Distributed In-Memory Computing on Binary RRAM Crossbar

LEIBIN NI, Nanyang Technological University, Singapore
HANTAO HUANG, Nanyang Technological University, Singapore
ZICHUAN LIU, Nanyang Technological University, Singapore
RAJIV V. JOSHI, IBM T. J. Watson Research Center, Yorktown Heights, New York
HAO YU, Nanyang Technological University, Singapore

The recent emerging resistive random-access memory (RRAM) can provide non-volatile memory storage but also intrinsic computing for matrix-vector multiplication, which is ideal for low-power and high-throughput data analytics accelerator performed in memory. However, the existing RRAM-crossbar based computing is mainly assumed as a multi-level analog computing, whose result is sensitive to process non-uniformity as well as additional overhead from AD-conversion and I/O. In this paper, we explore the matrix-vector multiplication accelerator on a binary RRAM-crossbar with adaptive 1-bit-comparator based parallel conversion. Moreover, a distributed in-memory computing architecture is also developed with according control protocol. Both memory array and logic accelerator are implemented on the binary RRAM-crossbar, where logic-memory pair can be distributed with protocol of control bus. Experiment results have shown that compared to the analog RRAM-crossbar, the proposed binary RRAM-crossbar can achieve significant area-saving with better calculation accuracy. Moreover, significant speedup can be achieved for matrix-vector multiplication in the neural-network based machine learning such that the overall training and testing time can be both reduced respectively. In addition, large energy saving can be also achieved when compared to the traditional CMOS-based out-of-memory computing architecture.

CCS Concepts: •**Hardware** → **Non-volatile memory; Emerging architectures;** •**Computing methodologies** → **Supervised learning by classification;**

Additional Key Words and Phrases: RRAM crossbar, L2-norm based machine learning, Hardware accelerator

1. INTRODUCTION

Future cyber-physical system requires efficient real-time data analytics [Kouzes et al. 2009; Wolpert 1996; Hinton et al. 2006; Müller et al. 2008; Glorot and Bengio 2010] with applications in robotics, brain-computer interface as well as autonomous vehicles. The recent works in [Huang et al. 2006; Coates et al. 2011] have shown a great potential for machine learning with significant reduced training time for real-time data analytics.

Hardware-based accelerator is currently practiced to assist machine learning. In traditional hardware accelerator, there is intensive data migration between memory and logic [Kumar et al. 2014; Park et al. 2013] caused both bandwidth and power walls. Therefore, for data-oriented computation, it is beneficial to place logic accelerators as close as possible to the memory to alleviate the I/O communication overhead [Wang et al. 2015]. The cell-level in-memory computing is proposed in [Matsunaga

---

This preliminary result of this work was published at ASP-DAC'16 [Ni et al. 2016].
The work of H. Yu was supported in part by Singapore NRF-CRP (NRF-CRP9-2011-01) and MOE Tier-2 (MOE2015-T2-2-013).
Author's addresses: L. Ni and H. Huang and Z. Liu and H. Yu, School of Electrical and Electronic Engineering, VIRUTUS, Nanyang Technological University, 637798, Singapore. R. V. Joshi, IBM T. J. Watson Research Center, Yorktown Heights, New York, 10598. Please make comments to haoyu@ntu.edu.sg.

DOI: 0000001.0000001

et al. 2009], where simple logic circuits are embedded among memory arrays. Nevertheless, the according in-memory logic that is equipped in memory cannot be made for complex logic function, and also the utilization efficiency is low as logic cannot be shared among memory cells. In addition, there is significant memory leakage power in CMOS based technology.

Emerging resistive random-access memory (RRAM) [Akinaga and Shima 2010; Kim et al. 2011; Chua 1971; Williams 2008; Strukov et al. 2008; Shang et al. 2012; Fei et al. 2012; Wong et al. 2012] has shown great potential to be the solution for data-intensive applications. Besides the minimized leakage power due to non-volatility, RRAM in crossbar structure has been exploited as computational elements [Kim et al. 2011; Liu et al. 2015]. As such, both memory and logic components can be realized in a power- and area- efficient manner. More importantly, it can provide a true in-memory logic-memory integration architecture without using I/Os. Nevertheless, the previous RRAM-crossbar based computation is mainly based on an analog fashion with multi-level values [Kim et al. 2012] or Spike Timing Dependent Plasticity (STDP) [Lu et al. 2011]. Though it improves computation capacity, the serious non-uniformity of RRAM-crossbar at nano-scale limits its wide applications for accurate and repeated data analytics. Moreover, there is significant power consumption from additional AD-conversion and I/Os mentioned in [Lu et al. 2011].

In this paper, we propose a distributed in-memory accelerator on binary RRAM-crossbar for machine learning. Both computational energy efficiency and robustness are greatly improved by a binary RRAM-crossbar for memory and logic units. The memory arrays are paired with the in-memory logic accelerators in a distributed fashion, operated with a protocol of control bus for each memory-logic pair. Moreover, different from the multi-leveled analog RRAM-crossbar, a three-step digitalized RRAM-crossbar is proposed in this paper to perform a digital matrix-vector multiplication. For the pure matrix-vector multiplication, simulation results show that $2.86\times$ faster speed, $105.6\times$ better energy efficiency, and $36.23\times$ smaller area can be achieved in binary RRAM-crossbar compared to the same implementation by CMOS-ASIC. For the machine learning based face recognition, the proposed accelerator is $4.34\times$ speed-up, $13.08\times$ energy-saving and $51.3\times$ area-saving compared to the CMOS-ASIC.

The rest of this paper is organized as follows. The L2-norm based machine learning algorithm and its major computational operations are discussed in Section 2. Section 3 shows the novel distributed RRAM-crossbar based in-memory computing architecture (XIMA). Section 4 introduces the binary RRAM-crossbar for matrix-vector multiplication. Section 5 presents the mapping details for matrix multiplication on the digitalized RRAM crossbar. Experimental results are presented in Section 6 with conclusion in Section 7.

2. BACKGROUND OF MACHINE LEARNING

The current data analytics is mainly based on machine learning algorithm to build a model to correlate input data with targeted output. Neural network is the common model to build [Haykin et al. 2009], and usually has two computational phases: training and testing. In the training phase, the weight coefficients of the neural network model are determined by minimizing the error between the trial and the targeted using the training input data. In the testing phase, the neural network with determined coefficients is utilized for the classification of the new testing data.

However, the input data may be in high dimension with redundant information. To facilitate the training, feature extraction is usually needed performed to represent the characteristic data with redundancy or dimension reduction.

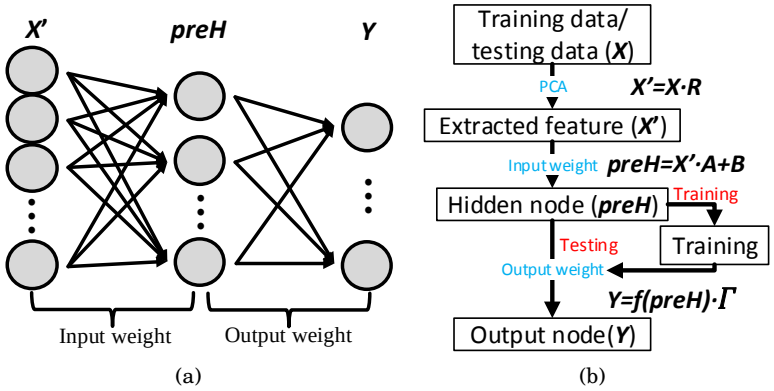


Fig. 1. (a) Single-layer neural network (b) General flow of neural network training and testing

2.1. Feature Extraction

In general, the feature of original data $\mathbf{X}$ can be extracted by projection,

$$\mathbf{X}' = \mathbf{R} \cdot \mathbf{X} \tag{1}$$

where $\mathbf{X}'$ is the extracted feature. The projection matrix $\mathbf{R}$ can be found with the use of principal/singular components, random embedding or convolution [Wold et al. 1987]. Matrix $\mathbf{R}$ is computed off-line and used for dimension reductions. We can treat $\mathbf{R}$ as a new basis to represent columns of $\mathbf{X}$; and remove those small values to minimize the total squared reconstruction error by

$$||\mathbf{X}' - \mathbf{R} \cdot \mathbf{X}||_2 \tag{2}$$

One can observe significant matrix-vector multiplications during the feature extraction as shown in (1).

2.2. Neural Network based Learning

After feature extraction, one can perform various machine learning algorithms [Suykens and Vandewalle 1999; LeCun et al. 2012; Huang et al. 2006] for data analytics. As shown in Fig. 1 for a typical neural network model, one needs to determine the network weights from training and then practice testing. We use n to represent the number of features with training input $X_f \in R^{N \times n}$. n is the training data size. The extracted feature will be input to the neural network with following relationship for the first layer output $\mathbf{preH}$:

$$\mathbf{preH} = \mathbf{X}_f\mathbf{A} + \mathbf{B},\ \mathbf{H} = g(\mathbf{preH}) = \frac{1}{1 + e^{-\mathbf{preH}}} \tag{3}$$

where $\mathbf{A} \in \mathbb{R}^{n \times L}$ and $\mathbf{B} \in \mathbb{R}^{N \times L}$ are randomly generated input weight and bias formed by a_{ij} and b_{ij} between $[-1, 1]$; $\mathbf{H}$ is the hidden-layer output matrix generated from the Sigmoid function $g(\cdot)$ for activation.

The training of neural network is to minimize error with an objective function below

$$\min_{\mathbf{\Gamma}}\ ||\mathbf{H\Gamma} - \mathbf{T}||_2^2 + \eta|||\mathbf{\Gamma}||_2^2 \tag{4}$$

where η is the regularized parameter and $\mathbf{T}$ is the label of training data.

One can solve (4) either by iterative backward propagation method [Werbos 1990] or direct L2-norm solver method for least-squares problem [Huang et al. 2006]. Compared to the L1 based sparse coding [Chen et al. 2015], (4) is more accurate in error

calculation. The output weight can be obtained as $||\tilde{\mathbf{H}}\mathbf{\Gamma} - \tilde{\mathbf{T}}||$, and can be solved as

$$\begin{aligned} \mathbf{\Gamma} &= (\tilde{\mathbf{H}}^T\tilde{\mathbf{H}})^{-1}\tilde{\mathbf{H}}^T\tilde{\mathbf{T}} = (\mathbf{H}^T\mathbf{H} + \eta\mathbf{I})^{-1}\mathbf{H}^T\mathbf{T} \\ where\ \tilde{\mathbf{H}} &= \begin{pmatrix} \mathbf{H} \\ \sqrt{\eta}\mathbf{I} \end{pmatrix},\ \ \tilde{\mathbf{T}} = \begin{pmatrix} \mathbf{T} \\ \mathbf{0} \end{pmatrix} \end{aligned} \tag{5}$$

Here $\tilde{\mathbf{H}} \in \mathbb{R}^{(N+L)\times L}$ is formed based on $\mathbf{H}$ and $\mathbf{I}$. For matrix $\mathbf{\Gamma}$, it is the solution of a least-square problem, where we adopt Cholesky decomposition to solve it [Higham 2009]. We have also analyzed the major computations of Cholesky decomposition for least-square problem, which will be discussed in Section 6.

As a result, in the testing phase, output node $\mathbf{Y}$ is calculated by already determined hidden node value and output weight value as

$$\mathbf{Y} = \mathbf{H} \cdot \mathbf{\Gamma} \tag{6}$$

The index of the maximum value in $\mathbf{Y}$ represents the class that the test data belongs to.

Based on the computation analysis on feature extraction and neural network, we can observe that matrix-vector multiplication is the dominant operation as shown in (1), (5) and (6). As such, a hardware accelerator to facilitate the matrix-vector multiplication is indeed the critical requirement for the efficient machine-learning based data analytics.

3. XIMA ARCHITECTURE

3.1. Overview of Distributed In-memory Computing

Conventionally, processor and memory are separate components that are connected through I/Os. With limited width and considerable RC-delay, the I/Os are considered the bottleneck of system overall throughput. As memory is typically organized in H-tree structure, where all leaves of the tree are data arrays, it is promising to impose in-memory computation with parallelism at this level. In this paper, we propose a distributed RRAM-crossbar in-memory architecture (XIMA). Because both data and logic units have uniform structure when implemented on RRAM-crossbar, half of the leaves are exploited as logic elements and are paired with data arrays. The proposed architecture is illustrated in Fig. 2. The distributed local data-logic pairs can form one local data path such that the data can be processed locally in parallel, without the need of being readout to the external processor.

Coordinated by the additional controlling unit called *in-pair control bus* the in-memory computing is performed in following steps. (1) logic configuration: processor issues the command to configure logic by programming logic RRAM-crossbar into specific pattern according to the functionality required; (2) load operand: processor sends the data address and corresponding address of logic accelerator input; (3) execution: logic accelerator can perform computation based on the configured logic and obtain results after several cycles; (4) write-back: computed results are written back to data array directly but not to the external processor.

With emphasis on different functionality, the RRAM crossbars for data storage and logic unit have distinctive interfaces. The data RRAM-crossbar will have only one row activated at one time during read and write operations, and logic RRAM-crossbar; however, we can have all rows activated spontaneously as rows are used to take inputs. As such, the input and output interface of logic crossbar requires AD/DA conversions, which could outweigh the benefits gained. Therefore, in this paper, we propose a conversion-free digital-interfaced logic RRAM crossbar design, which uses three lay-

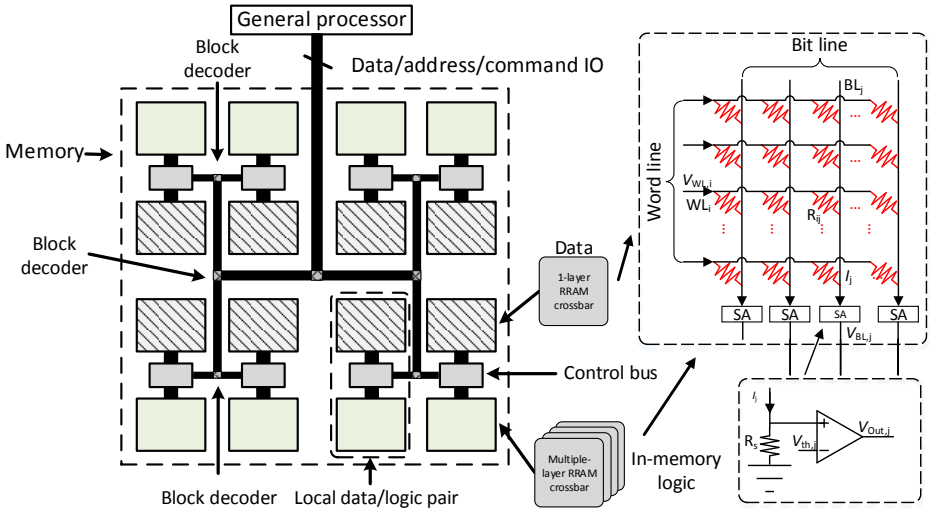


Fig. 2. Overview of distributed in-memory computing architecture on RRAM crossbar

ers of RRAM crossbars to decompose a complex function into several simple operations that digital crossbar can tackle.

3.2. Communication Protocol

The conventional communication protocol between external processor and memory is composed of *store* and *load* action identifier, address that routes to different locations of data arrays, and data to be operated. With additional in-memory computation capacity, the proposed distributed in-memory computing architecture requires modifications

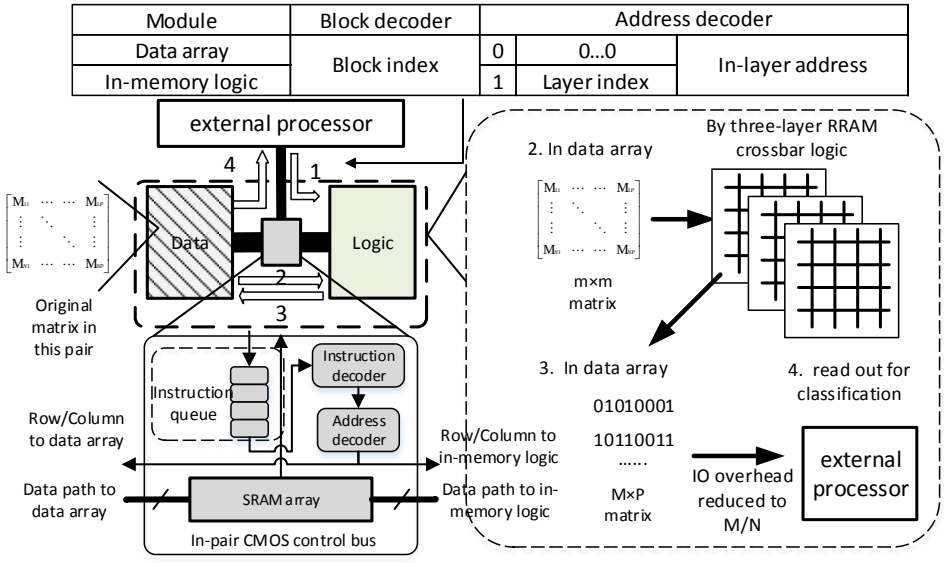


Fig. 3. Detailed structure of control bus and communication protocol

Table I. Protocols between external processor and control bus

Inst.	Op. 1	Op. 2	Action	Function
SW	Addr 1	Addr 2	Addr 1 data to Addr 2	store data, configure logic, in-memory results write-back
	Data	Addr	store data to Addr	
LW	Addr	-	read data from Addr	standard read
ST	Block Idx	-	switch logic block on	start in-memory computing
WT	-	-	wait for logic block response	halt while performing in-memory computing

on the current communication protocol. The new communication instructions are proposed in TABLE I, which is called in-pair control.

In-pair control bus needs to execute instructions in TABLE I. SW (store word) instruction is to write data into RRAMs in data array or in-memory logic. If target address is in data array, it will be a conventional write or result write-back; otherwise it will be logic configuration. LW (load word) instruction performs as conventional read operation. ST (start) instruction means to switch on the logic block for computing after the computation setup has been done. WT (wait) operation is to stop reading from instruction queue during computing.

Besides communication instructions, memory address format is also different from that in the conventional architecture. To specify a byte in the proposed architecture, address includes the following identifier segments. Firstly, the data-logic pair index segment is required, which is taken by block decoders to locate the target data-logic pair. Secondly, one-bit flag is needed to clarify that whether the target address is in data array or in-memory logic crossbar. Thirdly, if logic accelerator is the target, additional segment has to specify the layer index. Lastly, rest of address segment are row and column indexes in each RRAM-crossbar. An address example for data array and in-memory logic is shown in Fig. 3.

To perform logic operation, the following instructions are required to performed. Firstly, we store the required input data and RRAM values with SW operation. Secondly, an ST instruction will be issued to enable all the columns and rows to perform the logic computing. The WT instruction is also performed to wait for the completion of logic computing. At last, LW instruction is performed to load the data from the output of RRAM-crossbar.

3.3. Control Bus

Given the new communication protocol between general processor and memory is introduced, one can design the according control bus as shown in Fig. 3. The control bus is composed of an instruction queue, an instruction decoder, an address decoder and a SRAM array. As the operation frequency of RRAM-crossbar is slower than that of external processor, instructions issued by the external processor will be stored in the instruction queue first. They are then analyzed by instruction decoder on a first-come-first-serve (FCFS) basis. The address decoder obtains the row and column index from the instruction; and SRAM array is used to store temporary data such as computation results, which are later written back to data array.

4. BINARY RRAM-CROSSBAR FOR MATRIX-VECTOR MULTIPLICATION

In this paper, we implement matrix-vector multiplication in the proposed XIMA. It is one always-on operation in various data-analytic applications such as compressive sensing, machine learning. For example, the feature extraction can be achieved by multiplying Bernoulli matrix in [Wright et al. 2009].

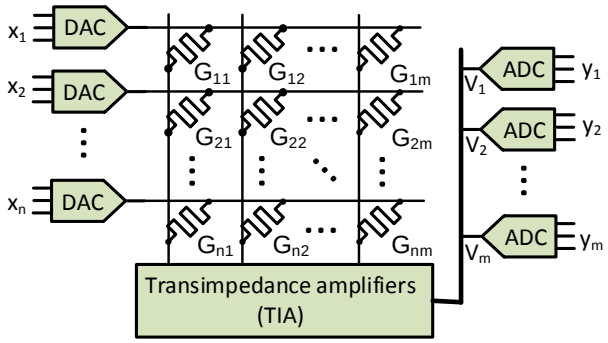


Fig. 4. Traditional analog-fashion RRAM crossbar with ADC and DAC

Matrix multiplication can be denoted as $Y = \Phi X$, where $X \in \mathbb{Z}^{N \times P}$ and $\Phi \in \{0,1\}^{M \times N}$ are the multiplicand matrices, and $Y \in \mathbb{Z}^{M \times P}$ is the result matrix.

4.1. RRAM Device and Crossbar

RRAM is a two-terminal device that can be observed in sub-stoichiometric transition metal oxides (TMOs) sandwiched between metal electrodes. Such a device can be used as non-volatile memory with state of ion resistance, which results in 2 non-volatile states: high resistance state HRS and low resistance state LRS. One can change the state from HRS to LRS or vice versa by applying a SET voltage (V_w) or a RESET voltage ($-V_w$).

The two states RHS and LRS represent 0 and 1, respectively. To read a RRAM cell, one can apply a read voltage V_r to the RRAM. The V_r and V_w follow

$$V_w > V_{th} > V_w/2 > V_r, \tag{7}$$

where V_{th} is the threshold voltage of the RRAM.

Because of the high density of RRAM device, one can build a crossbar structure as the array of RRAM [Kim et al. 2011; Kang et al. 2014; Fan et al. 2014; Gu et al. 2015; Srimani et al. 2015; Wang et al. 2014]. Such crossbar structure can be utilized as memory for high-density data storage. The memory array can be read or written by controlling the voltage of wordlines (WLs) and bitlines (BLs). For example, we can apply $V_w/2$ on the i^{th} WL and $-V_w/2$ on the j^{th} BL to write data into the RRAM cell on i^{th} row, j^{th} column.

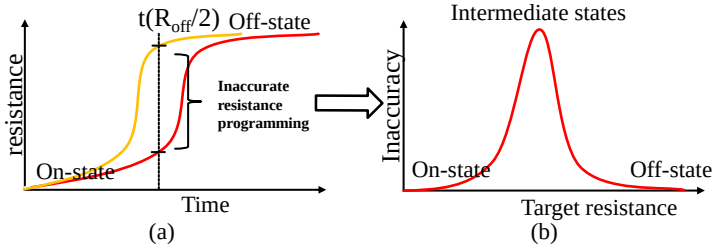


Fig. 5. (a) Switching curve of RRAM under device variations (b) Programing inaccuracy for different RRAM target resistances

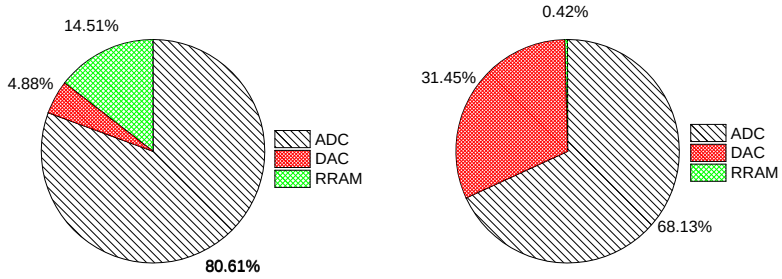


Fig. 6. (a) Power consumption of analog-fashion RRAM crossbar (b) Area consumption of analog-fashion RRAM crossbar

4.2. Traditional Analog RRAM Crossbar

The fabric of crossbar intrinsically supports matrix-vector multiplication where vector is represented by row input voltage levels and matrix is denoted by mesh of RRAM resistances. As shown in Fig. 4, by configuring Φ into the RRAM crossbar, analog computation $y = \Phi x$ by RRAM crossbar can be achieved.

However, such analog RRAM crossbar has two major drawbacks. Firstly, the programming of continuous-valued RRAM resistance is practically challenging due to large RRAM process variation [Joshi et al. 2011]. Specifically, the RRAM resistance is determined by the integral of current flowing through, which leads to a switching curve as shown in Fig. 5 (a). With the process variation, the curve may shift and leave intermediate values very unreliable to program, as shown in Fig. 5 (b). Secondly, the A/D and D/A converters are both timing-consuming and power-consuming. In our simulation, the A/D and D/A conversion may consume up to 85.5% of total operation energy in $65nm$ as shown in Fig. 6.

4.3. Proposed Digitalized RRAM Crossbar

To overcome the aforementioned issues, we propose a full-digitalized RRAM crossbar for matrix-vector multiplication. Firstly, as ON-state and OFF-state are much more reliable than intermediate values shown in Fig. 5, only binary values of RRAM are allowed to reduce the inaccuracy of RRAM programming. Secondly, we deploy a pure digital interface without A/D conversion.

In RRAM crossbar, we use V_{wl}^i and V_{bl}^j to denote voltage on ith wordline (WL) and jth bitline (BL). R_{off} and R_{on} denote the resistance of off-state and on-state. In each sense amplifier (SA), there is a sense resistor R_s with fixed and small resistance. The relation among these three resistance is $R_{off} \gg R_{on} \gg R_s$. Thus, the voltage on jth BL can be presented by

$$V_{bl}^j = \sum_{i=1}^{m} g_{ij} V_{wl}^i R_s \tag{8}$$

where g_{ij} is the conductance of R_{ij}.

The key idea behind digitalized crossbar is the use of comparators. As each column output voltage for analog crossbar is continuous-valued, comparators are used to digitize it according to the reference threshold applied to SA in Fig. 2,

$$O_j = \begin{cases} 1, \ if \ V_{bl}^j \geq V_{th}^j \\ 0, \ if \ V_{bl}^j < V_{th}^j \end{cases} \tag{9}$$

However, the issue that rises due to the digitalization of analog voltage value is the loss of information. To overcome this, three techniques are applied. Firstly, multi-thresholds are used to increase the quantization level so that more information can be preserved. Secondly, the multiplication operation is decomposed into three sub-operations that binary crossbar can well tackle. Thirdly, the thresholds are delicately selected at the region that most information can be preserved after the digitalization.

5. IMPLEMENTATION OF DIGITAL MATRIX MULTIPLICATION

In this section, hardware mapping of matrix multiplication on the proposed architecture is introduced. The logic required is a matrix-vector multiplier by the RRAM-crossbar. Here, a three-step RRAM-crossbar based binary matrix-vector multiplier is proposed, in which both the input and output of the RRAM-crossbar are binary data without the need of ADC. The three RRAM-crossbar step: parallel digitizing, XOR and encoding are presented in details as follows. As the output of an RRAM-crossbar array can be connected to the input of another RRAM-crossbar array, we can use multiple RRAM arrays in the logic block for the mappings. Here we use symbol s to denote the result of binary matrix-vector multiplication. Therefore, s follows

$$0 \leq s \leq N, \tag{10}$$

where N is the maximum result. To illustrate the three-step procedure more clearly, we will use the following matrix-vector multiplication as an example:

$$[00101011] \times [10111110]^{\mathrm{T}} = 3 \tag{11}$$

The output after the three-step procedure will be shown when $s = 3$ and $N = 8$.

5.1. Parallel Digitizing

The first step is called parallel digitizing, which requires $N \times N$ RRAM crossbars. The idea is to split the matrix-vector multiplication to multiple inner-product operations of two vectors. Each inner-product is produced by one RRAM crossbar. For each crossbar, as shown in Fig. 7, all columns are configured with same elements that correspond to one column in random Boolean matrix Φ, and the input voltages on word-lines (WLs) are determined by x. As $g_{on} \gg g_{off}$, current on RRAMs with high impedance are

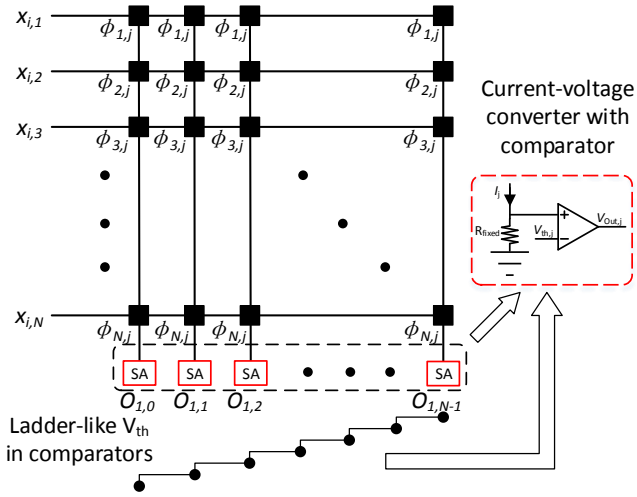


Fig. 7. Parallel digitizing step of RRAM crossbar in matrix multiplication

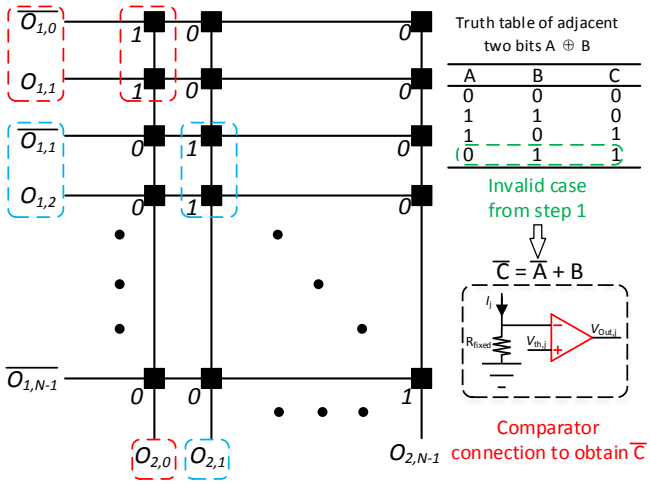


Fig. 8. XOR step of RRAM crossbar in matrix multiplication

insignificant, so that the voltages on BLs approximately equal to $kV_r g_{on} R_s$ according to Eq. (8) where k is the number of RRAM with in low-resistance state (g_{on}).

It is obvious that voltages on bit-lines (BLs) are all identical. Therefore, the key to obtain the inner-product is to set ladder-type sensing threshold voltages for each column,

$$V_{th,j} = \frac{(2j+1)V_r g_{on} R_s}{2}, \tag{12}$$

where $V_{th,j}$ is the threshold voltage for the j_{th} column. The $O_{i,j}$ is used to denote the output of column j in RRAM crossbar step i after sensing. Therefore, for the output we have

$$O_{1,j} = \begin{cases} 1, & j \leq s \\ 0, & j > s, \end{cases} \tag{13}$$

where s is the inner-product result. In other words, the first $(N-s)$ output bits are 0 and the rest s bits are 1 ($s <= N$). In our example, $x_i = [00101011]$ and $\phi_i = [10111110]$, and the corresponding output $O_1 = [11100000]$.

5.2. XOR

The inner-product output of parallel digitizing step is determined by the position where $O_{1,j}$ changes from 0 to 1. The XOR takes the output of the first step, and performs XOR operation for every two adjacent bits in $O_{1,j}$, which gives the result index. For the same example of $s = 3$, we need to convert the first-step output $O_1 = [11100000]$ to $O_2 = [001000000]$. The XOR operation based on RRAM crossbar is shown in Fig. 8. According to parallel digitizing step, $O_{1,j}$ must be 1 if $O_{1,j+1}$ is 1. Therefore, XOR operation is equivalent to the AND operation $O_{1,j} \oplus O_{1,j+1} = O_{1,j}\overline{O_{1,j+1}}$, and therefore we have

$$\overline{O_{2,j}} = \begin{cases} \overline{O}_{1,j} + O_{1,j+1}, & j < N-1 \\ \overline{O}_{1,j}, & j = N-1. \end{cases} \tag{14}$$

In addition, the threshold voltages for the columns have to follow

$$V_{th,j} = \frac{V_r g_{on} R_s}{2} \tag{15}$$

Eqs. (14) and (15) show that only output of s_{th} column is 1 on the second step, where s is the inner product result. Each crossbar in XOR step has the size of $N \times (2N-1)$.

5.3. Encoding

The third step takes the output of XOR step and produces s in binary format as an encoder. Therefore, O_3 should be in the binary format of s. In our example, $O_3 = [00000011]$ when $s = 3$. In the output of XOR step, as only one input will be 1 and others are 0, according binary information is stored in corresponding row, as shown in Fig. 9. Encoding step needs $N \times n$ RRAMs, where $n = \lceil \log_2 N \rceil$ is the number of bits in order to represent N in binary format. The thresholds for the encoding step are set following Eq. 15 as well.

For activation function in (3), the exponentiation and division operations can be implemented by look-up table (LUT). The output of XOR layer is the index of $\mathbf{preH}$. Therefore, the encoding layer performs as a LUT mapping process from $\mathbf{preH}$ to $\mathbf{H}$. For example, if we want to map $\mathbf{preH} = 3$ to $\mathbf{H} = 0.953$, we can make the RRAMs in column 3 as the binary format of 0.953 ($[11110100]$, no-signed, 8-bit fixed point with 256 scaling factor). As a result, the mapping process can be included in the encoding step.

5.4. Adding and Shifting for Inner-product Result

The output of encoding step is in binary format, but some processes are needed to obtain the final inner-product result. Adder and shifter are designed to complete this process as shown in Fig. 10. We suppose the original data is 8-bit and data dimension is 512, the workload of adder is 512 without any acceleration. With three-step RRAM-crossbar accelerator as pre-processing, the workload of adder can be significantly reduced to 9 ($\log_2 512$). Detailed comparison results will be shown in Section 6.

6. EXPERIMENTAL RESULT

6.1. Experiment Settings

The hardware evaluation platform is implemented on a computer server with 4.0GHz core and 16.0GB memory. Feature extraction is implemented by general processor,

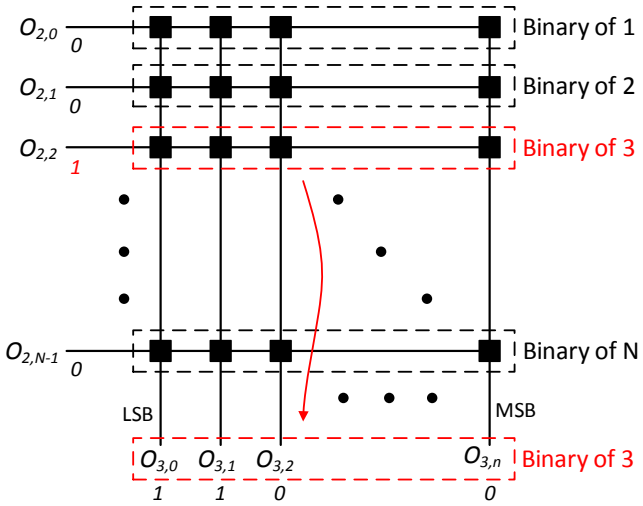


Fig. 9. Encoding step of RRAM crossbar in matrix multiplication

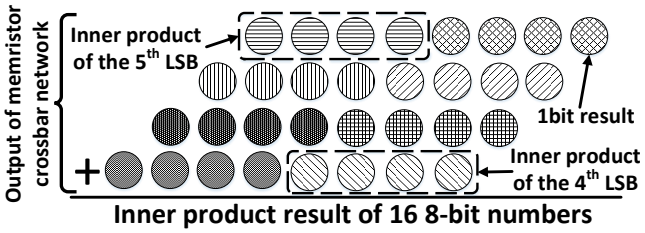


Fig. 10. RRAM-based inner-product operation

CMOS-based ASIC, non-distributed and distributed in-memory computing based on digitalized RRAM crossbar respectively. For the RRAM-crossbar design evaluation, the resistance of RRAM is set as 1kΩ and 1MΩ as on-state and off-state resistance respectively according to [Lee et al. 2008].

The general processor solution is performed by Matlab on a 4.0GHz desktop. For CMOS-ASIC implementation, we implement it by Verilog and synthesize with CMOS 65nm low power PDK. For RRAM-crossbar based solution, we verify the function in circuit level with SPICE tool NVMSPICE [Yu and Wang 2014]. By analyzing the machine learning algorithm, we obtain the basic operations and the number of RRAM-crossbar logic units required.

The working frequency of general processor implementation is 4.0GHz while the CMOS ASIC feature extraction design frequency is 1.0GHz. For in-memory computing based on the proposed RRAM crossbar, write voltage V_w is set as $0.8V$ and read voltage V_r is set as $0.1V$ as well as duration time of $5ns$. In addition, the analog computation on RRAM-crossbar is performed for comparison based on design in [Singh et al. 2007].

In the followings, we will discuss the computation complexity in machine learning first. Hereafter, the performance evaluation of pure matrix-vector multiplication and machine learning will be shown. The effects of scalability is also studied.

6.2. Computation Complexity Analysis

In the experiment, 200 face images of 13 people are selected from [Huang et al. 2007], with scaled image size 262 of each image $\mathbf{X}$. In PCA, feature size of image is further reduced to 128 by multiplying the matrix $\mathbf{R}$. The number of hidden node L and classes m are 160 and 13, respectively. Based on the experimental settings, computation complexity is analyzed with results shown in Fig. 11. 82% of computations are multiplication in output weight calculation, which is the most time-consuming procedure in neural network. Time-consumption of each process in neural network is introduced in Fig. 11(b). Since processes except activation function involve matrix-vector multiplication, we extracted this operation in the whole algorithm and found that 64.62% of time is consumed in matrix-vector multiplication, shown in Fig. 11(c).

6.3. General Performance Comparison of Matrix-vector Multiplication

To evaluate the performance of binary RRAM-crossbar for matrix-vector multiplication, we use the proposed architecture to accelerate dimension reduction of fingerprint images. 1,000 fingerprint images selected from [Tan and Sun 2010] are stored in memory with 328×356 resolution, with 8 bits in each pixel. To agree with patch size, random Bernoulli $N\times M$ matrix is with fixed N and M of 356 and 64, respectively. The original images can be seen as $X\in\mathbb{Z}^{N\times P}$ in matrix-vector multiplication. The detailed comparison is shown in Table II with numerical results including energy consumption and delay obtained for one image on average of 1,000 images. For the digitied XIMA imple-

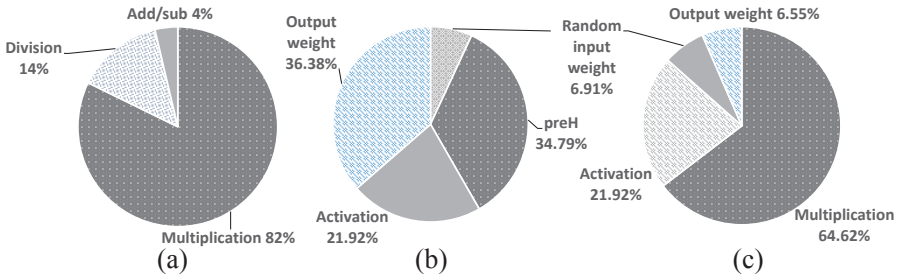


Fig. 11. (a) Time consumption breakdown for output weight calculation (b) neural network training computation effort analysis (c) Multiplication analysis for neural network training ($N = 200$, $n = 128$, $L = 160$ and $m = 13$)

Table II. Matrix-vector multiplication performance comparison under among software and hardware implementation

Implementation	General purpose processor (MatLab)	CMOS ASIC	Non-distributed digitalized XIMA	Distributed digitalized XIMA	Distributed analog XIMA
Area	$177mm^2$	$5mm^2$	$3.28mm^2$ (800 MBit RRAMs) + $0.088mm^2$ + $128\mu m^2$	$0.05mm^2$ (12 MBit RRAMs) + $0.088mm^2$ + 8192 μm^2	$8.32mm^2$
Frequency	4GHz	1GHz	200MHz	200MHz	200MHz
Cycles	-	69,632	Computing: 984 Pre-computing: 262,144	Computing: 984 Pre-computing: 4,096	Computing: 328 Pre-computing: 4,096
Time	$1.78ms$	$69.632\mu s$	Computing: $4{,}920ns$ Pre-computing: $1.311ms$	Computing: $4{,}920ns$ Pre-computing: $20.48\mu s$	Computing: $1{,}640ns$ Pre-computing: $20.48\mu s$
Dynamic power	$84W$	$34.938W$	RRAM: $4.71W$ Control-bus: $100\mu W$	RRAM: $4.71W$ Control-bus: $6.4mW$	RRAM: $1.28W$ Control-bus: $6.4mW$
Energy	$0.1424J$	$2.4457mJ$	RRAM: $23.17\mu J$ Control-bus: $0.131\mu J$	RRAM: $23.17\mu J$ Control-bus: $0.131\mu J$	RRAM: $2.1\mu J$ RRAM: $0.131\mu J$

mentation, we need to compute the area of RRAM cell, adding and shifting as well as the control bus. For analog XIMA, the majority of area is consumed by ADC/DACs and area of RRAM cell can be neglected.

Among hardware implementations, in-memory computing based on the proposed XIMA achieves better energy-efficiency than CMOS-based ASIC. Non-distributed XIMA (only one data and logic block inside memory) needs fewer CMOS control bus but large data communication overhead on a single-layer crossbar compared to distributed RRAM crossbar. Although distributed analog RRAM crossbar can achieve the best in energy perspective but has larger area compared to the digitalized one. Shown in Table II, RRAM crossbar in analog fashion only consumes $2.1\mu J$ for one vector mul-

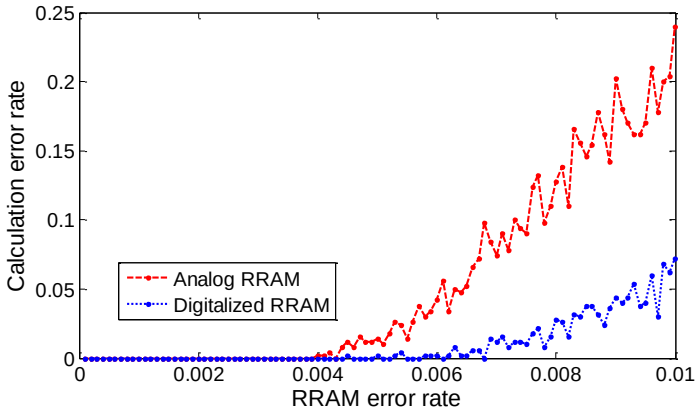


Fig. 12. Calculation error comparison between multi-leveled and binary RRAM

Table III. Face recognition performance comparison under among software and hardware implementation

Implementation	General purpose processor (MatLab)				CMOS-ASIC				Distributed in-memory RRAM-crossbar architecture			
Frequency	$4.0GHz$				$1.0GHz$				$200MHz$			
Area	$177mm^2$				$5.64mm^2$				$0.11mm^2$			
Power	$84W$				$39.41W$				$13.1W$			
Computation	PCA (1)	Input layer (3)	L2-norm (4)(5)	Output layer (6)	PCA (1)	Input layer (3)	L2-norm (4)(5)	Output layer (6)	PCA (1)	Input layer (3)	L2-norm (4)(5)	Output layer (6)
Cycles	-	-	-	-	195,000	121,900	12,256,400	12,440	7,680	4,800	566,400	486
Time	$1.56ms$	$0.98ms$	$92.5ms$	$0.1ms$	$195\mu s$	$121.9\mu s$	$12.26ms$	$12.4\mu s$	$38.4\mu s$	$24\mu s$	$2,832\mu s$	$2.4\mu s$
Energy	$131mJ$	$82mJ$	$7,770mJ$	$8.4mJ$	$7.68mJ$	$4.80mJ$	$483.2mJ$	$0.49mJ$	$0.5mJ$	$0.3mJ$	$37.1mJ$	$0.03mJ$
Speed-up	-				$7.56\times(95140\mu s:12589.3\mu s)$				$32.84\times(95140\mu s:2896.8\mu s)$			
Energy-saving	-				$16.11\times(7991.4mJ:496.17mJ)$				$210.69\times(7991.4mJ:37.93mJ)$			

tiplication while the proposed architecture requires $23.17\mu J$ because most of power consumption comes from RRAM in computing instead of ADCs. However, ADCs need more area so that RRAM crossbar with analog fashion is $8.32mm^2$ while the proposed one is only $0.15mm^2$ because of the high density of RRAM crossbar.

Calculation error of analog and digitalized RRAM crossbar are compared in Fig. 12, where M and N are both set as 256. Calculation error is very low when RRAM error rate is smaller than 0.004 for both analog and digitalized fashion RRAM. However, when RRAM error rate reaches 0.01, calculation error rate of analog RRAM crossbar goes to 0.25, much higher than the other one with only 0.07. As such, computational error can be reduced in the proposed architecture compared to analog fashion RRAM crossbar.

6.4. Scalability Study

Hardware performance comparison among CMOS-based ASIC, non-distributed and distributed XIMA with varying M is shown in Fig. 13. From area consumption perspective shown in Fig. 13(a), distributed RRAM-crossbar is much better than the

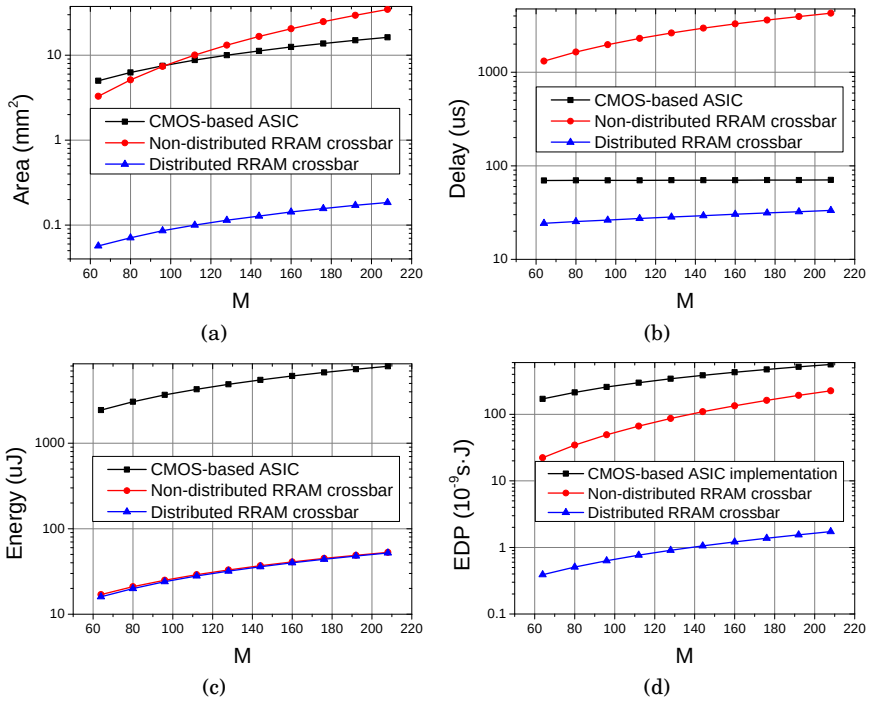


Fig. 13. Hardware Performance Scalability under Different Reduced Dimension for (a) area; (b) delay; (c) energy (d) EDP

other implementations. With increasing M from 64 to 208, its total area is from $0.057mm^2$ to $0.185mm^2$, approximately 100x smaller than the other two approaches. Non-distributed RRAM crossbar becomes the worst one when $M > 96$. From delay perspective shown in Fig. 13(b), non-distributed RRAM crossbar is the worst because it has only one control bus and takes too much time on preparing of computing. Delay of non-distributed RRAM crossbar grows rapidly while distributed RRAM crossbar and CMOS-based ASIC implementation maintains on approximately $21\mu s$ and $70\mu s$ respectively as the parallel design. For energy-efficiency side shown in Fig. 13(c), both non-distributed and distributed RRAM crossbar do better as logic accelerator is off at most of time. The proposed architecture also performs the best in energy-delay product (EDP) shown in Fig. 13(d). Distributed XIMA performs the best among all implementation under different specifications. The EDP is from $0.3 \times 10^{-9} s \cdot J$ to $2 \times 10^{-9} s \cdot J$, which is 60x better than non-distributed RRAM crossbar and 100x better than CMOS-based ASIC.

What is more, hardware performance comparison with varying N is shown in Fig. 14. Area and energy consumption trend is similar to Fig. 13. But for computational delay, the proposed architecture cannot maintain constantly as Fig. 13(b) because it needs much time to configure the input, but still the best among the three. Distributed XIMA still achieves better performance than the other two.

6.5. General Performance Comparison of Machine Learning based Face Recognition

In Table III, general performance comparisons among MatLab, CMOS-ASIC and RRAM-crossbar accelerator are introduced, and the acceleration of each procedure as

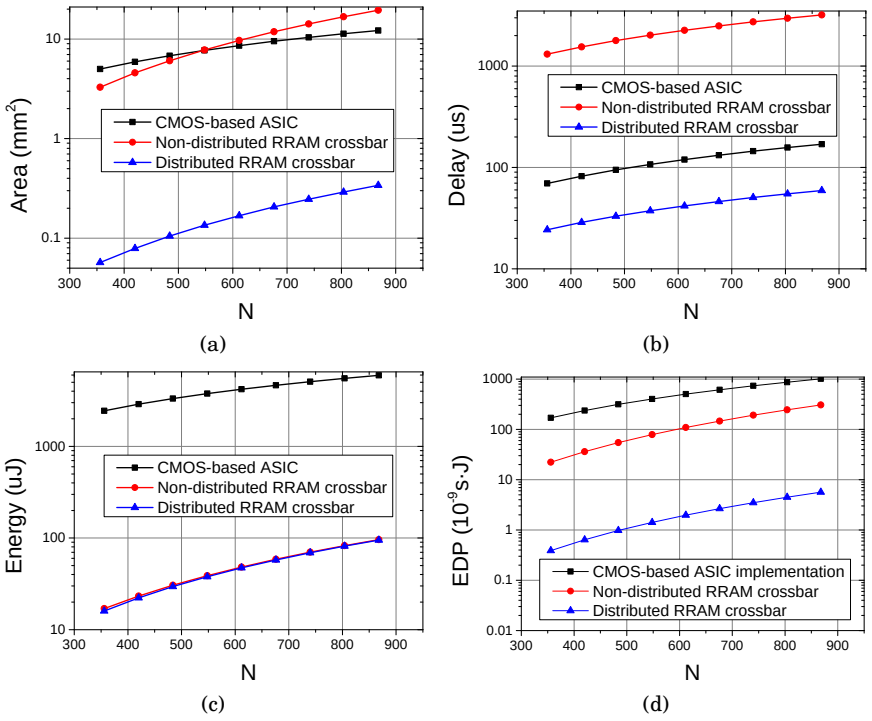
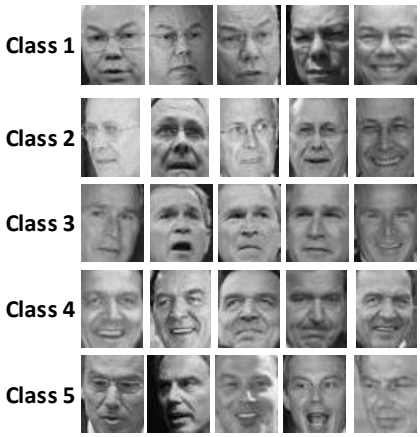


Fig. 14. Hardware Performance Scalability under Different Original Dimension for (a) area; (b) delay; (c) energy (d) EDP



Test cases/ classes	Test case 1 (Y) 	Test case 2 (Y) 	Test case 3 (Y) 
Class 1	1.518062	-0.79108	-0.58029
Class 2	-0.29803	-0.87155	-0.24397
Class 3	-0.95114	0.793256	-0.35867
Class 4	-0.65597	-0.44714	-0.70879
Class 5	-0.65955	0.262689	0.872497
⋮	⋮	⋮	⋮

Fig. 15. Training samples and prediction value $\mathbf{Y}$ (6) for face recognitions

the formula described in Section 2 is also addressed. Among three implementations, RRAM-crossbar architecture performs the best in area, energy and speed. Compared to MatLab implementation, it achieves $32.84\times$ speed-up, $210.69\times$ energy-saving and almost four-magnitude area-saving. We also design a CMOS-ASIC implementation with similar structure as RRAM-crossbar with better performance compared to MatLab. RRAM-crossbar architecture is $4.34\times$ speed-up, $13.08\times$ energy-saving and $51.3\times$ area-saving compared to CMOS-ASIC. The performance comparison is quite different from Table II, because we applied different designs (RRAM-crossbar size) is this two experiments according to the dimension of matrices.

The result of face recognition is shown in Fig. 15. Five training classes are provided as an example with three test cases. Each test face will be recognized as the class with the largest score (prediction result as the index of $Max(\mathbf{Y})$, marked in red color). In this example, case 1 is identified as class 1, while case 2 and 3 are classified into class 3 and 5, respectively.

7. CONCLUSION

In this paper, we presented the distributed in-memory matrix-vector multiplication accelerator using binary RRAM-crossbar for machine learning. The design of three-step digital matrix multiplier on the binary RRAM-crossbar is presented. Moreover, the distributed in-memory computing architecture is introduced with the according control protocol of the digital memory-logic pair. The performance of the mapped machine learning can be boosted by the proposed accelerator with significant improvement in speed and energy efficiency. Experiment results have shown that as for the matrix-vector multiplication, 72% smaller error can be observed when compared to the analog RRAM-crossbar. Moreover, $2.86\times$ speedup and $105.6\times$ power saving can be achieved when compared to the CMOS-ASIC. In addition, as for the machine learning based face recognition, $4.34\times$ speedup and $13.08\times$ power saving can be also achieved when compared to the CMOS-ASIC.

REFERENCES

Hiroyuki Akinaga and Hisashi Shima. 2010. Resistive random access memory (ReRAM) based on metal oxides. *Proc. IEEE* 98, 12 (2010), 2237–2251.

Pai-Yu Chen, Deepak Kadetotad, Zihan Xu, Abinash Mohanty, Binbin Lin, Jieping Ye, Sarma Vrudhula, Jae-sun Seo, Yu Cao, and Shimeng Yu. 2015. Technology-design co-optimization of resistive cross-point

array for accelerating learning algorithms on chip. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 854–859.

Leon O Chua. 1971. Memristor-the missing circuit element. *Circuit Theory, IEEE Transactions on* 18, 5 (1971), 507–519.

Adam Coates, Andrew Y Ng, and Honglak Lee. 2011. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*. 215–223.

Deliang Fan, Mrigank Sharad, and Kaushik Roy. 2014. Design and synthesis of ultralow energy spin-memristor threshold logic. *Nanotechnology, IEEE Transactions on* 13, 3 (2014), 574–583.

Wei Fei, Hao Yu, Wei Zhang, and Kiat Seng Yeo. 2012. Design exploration of hybrid cmos and memristor circuit by new modified nodal analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 20, 6 (2012), 1012–1025.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*. 249–256.

Peng Gu, Boxun Li, Tianqi Tang, Shimeng Yu, Yu Cao, Yu Wang, and Huazhong Yang. 2015. Technological exploration of rram crossbar array for matrix-vector multiplication. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 106–111.

Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. 2009. *Neural networks and learning machines*. Vol. 3. Pearson Education Upper Saddle River.

Nicholas J. Higham. 2009. Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics* 1, 2 (2009), 251–254. DOI:http://dx.doi.org/10.1002/wics.18

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

Gary B Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. 2007. *Labeled faces in the wild: A database for studying face recognition in unconstrained environments*. Technical Report. Technical Report 07-49, University of Massachusetts, Amherst.

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. 2006. Extreme learning machine: theory and applications. *Neurocomputing* 70, 1 (2006), 489–501.

Rajiv Joshi, Rouwaida Kanj, Peiyuan Wang, and Hai Helen Li. 2011. Universal statistical cure for predicting memory loss. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 236–239.

Jinfeng Kang, Bin Gao, Bing Chen, Pei-Yu Huang, FF Zhang, YX Deng, LF Liu, XY Liu, H-Y Chen, Z Jiang, and others. 2014. 3D RRAM: Design and optimization. In *Solid-State and Integrated Circuit Technology (ICSICT), 2014 12th IEEE International Conference on*. IEEE, 1–4.

Kuk-Hwan Kim, Siddharth Gaba, Dana Wheeler, Jose M Cruz-Albrecht, Tahir Hussain, Narayan Srinivasa, and Wei Lu. 2011. A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. *Nano letters* 12, 1 (2011), 389–395.

Yongtae Kim, Yong Zhang, and Peng Li. 2012. A digital neuromorphic VLSI architecture with memristor crossbar synaptic array for machine learning. In *SOC Conference (SOCC), 2012 IEEE International*. IEEE, 328–333.

Richard T Kouzes, Gordon A Anderson, Stephen T Elbert, Ian Gorton, and Deborah K Gracio. 2009. The changing paradigm of data-intensive computing. *Computer* 1 (2009), 26–34.

Vipin Kumar, Ritu Sharma, Erdal Uzunlar, Li Zheng, Rizwan Bashirullah, Paul Kohl, Muhannad S Bakir, and Azad Naeemi. 2014. Airgap interconnects: Modeling, optimization, and benchmarking for backplane, pcb, and interposer applications. *Components, Packaging and Manufacturing Technology, IEEE Transactions on* 4, 8 (2014), 1335–1346.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 9–48.

HY Lee, PS Che, TY Wu, YS Che, CC Wan, PJ Tzen, CH Lin, F Chen, CH Lien, and MJ Tsai. 2008. Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO2 based RRAM. In *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*. IEEE, 1–4.

Xiaoxiao Liu, Mengjie Mao, Beiye Liu, Hai Li, Yiran Chen, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, and others. 2015. RENO: a high-efficient reconfigurable neuromorphic computing accelerator design. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 1–6.

Wei Lu, Kuk-Hwan Kim, Ting Chang, and S. Gaba. 2011. Two-terminal resistive switches (memristors) for memory and logic applications. In *Design Automation Conference (ASP-DAC)*.

Shoun Matsunaga, Jun Hayakawa, Shoji Ikeda, Katsuya Miura, Tetsuo Endoh, Hideo Ohno, and Takahiro Hanyu. 2009. Mtj-based nonvolatile logic-in-memory circuit, future prospects and issues. In *Proceed-*

ings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, 433–435.

Klaus-Robert Müller, Michael Tangermann, Guido Dornhege, Matthias Krauledat, Gabriel Curio, and Benjamin Blankertz. 2008. Machine learning for real-time single-trial EEG-analysis: from brain–computer interfacing to mental state monitoring. *Journal of neuroscience methods* 167, 1 (2008), 82–90.

Leibin Ni, Yuhao Wang, Hao Yu, Wei Yang, Chuliang Weng, and Junfeng Zhao. 2016. An Energy-efficient Matrix Multiplication Accelerator by Distributed In-memory Computing on Binary RRAM Crossbar. In *Design Automation Conference, Asia and South Pacific (ASP-DAC)*.

Sunghyun Park, Masood Qazi, Li-Shiuan Peh, and Anantha P Chandrakasan. 2013. 40.4 fJ/bit/mm low-swing on-chip signaling with self-resetting logic repeaters embedded within a mesh NoC in 45nm SOI CMOS. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 1637–1642.

Yang Shang, Wei Fei, and Hao Yu. 2012. Analysis and modeling of internal state variables for dynamic effects of nonvolatile memory devices. *Circuits and Systems I: Regular Papers, IEEE Transactions on* 59, 9 (2012), 1906–1918.

Pratap Narayan Singh, Ashish Kumar, Chandrajit Debnath, and Rakesh Malik. 2007. 20mW, 125 Msps, 10 bit Pipelined ADC in 65nm Standard Digital CMOS Process. In *Custom Integrated Circuits Conference, 2007. CICC'07. IEEE*. IEEE, 189–192.

Tathagata Srimani, Bibhas Manna, Anand Kumar Mukhopadhyay, Kaushik Roy, and Mrigank Sharad. 2015. Energy Efficient and High Performance Current-Mode Neural Network Circuit using Memristors and Digitally Assisted Analog CMOS Neurons. *arXiv preprint arXiv:1511.09085* (2015).

Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. 2008. The missing memristor found. *nature* 453, 7191 (2008), 80–83.

Johan AK Suykens and Joos Vandewalle. 1999. Least squares support vector machine classifiers. *Neural processing letters* 9, 3 (1999), 293–300.

Tieniu Tan and Z Sun. 2010. CASIA-FingerprintV5. (2010). http://biometrics.idealtest.org/

Yuhao Wang, Hao Yu, Leibin Ni, Guang-Bin Huang, Mei Yan, Chuliang Weng, Wei Yang, and Junfeng Zhao. 2015. An Energy-efficient nonvolatile in-memory computing architecture for extreme learning machine by domain-wall nanowire devices. *Nanotechnology, IEEE Transactions on* 14, 6 (2015), 998–1012.

Yuhao Wang, Hao Yu, and Wei Zhang. 2014. Nonvolatile cbram-crossbar-based 3-d-integrated hybrid memory for data retention. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 22, 5 (2014), 957–970.

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.

Stanley R Williams. 2008. How we found the missing memristor. *Spectrum, IEEE* 45, 12 (2008), 28–35.

Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

David H Wolpert. 1996. The lack of a priori distinctions between learning algorithms. *Neural computation* 8, 7 (1996), 1341–1390.

H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. 2012. Metal–oxide RRAM. *Proc. IEEE* 100, 6 (2012), 1951–1970.

John Wright, Allen Y Yang, Arvind Ganesh, Shankar S Sastry, and Yi Ma. 2009. Robust face recognition via sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31, 2 (2009), 210–227.

Hao Yu and Yuhao Wang. 2014. *Design Exploration of Emerging Nano-scale Non-volatile Memory*. Springer.