ZARA: A Novel Zero-free Dataflow Accelerator for Generative Adversarial Networks in 3D ReRAM

Fan Chen, Linghao Song, Hai "Helen" Li, Yiran Chen
Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, USA
{fan.chen,linghao.song,hai.li,yiran.chen}@duke.edu

ABSTRACT

Generative Adversarial Networks (GANs) recently demonstrated a great opportunity toward unsupervised learning with the intention to mitigate the massive human efforts on data labeling in supervised learning algorithms. GAN combines a generative model and a discriminative model to oppose each other in an adversarial situation to refine their abilities. Existing nonvolatile memory based machine learning accelerators, however, could not support the computational needs required by GAN training. Specifically, the generator utilizes a new operator, called transposed convolution, which introduces significant resource underutilization when executed on conventional neural network accelerators as it inserts massive zeros in its input before a convolution operation. In this work, we propose a novel computational deformation technique that synergistically optimizes the forward and backward functions in transposed convolution to eliminate the large resource underutilization. In addition, we present dedicated control units - a dataflow mapper and an operation scheduler, to support the proposed execution model with high parallelism and low energy consumption. ZARA is implemented with commodity ReRAM chips, and experimental results show that our design can improve GAN's training performance by averagely 1.6× ~23× over CMOS-based GAN accelerators. Compared to state-of-the-art ReRAM-based accelerator designs, ZARA also provides 1.15 × ~2.1× performance improvement.

CCS CONCEPTS

• **Hardware → Hardware accelerators**;

KEYWORDS

GAN; 3D ReRAM; unsupervised learning

ACM Reference Format:
Fan Chen, Linghao Song, Hai "Helen" Li, Yiran Chen. 2019. ZARA: A Novel Zero-free Dataflow Accelerator for Generative Adversarial Networks in 3D ReRAM. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317936

1 INTRODUCTION

Supervised deep learning have been widely used in various modern artificial intelligence (AI) applications, spanning over image and speech recognition [9], object detection [6], natural language processing [3], etc. It has achieved near or even beyond human-level accuracy in classification problems [8, 11]. Training deep neural networks (DNNs), however, requires vast quantity of correctly labeled data. Such a fact greatly constrains the generalization of DNN models to the scenarios where the training data is difficult or costly to obtain.

Recent advances in unsupervised Generative Adversarial Networks (GANs) [7] offer a great opportunity to extend deep learning to applications that conventional supervised learning is not capable to handle [10, 13, 15, 20]. A GAN model consists of two neural networks, a generator that attempts to generate synthetic data from random noise and a discriminator that tries to distinguish the data created by the generator from the actual training samples. The generator and discriminator are co-trained against each other in an unsupervised manner until we receive a generator with strong generation capability and a discriminator with high classification accuracy.

Although GAN has quickly become a popular research topic since it was proposed in 2014 [7], there were few studies on how to accelerate the calculation of GAN models in computing community. Until recently, researchers began to realize that the special mathematical operator in GAN - transposed convolution (TCONV) could not be effectively executed in traditional DNN accelerators such as Eyeriss [2] and Pipelayer [18]. TCONV is a two-stage operation which augments a low-dimension input feature map to a high-dimension richer representation. It first inserts many zeros into the input data and then performs a convolution operation on the expanded feature map at the second stage. The inserted zeros contribute to more than 60% of multiplication and addition operations [25], resulting in severe resource underutilization when conventional DNN accelerators were employed. In this paper, we propose ZARA, a ReRAM-based PIM accelerator that fills the gaps in prior arts. Compared to previous work, we make the following contributions.

- we explore the dataflow and complex computing operations involved in GANs. We design a revolutionary computation deformation to completely eliminate inserted zeros in transposed convolution to improve the computation efficiency and resource underutilization.
- we propose a dataflow mapper and an operation scheduler to mitigate the discrepancy in computation latency caused by deformed flow of data. By leveraging the trade-off between performance and hardware resource, the mapper effectively maps the computation onto processing engines to maximize both spatial and temporal parallelism.
- we implement ZARA with 3D horizontal ReRAM-based architecture and evaluate its effectiveness using five recent

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317936

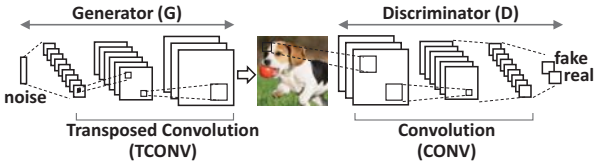

Figure 1: A GAN system.

GAN models on five distinct applications. We show that ZARA delivers significant speedup and energy saving over state-of-the-art GAN accelerators.

2 PRELIMINARY

2.1 GAN and DCGAN

In a GAN framework, two DNNs compete with each other in a minimax game: a generative model G generates synthetic samples from random noise to simulate the real samples, and a discriminative model D distinguishes the generated samples from the real ones. The two models are trained alternatively until they reach an equilibrium. In theory, any two networks with inverse structures can be trained as opponents in a GAN system to refine their abilities. However, the inferior stability of GAN imposes strict constraints on selection of proper networks. Inspired by the recent success of convolutional neural networks (CNNs), deep convolutional generative adversarial networks (DCGAN) [15] was proposed. DCGAN features several principles of network architectures to ensure a stable training, such as 1) replacing pooling layers with strided convolutions in D and transposed convolutions in G, 2) removing fully-connected layer, and 3) using Rectified Linear Unit (ReLU) and LeakyReLU in activation functions in G and D, respectively. Most of the current GAN variations are based on or at least partially based on DCGAN [10, 20, 22] because of its good stability. As illustrated in Figure 1, D uses conventional convolution (CONV) to downsample the input and produces a binary classification. In contrast, G performs transposed convolution (TCONV), which is fundamentally different from CNOV, to map a uniform noise to a high-dimension representation. Table 1 summarizes the notations that are commonly used for explanation of the layer structure and computation of a GAN. Due to page limit, we do not give the full descriptions of those parameters or the mathematical model of the GAN's training process. We refer the interested readers to [1, 18].

Compared to CONV, TCONV first transforms its input to a larger representation by zero paddings around the borders and zero insertions between adjacent rows/columns and then performs performs

Table 1: Notations used for explanation of a GAN.

Symbols/equations		Description
W_l, b_l		kernel matrix, bias of the l^{th} layer
h		activation function
u_l, d_l		ourput FP before, after activation function
δ_l		error matrix of the l^{th} layer
∇W_l		weight derivative of the l^{th} layer
$u_l = W_l d_{l-1} + b_l$	(1)	forward function
$d_l = h(u_l)$	(2)	activation function
$\delta_{l-1} = {W_l}^T \delta_l \circ h'(u_l)$	(3)	error backpropagation
$\nabla W_l = d_{l-1} {\delta_l}^T$	(4)	weight partial derivative
$\nabla b_l = \delta_l$	(5)	bias partial derivative
I_W, I_H, I_C		width, height, channels of input FP
K_W, K_H, K_C, K_N		width, height, channels, numbers of kernels
O_W, O_H, O_C		width, height, channels of output FP
S_W, S_H		TCONV stride on width, height
P_W, P_H		TCONV padding on width, height

$\circ$ *denotes element-wise multiplication.*

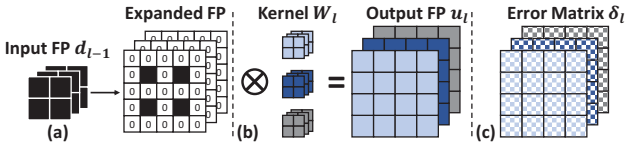

Figure 2: Transposed convolution (TCONV). (a) zero-insertion stage for a 2×2 input FP with stride $S_W = S_H = 2$. (b) convolution between the expanded input and kernels. (c) error matrix for backpropagation.

convolutional computation on the expanded input. The padding on width and height are denoted as P_W and P_H, respectively. The stride on width (height) is denoted as S_W (S_H), representing that $S_w - 1$ ($S_H - 1$) zeros need to be inserted between two adjacent columns (rows). In this way, only approximately $1/S_W$ ($1/S_H$) of the source operands within a row (column) are non-zero and contributes to the final result. For instance, Figure 2(a) illustrates the zero-insertion step for a 2×2 multi-dimensional input feature map (FP) with stride $S_W = S_H = 2$ and padding $P_W = P_H = 1$. Then 3 multi-dimensional kernels (a.k.a. weight matrix) convolve with this transformed input FP and correspondingly generate 3 output FPs with a size of 4×4, as shown in Figure 2(b). In this example, 63 zeros are inserted to each input FP, that is, only 12/75 (16%) of the source operands are valid input and contributes to the final results.

As a training framework, a backward pass (represented by Equation (3)-(5) in Table 1) is essential for updating a GAN model. Figure 2(c) shows the error matrix, which has the same size with output FP and can be back-propagated to the previous layer. The error and the expanded FPs are then multiplied at each layer to calculate the partial derivative of weights (∇W) and bias (∇b).

2.2 ReRAM-based Vector-Matrix Multiplier

Resistive Random Access Memory (ReRAM) is an emerging non-volatile memory that uses the resistance of a dielectric solid-state device to store data. Recent works [1, 17, 18] demonstrated that vector-matrix multiplication (VMM), which is the major operations in neural network models, can be efficiently conducted using a ReRAM crossbar with more than 100× efficiency improvement over conventional designs. Figure 3(a) illustrates a 2D ReRAM-based VMM. Each input on the wordlines (WLs) of a two-dimensional ReRAM crossbar is connected to all the bitlines (BLs) via ReRAM cells. The elements of the matrix are represented as the conductance of the ReRAM cells and the input vector is represented by the input voltage on the WLs. The output currents from the BLs, hence, represents the result of vector-matrix multiplication. By adding

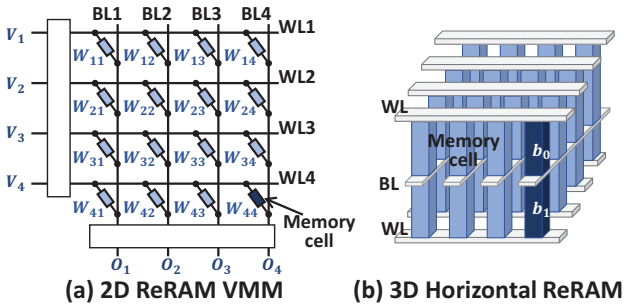

Figure 3: The ReRAM basics.

some necessary peripheral circuits (e.g., analog-digital converters, sample-and-hold units, shift-and-add units, etc.) onto the ReRAM crossbar, we are able to effectively implement the calculations involved in the neural networks.

Previous work [13] demonstrated a DNN inference accelerator based on 128 × 128 2-bit Multi-level Cell (MLC) ReRAM arrays with a lifetime of only one day. For a DNN learning accelerator, the life problem will be more severe as the high precision and frequent writes required by the training process quickly wear the ReRAM cells out. In our work, we choose 3D horizontal Single-bit Cell (SLC) ReRAM crossbar as the basic computing unit to balance the computation density and lifetime of the design. As shown in Figure 3(b), a 2-bit cell in 3(a) can be split into two 1-bit cells. These two designs have the exactly same peripheral circuits but the lifetime of ReRAM arrays can achieve $\sim 10^3\times$ improvement [13].

3 RELATED WORKS AND MOTIVATION

Zhang *et. al* [26] proposed to accelerate TCONV on FPGA. Fcn-engine [23] and Red [5] further realizes a fully convolutional accelerator that can handle both CONV and TCONV operations using unified processing elements (PEs). These work only focus on inference task but do not support training function. ReGAN [1] implemented a full-fledged GAN accelerator using emerging ReRAM but it adopts the inefficient zero-insertions in TCONV and neglects the redundant operations. Zhang *et. al* [19] proposed a CMOS-based GAN accelerator where zero operands can be skipped by carefully mapping the GANs onto the PEs. Its dataflow, however, imposes strong limitations on the topology of the GANs (i.e. $S_W = S_H = 2$), making it very inflexible to serve various GAN variations. Flexi-GAN [24] and GANAX [25] reorder the output computation and allocate computing rows with similar patterns of zeros to adjacent PEs for efficient TCONV execution. To support their reorganized dataflow, these two works both implement decoupled access and execute micro-engines to support interleaving MIMD and SIMD operations, resulting in increased design complexity. LerGAN [14] presented kernel reshaping scheme to handle zero-related computation in GAN and implemented the proposed design with 3D-connected Process-in-Memory architecture in ReRAM. The forward and backward functions in TCONV are considered as two independent processes in the reshaping scheme proposed in LerGAN, so the optimization is not explored to the maximum. The PIM architecture employs an ideal 3D connection technology that is difficult to implement even with the most advanced circuit technologies available today. Different from all the above designs, our work completely eliminates the zero-insertion related in TCONV computations. We collaboratively optimize the forward and backward propagation with a novel deformation scheme and implemented the proposed design with commodity ReRAM chips.

4 ZARA

4.1 Overview of ZARA

Figure 4 illustrates the ZARA workflow. ZARA takes in a high-level specification of a GAN model which defines network typologies, number of parameters and other information of both the generative model and the discriminative model. We propose to deform the TCONV computation by decomposing the original kernel matrix into multiple sub-kernels which convolve on the input FP directly without zero-insertions. The output FPs can then be periodic shuffled and combined to obtain equivalent output results. The TCONV deformation results in discrepancy in computation latency, which breaks the pipelined execution model. Therefore, we carefully design a **Mapper** and a **Scheduler** to effectively map and schedule the operations of the optimized dataflow onto **Processing Engines** (PEs) to eliminate the discrepancy in execution time. We will provide the details of each components in the following sections.

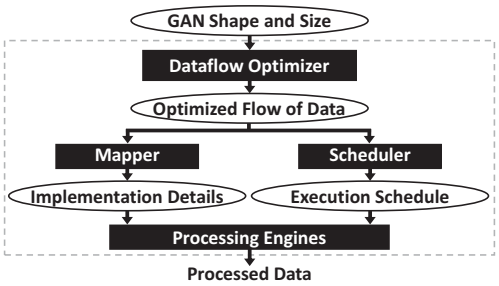

Figure 4: Workflow of ZARA.

4.2 Dataflow Optimizer

Due to the zero insertions in TCONV, executing GAN by following conventional convolution dataflow may result in over 60% resource underutilization. To solve this issue, we first studied the computational characteristics of TCONV. Taking the example in Figure 5(a), we observed that the convolution between the expanded FP and the kernels only contains four distinct valid patterns, represented by different colors. The elements in odd output rows only related to the elements in the even rows of the kernel weight matrix, whereas the even output rows consume elements in the odd rows of the weight matrix. Building upon this observation, we propose to pre-classify the weight kernels into multiple subsets. Convolution execution occurs only between the input and relevant subsets in both the forward and backward phases of a TCONV, thereby alleviating the aforementioned inefficiency in TCONV execution. Below we will detail the proposed data flow optimization scheme.

Forward function dataflow optimization is illustrated in Figure 5(b). Note that we assume $P_W < S_W - 1$ and $P_H < S_H - 1$, which is valid in practice. In this case, the pre-classification of the weight kernels is completely dependent on S_W and S_H. In the special case where the assumption is not true, some boundary subsets need to be added to produce boundary elements in the output FP. In this example, the kernels are first categorized into $S_W \times S_H$ subsets. In this way, the convolution between expanded input FP and kernels translates into the convolution of original input FP with multiple

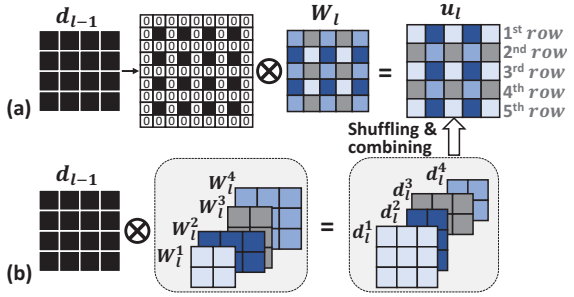

Figure 5: (a) 2D TCONV operation for a 4×4 input and a 5×5 kernel. (b) Optimized dataflow for performing 2D TCONV.

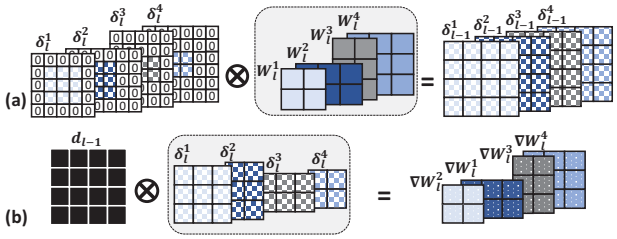

Figure 6: Optimized dataflow of (a) error backpropagation. (b) weight derivative.

kernel subsets. The output FP is thus obtained by shuffling the elements in multiple subsets of output FPs and combining them together.

Error propagation from layer l is performed inversely to layer $l-1$ as denote in Equation (3) in Table 1. $h^{'}(u_l)$ denotes partial derivative of activation function. Note that in DCGAN, the activation function used for G is ReLU, for D is LeakyReLU, which means that $h^{'}(u_l)$ is either 0 or a constant. Hence, the $\circ$ can be realized by AND operator. For error backpropagation, the transposed weight matrix W_l needs to convolve with error matrix δ_l. Based on the deformed dataflow in the forward function, we learn that each output element in layer l is only related to a subset of kernels. Therefore, to get the error for layer $l-1$, we can just convolve δ_l with the corresponding subset of weight matrix. As depicted in Figure 6(a), mathematically, we pad K_H-1 (K_W-1) rows (columns) of zeros on the edges and then perform the convolutions accordingly. The ineffectual calculation with zero operand on the edge can be handled by the scheduler with simple zero skipping method. δ_{l-1} is then obtained by adding four error sub-matrix element-wisely.

Partial derivatives of weights and bias are represented by Equation (4) and (5) in Table 1. The partial derivatives of bias can be easily obtained from δ_{l-1}. The partial derivatives of weight matrix can be obtained by the convolution illustrated in Figure 6(b). In this case, the error sub-matrix can be viewed as kernels which convlve with the output FP of layer $l-1$. The results are then be used to update the weight matrix and bias in layer l.

4.3 Mapper and Scheduler

Previous works [1, 17, 18] have shown that convolutions can be represent as matrix multiplications through the Toeplitz matrix. The data input and the Toeplitz matrix can be then mapped onto the ReRAM crossbars. Take the example in Figure 5, we assume the kernel channels and number of kernels are $K_C = 64$ and $K_N = 512$, respectively. Figure 7 (a) illustrate the naïvely mapping for ${W_l}^1$ and ${W_l}^2$ on to ReRAM crossbars with a size of 128×128. We find that decomposing kernel matrix is efficient to eliminate zero operands in TCONV computation but breaks the regular execution models. The different sizes of output FPs lead to unbalanced runtime – 9 cycles to produce ${d_l}^1$ and 6 cycles for ${d_l}^2$. As such, if one uses a convolution accelerator for TCONV operations, the processing engines that are performing the operations for a kernel sub-matrix with fewer number of output have to remain idle until the operations for other kernel sub-matrix finish, making it impossible for pipelined execution as in [1, 18]. To address this challenges, we dedicated a mapper unit and a scheduler unit to deal with the variability in computation latency. The mapper unit effectively mapping the optimized flow of data on to PEs to eliminate the discrepancy in

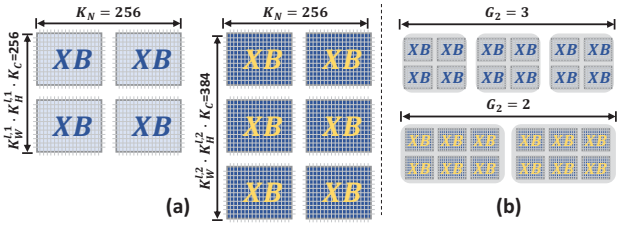

Figure 7: (a) Mapping kernel sub-matrix onto VMM. (b) Spatial parallelism.

execution time, while the scheduler manages the operation flow. These two units work together to maximize the benefits of both spatial and temporal parallelism.

In general, the mapper utilize the trade-off between hardware resources and performance to realize spatial parallelism. As shown in Figure 7(b), we define parallelism degree as G, representing G copies of the naïve implementation will be duplicated. In this example, we have $G_1 = 3$ and $G_2 = 2$ to ensure consistent computing time. The scheduler schedules GAN training accordingly in a pipelined fashion to exploit the temporal parallelism.

4.4 ZARA PIM

Figure 8 shows the architecture of one ZARA node. ZARA leverages ReRAM crossbars to perform in-memory GAN execution. At the top level, each ZARA node consists of a ReRAM memory to store input/output values, a mapper and a scheduler that maps the optimized dataflow onto target PEs and controls the computation flow, respectively, an IO interface to communicate with other ZARA nodes, and a number of processing engines (PEs) connected via on-chip mesh. Each PE contains multiple vector-matrix multipliers, a ReRAM buffer to cache temporal data, activation units to perform nonlinear function, and output register to aggregate results, all connected with a shared bus. A PE also has simple algorithm and logic units (sALU) and shift&add units. Each VMM has a few ReRAM crossbars which shares an ADC, a number of 1-bit DAC, sample & hold units and shift & add units. The crossbar arrays within a VMM shares a driver by global wordline. At a time, only 1 crossbar array in a VMM can be activated and used to perform in-situ vector-matrix multiplications. The results can be selected via a bitline multiplex. The details of these components are summarized as the follows:

Shift and add (S&A). For a large matrix that can not fit in a single PE, the input and the output shall be partitioned and grouped into

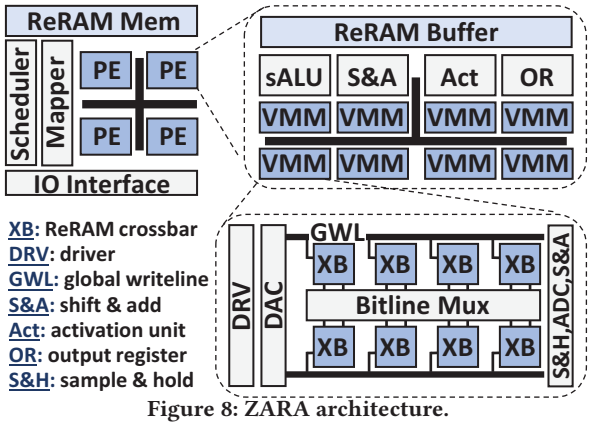

Figure 8: ZARA architecture.

multiple PEs. The output of each PE is a partial sum, which is collected horizontally and summed vertically via a shift-and-add unit to generate the actual results.

Activation unit (Act). It implements the activation function used in a GAN application. In this work we focus on Rectified Linear Unit (ReLU) and LeakyReLU as they are most widely generator and discriminator, respectively. We realize the activation unit as a look up table (LUT) as previous work [1, 18].The LUT can be bypassed in certain scenarios, e.g., when a large matrix is mapped to multiple crossbar arrays.

Special algorithm and logic units (sALU). The sALU provides three types of functions for data pre- and post-processing in this work: (i) vectorvector multiplication used in error back-propagation; (ii) a scalar multiplication; and (iii) element-wise addition of multiple error sub-matrix in the error propagation pass.

Driver (DRV). The driver is used to program the weights to ReRAM cells before computation and input the data onto writlines for vector-matrix multiplication.

Sample-and-hold (S&H). It captures the bitline current, converts the current to a voltage and sends the voltage to an analog-digital converter (**ADC**) unit.

Digital-analog converter (DAC). It converts digital inputs into corresponding voltages applied to each WL. In this work we assume that every WL receives one-bit of its input voltage each cycle. As such, the expensive DAC can be realized by an inverter [17].

Analog-digital converter (ADC). ADCs convert the analog signals produced by ReRAM crossbars to digital output results. It costs > 60% power consumption in a VMM [17]. In this work, we share a ADC across 8 ReRAM crossbars to amortize the overhead of expensive ADCs.

5 EXPERIMENTAL SETUP

Benchmarks. We evaluated the ZARA architecture using four state-of-the-art GANs. Table 2 summarizes the evaluated networks, dataset, and the topological structures of G and D that are represented by the numbers of CONV and TCONV layers of the adopted generative and discriminative models. For comparison purpose, we also include fully convolution network [12] in evaluation.

Schemes. We compared the proposed ZARA architecture with four counterparts shown in Table 3. The ADC, DAC, S&H, S&A, activation logic designs and the same 32nm process technology used in an existing ReRAM DNN accelerator [17] are adopted in our evaluations. The bus and connections are modeled and estimated using Cadence Virtuoso with TSMC 32nm technology. We used NVSim [4] to estimate the latency, power and area of ReRAM arrays. We adopted the ReRAM cell model from [21] with a size of 256×256. Each 3D ReRAM crossbar shares 256 1-bit inverters to replace the expensive DACs [17]. In this design, each PE contains 4 VMMs, while each VMM has 8 ReRAM crossbars.

Table 2: Benchmarks (C: CONV layer; T: TCONV layer; F: fully connected layer.)

Model	Year	Dataset	D Topology	G Topology
FCN [12]	2015	POSCAL VOC	5C, 2F	N/A
DCGAN [15]	2016	LSUN	4C, 1F	1F, 4T
iGAN [16]	2016	CIFAR-10	3C, 1F	1F, 3T
3DGAN [22]	2016	IKEA	5C, 1F	1F, 4T
ArtGAN [20]	2017	CIFAR-10	6C, 1F	1F, 5T

Table 3: Simulated scheme comparison

Name	Year	Description
FCN-engine [23]	2018	CMOS TCONV accelerator
GANAX [25]	2018	CMOS GAN acclerator with zero skipping
Pipelayer [18]	2017	ReRAM-based CNN training accelerator
ReGAN [1]	2018	ReRAM-based GAN accelerator without zero skipping

Energy and area model. We implement the ZARA microarchitectural units using Verilog, including sALU, S&A, Act, the mapper, the scheduler, the connection and bus, and other logic hardware units. We use TSMC 32nm standard cell library and Synopsys Design Compiler to synthesize these units to obtain the area, delay, and energy numbers.

6 EVALUATION

Comparisons to a CPU and a GPU. Figure 9(a) shows the performance comparison between a CPU (Intel E5-2630 v3 8-core), a GPU (NVIDIA Geforce GTX 1080) and ZARA. The results are normalized to the CPU's run time. ZARA achieves the best performance across all the benchmarks. Such significant performance improvement comes from two reasons: 1) ZARA eliminates the redundancy of zeros in input FPs and pipelines the execution after removing the discrepancy in computation cycles; and 2) the in-situ analog vector-matrix multiplication reduces the off-chip memory data accesses. Compared to the CPU and GPU platforms, ZARA achieves averagely 146× and 7.6× performance improvement, respectively.

Figure 9(b) presents the energy comparison between a CPU, a GPU and ZARA. The results are normalized to the CPU's energy consumption when training each applications of interest. Thanks to the high energy-efficiency of ReRAM arrays, ZARA provides averagely 87.5× and 3.6× energy savings compared to the CPU and GPU platforms. The results also show that the energy saving of ZARA increases with the complexity of generative models. Thus, ArtGAN on cifar10 obtains the largest energy saving, i.e., 111× and 4.1× compared to the CPU and GPU counterparts, respectively.

Performance improvement over other DNN accelerators. The performance comparison of all accelerators is shown in Figure 10(a). For simple benchmarks like FCN, FCN-engine and GANAX offer performance comparable to the ReRAM-based accelerators such as Pipelayer and ReGAN. For complex benchmarks, the ReRAM-based accelerators significantly outperform the CMOS-based accelerator. This result is mainly caused by the limited hardware resources of the CMOS-based accelerators. On average, ZARA achieves 1.6× ∼ 23× speedup over the CMOS-based accelerators. Compared with the

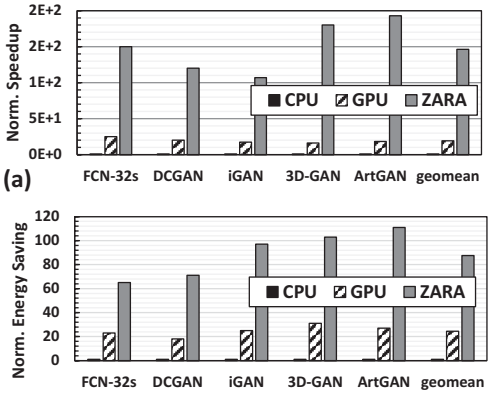

Figure 9: Comparison to CPU and GPU.

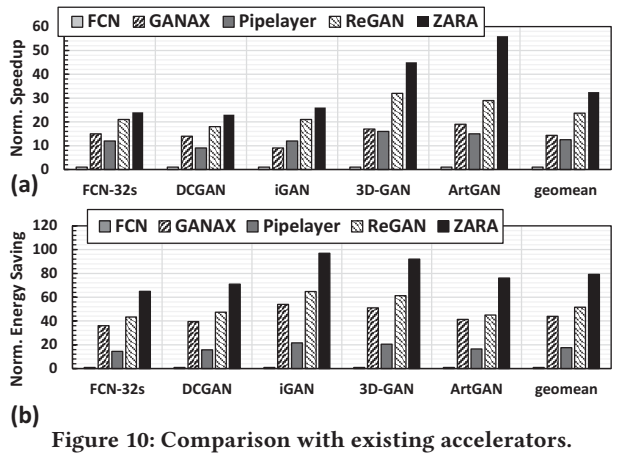

Figure 10: Comparison with existing accelerators.

ReRAM-based Pipelayer and ReGAN, ZARA achieves 2.1× and 1.15× performance speedup, respectively.

Energy improvement over other DNN accelerators. We compare the energy consumption of all the concerned DNN accelerators, as shown in Figure 10(b). Among all the accelerators, ZARA consumes the least energy in GAN executions. Compared with the ReRAM-based ReGAN and Pipeleyer, on average, the energy savings of ZARA are 4.5× and 1.5×, respectively. Compared with the CMOS-Based FCN-engine and GANAX, the energy savings of ZARA are as high as 78× and 1.8×, respectively.

Reliability improvement over ReRAM-based DNN accelerators. Figure 11 shows by adopting SLC 3D ReRAM, the lifetime of ZARA can be as long as 3.4 years when constantly training GAN while the lifetime of Pipelayer and ReGAN are only 3 months and 5 months, respectively.

7 CONCLUSION

In this work, we propose a process-in-memory accelerator for generative adversarial networks. We first present a novel computation deformation for transposed convolution to synergistically optimizes both forward and backward functions to eliminate the large resource underutilization due to zero-insertions. This deformed flow of data inevitably leads to discrepancy in computation latency. We then proposed a unique dataflow mapper and operation scheduler by leveraging both the spatial and temporal parallelism. Different from previous research, we use ReRAM-based in-memory computation since ReRAM is capable of both computation and storage, significantly reducing the data movement. Compared to state-of-the-art DNN accelerator designs, our design can substantially improve GAN's training performance without requiring additional computing resources.

ACKNOWLEDGMENTS

This work was supported in part by NSF 1725456, NSF1744082 and DOE DE-SC0017030. 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.

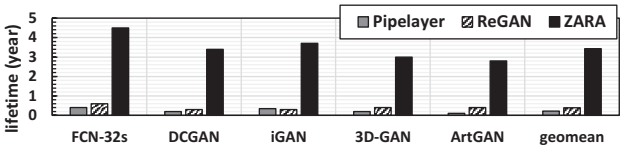

Figure 11: Lifetime comparison.

REFERENCES

[1] F. Chen, L. Song, and Y. Chen. 2018. ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks. In *ASP-DAC*.
[2] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*.
[3] Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *ICML*.
[4] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. 2012. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
[5] Zichen Fan, Ziru Li, Bing Li, Yiran Chen, and Helen Hai Li. 2019. RED: A ReRAM-based Deconvolution Accelerator. In *DATE*.
[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*.
[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NIPS*. Curran Associates, Inc.
[8] Benjamin Graham. 2014. Fractional Max-Pooling. *ArXiv e-prints*.
[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*.
[10] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. 2017. Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. *ArXiv e-prints*.
[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE*.
[12] J. Long, E. Shelhamer, and T. Darrell. 2015. Fully convolutional networks for semantic segmentation. In *CVPR*.
[13] Qian Lou, Wujie Wen, and Lei Jiang. 2018. 3DICT: A Reliable and QoS Capable Mobile Process-in-memory Architecture for Lookup-based CNNs in 3D XPoint ReRAMs. In *ICCAD*.
[14] H. Mao, M. Song, T. Li, Y. Dai, and J. Shu. 2018. LerGAN: A Zero-Free, Low Data Movement and PIM-Based GAN Architecture. In *MICRO*.
[15] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ArXiv e-prints*.
[16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. *ArXiv e-prints*.
[17] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *ISCA*.
[18] L. Song, X. Qian, H. Li, and Y. Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *HPCA*.
[19] M. Song, J. Zhang, H. Chen, and T. Li. 2018. Towards Efficient Microarchitectural Design for Accelerating Unsupervised GAN-Based Deep Learning. In *HPCA*.
[20] Wei Ren Tan, Chee Seng Chan, Hernan Aguirre, and Kiyoshi Tanaka. 2017. ArtGAN: Artwork Synthesis with Conditional Categorical GANs. *ArXiv e-prints*.
[21] W. Wen, L. Zhao, Y. Zhang, and J. Yang. 2017. Speeding up crossbar resistive memory by exploiting in-memory data patterns. In *ICCAD*.
[22] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*.
[23] Dawen Xu, Kaijie Tu, Ying Wang, Cheng Liu, Bingsheng He, and Huawei Li. 2018. FCN-engine: Accelerating Deconvolutional Layers in Classic CNN Processors. In *ICCAD*.
[24] Amir Yazdanbakhsh, Michael Brzozowski, Behnam Khaleghi, Soroush Ghodrati, Kambiz Samadi, Nam Sung Kim, and Hadi Esmaeilzadeh. 2018. FlexiGAN: An End-to-End Solution for FPGA Acceleration of Generative Adversarial Networks. *FCCM*.
[25] Amir Yazdanbakhsh, Kambiz Samadi, Nam Sung Kim, and Hadi Esmaeilzadeh. 2018. GANAX: A Unified MIMD-SIMD Acceleration for Generative Adversarial Networks. In *ISCA*.
[26] Xinyu Zhang, Srinjoy Das, Ojash Neopane, and Ken Kreutz- Delgado. 2017. A Design Methodology for Efficient Implementation of Deconvolutional Neural Networks on an FPGA. *ArXiv e-prints*.