RED: A ReRAM-based Efficient Accelerator for Deconvolutional Computation

Ziru Li*, Bing Li*§, *Member, IEEE,* Zichen Fan, and Hai (Helen) Li, *Fellow, IEEE*

***Abstract*—Deconvolution is a key component in contemporary neural networks, especially generative adversarial networks (GANs) and fully convolutional networks (FCNs). Due to extra operations of deconvolution compared to convolution, considerable degradation of performance as well as energy efficiency is incurred when implementing deconvolution on the existing resistive random access memory (ReRAM)-based processing-in-memory (PIM) accelerators. In this work, we propose a ReRAM-based accelerator design, RED, for providing high-performance and low-energy deconvolution. We analyze the deconvolution execution on the existing ReRAM-based PIMs and utilize its interior computation pattern for design optimization. RED includes two major contributions: pixel-wise mapping scheme and zero-skipping data flow. Pixel-wise mapping scheme removes the zero insertion and performs convolutions over several ReRAM arrays and thus enables parallel computations with non-zero inputs. Zero-skipping data flow, assisted with customized input buffers design, enhances the computation parallelism and input data reuse. In evaluation, we compare RED against the existing ReRAM-based PIMs and CMOS-based counterpart with a variety of GAN and FCN models, each of which contains multiple deconvolution layers. The experimental results show that RED achieves a 4.0× − 56.16× speedup and a 1.05× − 18.17× energy efficiency improvement over previous related accelerator designs.**

***Index Terms*—Accelerator, deconvolution, energy efficiency, neural networks, resistive memory.**

I. Introduction

DEEP neural networks (DNNs) have achieved striking success in many applications, especially on image classification and object detection. Many of these designs were based on convolutional neural networks (CNNs), which extract high-dimensional information from input data for classification. In recent years, new types of neural networks, like generative adversarial networks (GANs) [1] and fully convolutional networks (FCNs) [2], have emerged and been used in a variety of domains, such as robotics [3], [4], autonomous driving [5], [6], and medicine [7].

This work was in part supported by US Department of Energy (DOE) under grant SC0017030 and National Science Foundation (NSF) under grant CCF-1725456. Bing Li acknowledges the National Academy of Sciences (NAS), USA for awarding the NRC research fellowship.

*First author. §Corresponding author. Z. Li and H. Li are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA. e-mails: {ziru.li, hai.li}@duke.edu. B. Li now is with Capital Normal University, Beijing, China. email: bing.li@cnu.edu.cn. This work was completed when she was a Post-Doc at Duke University. Z. Fan now is with University of Michigan, email: zcfan@umich.edu. This work was completed when he was in his internship at Duke University.

The preliminary manuscript was published in the proceedings of the 2019 Design, Automation & Test in Europe (DATE).

Manuscript received July 30, 2019.

The deployment of DNNs in traditional computing architecture further aggravates the memory and power bottlenecks. Therefore, novel paradigms like processing-in-memory (PIM) have been intensively investigated for efficient DNN execution [8]–[10]. Among various technologies to implement PIM designs, resistive random access memory (ReRAM) demonstrates an outstanding computation capability and is regarded as a competitive technology [11]–[16]. ReRAM has been studied as a high-density, low-power, and low-leakage memory device [17], [18]. Its crossbar structure naturally mimics the analog matrix-vector multiplication (MVM) [19], [20]. Since MVM is the dominant operation in DNNs, ReRAM-based PIM substantially improves the execution speed and the energy efficiency for DNN applications [11]–[16], [21].

GANs and FCNs have been applied to relatively complicated tasks instead of ordinary CNNs. In these models, *deconvolution* layers, aka *transpose convolution* layers [22], are used to operate up-sampling on input feature maps. Compared to the traditional convolution, deconvolution has two different arithmetic implementations, both of which append additional operations to convolution: (1) *zero-padding deconvolution* expands the size of feature maps by performing zero-padding on input feature maps before convolution, inducing redundant operations on these zero values; (2) *padding-free deconvolution* multiplies the pixels of input feature maps with kernel matrices, without the need of zero-insertion, but extra operations are required after multiplications, *i.e.*, summation and cropping. Despite the fact that there is a convolutional operation within a deconvolution, zero-padding deconvolution leads to massive redundant operations if the deconvolutional layer is implemented on the existing ReRAM-based accelerators, resulting in under-utilization of computing resources. The padding-free deconvolution is friendly to CMOS-based accelerators [23], but the extra operations lead to increased circuits overhead in ReRAM-based PIMs.

This work aims to develop an efficient ReRAM-based accelerator to support deconvolution by eliminating redundant and extra operations. We propose a *pixel-wise mapping* scheme to remove the zero-insertion stage and the redundant operations required in zero-padding deconvolution, and *zero-skipping data flow* to elevate execution efficiency. Meanwhile, an innovative input buffer design is developed to maximize input data reuse and to improve computation parallelism. By integrating these designs, we propose RED, a ReRAM-based accelerator tailored for deconvolutional computation. The primary contributions of this work are as follows:

1) We analyze the deconvolution operation on existing ReRAM-based PIMs. The zero-padding deconvolution

results in under-utilization of computing resources due to massive zero insertion. The padding-free deconvolution introduces additional add-on operations and incurs extra circuit cost.

2) We propose the *pixel-wise mapping* scheme that maps kernel weights to ReRAM arrays at a fine-grained level by exploiting the computation modes derived from convolution between the used input pixels and kernel weights.
3) We propose the *zero-skipping data flow* facilitated with a novel *input buffer* design to maximize the input data reuse and further elevate computation parallelism.
4) We evaluate the efficiency of RED with the typical GAN and FCN models on a variety of applications. We compare it against the prior ReRAM-based PIM implementations for the zero-padding and padding-free designs [24], as well as several state-of-the-art deconvolution accelerators [23], [25]–[27].

The rest of this paper is organized as follows. We first introduce deconvolutional computation, ReRAM-based PIM, and related works of accelerator design in Section II and then present the challenges when using existing ReRAM-based PIMs to accelerate deconvolution in Section III. Section IV describes the proposed RED architecture. In Section V, we evaluate our proposed RED design and compare it with prior ReRAM-based PIM designs and the CMOS-based counterpart. Finally, we conclude this work in Section VI.

II. Background

A. Deconvolutional Computation

Deconvolution or *transpose convolution* is extensively employed in generative adversarial networks (GANs) and fully convolutional networks (FCNs). For instance, the generator network in a GAN uses a stack of deconvolutional layers to generate the sample images from the noise-like inputs. The deconvolutional layers in FCNs substitute the fully connected layers in conventional CNNs. FCNs can produce the output with the same dimensions as the input image for pixel-wise prediction or semantic segmentation when conducting the inference tasks. This work primarily focuses on the acceleration of the inference of GANs and FCNs.

Unlike the conventional convolutional layer that extracts features and produces abstract information, a deconvolutional layer up-samples its input and generates an output with a larger size than the input. Fig. 1 presents a portion of a typical FCN [2], where a deconvolutional layer DECONV1 follows a series of convolutional layers. As shown in the figure, the input feature map (FP) of CONV15 has a size of 22×22 and the output FP attains 16×16, while the input FP size of DECONV1 is 16×16 and its output FP size is 34×34.

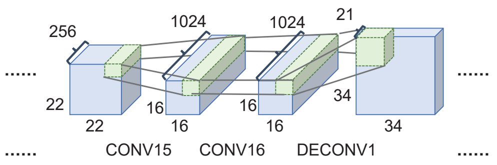


Fig. 1. The sizes of the feature maps changes along layers in an example FCN [2].

TABLE I
NOTATIONS USED IN DECONVOLUTIONAL OPERATION.

Notation	Description
I_h, I_w, C	The height, width and number of channels of input feature map.
O_h, O_w, M	The height, width and number of channels of output feature map.
K_h, K_w, M	The height, width and number of kernels.
S, P	The stride and padding sizes of deconvolution.
I_{pad_h}, I_{pad_w}	The height, width of padded input.
S'	The number of inserted zeros in zero-padding deconvolution.
P'	The size of padded border in zero-padding deconvolution.

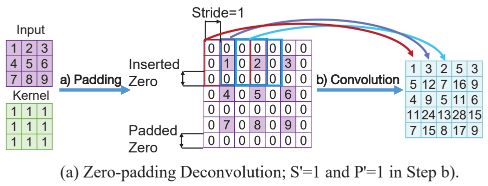
Note: In neural networks discussed in this work, $I_h = I_w$, $O_h = O_w$, $K_h = K_w$, and $I_{pad_h} = I_{pad_w}$. In the following manuscript, we will omit the subscripts and use I, O, K and I_{pad} in some places for simplicity.

There are two categories of algorithms to perform deconvolution. In this work, the two algorithms are denoted as *zero-padding* and *padding-free*, respectively. Here, an element in an input or an output is referred as a *pixel*. Table II-A lists the notations used for explaining the deconvolutional algorithms. Fig. 2 illustrates the calculation phases of zero-padding and padding-free algorithm for a 2D deconvolution, which is equivalent to a 3D deconvolution when $\mathrm{M} = 1$ and $\mathrm{C} = 1$. In this figure, $\mathrm{I} = 3$, $\mathrm{O} = 5$, $\mathrm{K} = 3$, $\mathrm{S} = 2$, and $\mathrm{P} = 1$. The number in a cell represents its element value.

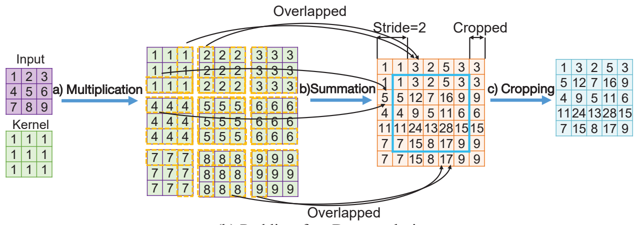
As illustrated in Fig. 2(a), the zero-padding deconvolution includes two steps:

a) Padding: an input FP is enlarged by inserting zeros between any two adjacent input pixels along the row and column directions and adding one $P' \times P'$ border of zeros surrounding the input FP. The dimension of padded input is denoted as I_{pad}.
b) Convolution: the conventional convolution of the padded input and a kernel is performed with the unit stride to generate the output FP. The number of the kernels is represented as M.

The example in Fig. 2(a) adds 1×1 zero-value borders to



(a) Zero-padding Deconvolution; S'=1 and P'=1 in Step b).



(b) Padding-free Deconvolution.

Fig. 2. Illustrative examples of 2D deconvolution algorithms: (a) zero-padding and (b) padding-free. Here, $\mathrm{K} = 3$, $\mathrm{I} = 3$, $\mathrm{S} = 2$ and $\mathrm{P} = 1$, $\mathrm{S}' = 1$ and $\mathrm{P}' = 1$.

the four sides of the input matrix and grows the 3×3 input FP to the size of 7×7. Then a 3×3 kernel with a stride of 1 is applied to the padded input for convolution operation, generating an output in a size of 5×5. In this example, the total number of multiplications is 225, while most of them are calculated with zero values. The multiplications on the inserted zeros are unnecessary and should be skipped. In general, the relationship between the sizes of input and output FPs can be described as follows:

$$\begin{aligned} & I = \frac{O + 2 \times P - K}{S} + 1 && \text{(a)} \\ & I_{pad} = O + K - 1 && \text{(b)} \\ & S' = S - 1 && \text{(c)} \\ & P' = K - 1 - P && \text{(d)} \end{aligned} \qquad (1)$$

The convolution for the padded input is the same as traditional convolution. Thus, the zero-padding is easy to implement for software design by directly invoking the convolutional function. However, the increased size of padded input introduces overhead on storage and the zeros induce redundant operations and resource under-utilization.

Fig. 2(b) illustrates the padding-free deconvolutional algorithm, which has the following three major steps:

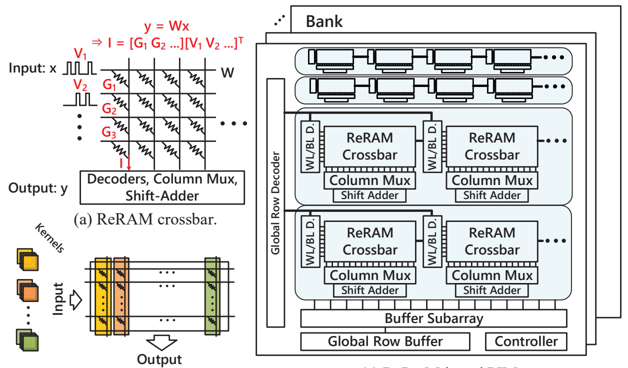
a) Multiplication: the input pixels of an input FP are multiplied with the kernel matrix, and afterwards, the products of different channels are accumulated to generate a group of matrices with the same size as kernels;
b) Summation: the overlapped rows and columns of the generated matrices in step a) are added, and those elements that share the same output coordinates are summed together;
c) Cropping: the border of matrix generated in step b) is cropped, and the final output matrix is produced.

As shown in Fig. 2(b), element-wise multiplications are conducted between every element of the input matrix and kernel. This step results in 9 matrices, the size of which is equal to the size of the kernel. Then the overlapped elements according to their coordinate in the final matrix are added together. For example, the third column of the first matrix will be added with the first column of the second matrix, and the last row of the first matrix will be summed with the first row of the fourth matrix. Step b) produces a 7×7 matrix, the border of which is then cropped. The final matrix has a size of 5×5. In this example, the entire operation contains 81 multiplications and extra 36 summations on the overlapped pixels.

Padding-free introduces two additional operations, summation and cropping. Moreover, the intermediate results in step a) and step b) exacerbate the storage overhead. Previously, Xu *et al.* [23] successfully utilized the padding-free algorithm on CMOS-based hardware for efficient deconvolutional computation. As we shall show in Section III, completing these operations on the existing ReRAM-based accelerators incurs a considerable overhead.

B. ReRAM-based PIM

As a nanoscale memory technology, the advantages of ReRAM include simple structure, large capacity and low read/write latency [17], [18]. What's more, ReRAM crossbar demonstrates high efficiency in performing matrix-vector multiplication (MVM) operations [19], [20]. As illustrated in Fig. 3(a), a matrix is stored in a ReRAM crossbar, in which the cell conductance values correspond to the weight elements of the matrix. During the MVM operation, an input vector in the form of input voltages are supplied as the input of wordlines along the row direction simultaneously. The currents flowing out from the bitlines along columns denote the output vector, which can remain in analog format or be converted into a digital vector via the read-out periphery circuits.



(a) ReRAM crossbar. (b) Kernels mapping. (c) ReRAM-based PIM.

Fig. 3. The fundamental ReRAM-based PIM architecture.

The ReRAM-based processing-in-memory (PIM) designs present high efficiency in matrix-based applications. A variety of ReRAM-based PIM designs have been proposed for accelerating the CNN inference and training [11]–[13], [15], [16]. When performing the convolution operation, the kernels are usually mapped onto one or several ReRAM crossbar arrays to maximize the computing parallelism [13], [14]. An exemplary mapping scheme used by ReRAM-based accelerator designs in [13], [24] is shown in Fig. 3(b). Here, M kernels are stored on M adjacent columns in a ReRAM crossbar. The weights of C channels are spread into a one-dimension vector and stored in one column. Fig. 3(c) depicts a full ReRAM-based PIM architecture [12], [13]. The design is based on the main memory structure with dedicated periphery circuits. ReRAM crossbars can alternate between storage and computing mode according to the system configuration [12].

C. Related Works

1) Deconvolutional Accelerators: Numerous efforts explored FPGA or ASIC based deconvolutional accelerator designs. For example, the FPGA-based GAN accelerator [27] exploits the dataflow optimization to remove zero operations and increase data reuse. However, the scalability of the FPGA-based accelerator is constrained by the limited on-chip storage. GANAX [26] is an SIMD-MIMD accelerator for GAN. The design reorganizes the computations along the rows of output and kernel to enable the parallel distinct computation flows, which inevitably increases hardware complexity. FCN-Engine [23] is an accelerator for deconvolution by exploring the padding-free algorithms to remove the redundancy of

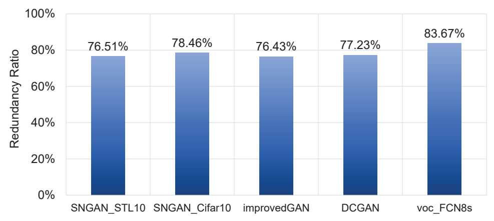


Fig. 4. The zero redundancy ratio in the zero-padding deconvolution for the neural networks evaluated in this work.

the input FP. However, it requires additional storage to hold the temporary output, as well as computing components to perform the summation.

2) ReRAM-based Accelerators: ReRAM-based CNN accelerators have been extensively studied for both inference and training. ISAAC [11] employs ReRAM crossbars to perform MVM operations and designs a pipeline for CNN inference acceleration. PRIME [12] enhances the nueral network inference by assigning a portion of crossbars dedicated to intermediate result storage. AtomLayer [14] attempts to compute one convolutional layer at a time to avoid the long latency and bubble induced by deep pipeline. PipeLayer [13] firstly optimizes the CNN training by collaborating intra-layer and inter-layer parallelism into the design. TIME [15] is designed for CNN training and further eliminates the amount of write operations on ReRAM. None of these works are aware of or optimize for the particular computation patterns in deconvolution.

ReGAN [24] is a ReRAM-based pipelined GAN accelerator. It performs the zero-padding deconvolution and thus incurs significantly redundant operations. LerGAN [25] handles the zero-related scenarios in GAN training by storing all the partial weight matrices involved in convolution operations. As many weights will be reused repeatedly, the approach introduces a large storage overhead. Our RED accelerator uses ReRAM crossbars to conduct the low-overhead convolutional arithmetic and specially optimizes for improving the deconvolution execution efficiency.

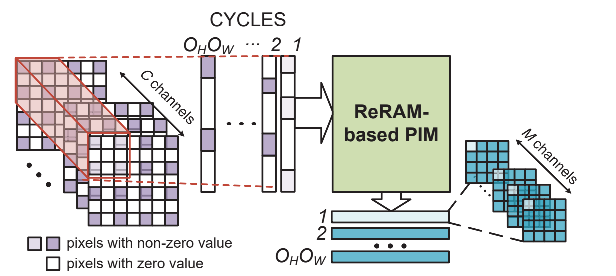
III. Inefficient Deconvolutional Computation on ReRAM-based Accelerators

In this section, we discuss the inefficiency of the zero-padding deconvolution and padding-free deconvolution on the existing ReRAM-based accelerators.

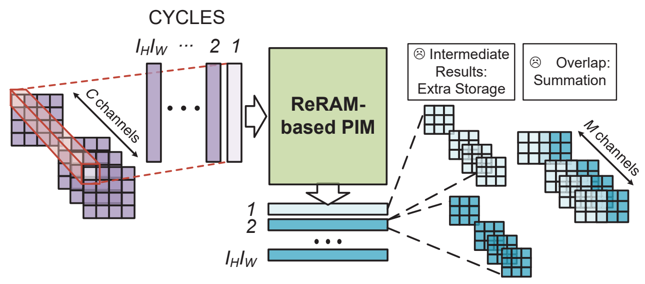
A. Redundant Zero-Value Operations

The zero-padding deconvolution inserts extra zeros into the input image, which leads to a large number of redundant operations. The relationship of I and I_{pad} can be calculated by using Equation 1(a,b).

There are two kinds of padded zeros: the zeros inserted between adjacent input pixels and those added at the four edges. The padded input FPs are sparse, and multiplications on zero pixels are not necessary. We use the redundancy ratio to denote the proportion of the zero-related operations. The redundancy ratio in a deconvolutional layer is defined as the proportion of padded zeros over the entire input FPs.



(a) ReRAM-based zero-padding deconvolution.



(b) ReRAM-based padding-free deconvolution.

Fig. 5. Deconvolution on existing ReRAM-based accelerators.

In a single deconvolution layer with stride S, for example, the redundancy ratio can be approximated to $(1 - 1/S^2)$, when the first kind of padded zeros are dominant. Fig. 4 summarizes the average geometric redundancy ratio of the deconvolutional layers in our benchmarks[1]. The redundancy ratio of SNGAN_STL10 is as high as 76.51% and that of FCN reaches 83.67%.

Fig. 5(a) illustrates the implementation of the zero-padding deconvolution on a ReRAM-based PIM [24], which is the same as the standard convolutional computation described in Section II-B. In each cycle, one input vector with padded zeros is fed into ReRAM-basd PIM for computation and each element in the produced output vector corresponds to one-pixel information of M output FPs. As such, it will take $O_h \times O_w$ cycles to complete the computation of M output FPs in the shape of $O_h \times O_w$.

B. Inefficiency of Padding-Free Deconvolution

The padding-free algorithm, as an alternative deconvolution implementation, avoids to insert redundant zeros and potentially improves the hardware implementation efficiency. A previous study [23] showed that the padding-free deconvolution achieved up to $44.9\times$ performance improvement when it is deployed on the CMOS-based platforms. However, performing the padding-free algorithm on a ReRAM-based PIM requires the extra storage overhead and non-trivial efforts in circuitry modification. Fig. 5(b) gives an example of ReRAM-based padding-free implementation. The padding-free algorithm involves the element-wise multiplication of the input and the kernel and the summation of the overlapped output. The weight elements in one channel of the kernel are spread into

[1]The details of the parameters and configuration of these networks shall be given in Section V.

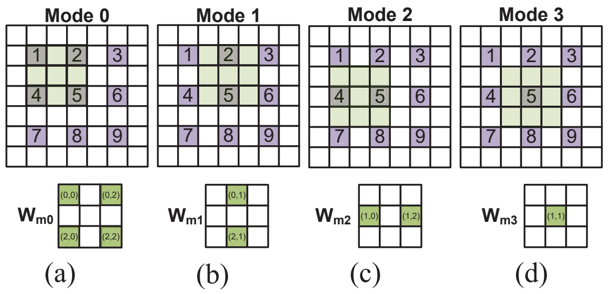


Fig. 6. The four computation modes in 2D deconvolution when $K = 3$ and $S = 2$.

a vector and the same row of crossbars. Each input FP is converted into a vector whose elements are supplied to the crossbar in sequence. Each output generated from ReRAM-based PIM has C matrix in the shape of $K_h \times K_w$. In the duration of computing, output buffers are necessary to place the intermediate outputs, incurring extra overhead. When intermediate outputs are ready, they will be injected into an adder tree to calculate the summation of the overlapped elements. Moreover, extra circuits are needed to conduct the cropping operations to obtain the final results.

Compared to the zero-padding deconvolution, the padding-free deconvolution requires extra storage and computing components. Accordingly, the padding-free deconvolution is expected to have a much higher energy consumption than the zero-padding deconvolution.

IV. ReRAM-based Deconvolutional Accelerator

Our proposed ReRAM-based PIM accelerator for deconvolution, namely **RED**, leverages the relation of different computation modes in deconvolution operation and consists of two orthogonal schemes: *pixel-wise mapping* and *zero-skipping data flow*. In this section, we will elaborate the details of the RED design and discuss the trade-off in its usage.

A. Computation Modes

We revisit the zero-padding algorithm and utilize the inherent computation modes within the deconvolution to design our accelerator. This subsection formally describes and concludes the computation modes in deconvolution operation.

Fig. 6 illustrates the four computation modes when sliding the kernel within an input FP, with an example of 2D deconvolution, in which $S = 2$, $I = 3$ and $K = 3$. In each sub-figure, the top large grid refers to the padded input FP, whose non-zero and zero pixels are denoted in purple and white colors, respectively. Each non-zero element in the original 3×3 input is numbered by $1 \sim 9$. The small grid in a sub-figure indicates the usage of the weight element in the kernel, in which only the green bricks labeled with the coordinates are in use in the current computation mode.

As shown by Fig. 6(a), `Mode 0` is an MVM operation with two 2×2 matrices. The weight matrix W_{m0} is a sub-block of the kernel, which consists of four weight elements: $w(0,0)$, $w(0,2)$, $w(2,0)$ and $w(2,2)$. `Mode 1` in Fig. 6(b) corresponds to the deconvolution by sliding the kernel horizontally one step from the position in `Mode 0`. `Mode 1` performs the deconvolution with two 2×1 matrices. The weight matrix W_{m1} is composed of two weight elements: $w(0,1)$ and $w(2,1)$, which do not contain the shared values with W_{m0}. Similarly, Fig. 6(c,d) demonstrates the operations when moving the kernel window down one grid from the positions in Fig. 6(a,b), respectively. The weight matrices in these deconvolution modes are W_{m2} and W_{m3}, respectively.

TABLE II
Notations Used in Computation Mode i. $i \in \{0, 1, \dots S^2 - 1\}$.

Notations	Explanation
W_{mi}	Weight matrix in computation mode i.
N_i	Size of weight matrix W_{mi}.
L_{Ri}, L_{Ci}	Length of row and column of W_{mi}.

TABLE III
Weight Matrix W_{mi} in Computation Mode i. $i \in \{0, 1, \dots S^2 - 1\}$.

Notations	Definitions & Values
W_{mi}	$W_{mi} \subseteq W$, $\bigcup_i W_{mi} = W$, $\bigcap_i W_{mi} = \varnothing$ W_{mi}=$\{w \mid w = w[k_h, k_w, c, m]\}$
L_{Ri}	$\left\lceil \frac{K - \lfloor i \div S \rfloor}{S} \right\rceil$
L_{Ci}	$\left\lceil \frac{K - i \mod S}{S} \right\rceil$
$w[k_h, k_w, c, m]$	$k_h = \lfloor i \div S \rfloor + n \times S$ $k_w = i \mod S + u \times S$ $n \in \{0, 1, \dots, L_{Ri} - 1\}$, $u \in \{0, 1, \dots, L_{Ci} - 1\}$ $c \in \{1, 2, \dots, C\}$, $m \in \{1, 2, \dots M\}$

TABLE IV
Weight Matrices in Computation Modes of Fig. 6

Computation Mode	Weight Matrix
Mode 0	$N_0 = 2^2 = 4$; $W_{m0} = \{w(0,0), w(0,2), w(2,0), w(2,2)\}$
Mode 1	$N_1 = 1 \times 2 = 2$; $W_{m1} = \{w(0,1), w(2,1)\}$
Mode 2	$N_2 = 2 \times 1 = 2$; $W_{m2} = \{w(1,0), w(1,2)\}$
Mode 3	$N_3 = 1^2 = 1^2 = 1$; $W_{m3} = \{w(1,1)\}$

There are three important features regarding about the weight matrices in these computation modes:

- The union of these weight matrices is equal to the kernel.
- The weight matrices in different computation modes do not share any elements.
- During the entire process, only one of $W_{m0} \sim W_{m3}$ is in use at a time. When the kernel slides two steps horizontally and vertically from the initial `Mode 0`, W_{m0} is reused in the corresponding calculations.

There exist S^2 computation modes for a deconvolutional layer with a stride of S. When the weight element in W_{mi} is denoted with $w[k_h, k_w, c, m]$, k_h and k_w can be calculated by:

$$\begin{aligned} k_h &= \lfloor i/S \rfloor + n \times S, n \in [0, 1, \dots, L_{Ri} - 1] \\ k_w &= i \mod S + u \times S, u \in [0, 1, \dots, L_{Ci} - 1] \end{aligned}, \tag{2}$$

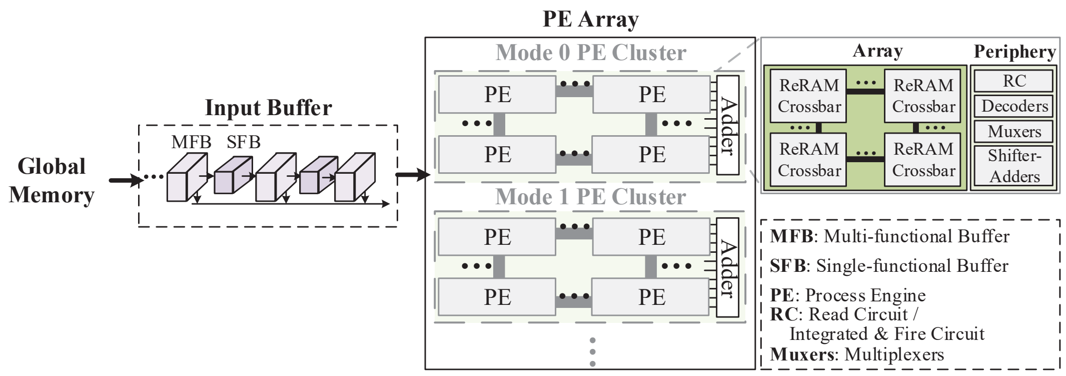


Fig. 7. The proposed RED architecture.

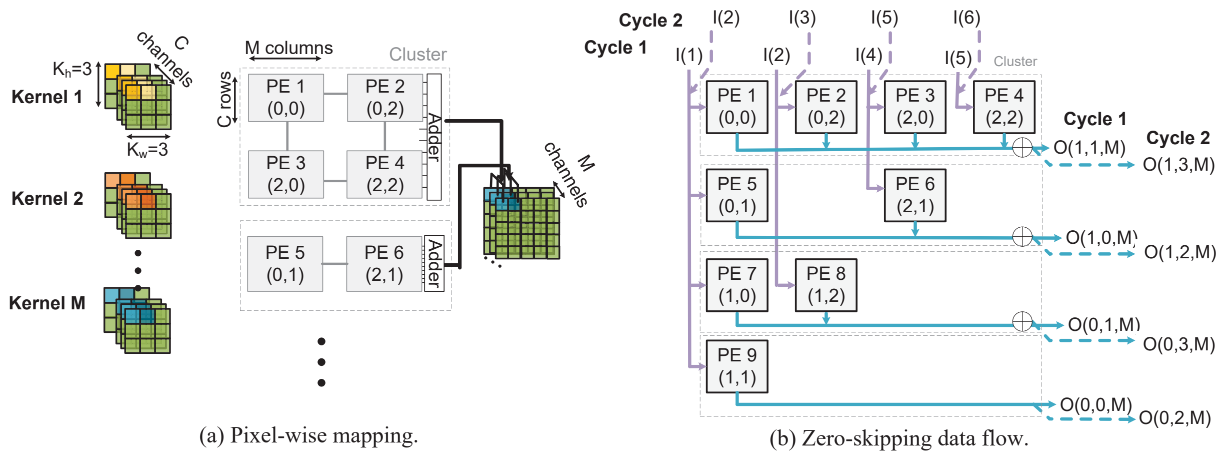


Fig. 8. The illustration of pixel-wise mapping (a) and zero-skipping data flow (b).

where L_{Ri} and L_{Ci} respectively denote the lengths of row and column of W_{mi} in those computation modes. L_{Ri} and L_{Ci} can be obtained by:

$$\begin{aligned} L_{Ri} &= \left\lceil \frac{K - \lfloor i/S \rfloor}{S} \right\rceil \\ L_{Ci} &= \left\lceil \frac{K - i \mod S}{S} \right\rceil \end{aligned}. \tag{3}$$

Given a deconvolution with a stride of S and a padding step of K, the notations used in the computation modes are given in Table II and the definitions and values related to weight matrix W_{mi} in `Mode` i are defined in Table III. The example in Fig. 6 has four inherent computation modes, and the weight matrices are shown in Table IV.

B. Overall Architecture

By utilizing the computation modes in deconvolution, we propose RED to tailor for deconvolution. Fig. 7 illustrates the high-level diagram of the RED architecture. There are two types of primary components: *local input buffer* and *ReRAM-based processing engine (PE) arrays.*

The local input buffer shared among PEs is designed to assist the skipping of the inserted zeros. The input buffer consists of two kinds of registers: multi-functional buffer (MFB) and single-functional buffer (SFB). To maximize the reuse of input data, the MFBs within the input buffer can switch between two work modes—*shift* and *buffer*. In the shift mode, MFBs and SFBs shift out the requested data to perform computations. In the buffer mode, MFBs load data from global memory and/or feed data to PE arrays.

The PEs in the PE array are communicated over the high-throughput on-chip data bus. Each PE contains a few ReRAM crossbars to conduct the MVM operations, and peripheral circuits such as read circuits, decoders, column multiplexers, and shift-and-add units. The weight matrices of each computation mode may occupy multiple PEs to complete the operations. These PEs that conduct the operations for the same computation mode logically constitute a PE cluster. The outputs from one cluster are selected via the multiplexers and fed into an adder to produce the aggregated results.

In the following subsections, we elaborate the *pixel-wise mapping* scheme, the *zero-skipping data flow*, as well as the local input buffer design of the RED architecture. These designs cooperate to improve the performance and energy efficiency of deconvolution.

C. Pixel-wise Mapping

We propose the *pixel-wise mapping* to eliminate the zero-related redundancy and enable the parallel execution of the computation modes. As indicated in Table I, the size of a kernel is K^2, the number of channel is C, the number of kernels is M, and the computing stride is S. We use the four dimensional coordinate $w(k_h, k_w, c, m)$ to represent a weight element. Our mapping scheme distributes the operations of this deconvolutional layer into K^2 PEs. Each PE holds the weight elements whose coordinates have the same value of both k_h and k_w. The columns of crossbars in PEs hold the elements from the same channel while their rows are allocated to the elements from the same kernel.

Fig. 8(a) show the pixel mapping for deconvolution with $K = 3$. PEs 1∼4 store weight matrices in Mode 0, *i.e.* W_{m0}. For example, PE 1 holds $w(0,0)$. Likewise, PE 2, PE 3 and PE 4 correspond to $w(0,2)$, $w(2,0)$, and $w(2,2)$, respectively. In the similar way, the weight matrices of other computation modes, W_{m1}, W_{m2} and W_{m3} are stored in PEs 5∼6, PEs 7∼8 and PE 9, respectively. The pixel-mapping scheme matches the pixel locations with the corresponding PEs and have multiple computation modes executed in parallel. In this example, the outcomes from the four computation modes are generated in one computing cycle.

D. Zero-skip Data Flow

Based on the pixel-wise mapping scheme, we further develop the zero-skipping data flow to skip the zero-insertion and maximize data reuse during the deconvolution computation. Only the original input values are distributed to the corresponding PEs in different computation modes, which maximizes data reuse and enables simultaneous execution.

Fig. 8(b) illustrates the operation for the given example in Fig. 6 with $\mathrm{stride} = 2$ and $\mathrm{K} = 3$. According to the pixel-wise mapping, the weight matrices in the four computation modes are mapped to 9 PEs. Here, $\mathbf{I}(i)$ denotes the $\mathrm{i^{th}}$ input vector which has C pixels from C channels, respectively. For brevity, the PEs along the same column in the figure take the same inputs, and the outputs from the PEs on the same row will be added together. In every cycle, all 9 PEs are in operation by coordinating with the corresponding input vectors. As illustrated in Fig. 8(b), in $\mathrm{Cycle}\ 1$, $\mathbf{I(1)}$ is applied to PE 5, PE 6, PE 8 and PE 9, $\mathbf{I(2)}$ is provided to PE 2 and PE 8, $\mathbf{I(4)}$ is taken by PE 3 and PE 6, and $\mathbf{I(5)}$ goes to PE 4. Their outputs will be merged together for the final deconvolution results, *i.e.*, $\mathbf{O(0,0,M)}$, $\mathbf{O(0,1,M)}$, $\mathbf{O(1,0,M)}$ and $\mathbf{O(1,1,M)}$. In the next cycle, RED continues to compute with the next batch of non-zero inputs, *e.g.*, $\mathbf{I(2)}$, $\mathbf{I(3)}$, $\mathbf{I(5)}$ and $\mathbf{I(6)}$ will be applied in $\mathrm{Cycle}\ 2$. The zero-skipping data flow increases the computation parallelism of this example $4\times$ and eliminates the zero-redundancy. Ideally, the zero-skipping data flow can achieve $S^2\times$ increase of the computation parallelism, without considering the limitation of PE resources.

E. Input Buffer Design

To facilitate the zero-skipping data flow, we design an input buffer. Fig. 9 depicts its overall structure, in which the MFBs and SFBs are connected alternatively to form a chain. An MFB can operate in *shift* and *buffer* modes. In the shift mode, it receives data from its left side and passes to the right side. In the buffer mode, it submits the stored data onto the data bus, which will transfer the data to the PE array. An SFB performs only the right-shift function and acts like a pipe between the adjacent MFBs.

The size of MFBs is supposed to be the largest row size of the weight matrices in the computation modes. SFBs are required to contain a row of data from the input FP. Based on these criteria, the number of MFBs and SFBs and the number of their entry can be determined.

When feeding the data into the PE array, the input buffer can have the following three work patterns:

- *Initialization*: The first group of inputs for the computation modes are fetched and stored into MFBs and SFBs.
- *Load*: MFBs work in the buffer mode and load the stored data onto the data bus. Then the data bus distributes the data to the PE array for the parallel operations. SFBs are idle in this work pattern.
- *Shift*: MFBs and SFBs execute right-shifts simultaneously. The data in the last MFB are shifted out.

After the initialization and the first load, the input buffer alternates its work pattern between load and shift until the completion of a layer's computation. The data at the last entry in the last MFB will be used once and then shifted out. Other data in the input buffer will be reused with the assist of the SFBs. As such, the input buffer design enables the data reuse and lowers the cost of remote memory access. Once the computation of the PE array starts, the load and shift in the input buffer is processed simultaneously with the computation in PE arrays. As a shift work pattern is always followed by a load during operations, we use *shift-load* to indicate this situation. When kernels shift over the input FPs vertically, two shifts are required to load a new row that are followed by one load. We use *Shift2-Load* to explicitly denote this action. In the following, we will use two examples to explain the work patterns and execution procedure of the proposed input buffer design.

1) Example I: This is an example for the deconvolution in Fig. 6 with $I = 3$. The corresponding mapping and data flow results are shown in Table V. The execution procedure of the input buffer is demonstrated in Fig. 10(a). Its input buffer is composed of two MFBs and one SFBs, each of which has two entries to hold the inputs. Different colors of inputs indicates that they are from different rows of the input FP. The buffers carry out initialization and load in sequence to prepare inputs for the computation of $\mathrm{Cycle}\ 1$. When the PE array executes the computation of $\mathrm{Cycle}\ 1$, the input buffer is occupied with

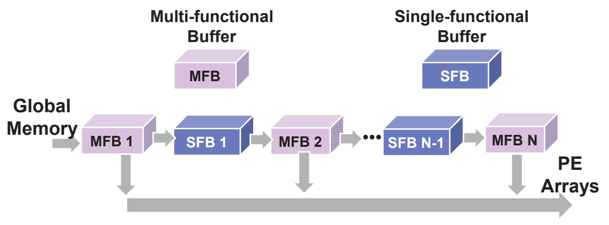
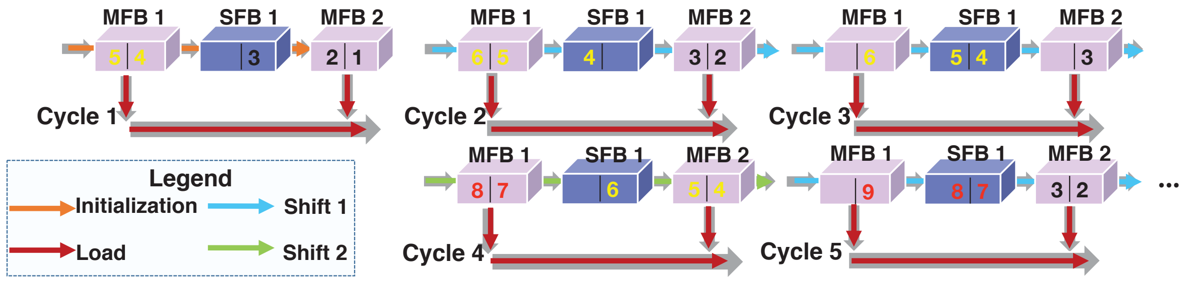


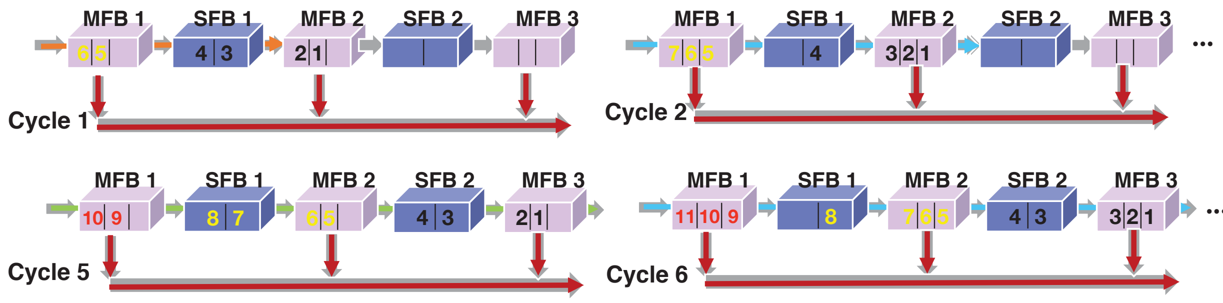
Fig. 9. The input buffer design.

TABLE V
LOADED INPUTS PER COMPUTING CYCLE FOR THE DECONVOLUTION IN FIG. 6 AND THE FIRST LAYER OF DCGAN [28] IN RED.

Example in Fig. 6	**Cycles**	1	2	3	4	5	···
	Inputs	1, 2, 4, 5	2, 3, 5, 6	3, 6	4, 5, 7, 8	5, 6, 8, 9	···
DCGAN_Deconv1	**Cycles**	1	2	···	5	6	···
	Inputs	1, 2, 5, 6	1, 2, 3, 5, 6, 7	···	1, 2, 5, 6, 9, 10	1, 2, 3, 5, 6, 7, 8, 9, 10	···



(a) The parameter values in this example are the same as those shown in Fig. 7.



(b) In this example, the values of stride (S), kernel size (K), and input width/height (I) are from the first deconvolution layer of DCGAN. S = 2, K = 5, I = 4.

Fig. 10. The example of input buffer design and execution procedures. Cycle here denotes the computing cycle. The number i in box denotes the i^{th} element in the input FP.

the inputs for Cycle 2. As shown in Fig. 10(a), MFBs switch into the shift mode and the input buffer performs the *shift-load* procedure. In the shift procedure, all buffers move the stored data one step toward the right; then inputs for Cycle 2 are ready in MFBs. The load procedure sends the inputs saved in MFBs to PE arrays. Likewise, another pair of shift-load of input buffers have the inputs for Cycle 3 ready. Then a *Shift2-Load* is executed by the input buffers to provide input for Cycle 4. For the sake of brevity, we only show the execution flow of the first five cycles.

2) Example II: Fig. 10(b) illustrates the first deconvolutional layer in DCGAN with where $S = 2$, $K = 5$, and $I = 4$. Three three-entry MFBs and two two-entry SFBs cooperate with each other in the deconvolution execution. First, the initialization operation fetches the first group of input pixels, and the load operation feeds the input data for Cycle 1 from MFBs onto the data bus. Subsequently, one *shift-load* action is carried out to provide the input data for Cycle 2. After Cycle 4, the input buffers perform a pair of *Shift2-Load* action, the input vectors 1, 2, 5, 6, 9, and 10 are loaded onto the data bus and transferred to the PE array. The *shift-load* actions in the input buffers overlap with the computation in the PE array.

The proposed input buffer design benefits from three perspectives. First, it maximizes the data reuse between computing cycles, and therefore, reduces the global memory access. Second, it employs less storage resources compared with previous designs [14]. Third, the input buffer design supports the simultaneous operations of the PE array by directly receiving data from MFBs, leading to higher computation parallelism and efficiency.

F. Programming Interface

In our proposed RED, the entire network model is executed by layers. The execution of a single deconvolutional layer is completed in several computing cycles. In each computation cycle, multiple computation modes are executed in parallel. One group of inputs for the computation modes are fetched from the global memory to the input buffer, which feeds the data into the PE array to perform computation. The output buffer holds the intermediate data outputs and transfers them back to the global memory once all the computations complete.

We provide a set of functions as the specific programming interface for the deconvolutional layer. These functions control the dataflow within RED as well as the data transfer between the global memory and RED. The controller driven by these functions works as a finite state machine and controls the data transfer between PE array and the global memory, as well as the data movement within the PE array. The function *Load_Weight* is used to load the pretrained weight to the PE array. Three specific functions are applied to operate the dataflow, including *Initialize_Data*, *Load_Data* and *Shift_Data*: *Initialize_Data* transfers data from the global memory to the input buffer of RED, *Load_Data* is activated to enable the input buffer to feed the data into the PE array and start computation simultaneously, and *Shift_Data* enables the input data to shift the data. After all the computation cycles are finished, the intermediate data in the output buffer are transferred back to the global memory by executing the function *Trans_to_GM*. In this way, the inference of a single deconvolutional layer is completed, and the output of this layer is stored in the global memory, ready for the computation of next convolutional or deconvolutional layer.

G. Discussion

The proposed pixel-wise mapping and zero-skipping data flow schemes facilitate the parallelism of deconvolutional computation. As aforementioned, RED can achieve $\mathrm{stride}^2\times$ computation parallelism by skipping all the redundant zeros. Previously, we interpreted the RED architecture with the examples of the deconvolutional layer with $\mathrm{stride} = 2$, which is a typical configuration of deconvolutional layers in GANs and FCNs. When the stride of the deconvolutional layer is larger than 2, the number of computation modes increases and the computation parallelism could be even higher.

The augmentation of stride is also followed by the expansion of kernel size. For example, in a typical deconvolutional layer with $\mathrm{stride} = 8$ in FCN [2], the size of kernel is 16×16. To achieve the maximum computation parallelism of $64\times$, 256 PEs are needed. Integrating such a large number of PEs might not be practical due to area and power constraints. We can modify the data flow to address such a situation. For instance, when K^2 PEs are needed while only $\frac{K^2}{2}$ is available, the data flow can be modified as follows:

$$\begin{aligned}\text{Cycle 1}: \quad & I_n[c]_{c=1,\ldots,C} = I_{2n,ori}[c]_{c=1,\ldots,C};\\ & I_n[c]_{c=C+1,\ldots,2C} = 0;\\ \text{Cycle 2}: \quad & I_n[c]_{c=1,\ldots,C} = 0;\\ & I_n[c]_{c=C+1,\ldots,2C} = I_{2n+1,ori}[c]_{c=1,\ldots,C};\end{aligned} \tag{4}$$

Where I_n presents the input data flow of the n^{th} PE, and $I_{2n,ori}$ and $I_{2n+1,ori}$ denote the input data flow of the PEs in the original pixel-wise mapping scheme ($0 \leq n < \frac{K^2}{2}$).

V. Experiments

A. Experimental Setup

We modify the NeuroSim+ [29] to evaluate the performance and energy consumption of RED. The system employs one-transistor-one-ReRAM (1T1R) cell at $65\mathrm{nm}$ technology node and runs at 2 GHz clock frequency. We adopt the device model with $R_{on}/R_{off} = 200k\Omega/1M\Omega$ and $2V$ write voltage. The input buffer is simulated with CACTI [30].

TABLE VI
Benchmarks used in this work.

Network Name	Dataset	Number of Deconv Layers	Number of Conv Layers
DCGAN	LSUN	4	1
ImprovedGAN	CIFAR-10	3	1
SNGAN_STL10	STL-10	3	2
SNGAN_cifar10	CIFAR-10	3	2
voc_FCN8s	PASCAL VOC	3	18

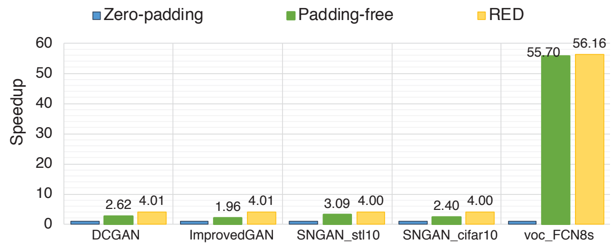
The evaluating benchmark covers a set of representative neural network models including GANs and FCNs. The structure of models are identical to those original settings. The same datasets are used for training and inference tasks. The structure details of networks are summarized in Table VI.

DCGAN [28] is a classical GAN and its generator is used to generate the 64×64 color images from LSUN-bedroom dataset [31]. ImprovedGAN is a semi-supervised GAN [32]. The model structure is based on DCGAN, while the number of the deconvolutional layers decreases from four to three. Furthermore, we employ two variants of SNGAN [33]: SNGAN_STL-10 [34] and SNGAN_cifar-10 [35]. SNGAN is a latest GAN network that is capable of generating high-quality images. SNGAN_STL-10 and SNGAN_STL-10 use different datasets, which are different in the size of input FP. Voc_FCN8s [2] is a representative fully connected network, which is used for semantic segmentation. We use the $\mathrm{stride} = 8$ version of FCN that is validated on PASCAL VOC dataset [36]. The deconvolutional layers in FCNs have fewer input channels, larger stride and larger kernel size compared to GANs, which contribute to the diversity of the selected benchmark.

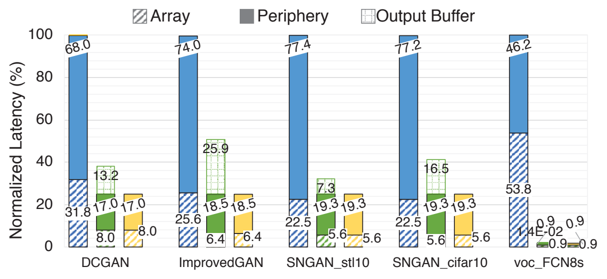
We compare RED with the padding-free design and the zero-padding design. The three implementations are evaluated in terms of execution time and energy consumption. All results are normalized to those of the zero-padding design. The comparison of deconvolutional layers is the main focus. We also present the breakdown results that separate the contributions of array, periphery circuitry and input buffer. At the end, we compare the computation and power efficiency of RED with the latest deconvolution accelerator designs [23], [25]–[27].

B. Execution time

During the evaluation, input and output buffer access and computation are scheduled in a pipeline manner referring to prior studies [11], [13], [37]. The execution time of the zero-padding and RED include both the array latency and the periphery latency. As the input buffer access and deconvolution execution can be processed in pipelined manner, the latency caused by the buffer access is omitted. However, the padding-free implementation uses output buffer to save the intermediate output. Although these store operations can work concurrently with the deconvolution execution, the drain time of the pipeline process induced by the output buffer is not trivial. Thus, we include the output buffer access latency in

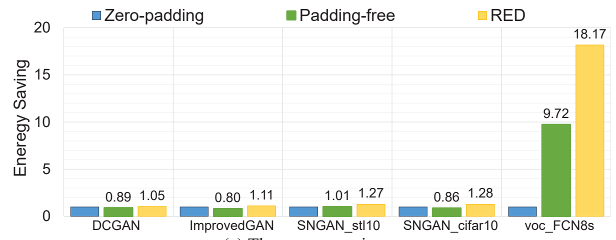


(a) The speedup performance.

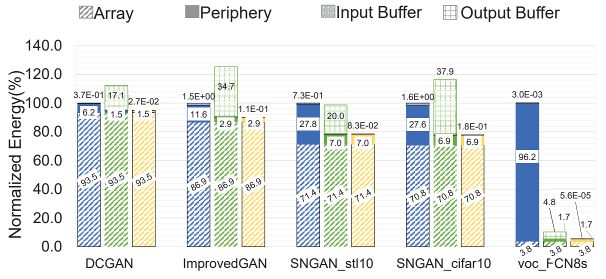


(b) The execution time breakdown

Fig. 11. The execution time comparison.



(a) The energy saving.



(b) The energy breakdown.

Fig. 12. The energy comparison.

the last computation cycle into the total execution time of the padding-free implementation.

Fig. 11(a) compares the execution time of the three implementations. Because of the same number of ReRAM crossbars utilized in a computation cycle, the three implementations have the same execution time for a single computation. Due to the zero redundancy in the input data, the zero-padding design requires more cycles to accomplish the computation of a deconvolutional layer. As interpreted in Section III, the zero-padding design takes $O_h \times O_w$ cycles to complete the computation of M output FPs in the shape of $O_h \times O_w$, while the padding-free design and RED will take $I_h \times I_w$ cycles. As shown in Fig. 11(a), RED reaches a $4\times \sim 56.16\times$ speedup compared to the zero-padding design. RED achieves a substantial improvement of execution time especially in FCN implementations, when the stride of deconvolutional computation is relatively large. Note that the deconvolutional layers in voc-FCN8s have different strides (S = 2 or S = 8).

The padding-free design and RED have the same number of computation cycles, leading to the same array latency and periphery latency. RED saves $29\% \sim 104\%$ execution time compared to the padding-free design. The slight speedup mainly comes from the removal of the extra operations of output data and the output buffer latency.

Fig. 11(b) shows the breakdowns of the execution time. The relationship between the execution time of the three implementations is as expected. The array latency and periphery latency of the zero-padding design is $\sim S^2\times$ compared to the other two designs, as interpreted above. The output buffer of the padding-free design costs $29\% \sim 104\%$ extra execution time. When the data size of the intermediate output in the last deconvolutional layer is relatively larger, as in ImprovedGAN [32] and SNGAN_cifar-10 [35], the total access time of the output buffer is considerable.

C. Energy consumption

Fig. 12(a) compares the energy consumption of the three designs. The energy evaluation considers all the components in accelerators, including arrays, peripheral circuitry, input buffers, etc. Here the overhead induced by the output buffer is included for the padding-free design.

The zero-padding design introduces zero-redundancy to the input dataflow, and consequently requires a much larger input buffer to hold the input dataflow. So the energy consumption for a single buffer access is relatively large. Moreover, the zero-padding design needs more cycles to complete the deconvolutional computation and the input data size in each cycle is larger. So the number of input buffer access is relatively large. Thus, the input buffer energy is larger than that of the other designs. RED doesn't show great advantage over the zero-padding design in GAN execution because the input data size of GANs is relatively small. FCN involves more input zero-redundancy of deconvolutional layers so the energy consumption of input buffer of RED reduces more significantly.

In the padding-free design, the storage of intermediate outputs involves writing to the output buffer, which introduces considerable energy consumption. So RED can achieve an $18\% \sim 49\%$ energy efficiency promotion compared to the padding-free design. This promotion is more significant when the data size of the intermediate output is relatively large, as in ImprovedGAN [32] and SNGAN_cifar-10 [35].

The breakdowns of the major components are shown in Fig. 12(b). Due to the same number of ReRAM crossbars utilized in a computation cycle, the padding-free design and RED have the same array energy consumption and periphery energy consumption. The zero-padding design involves more computation cycles. It has the same total array energy as RED because the zero-redundancy in the input dataflow doesn't in-

TABLE VII
COMPARISON WITH EXISTING DECONVOLUTION ACCELERATORS.

	Computation Efficiency ($\mathrm{GOPs}/(\mathrm{s} \times \mathrm{mm}^2)$)	Energy Efficiency (GOPs/W)
RED (Avg.)	8340.72	41457.71
FCN-Engine [23]	235.4	1412.5
LerGAN [25]	N/A	3588.92
GANAX [26]	N/A	2740.90
FPGA [27]	N/A	73.11
CPU	N/A	0.74
Tesla P100 (GPU)	N/A	14.06

troduce extra array energy consumption. The periphery energy of the zero-padding design is larger because of the larger number of computation cycles. The zero-padding design shows more energy of the input buffer, and the padding-free design involves a considerable output buffer energy consumption, as interpreted above.

D. Comparison with existing deconvolution accelerators

The estimated area of RED is $16.63mm^2$. Based on the experiment results of energy consumption and execution time, we calculate RED's computation efficiency and energy efficiency (Table VII). We compare RED with CPU and GPU platforms, as well as several state-of-the-art deconvolution accelerators [23], [25]–[27]. The computation efficiency of some baselines is not available due to the lack of the runtime or area data. The comparison in Table VII shows that RED achieves approximately $35.45\times$ and $6.67\times$ improvements in computation efficiency and energy efficiency, respectively.

VI. CONCLUSION

This work introduces RED, a ReRAM-based deconvolution accelerator with high energy efficiency and throughput. RED utilizes a performance-friendly mapping scheme and a redundancy-free data flow to reduce considerable latency and energy consumption caused by zero-insertion or extra operations in the conventional deconvolution implementations. A specialized input buffer design is also proposed to support the data flow. Experimental evaluation shows that RED outperforms the existing ReRAM-based accelerators for the common deconvolutional computation algorithms, with up to a 4-56.16× speedup and a 1.05-18.17× energy consumption reduction.

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[3] M. Mancini, G. Costante, P. Valigi, and T. A. Ciarfuglia, "Fast robust monocular depth estimation for obstacle detection with fully convolutional networks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4296–4303.

[4] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[5] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 132–142.

[6] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *arXiv preprint arXiv:1608.07916*, 2016.

[7] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[8] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[9] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 273–287.

[10] B. Li, B. Yan, and H. Li, "An Overview of In-memory Processing with Emerging Non-volatile Memory for Data-intensive Applications," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '19. New York, NY, USA: ACM, 2019, pp. 381–386. [Online]. Available: http://doi.acm.org/10.1145/3299874.3319452

[11] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, jun 2016, pp. 14–26. [Online]. Available: http://ieeexplore.ieee.org/document/7551379/

[12] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *SIGARCH Comput. Archit. News*, vol. 44, 2016, pp. 27–39.

[13] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined ReRAM-based accelerator for deep learning," *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 541–552, 2017.

[14] X. Qiao *et al.*, "Atomlayer: A universal ReRAM-based CNN accelerator with atomic layer computation," in *DAC*, 2018, p. 103.

[15] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "TIME: A training-in-memory architecture for RRAM-based deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 834–847, 2018.

[16] B. Li *et al.*, "ReRAM-based accelerator for deep learning," in *DATE*, 2018, pp. 815–820.

[17] C. J. Xue, G. Sun, Y. Zhang, J. J. Yang, Y. Chen, and H. Li, "Emerging non-volatile memories: opportunities and challenges," in *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2011, pp. 325–334.

[18] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, 2011.

[19] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor crossbar-based neuromorphic computing system: A case study," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 10, pp. 1864–1878, 2014.

[20] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Proceedings of the 53rd annual design automation conference*. ACM, 2016, p. 19.

[21] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, and Y. Xie, "FPSA: A Full System Stack Solution for Reconfigurable Reram-based NN Accelerator Architecture," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 733–747.

[22] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[23] D. Xu *et al.*, "FCN-Engine: Accelerating deconvolutional layers in classic cnn processors," in *ICCAD*, 2018.

[24] F. Chen *et al.*, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *ASP-DAC*, 2018, pp. 178–183.

[25] H. Mao, M. Song, T. Li, Y. Dai, and J. Shu, "LerGAN: A Zero-Free, Low Data Movement and PIM-Based GAN Architecture," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 669–681.

[26] A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "GANAX: A Unified MIMD-SIMD Acceleration for Generative Adversarial Networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 650–661. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00060

[27] M. Song *et al.*, "Towards efficient microarchitectural design for accelerating unsupervised GAN-based deep learning," in *HPCA*. IEEE, 2018, pp. 66–77.

[28] A. Radford *et al.*, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv:1511.06434*, 2015.

[29] P. Y. Chen *et al.*, "NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *IEDM*, 2018, pp. 6–1.

[30] H.-P. D. Company, "CACTI 6.5," 2014. [Online]. Available: http://www.hpl.hp.com/research/cacti

[31] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[32] T. Salimans *et al.*, "Improved techniques for training GANs," in *NIPS*, 2016, pp. 2234–2242.

[33] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[34] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[35] A. Krizhevsky *et al.*, "The CIFAR-10 dataset," 2014.

[36] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge 2012 (voc2012) results (2012)," in *URL http://www. pascal-network. org/challenges/VOC/voc2011/workshop/index.html*, 2011.

[37] Z. Fan, Z. Li, B. Li, Y. Chen, and H. H. Li, "RED: A ReRAM-based deconvolution accelerator," in *2019 Design, Automation and Test in Europe (DATE)*, 2019.

Ziru Li received the Bachelor of Engineering (2019) in Electronic Engineering from Tsinghua University, Beijing, China. He is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at Duke University, Durham, NC, USA. His research interests mainly focus on mixed-signal integrated circuit design for high energy efficient computing.

Bing Li (M'17) received the B.S. degree in Computer Science and Technology from the Minzu University of China in 2010 and the Ph.D. degree in Computer Architecture from the Institute of Computing Technology, University of Chinese Academy of Sciences in 2016. After completing her postdoctoral research at Duke University, Durham, NC, USA, Bing Li joined the Capital Normal University, China as a tenure-track Associate Professor in 2019. Her research interests include computer architecture, memory design, neural network acceleration, and neuromorphic computing systems.

Zichen Fan received the B.S. degree from Tsinghua University, Beijing, China, in 2019. He is currently working toward the Ph.D. degree in the Michigan Integrated Circuit Lab (MICL), University of Michigan, Ann Arbor, MI, USA. His current research interests include algorithm-hardware co-design and energy-efficient digital system design.

Hai (Helen) Li (M'08-SM'16-F'19) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from the Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA, in 2004. Dr. Li currently is the Clare Boothe Luce Associate Professor with the Department of Electrical and Computer Engineering at Duke University, Durham, NC, USA. She has authored or co-authored more than 200 papers in peer-reviewed journals and conferences and a book entitled Nonvolatile Memory Design: Magnetic, Resistive, and Phase Changing (CRC Press, 2011). Her current research interests include neuromorphic computing systems, machine learning and deep neural networks, memory design and architecture, and cross-layer optimization for low power and high performance.

Dr. Li is a Distinguished Lecturer of the IEEE CAS society and a distinguished speaker of ACM. Dr. Li is a recipient of the NSF Career Award, DARPA Young Faculty Award (YFA), and TUM-IAS Hans Fisher Fellowship from Germany. She received seven best paper awards and additional seven best paper nominations from international conferences. Dr. Li serves as Associate Editor of IEEE TCAD, IEEE TVLSI, IEEE TCAS-II, IEEE TMSCS, ACM TECS, IEEE CEM, ACM TODAES, and IET-CPS. She was the General Chair or Technical Program Chair of multiple IEEE/ACM conferences and the Technical Program Committee members of over 30 international conference series. Dr. Li is an IEEE fellow and a distinguished member of the ACM.