REBOC: Accelerating Block-Circulant Neural Networks in ReRAM

Yitu Wang*†, Fan Chen‡, Linghao Song‡, C. -J. Richard Shi*†§, Hai "Helen" Li‡, Yiran Chen‡
*State Key Laboratory of ASIC and Systems, Fudan University, Shanghai, China
†Institution of Brain-inspired Circuits and Systems, Fudan University, Shanghai, China
‡Department of Electrical and Computer Engineering, Duke University, Durham NC, U.S.A.
§Department of Electrical and Computer Engineering, University of Washington, Seattle WA, U.S.A.
Email: ytwang16@fudan.edu.cn, {fan.chen, linghao.song, hai.li, yiran.chen}@duke.edu, cjshi@uw.edu

***Abstract*—Deep neural networks (DNNs) emerge as a key component in various applications. However, the ever-growing DNN size hinders efficient processing on hardware. To tackle this problem, on the algorithmic side, compressed DNN models are explored, of which block-circulant DNN models are memory efficient and hardware-friendly; on the hardware side, resistive random-access memory (ReRAM) based accelerators are promising for in-situ processing of DNNs. In this work, we design an accelerator named REBOC for accelerating block-circulant DNNs in ReRAM to reap the benefits of light-weight models and efficient in-situ processing simultaneously. We propose a novel mapping scheme which utilizes *Horizontal Weight Slicing* and *Intra-Crossbar Weight Duplication* to map block-circulant DNN models onto ReRAM crossbars with significant improved crossbar utilization. Moreover, two specific techniques, namely *Input Slice Reusing* and *Input Tile Sharing* are introduced to take advantage of the circulant calculation feature in block-circulant DNNs to reduce data access and buffer size. In REBOC, a DNN model is executed within an intra-layer processing pipeline and achieves respectively $96\times$ and $8.86\times$ power efficiency improvement compared to the state-of-the-art FPGA and ASIC accelerators for block-circulant neural networks. Compared to ReRAM-based DNN accelerators, REBOC achieves averagely $4.1\times$ speedup and $2.6\times$ energy reduction.**

I. INTRODUCTION

DNNs play a key role in various application domains including computer vision, natural language processing, and speech recognition. DNN applications involve millions to billions of matrix-vector multiplication (MVM) and therefore require intensive computational and memory resources. The ever-growing DNN model size has exacerbated this phenomenon, making the deployment of DNNs on hardware challenging, especially on end devices where the available computation and memory resources are limited and the power budget is constrained.

For efficient DNN processing, compressed DNN models are explored [1] to achieve light-weight models at the algorithm level. However, the random weight sparsity introduced in the model compression stage and the consequent poor access locality hinder efficient computation on processing devices. Therefore hardware-friendly DNN model compression is proposed, of which block-circulant DNNs [2] are a promising candidate solution for efficient hardware deployment. From a hardware perspective, data movement has become a major performance and energy bottleneck in data-intensive DNN applications, and tremendous efforts have been made to memory-centric accelerator design [3]–[6]. Compared with traditional CMOS-based designs [3], [4] where logic is distributed near or inside the memory system, the implementations based on the emerging ReRAM technology [5], [6] typically utilize the memory itself as processing elements (PEs) by taking advantage of their dual capabilities of both computation and storage.

Block-circulant DNNs are adapted as the target models by state-of-the-art customized accelerators such as CirCNN [2] on FPGA/ASIC and STICKER-T [7] in ASIC. These two works heavily rely on the Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) for computation. However, compared to real number multiplication, multiplication of FFT requires four real multiplications and two real additions. In addition, in order to perform multiplication or convolution for one layer, an FFT needs to be performed before the multiplication between the weight and the feature map, and an additional IFFT is required after the multiplication, which introduces significant computational and memory overhead.

In this work, we make an important observation that *block-circulant attribute does not necessarily require FFT/IFFT*, which distinguishes our work from prior arts [2], [7]. In general, the block-circulant DNN only favors FFT/IFFT in the sequential processing scenario where only one processing thread or unit is available. For DNN accelerator architectures where massive parallelisms for MVM are available, especially for ReRAM-based accelerators, we need a new paradigm for processing block-circulant DNNs rather than the naive invocation of FFT/IFFT. We propose REBOC, the *first* accelerator for processing block-circulant DNNs in ReRAM. REBOC utilizes the massive parallel MVMs in ReRAM rather than FFT/IFFT. We make the following contributions:

- We propose a novel mapping scheme with a shift method, which utilizes *Horizontal Weight Slicing* and *Intra-Crossbar Weight Duplication* to map the block-circulant DNN model onto ReRAM crossbars with high crossbar utilization.
- We propose *Input Slice Reusing* and *Input Tile Sharing* to take advantage of the circulant calculation feature in block-circulant DNN models to reduce data access and buffer size. We also design an intra-layer pipeline for achieving high processing throughput.
- We evaluate REBOC against state-of-the-art accelerators. REBOC achieves respectively, $96\times$ and $8.86\times$ power efficiency improvement, compared to the state-of-the-art FPGA and ASIC accelerators for block-circulant neural networks. Compared to ReRAM-based DNN acclerators, REBOC also achieves averagely $4.1\times$ speedup and $2.6\times$ energy reduction.

978-3-9819263-4-7/DATE20/©2020 EDAA

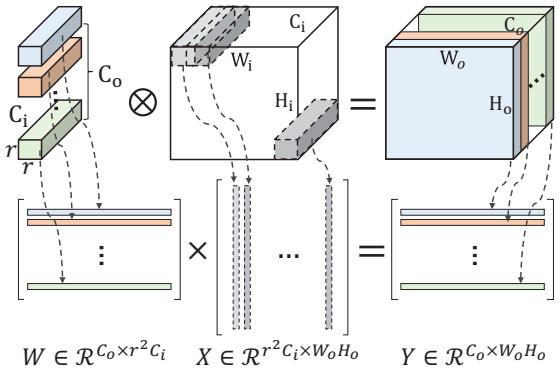

Fig. 1. Convertion to matrix multiplication through Toeplitz.

II. Background and Motivation

A. Basics of Deep Neural Networks

A typical deep neural network (DNN) model is constructed by cascading multiple convolutional (CONV) layers for feature extraction and then placing one or more fully connected (FC) layers at the final stage for classification. A non-linear activation layer, such as sigmoid unit, rectified linear unit (ReLU), leaky ReLU unit, etc., is optionally located after a linear CONV layer to enhance the representation capability. In general, the CONV and FC layers account for $> 90\%$ [6] of computing and storage resources and are therefore the primary target of accelerator design. The basic computation in the FC layer can be formulated as matrix-vector multiplication (MVM): $\boldsymbol{y} = \phi(\boldsymbol{W}\boldsymbol{x})$, where $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ is the weight matrix, which connects input vector $\boldsymbol{x} \in \mathbb{R}^n$ to output vector $\boldsymbol{y} \in \mathbb{R}^m$, $\phi(\cdot)$ is an element-wise nonlinear activation function. Note that we omit the bias for simplicity.

In CONV layers, parameters are shared spatially by sliding a set of weight kernels across the width and height of the input feature map (FP). Hence the number of parameter is significantly reduced. As demonstrated in Fig. 1, CONV operations are converted into matrix multiplication $\boldsymbol{Y} = (\boldsymbol{W}\boldsymbol{X})$ through the Toeplitz matrix supported by current software [8], [9] and most of the customized [5], [6] DNN acclerators.

B. Circulant DNN Model

A circulant DNN model based on circulant projection [10] can significantly compress the conventional DNN model with controllable storage saving and computation reduction. The key idea is to approximate a matrix operation $\boldsymbol{y} = \phi(\boldsymbol{W}\boldsymbol{x})$ with $\hat{\boldsymbol{y}} = \phi(\boldsymbol{W}^{\boldsymbol{r}}\boldsymbol{x})$ where $\boldsymbol{W}^{\boldsymbol{r}} = circ(w)$ is a circulate matrix defined by a vector $w = (w_1, w_2, \cdots, w_k)$. We illustrate the concept of circulant projection in Fig. 2 and refer interested readers to [10]. CirCNN [2] further proposes a finer-grained block-circulant method to improve efficiency by pre-classifying the original matrix into sub-matrices and applying compression thereafter. In general, the compression ratio and resulting accuracy deduction are determined by block size k. In this work, we adopt the block-circulant method in [2] as the target circulant DNN compression model.

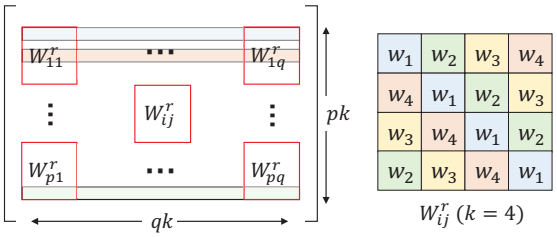

Fig. 2. Block-circulant weight matrix.

C. Related Work and Motivation

Previous works [2], [7] first train DNN weights using the block-circulant pattern. MVM operations can be then transformed into FFT in the frequency domain by leveraging the circulant convolution theorem [2]. At the end of each matrix computation, an IFFT is needed to convert the result back to the original domain. Thanks to the small-footprint FFT/IFFT kernels, promising energy and performance improvement have been demonstrated in both FPGA [2] and ASIC [7] designs.

In this work, we explore the opportunity of implementing circulant DNN models with the emerging ReRAM memory by taking advantage of its in-situ computing capability [5], [6], [11]. In contrast to previous FFT-based approaches, we identify that *block circulant attribute does not necessarily require FFT/IFFT*, especially for ReRAM-based designs which feature massive parallel MVM processing capability. In ReBoc, MVM operations are computed directly, without the need of converting back and forth between the original domain and the frequency domain, since the FFT-based approach introduces huge computational and memory overhead, and even worse, *it does not always leads to actual computation reduction*. More specifically, each complex multiplication in FFT requires four real multiplications and two real additions with higher precision. Therefore, for CONV layers, which typically require smaller block size n (< 32) to preserve reasonable accuracy [2], [7], FFT-based approach actually increases the computation complexity rather than reducing it.

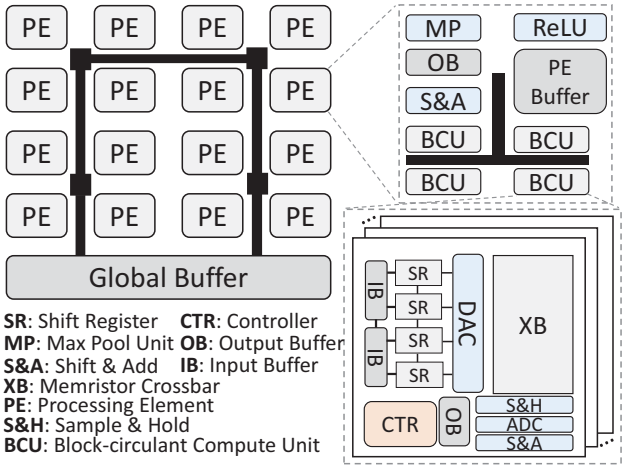

Fig. 3. ReBoc overall architecture.

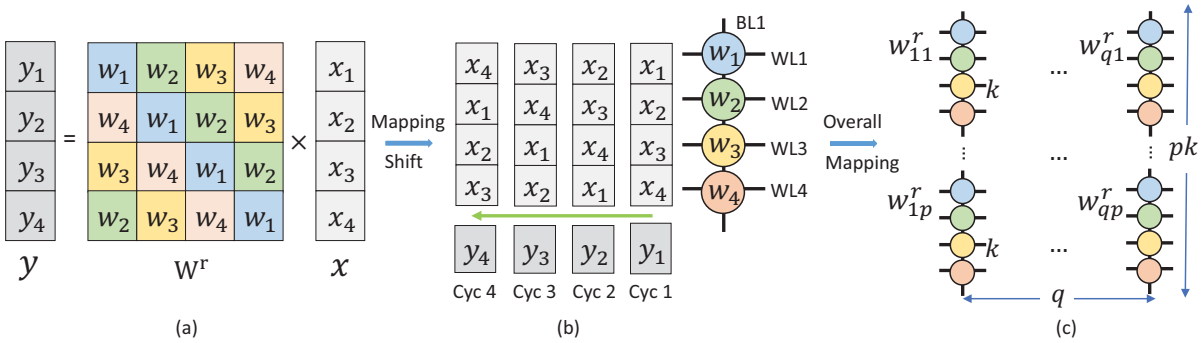

Fig. 4. Shift method and mapping scheme.

III. Design

Fig. 3 illustrates the top-level diagram of ReBoc, which consists of a set of identical processing engines (PEs) organized in a 2D array and connected through an on-chip mesh. Each PE is composed of 4 block-circulant compute units (BCUs) connected with a shared bus, an output buffer (OB), max pool unit (MP), shift and add (S&A) and rectified linear unit (ReLU). Input FP and intermediate data are temporally stored in a PE buffer within each PE. Note that the size of each PE buffer is $16\times$ smaller than the eDRAM employed in ISSAC [5] thanks to the unique data reuse system in ReBoc. Each BCU is comprised of 8 ReRAM crossbars (XBs) and each XB consists of an ADC, a set of 1-bit DAC and other necessary peripheral circuitry. In particular, we dedicate a series of shift registers (SRs) and inter-connected input buffers (IBs) to support the shift-based MVM and data reuse scheme. The number of circulant shifts and detailed data reuse approach are orchestrated by a controller (CTR).

A. Shift Method and Mapping Scheme

Previous ReRAM-based NN accelerators [5], [6], [11], [12] flatten kernel weights and then map them onto crossbars. Input FPs are re-organized accordingly and served as the wordline (WL) voltage. Note we assume that one crossbar cell can store one weight and defer the discussion on practical crossbar implementations in the following section. A naïve mapping following the previous method is shown in Fig. 4 (a). Clearly, the circulant characteristic is not exploited. Therefore, we propose a novel mapping scheme to store the representative vectors of each block-circulant matrix into a crossbar array, which leads to k time storage reduction.

As illustrated in Fig. 4 (b), the weight representative vector $w^r = [w_1, w_2, w_3, w_4]$ of $\boldsymbol{W^r}$ is mapped to a crossbar column. To avoid expensive ReRAM writes [13] caused by w shifts, the weight representative vector w remains stationary in our design, and we circularly shift the input vector accordingly. More specifically, we store the input vector $x = [x_1, x_2, x_3, x_4]$ in the SR and apply it onto WLs when computation begins. Therefore, we obtain y_1 at the end of the first cycle. In the next clock cycle, we shift the elements of x circularly by one position in the SR and then multiply it by w to get the second element y_2. In this example, we obtain the output vector $y = [y_1, y_2, y_3, y_4]$ in four cycles. We can see that the number of shifts depend on the size of block-circulant matrix k.

As explained in Section II, the computation in the FC and CONV layers are represented as matrix-vector multiplication and the weight matrix can be further compressed into $p\times q$ block-circulant sub-matrices with a block size of $k\times k$ as illustrated in Fig. 2. We denote the representative vector of block-circulant matrix W^r_{ij} as w^r_{ij}. For each sub-matrix, we follow the previous described mapping methods, and the overall mapping is conceptually demonstrated in Fig. 4 (c). To maintain the spatial position in the original matrix, $w^r_{i:}$ and $w^r_{:j}$ are respectively mapped onto the same wordline(s) and bitline of a ReRAM crossbar. In general, we map $(p\times k) \times (q\times k)$ weights onto a $(p\times k)\times q$ ReRAM crossbar with k time memory reduction. Here, k can be greater or equal to 128 for FC layers, while CONV layers typically utilize a k smaller than 32 [2], [7].

Optimization of Block-circulant Mapping Scheme: As mentioned above, block-circulant compression can effectively reduce memory requirements by a factor of k. However, when taking the actual size of a crossbar into account, the resulting matrix will occupy a narrower crossbar width and ultimately lead to resource underutilization. In the following analysis, we assume that the crossbar size is 128×128 and each ReRAM cell can store 2 bits. For a CONV layer with 256 filters and a block size of 16, only 25% of the crossbar is utilized. To address this challenge, we propose *Horizontal Weight Slicing* to allocate different bit slices of a weight to adjacent bitlines in the same crossbar. This forced adjacency can effectively improve crossbar utilization. In addition, we explore the trade-off between hardware and performance by *Intra-Crossbar Weight Duplication*.

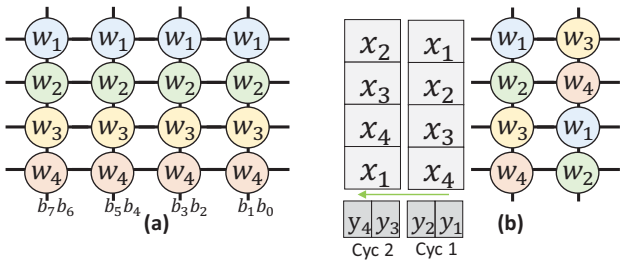

Fig. 5. Optimization of block-circulant mapping scheme (a) Horiziontalized weights (b) Intra-crossbar weight duplication.

Horizontal Weight Slicing *(HWS)*. Given the limited precision of ReRAM cells and analog circuits, each matrix element is mapped to multiple crossbars, and the resulting partial sums are combined by subsequent shift-and-add (S&A) reduction networks in previous works [5], [6]. However, in order to mitigate low bitline utilization caused by block-circulant compression, we devise a horizontal weight slicing scheme that reaps the benefits of increased utilization and shared input FP at the same time. As an example, *HWS* is shown in Figure 5(a), every 2-bit matrix coefficient is sequentially mapped onto a crossbar cell in the same wordline. Note that in this design we use 2-bit cells, the technique can be generalized to many-bit ReRAM devices. The side benefit for *HWS* is that input FPs can be shared without being distributed to and buffered in multiple crossbars, reducing the communication overhead and input buffer requirement.

Intra-Crossbar Weight Duplication *(ICWD)*. The trade-off between hardware and performance is a common adjustment factor exploited in ReRAM-based accelerator design [6], [11] to realize spatial parallelism. However, circulant DNN models require a modified parameter replication approach to support their circulant execution model. A conceptual example of the proposed Intra-Crossbar Weight Duplication is demonstrated in Figure 5(b). Instead of duplicating the original weight representative vector w^r, a circulant form of w^r is stored in the same XB. In this example, we have a duplication degree of $g = 2$, resulting in $2\times$ execution time reduction at the cost of $2\times$ hardware overhead. *ICWD* can be utilized with a flexible g considering a specific DNN topology, compression ratio and available hardwre resource.

B. Input Data Reusing and Sharing

Analysis of Redundant Data Access to External Memory: Due to the overlapped computation of CONV layers and the mapping scheme proposed above (weight representative vectors are stationary on crossbars), most of the FP elements need to be accessed for more than once as input. Usually, we need to reorganize the 3D FP into a 2D matrix with many duplicated FP elements as shown in Fig. 1. However, the reorganization brings two problems. First, it needs $(r^2H_oW_o - H_iW_i)C_i$ external memory space to store the duplicated FP elements [14]. Second, it leads to more access to the external memory, which consumes much energy. To solve this problem, we propose two methods: *Input Slice Reusing* and *Input Tile Sharing*. We use the example shown in Fig. 6 to explain. In this example, the width and height of the FP are both 5, and the number of input channels is 128. The filter size is 3×3 and the stride is 1. We denote $X(i,.,.), i = 1,2,3,4,5$ as the input tiles, and $X(i,j,.), j = 1,2,3,4,5$ as the input slices. Each input slice is composed of 128 FP elements. The relationship between input tiles and input slices is also illustrated in Fig. 6. It is obvious that the 2D matrix has duplicated input tiles and in each input tile, there are also duplicated input slices.

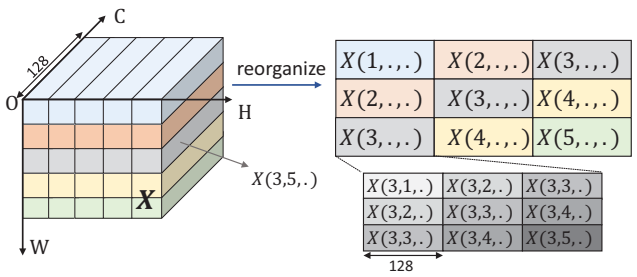

Fig. 6. Input feature map reorganization.

Input Slice Reusing *(ISR)*. REBOC uses input buffers (IBs) to implement *ISR*. As Fig. 7(a) shows, at the beginning, input slices 1, 2 and 3 are, respectively, allocated in three connected IBs. After SRs fetch the input slices from IBs, input slice 2 and 3 are transferred to the top IBs for data reuse. Meanwhile, input slice 1 is exhausted from the top IB and input slice 4 is allocated into the bottom IB. Finally, input slice 2 is exhausted and input slice 5 is allocated, and input slices 3 and 4 are reused. In this way, redundant data access and extra memory space requirement of the input slices in each input tile can be avoided.

Input Tile Sharing *(ITS)*. Fig. 7(b) illustrates that for one CONV layer whose filter width and height are both 3, we can partition the crossbars into 3 groups – XB_G1, XB_G2 and XB_G3 – to store the CONV layer weight representative vectors. Once one input tile is fetched, it is shared by all the corresponding crossbar groups in processing. For example, once the gray input tile $X(3,.,.)$ in Fig. 7(b) is accessed from the PE buffer, it can be used as input of XB_G1, XB_G2 and XB_G3 at the same time. Therefore, after completing computation of input tiles with their matched weight representative vectors in crossbar groups, we can get corresponding partial sums (*psum*s). We can get $psum_{3,1}$, $psum_{3,2}$ and $psum_{3,3}$ in the case of the gray input tile $X(3,.,.)$. As shown in Fig. 7(b), with $psum_{1,1}$, $psum_{2,2}$ and $psum_{3,3}$, we can get next layer's input tile $X^{next}(1,.,.)$ (before pooling and ReLU). Although in some cases we should store part of $psum$s in the global buffer because of the limited buffer resource in each PE, *ITS* reduces the redundant access to external memory of the duplicated input.

C. Intra-layer Pipeline Design

Pipeline has been widely exploited in most state-of-art ReRAM-based accelerators [5], [6], [11]. In REBOC, we support intra-layer pipeline (*ILP*), where different parts of different layers from the same frame can be processed simul-

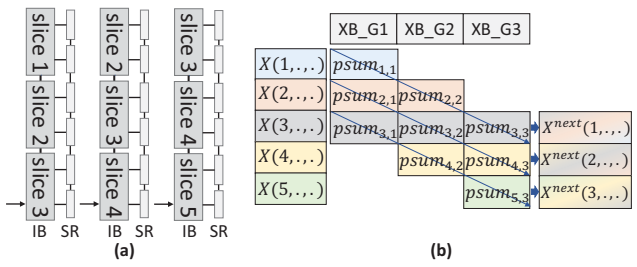

Fig. 7. Input data reusing and sharing (a) Input slice reusing (b) Input tile sharing.

TABLE I
MAPPING AND INTERMEDIATE STORAGE OF DIFFERENT ACCELERATORS

Accelerators	Mapping Scheme	Intermediate storage
Atomlayer [12]	Row-disjoint	$H_o{\cdot}W_o{\cdot}C_o$
ISSAC [5]	Overlap	$(H_o{\cdot}(r-1)+r){\cdot}C_o$
Pipelayer [6]	Overlap	$H_o{\cdot}W_o{\cdot}C_o{\cdot}r^2$
ILP	WRV	$H_o{\cdot}C_o{\cdot}r^2$
ILP+ISR	WRV	$H_o{\cdot}(r-1){\cdot}C_o + r{\cdot}C_o$
ILP+ITS	WRV	$r{\cdot}C_o$
ILP+ISR+ITS	WRV	C_o

TABLE II
NETWORK TOPOLOGY

Dataset	NN model	MAC bit-precision	Block-size
MNIST	LeNet	FC:2bit Conv:4bit	FC:128 Conv:16
ImageNet	AlexNet	FC:4bit Conv:4bit	FC:128 Conv:16
DataDJI	TinyYolo	Conv:6bit	Conv:32
CIFAR-10	CifarNet	FC:2bit Conv:4bit	FC:128 Conv:16

taneously for high throughput and the saving of memory space for intermediate results. Thus, *ILP* can also reduce the access to external memory for energy saving.

The partial outputs of layer $i-1$ are stored in PE buffer and used as the input of layer i. Once C_i input elements (C_o output elements of layer $i-1$) show up as an input slice, which is a part of the input tile, they can be allocated to the corresponding input buffer for *ITS* and then for *ISR*. After processing $r-1$ input tiles and one input slice ($X(1,.,.)$, $X(2,.,.)$ and $X(3,1,.)$ in Fig. 7), we can get the input slices of layer $i+1$ consecutively. To get a balanced pipeline, we also adopt intra-layer parallelism like Pipelayer [6] and the parallelism granularity depending on the MACs of different layers of the processed NN.

Table I compares the mapping scheme and intermediate storage of different accelerators and techniques we proposed. WRV denotes the mapping scheme of weight representative vectors. Note that the intermediate storage means the memory of intermediate results that can be directly used in the next layer's processing. The extra $psum$ access to global buffer of *ITS* is not included here but we consider the overhead in our evaluation. It is obvious that, with *ISR* and *ITS*, REBOC's intermediate storage is much smaller than that of others. Therefore, REBOC's PE buffer need not to be as large as eDRAM buffer of ISSAC. And in fact, under most circumstances, ISSAC's 64 KB buffer cannot be fully exploited. Thus, our design has better area and power efficiency.

IV. EVALUATION

A. Benchmarks

We evaluate the proposed REBOC accelerator on four datasets: MNIST [15], ImageNet [16], CIFAR-10 [17] and DataDJI [18]. The networks used in our evaluation are LeNet on MNIST, AlexNet on ImageNet, CifarNet on CIFAFR-10, and TinyYolo on DataJI. For fair comparison and accuracy, we use different configurations of MAC bit-precision and block size for different NNs in our evaluation, which are almost the same as Sticker-T [7]. Table II illustrates the details of the configurations.

TABLE III
SIMULATION CONFIGURATION

DRAM	128GB	**Global Buffer**	128KB
Base Clock	1200MHz	**PE Buffer**	4KB
IB per XB	256B	**SR per XB**	128B
Crossbar Size	128 × 128	**#XBs per CU**	8
#CUs per PE	4	**#PEs**	16

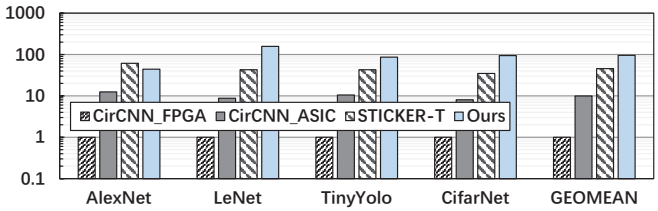

Fig. 8. Normalized power efficiency comparison.

B. Experiment Setup

We build a ReRAM simulator based on NVSim [19] to evaluate REBOC. The configuration of the simulation is shown in Table III. The models of buffers and register are extracted from CACTI [20] with a 65nm technology. For ADC energy and sensing time, we extract data from [21]. We adopt ReRAM write/read latency and energy from [22] as 29.31ns/50.88ns, 1.08pJ/3.91nJ. We also build a Pipelayer-like accelerator, which does not support intra-layer pipeline and the data reuse system we proposed, using the same hardware configurations. When comparing REBOC with ASIC/FPGA, the baseline is the FPGA implementation of Cir_CNN [2].

C. Comparison with ASIC/FPGA

Power efficiency ($GOPs/s/W$) is the metric to compare REBOC with [2], [7]. The configurations of MAC bit-precision of different NNs are shown in Table II. The MAC bit-precision of CirCNN_FPGA and CirCNN_ASIC are 16bit because if the weights are quantized to 4bit, the accuracy degrades much (the accuracy of AlexNet is lower than 20%) because in CirCNN architecture the configuration bit-precision of FFT/IFFT and MAC should be the same but FFT/IFFT need higher bit-precision. In [7], the configuration is more flexible, which allows FFT/IFFT have higher bit-precision than MAC. As shown in Fig. 8, on average, the power efficiency of REBOC is 96× higher than CirCNN_FPGA, 8.86× higher than CirCNN_ASIC and 1.93× higher than STICKER-T for the four benchmarks in our evaluation. Firstly, this is because REBOC does not need to fetch weights from external memory, and we can delicately map the weights onto crossbars to make PEs get maximally exploited and duplicate the weights to make the pipeline balanced. Thus, REBOC's performance is harder to be weakened by the variations of operations of different layers or NN models. However, due to their architecture design, the power efficiency of Cir_CNN [2] and STICKER-T [7] vary and the hardware resource of them cannot be fully exploited simultaneously when processing different NNs. The performance of FFT module is the bottleneck because FFT module has much poorer computation parallelism and needs

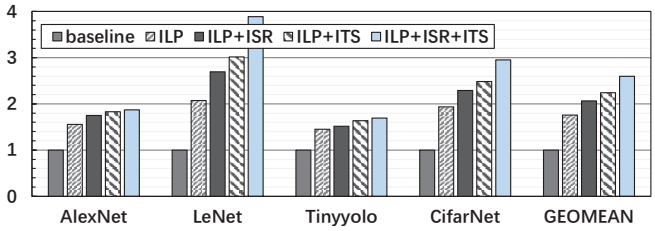

Fig. 9. Normalized energy saving of CNN layers.

higher bit-precision. REBOC has lower power efficiency than STICKER-T for AlexNet because in AlexNet, nearly 90% of parameters belong to the FC layers. Thus, when processing the large FC layers with large block size (eg., 128), STICKER-T can get both of its FFT module and MAC module maximally exploited to achieve better power efficiency.

D. Comparison with ReRAM-based Accelerator

Energy Results. Fig. 9 illustrates the energy consumption of CONV layers in the benchmarks. The average energy saving of REBOC for the benchmark's Conv layers is $2.6\times$. For each benchmark, we report the energy consumption of intra-layer pipelined REBOC (*ILP*), intra-layer pipelined REBOC with input slice reusing (*ILP+ISR*), intra-layer pipelined REBOC with input tile sharing (*ILP+ITS*) and intra-layer pipelined REBOC with both input slice reusing and input tile sharing (*ILP+ISR+ITS*). Intra-layer pipeline achieves energy savings because it does not need to store all the intermediate results of one layer in the global buffer or DRAM and re-access them before processing next layer as the baseline does. *ITS* and *ISR* further reduce the access to external memory to save more energy.

Performance Results. As described above, *ILP* can speed up processing Conv layers of one frame and the speedup is nearly a direct function of the nunmber of Conv layers in benchmarks. Thus, intra-layer pipelined REBOC achieves the highest speedup for TinyYolo, improving the performance to $7.69\times$. *ITS* and *ISR* do not achieve obvious improvement for speedup because compared with the latency of computation and ADC sensing time, the latency of accessing data from the buffer is relatively tiny. However, the combination of *ITS* and *ISR* can still bring $1.06\times$ speedup to the *ILP*. On average, REBOC achieves $4.1\times$ speedup.

V. CONCLUSION

In this work, we proposed REBOC, a ReRAM based accelerator to efficiently process block-circulant neural networks, which utilizes massive parallel MVM processing in ReRAM for computation rather than the naive FFT/IFFT. For REBOC, we developed Horizontal Weight Slicing and Intra-Crossbar Weight Duplication to map block-circulant DNN models onto ReRAM crossbars efficiently. Input Slice Reusing and Input Tile Sharing are particularly designed to reduce the data access and buffer size. An intra-layer processing pipeline was also proposed for high throughput. From the evaluation, REBOC achieved significant speedup and power efficiency compared with state-of-the-art accelerators for block-circulant neural networks and ReRAM based accelerators.

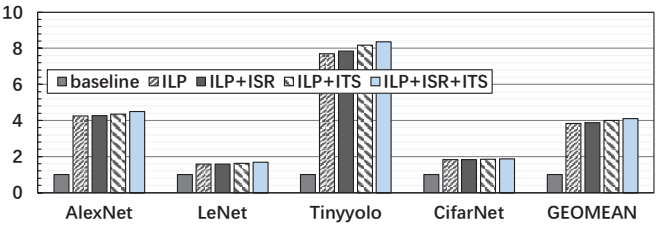

Fig. 10. Normalized speedup of CNN layers.

VI. ACKNOWLEDGEMENTS

This work was partially supported by the Science and Technology Commission of Shanghai Municipality under Grants 16JC1420300 and 2018SHZDZX01. This work was also supported in part by NSF-1910299 and NSF-1822085. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of grant agencies or their contractors.

REFERENCES

[1] S. Han *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *ICLR*, 2016.
[2] C. Ding *et al.*, "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices," in *MICRO*, 2017.
[3] Y. Chen *et al.*, "Diannao family: Energy-efficient hardware accelerators for machine learning," *Commun. ACM*, 2016.
[4] A. Farmahini-Farahani *et al.*, "Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules," in *HPCA*, 2015.
[5] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[6] L. Song *et al.*, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *HPCA*, 2017.
[7] J. Yue *et al.*, "7.5 A 65nm 0.39-to-140.3TOPS/W 1-to-12b Unified Neural Network Processor Using Block-Circulant-Enabled Transpose-Domain Acceleration with 8.1 × Higher TOPS/mm^2 and 6T HBST-TRAM-Based 2D Data-Reuse Architecture," in *ISSCC*, 2019.
[8] "Caffe." https://caffe.berkeleyvision.org/.
[9] "Tensorflow." https://www.tensorflow.org/.
[10] Y. Cheng *et al.*, "An exploration of parameter redundancy in deep networks with circulant projections," in *ICCV*, 2015.
[11] F. Chen *et al.*, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *ASP-DAC*, 2018.
[12] X. Qiao *et al.*, "AtomLayer: A Universal ReRAM-Based CNN Accelerator with Atomic Layer Computation," in *DAC*, 2018.
[13] M. J. Marinella *et al.*, "Multiscale co-design analysis of energy, latency, area, and accuracy of a reram analog neural training accelerator," *JETCAS*, 2018.
[14] A. Anderson *et al.*, "Low-memory GEMM-based convolution algorithms for deep neural networks," *arXiv*, 2017.
[15] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.
[16] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
[17] S. K. Esser *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *PNAS*, 2016.
[18] C. Ding *et al.*, "Req-yolo: A resource-aware, efficient quantization framework for object detection on fpgas," in *FPGA*, 2019.
[19] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, 2012.
[20] N. Muralimanohar *et al.*, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," 2007.
[21] T.-H. Yang *et al.*, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *ISCA*, 2019.
[22] D. Niu *et al.*, "Design of cross-point metal-oxide reram emphasizing reliability and cost," in *ICCAD*, 2013.