^{(k)}_{i-1,W}; x^{(k)}_{i-1,1}, 0, x^{(k)}_{i,W})$. In these cycles, the Convolvor of the k^{th} layer is also in the S mode; while for the Line Buffer of the $(k+1)^{\text{th}}$ layer, there is no valid input. Fortunately, we find that the zero padding of next Conv Layer can just exploit this cycle. And in the next cycle, i.e. Cycle $T_{m(W+1)+2}$ $(m=0,1,...)$, the Convolver of the k^{th} layer recovers to awake (A); while the Convolver of the $(k+1)^{\text{th}}$ layer begins to sleep (S) for line feed. In this way, the "line feed" problem is solved by utilizing the extra sleep cycle in each layer for zero padding, and the works in fully pipelined parallelism without waiting. Based on this structure, **we achieve the theoretical fewest cycle amount for Conv-Conv pipeline connections**.

2) Conv-Pooling-Conv: For the Pooling Layers in VGG11, the "kernel size" is set as 2×2; the "stride" is set as 2, and zero padding does not exist. As stride is larger than 1, the pooling circuit will work for one row when every s rows are ready. In Conv-Pooling pipeline, just as shown in Fig. 4, the k^{th} Pool Circuit sleep from Cycle T_{W+3} to T_{2W+1}. For the awaken row, the pooling circuit will work once every s data are sent into the pooling Line Buffer. As shown in Fig. 4, the k^{th} Pool Circuit awakens one cycle and sleeps one cycle from Cycle T_3 to T_{W+1}. Although the problem of "line feed" also exists, the sleep cycles can be hidden into with the "sleep row", and zero

	T_0	T_1	T_2	...	T_W	T_{W+1}	T_{W+2}	T_{W+3}	
$Conv_K$ LB Input	$x^{(k)}_{2,1}$	$x^{(k)}_{2,2}$	$x^{(k)}_{2,3}$	...	$x^{(k)}_{2,W}$	0	$x^{(k)}_{3,1}$	$x^{(k)}_{3,2}$	...
$Conv_K$ Xbar	S	A	A	...	A	A	S	A	...
$Conv_{K+1}$ LB Input	0	$x^{(k+1)}_{1,1}$	$x^{(k+1)}_{1,2}$	...	$x^{(k+1)}_{1,W-1}$	$x^{(k+1)}_{1,W}$	0	$x^{(k+1)}_{2,1}$	...
$Conv_{K+1}$ Xbar	S	S	A	...	A	A	A	S	...

Fig. 3: The Dataflow of Conv-Conv. The first line shows the Line Buffer's input data of previous Conv Layer in each cycle, and the third line shows the input data of next Conv Layer. The second and forth line show whether the Convolvers are Awake (A) or Sleep (S).

	T_0	T_1	T_2			...	T_W	T_{W+1}	T_{W+2}	T_{W+3}	...	T_{2W+1}
$Conv_K$ LB Input	0	$x^{(k)}_{3,1}$	$x^{(k)}_{3,2}$	$x^{(k)}_{3,3}$	$x^{(k)}_{3,4}$	...	$x^{(k)}_{3,W}$	0	$x^{(k)}_{4,1}$	$x^{(k)}_{4,2}$	...	$x^{(k)}_{4,W}$
$Conv_K$ Xbar	A	S	A	A	A	...	A	A	S	A	...	A
$Pool_K$ LB Input	$y^{(k)}_{1,W}$	X	$y^{(k)}_{2,1}$	$y^{(k)}_{2,2}$	$y^{(k)}_{2,3}$	...	$y^{(k)}_{2,W-1}$	$y^{(k)}_{2,W}$	X	$y^{(k)}_{3,1}$	...	$y^{(k)}_{3,W-1}$
$Pool_K$ Circuit	A	S	S	A	S	...	S	A	S	Sleep until new data coming		
$Conv_{K+1}$ LB Input	0	0	0	$x^{(k+1)}_{1,1}$	0	...	0	$x^{(k+1)}_{1,\frac{W}{2}}$	0	0		

Fig. 4: The Dataflow of Conv-Pooling. The first line shows the Line Buffer's input data of previous Conv Layer in each cycle; the third line shows the input data of next Pooling Layer; the second and forth line show whether the Convolvers are Awake (A) or Sleep (S).

padding is not introduced in Pooling Layer, as shown in Cycle T_1 and Cycle T_{W+2}. While for the Pooling-Conv Line Buffer of the next layer, it is just the turn of zero padding in this cycle like Conv-Conv pipeline.

Finally, in the pipeline implementation, the total cycle amount for one complete forward process is shown as in Eq. 6.

$$T_{\text{pip}} = (W^{(1)}+p)\cdot(H^{(1)}+2p) + \sum_{i\in\text{Conv}}^{i>1}(W^{(i)}+p) + \sum_{i\in\text{Pool}} 1 + \sum_{j\in\text{FC}} 1 \tag{6}$$

$(W^{(1)}+p)\cdot(H^{(1)}+2p)$ is the computation cycle amount for the first Conv Layer. After that, once the cascaded layer is a Conv Layer, $(W+p)$ cycles are needed for computing the last row. Otherwise, only one extra cycle is needed for computing the last pixel of next Pooling Layer, or to perform a FC Layer. The pipelined cycle amount is much fewer than the straight forward layer-by-layer design whose cycle amount is:

$$\sum_{i\in\text{Conv}}(W^{(i)}+p)\cdot(H^{(i)}+2p) + \sum_{i\in\text{Pool}}(W^{(i+1)}H^{(i+1)}) + \sum_{j\in\text{FC}} 1 \tag{7}$$

IV. Experimental Results

A. Experiment Setup

In this section, the models of LeNet and AlexNet are respectively used on the dataset of MNIST and ImageNet. The multi-bit model is achieved by dynamically quantizing [4] the well-trained floating-point model into 8 bits; while the BCNN model is achieved by following the training algorithm of BinaryNet [11]. Single crossbar size is set as $(M,N)=(128,128)$. If one crossbar pair is not large enough to store all parameters of one layer, parameter splitting is done as shown in III-A. For the multi-bit CNN RRAM-based accelerator, 8-bit RRAM devices and 8-bit interfaces are introduced. While the BCNN system is implemented as proposed in Sec. III: The same bit-level RRAM devices are used as in multi-bit CNN system; and the

TABLE I: Error Rate of LeNet on MNIST: Device Variation Effects Under Different Weight Bit-Levels

Weight Bit Level	RRAM Bit Level[a]	RRAM Used in Full Bit-level Mode		RRAM Used in Binary Mode	
		No Variation	With Variation	No Variation	With Variation
8 bit	7 bit	0.58%	0.58%	0.73%	0.74%
6 bit	5 bit	0.60%	0.59%		0.75%
4 bit	3 bit	0.80%	1.21%		0.75%
2 bit	1 bit	90.67%	89.10%		0.86%

[a] The bit-level of RRAM devices is 1bit less than that of the weight parameters because of the signal splitting.

TABLE II: Amount and Processing Count of Computing Units, Interfaces and Buffers

Module	Layer	Amount	Processing Count
RRAM cell	Conv	$(h \cdot w \cdot C_{\text{in}}) \cdot C_{\text{out}} \cdot X_{\text{out}} \cdot X_{\text{out}}$	$H_{\text{out}} \cdot W_{\text{out}}$
DAC	Conv	$(h \cdot w \cdot C_{\text{in}}) \cdot X_{\text{out}}$	$H_{\text{out}} \cdot W_{\text{out}}$
SA&ADC	Conv	$C_{\text{out}} \cdot X_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
Feature Map Buffer	Conv	$h \cdot w \cdot C_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
Line Buffer	Conv	$h \cdot W_{\text{in}} \cdot C_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
Line Buffer	Pooling	$h \cdot W_{\text{in}} \cdot C_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
RRAM Cell	FC	$C_{\text{in}} \cdot C_{\text{out}} \cdot X_{\text{out}} \cdot X_{\text{out}}$	1
DAC	FC	$C_{\text{in}} \cdot X_{\text{out}}$	1
SA&ADC	FC	$C_{\text{out}} \cdot X_{\text{in}}$	1
Feature Map Buffer	FC	C_{in}	1

TABLE III: Area and Power Cost of Circuit Elements

	Area	Power(mW)
1T1R RRAM device	$(1+\frac{W}{L}) \cdot 3F^2$	0.052[b]
0T1R RRAM device	$4F^2$	0.06[b]
8bit DAC	$3096T$[a] [16]	30 [17]
Sense Amplifier	$244T$ [16]	0.25 [18]
8bit ADC	$2550T$+$1k\Omega(\approx 450T)$ [16]	35 [19]
4bit ADC	$72T$ [20]	12 [20]
8bit SUB	$256T$	2.5×10^{-6}[c]
1bit ADC	$244T$	1.73 [21]
32bit SRAM Cache	-	0.064[c]

[a] $T = W/L \cdot F^2$, where W/L =3, and the technology node F =45nm.
[b] The power consumption of RRAM cell is estimated by $V_{\text{avg}}^2 g_{\text{avg}}$, where $g_{\text{avg}} = \sqrt{g_{\text{on}} g_{\text{off}}}$ [22]
[c] The energy consumption of digital arithmetic logics and memory access refer to the energy table under 45nm CMOS technology node [23]. The system clock is assumed to be 100MHz, which is determined by the speed of ADC/DACs and the latency of RRAM crossbar [24].

TABLE IV: Energy and Area Estimation of Different RRAM-based Crossbar PEs

Database	Performance	CNN	BCNN	Saving
MNIST	Energy(uJ/img)	18.39	13.55	26.3%
	Area (mm^2)	0.054	0.060	-11.1%
ImageNet	Energy(uJ/img)	5444.85	2275.34	58.2%
	Area (mm^2)	21.25	9.19	56.8%

interface is binary when matrix splitting is not necessary, 4 bits when necessary.

In this section, we first explore the effect of variation under different weight bit levels; then a comparison on system efficiency is made between BCNNs on RRAM and multi-bit CNNs on RRAM.

B. Accuracy: Effects of Device Variation Under Different Bit-Levels

Variation exists when mapping weight parameters to RRAM devices since it is one conductance range (not a specific conductance value) that represents one fixed-point number. When one RRAM device is able to represent N bits, i.e. 2^N conductance ranges represent 2^N fixed-point weights respectively. For the k$^{\text{th}}$ conductance range, $g_{(k)}$ represents the center conductance, and $(g_{(k)} - \Delta g, g_{(k)} + \Delta g)$ represents the conductance range, i.e. the device variation δg ranges from $(-\Delta g, \Delta g)$. According to previous physical measurement results [25], we assume that the variation range Δg is the same for each conductance range. When the RRAM device is used in the **binary mode**, only two conductance ranges are picked from 2^N ones. In this way, the expectation of $(\delta g/g)$ can be smaller than in the case that 2^N ranges are all in use (we just name it as **full bit-level mode**), thus introducing less computing error for matrix-vector multiplication.

LeNet on the MNIST dataset is demonstrated as case study to show the effects of device variation under different weight bit-levels. Without considering device variation, a precise mapping is made from quantized fixed-point weight parameters to RRAM conductances in the full bit-level mode. In this way, the increasing recognition error rate mainly results from the quantization error. While in the binary mode, the recognition performance keeps the same for RRAM of different bit-levels when neglecting device variation, though the recognition error is a bit higher than that of full bit-level mode in the case of 7bit and 5bit RRAM, as listed in Table. I.

When considering device variation, the recognition performance in the binary mode shows better robustness: In binary mode, device variation introduces less than 0.01% error rate increase in case of 3bit (or larger bit-level) RRAM; while in full bit-level mode, the recognition performance in 3bit RRAM becomes worse than that in binary mode due to larger effect of device variation.

C. Area and Energy Estimation Under Different Bit-Levels

Network models of LeNet on MNIST and AlexNet on ImageNet are demonstrated in the area and energy estimation. Moreover, we also profile the area and energy distribution among different circuit elements and among different layers on AlexNet. In our estimation, the crossbar-based computing units and the buffers are considered; while the consumption of interconnections are neglected. The amount and the processing count of each module are listed as in Table. II. Because of the sliding window operation, modules in Conv layers process $H_{\text{out}} \cdot W_{\text{out}}$ times in one forward process. The area and power consumption of each circuit elements are listed in Table. III.

The area and energy estimation is shown in Table. IV. The experimental results show that BCNN on RRAN saves 58.2% of energy and 56.8% of area consumption for AlexNet on ImageNet compared with multi-bit CNN. Whether for binary or multi-bit CNNs, the output interface takes up the most part on energy and area consumption. The area and energy distribution is shown in Fig. 5. In terms of area distribution among all layers, the FC layers take up the most part since the FC layers take up most of the the weight parameter of the whole CNN. While in terms of energy distribution, the Conv layers take up the most part. This is because the sliding window of each Conv layer has to sweep through the whole featue map in multiple process counts; but FC layers only process once. Comparing the area and energy distribution between BCNN and multi-bit CNN, the overhead of input interface is mostly saved; meanwhile, the overhead of output interface is saved when the bit level of the partial sum decreases in the case of matrix splitting.

V. Conclusion

In this paper, an RRAM crossbar-based accelerator is proposed for BCNN forward process. Moreover, the special design for BCNN is well discussed, especially the matrix splitting problem and the pipeline implementation. The robustness of BCNN on RRAM under device variation are demonstrated. Experimental results show that BCNN introduces negligible recognition accuracy loss for LeNet on MNIST. For AlexNet on ImageNet, the RRAM-based BCNN accelerator saves 58.2% energy consumption and 56.8% area compared with multi-bit CNN structure.

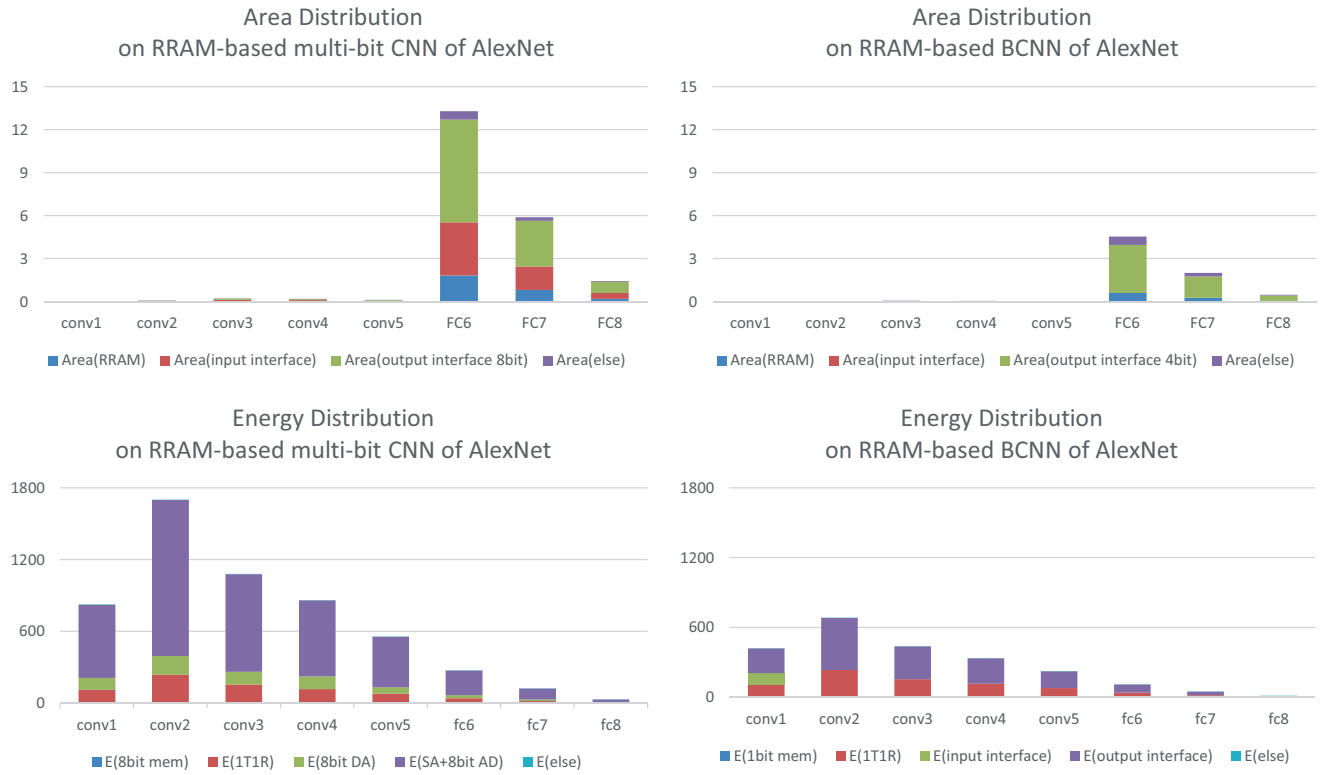

Fig. 5: Power and Area Distribution on AlexNet

References

[1] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] J. Fan *et al.*, "Human tracking using convolutional neural networks." *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1610–1623, 2010.

[3] A. Karpathy *et al.*, "Deep visual-semantic alignments for generating image descriptions," in *Computer Vision and Pattern Recognition*, 2015.

[4] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*, 2016, pp. 26–35.

[5] B. Li *et al.*, "Merging the interface: Power, area and accuracy co-optimization for rram crossbar-based mixed-signal computing system," in *DAC*, 2015, p. 13.

[6] L. Xia *et al.*, "Selected by input: Energy efficient structure for rram-based convolutional neural network," in *DAC*, 2016.

[7] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, vol. 43, 2016.

[8] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ISCA*, 2016.

[9] F. Alibart *et al.*, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[10] R. Degraeve *et al.*, "Causes and consequences of the stochastic aspect of filamentary rram," *Microelectronic Engineering*, vol. 147, pp. 171–175, 2015.

[11] M. Courbariaux *et al.*, "Binarized neural network: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[12] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," *arXiv preprint arXiv:1603.05279*, 2016.

[13] S. Ioffe *et al.*, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[14] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM Sigplan Notices*, vol. 49, no. 4, 2014, pp. 269–284.

[15] Y. Y. Chen *et al.*, "Understanding of the endurance failure in scaled hfo 2-based 1t1r rram through vacancy mobility degradation," in *IEDM*, 2012, pp. 20–3.

[16] R. St. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 505–516, 2014.

[17] J. Proesel *et al.*, "An 8-bit 1.5 gs/s flash adc using post-manufacturing statistical selection," in *CICC*, 2010, pp. 1–4.

[18] S. Gupta *et al.*, "Simulation and analysis of sense amplifier in submicron technology."

[19] S. Y.-S. Chen *et al.*, "A 10b 600ms/s multi-mode cmos dac for multiple nyquist zone operation," in *2011 Symposium on VLSI Circuits-Digest of Technical Papers*, 2011.

[20] S. S. Chauhan, S. Manabala, S. Bose, and R. Chandel, "A new approach to design low power cmos flash a/d converter," *International Journal of VLSI design & Communication Systems (VLSICS)*, vol. 2, no. 2, p. 10C108, 2011.

[21] Siddharth *et al.*, "Comparative study of cmos op-amp in 45nm and 180 nm technology," *Journal of Engineering Research and Applications*, vol. 4, pp. 64–67, 2014.

[22] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.

[23] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[24] S.-S. Sheu *et al.*, "A 4mb embedded slc resistive-ram macro with 7.2 ns read-write random access time and 160ns mlc-access capability," in *ISSCC*, 2011.

[25] S. R. Lee *et al.*, "Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory," *Digest of Technical Papers - Symposium on VLSI Technology*, pp. 71–72, 2012.