Chapter 1
Distributed In-Memory Computing on Binary Memristor-Crossbar for Machine Learning

Hao Yu, Leibin Ni, and Hantao Huang

Abstract The recent emerging memristor 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 memristor-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 chapter, we explore the matrix-vector multiplication accelerator on a binary memristor-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 memristor-crossbar, where logic-memory pair can be distributed with protocol of control bus. Experiment results have shown that compared to the analog memristor-crossbar, the proposed binary memristor-crossbar can achieve significant area-saving with better calculation accuracy. Moreover, significant speedup can be achieved for matrix-vector multiplication in the neuron-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.

Hao Yu
School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore e-mail: haoyu@ntu.edu.sg

Leibin Ni
School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore e-mail: nile0001@e.ntu.edu.sg

Hantao Huang
School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore e-mail: hhuang013@e.ntu.edu.sg

1.1 Introduction

Future cyber-physical system requires efficient real-time data analytics [19, 46, 13, 30, 9] with applications in robotics, brain-computer interface as well as autonomous vehicles. The recent works in [14, 5] 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 [22, 31] 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 [42]. The cell-level in-memory computing is proposed in [29], 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 memristor[1, 17, 4, 44, 35, 32, 8] has shown great potential to be the solution for data-intensive applications. Besides the minimized leakage power due to non-volatility, memristor in crossbar structure has been exploited as computational elements [17, 27]. 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 memristor-crossbar based computation is mainly based on an analog fashion with multi-level values [18] or Spike Timing Dependent Plasticity (STDP) [28]. Though it improves computation capacity, the serious non-uniformity of memristor-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 [28].

In this chapter, we propose a distributed in-memory accelerator. Both computational energy efficiency and robustness are greatly improved by a binary memristor-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 memristor-crossbar, a three-step digitalized memristor-crossbar is proposed in this chapter to perform a digital matrix-vector multiplication. In addition, a 3D CMOS-memristor accelerator is also proposed for machine learning. The area overhead can be reduced due to the 3D architecture. CMOS based operations can be implemented after the memristor-crossbar process in such architecture.

1.2 Background of Machine Learning

The current data analytics is mainly based on machine learning algorithm and computational intelligence to build a model to correlate input data with targeted output [39, 40]. Features extraction are also perfromed to extract the key information for data analytics in Neural Network. Neural network is the common model to build [11], 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.

To speed-up the training process, we tackle this challenge from two perspectives. Firstly, we propose a general incremental machine learning architecture with minimal tuning of parameters as shown in Fig. 1.1, which is mainly based on incremental least-squares solution. Secondly, we analyze the key complexity of each learning step and propose a hardware friendly algorithm to explore the parallelism with minimized hardware operational complexity.

1.2.1 Feature Extraction

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

$$\mathbf{X}' = \mathbf{R} \cdot \mathbf{X} \tag{1.1}$$

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 [45]. 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

$$||\mathbf{X}' - \mathbf{R} \cdot \mathbf{X}||_2 \tag{1.2}$$

One can observe significant matrix-vector multiplications during the feature extraction as shown in (1.1).

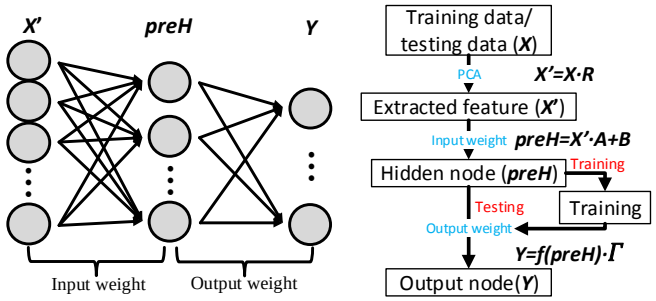

Fig. 1.1 (a) Single-layer neural network (b) General flow of neural network training and testing

1.2.2 Neural Network based Learning

After feature extraction, one can perform various machine learning algorithms [36, 23, 14] for data analytics. As shown in Fig.1.1 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 **preH**:

$$\mathbf{preH} = \mathbf{X}_f\mathbf{A} + \mathbf{B},\ \mathbf{H} = g(\mathbf{preH}) = \frac{1}{1+e^{-\mathbf{preH}}} \tag{1.3}$$

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

$$\min_{\boldsymbol{\Gamma}}\ ||\mathbf{H}\boldsymbol{\Gamma} - \mathbf{T}||_2^2 + \eta|||\boldsymbol{\Gamma}||_2^2 \tag{1.4}$$

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

One can solve (1.4) either by iterative backward propagation method [43] or direct L2-norm solver method for least-squares problem [14]. The output weight can be obtained as $||\tilde{\mathbf{H}}\boldsymbol{\Gamma} - \tilde{\mathbf{T}}||$, and can be solved as

$$\begin{aligned} \boldsymbol{\Gamma} &= (\tilde{\mathbf{H}}^T\tilde{\mathbf{H}})^{-1}\tilde{\mathbf{H}}^T\tilde{\mathbf{T}} = (\mathbf{H}^T\mathbf{H} + \eta\mathbf{I})^{-1}\mathbf{H}^T\mathbf{T} \\ &where\ \tilde{\mathbf{H}} = \begin{pmatrix}\mathbf{H}\\ \sqrt{\eta}\mathbf{I}\end{pmatrix},\quad \tilde{\mathbf{T}} = \begin{pmatrix}\mathbf{T}\\ \mathbf{0}\end{pmatrix} \end{aligned} \tag{1.5}$$

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 [12]. We have also analyzed the major computations of Cholesky decomposition for least-square problem, which will be discussed in Section 1.5.

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

$$\mathbf{Y} = \mathbf{H} \cdot \mathbf{\Gamma} \tag{1.6}$$

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 (1.1), (1.5) and (1.6). As such, a hardware accelerator to facilitate the matrix-vector multiplication is indeed the critical requirement for the efficient machine-learning based data analytics.

1.2.3 Incremental Least-square Solver based Learning

The objective function (1.4) is a least-squares problem and can be solved using backwards propagations (BP) or direct solution based on matrix operations. Since our target is to have incremental learning with latest training samples, iterative gradient based backwards propagation is slow comparing to pseudo-inverse solutions [14], therefore, BP will not be elaborated in details. In fact, as discussed in the next sections, our proposed 3D multi-layer CMOS-memristor architecture can accelerate the matrix-vector multiplications, which will also benefit BP based neural network training method.

Equation 1.5 shows how to obtain the output weight $\boldsymbol{\Gamma}$. The symmetric positive definite matrix $\tilde{\mathbf{H}}^T\tilde{\mathbf{H}}$ is decomposed into $\mathbf{QPQ^T}$. $\mathbf{Q}$ is a lower trangular matrix with diagonal elements $q_{ii} = 1$ and $\mathbf{P}$ is a positive diagonal matrix. Such method can maintain the same memory space as Cholesky factorization but need not perform square root extraction, as the square root of $\mathbf{Q}$ is resolved by diagonal matrix $\mathbf{P}$ [20]. Here, we use $\mathbf{H}_l$ to represent the matrix decomposition at l iteration where $l \leq L$ as below

$$\begin{aligned} \tilde{\mathbf{H}}_l^T\tilde{\mathbf{H}}_l &= \left[\tilde{\mathbf{H}}_{l-1}\ h_l\right]^T\left[\tilde{\mathbf{H}}_{l-1}\ h_l\right] \\ &= \begin{pmatrix} \tilde{\mathbf{H}}_{l-1}^T\tilde{\mathbf{H}}_{l-1} & \mathbf{v}_l \\ \mathbf{v}_l^T & g \end{pmatrix} \end{aligned} \tag{1.7}$$

where $(\mathbf{v}_l, g)$ is a new column generated from new hidden node output $h_l^T h_l$, compared to $\tilde{\mathbf{H}}_{l-1}^T\tilde{\mathbf{H}}_{l-1}$. Therefore, we can find

$$\begin{aligned}&\mathbf{Q}_l\mathbf{P}_l\mathbf{Q}_l^T\\&=\begin{pmatrix}\mathbf{Q}_{l-1} & 0\\ \mathbf{z}_l^T & 1\end{pmatrix}\begin{pmatrix}\mathbf{P}_{l-1} & 0\\ 0 & p\end{pmatrix}\begin{pmatrix}\mathbf{Q}_{l-1}^T & \mathbf{z}_l\\ 0 & 1\end{pmatrix}\end{aligned} \tag{1.8}$$

As a result, we can easily calculate the $\mathbf{z}_l$ and scalar p for Cholesky factorization as

$$\mathbf{Q}_{l-1}\mathbf{P}_{l-1}\mathbf{z}_l = \mathbf{v}_l,\ p = g - \mathbf{z}_l^T\mathbf{P}_{l-1}\mathbf{z}_l \tag{1.9}$$

where $\mathbf{Q}_l$ and $\mathbf{v}_l$ is known from (1.7), which means we can continue use previous factorization result and update only according part. Please note that Q_1 is 1 and P_1 is $\tilde{\mathbf{H}}_1^T\tilde{\mathbf{H}}_1$.

As a conclusion, we have elaborated the basic learning on neural network and optimize Cholesky decomposition to solve the incremental least-squares problem. We have found the major computations are matrix-vector multiplications such as layer output in Equation 1.3, 1.6 and also Cholesky decomposition (Equation 1.7, 1.8, 1.9). Therefore, the proposed 3D multi-layer CMOS-memristor architecture is designed to accelerate matrix-vector multiplication, which can be also extended to BP based training method, where matrix-vector operation is the major computation [6].

1.3 Memristor-crossbar based Accelerator

1.3.1 Distributed In-memory Computing Architecture

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 work, we propose a distributed memristor-crossbar in-memory architecture (XIMA). Because both data and logic units have uniform structure when implemented on memristor-crossbar, half of the leaves are exploited as logic elements and are paired with data arrays. The proposed architecture is illustrated in Fig. 1.2. 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 memristor-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

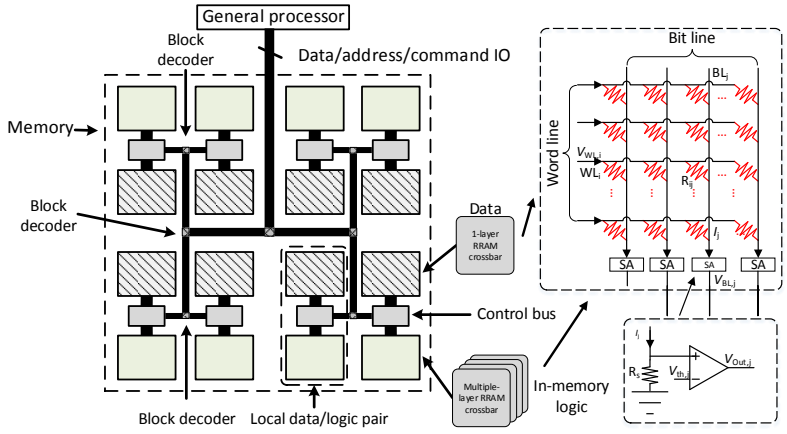

Fig. 1.2 Overview of distributed in-memory computing architecture on memristor-crossbar

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 memristor crossbars for data storage and logic unit have distinctive interfaces. The data memristor-crossbar will have only one row activated at one time during read and write operations, and logic memristor-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 memristor crossbar design, which uses three layers of memristor crossbars to decompose a complex function into several simple operations that digital crossbar can tackle.

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 on the current communication protocol. The new communication instructions are proposed in TABLE 1.1, which is called in-pair control.

In-pair control bus needs to execute instructions in TABLE 1.1. SW (store word) instruction is to write data into memristors 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.

Table 1.1 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

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 memristor-crossbar. An address example for data array and in-memory logic is shown in Fig. 1.3.

To perform logic operation, the following instructions are required to performed. Firstly, we store the required input data and memristor 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

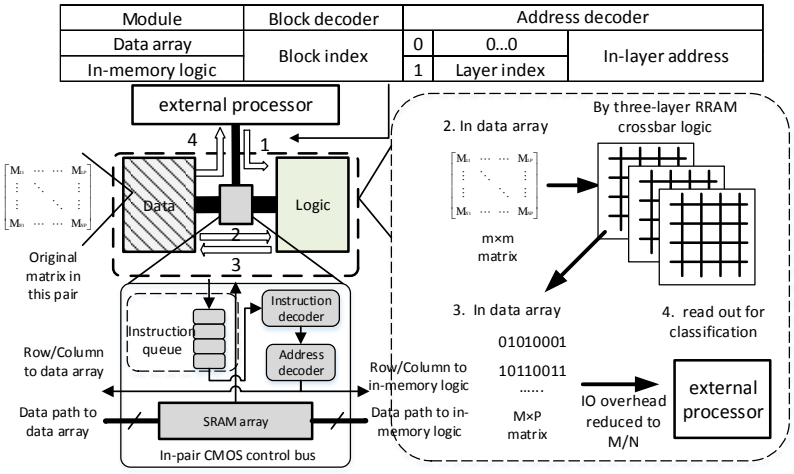

Fig. 1.3 Detailed structure of control bus and communication protocol

performed to wait for the completion of logic computing. At last, LW instruction is performed to load the data from the output of memristor-crossbar.

Given the new communication protocol between general processor and memory is introduced, one can design the according control bus as shown in Fig. 1.3. The control bus is composed of an instruction queue, an instruction decoder, an address decoder and a SRAM array. As the operation frequency of memristor-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.

1.3.2 3D CMOS-memristor Architecture

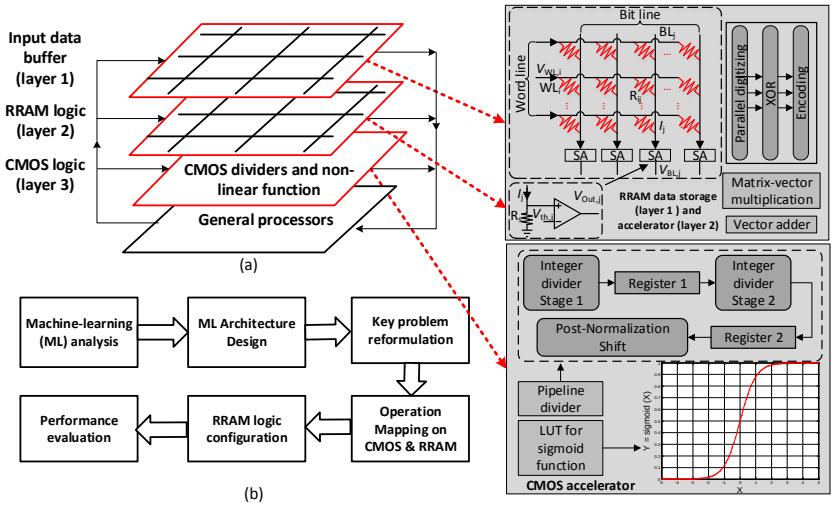

Fig. 1.4 (a) 3D multi-layer CMOS-memristor accelerator architecture; (b) Incremental machine learning algorithm mapping flow on proposed accelerator

Recent work [38] has shown that the 3D integration supports heterogeneous stacking because different types of components can be fabricated separately, and layers can be stacked and implemented with different technologies. Therefore, stacking non-volatile memories on top of microprocessors enables cost-effective heterogeneous integration. Furthermore, works in [25, 3] have also shown the feasibility to stack memristor on CMOS to achieve smaller area and lower energy consumption.

The proposed 3D multi-layer CMOS-memristor accelerator with three layers is shown in Fig. 1.4(a). This accelerator is composed of a two-layer memristor-crossbar and a one-layer CMOS circuit. As Fig. 1.4(a) shows, layer 1 of memristor-crossbar is implemented as a buffer to temporarily store input data to be processed. Layer 2 of memristor-crossbar performs logic operations such as matrix-vector multiplication and also vector addition. The details of implementation will be introduced in Section 4. Note that buffers are designed to separate resistive networks between layer 1 and layer 2. The last layer of CMOS contains read-out circuits for memristor-crossbar and performs as logic accelerators designed for other operations besides matrix-vector multiplication, including pipelined divider, look-up table (LUT) designed for division operation and activation function in machine learning.

Moreover, Fig. 1.4(b) shows the work flow for incremental machine learning based on the proposed architecture. Firstly, detailed architecture of machine learning (ML) (e.g. number of layers and activation function) is determined based on the accuracy requirements and data characteristics. Secondly, operations of this machine learning algorithm are analyzed and reformulated so that all the operations can be accelerated in 3D multi-layer CMOS-memristor architecture as illustrated in Fig. 1.4(a). Furthermore, the bit-width operating on memristor-crossbar is also determined by balancing the accuracy loss and energy saving. Finally, logic operations on memristor-crossbar and CMOS are configured based on the reformulated operations, energy saving and speed-up.

Such a 3D multi-layer CMOS-memristor architecture has advantages in three manifold. Firstly, by utilizing memristor-crossbar for input data storage, leakage power of memory is largely removed. In a 3D architecture with TSV interconnection, the bandwidth from this layer to next layer is sufficiently large to perform parallel computation. Secondly, memristor-crossbar can be configured as computational units for the matrix-vector multiplication with high parallelism and low power. Lastly, with an additional layer of CMOS-ASIC, more complicated tasks such as division and non-linear mapping can be performed. As a result, the whole training process of machine learning can be fully mapped to the proposed 3D multi-layer CMOS-memristor accelerator architecture towards real-time training and testing.

1.4 Binary memristor-crossbar for Matrix-vector Multiplication

In this work, we implement matrix-vector multiplication on binary memristor-crossbar. 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 [47].

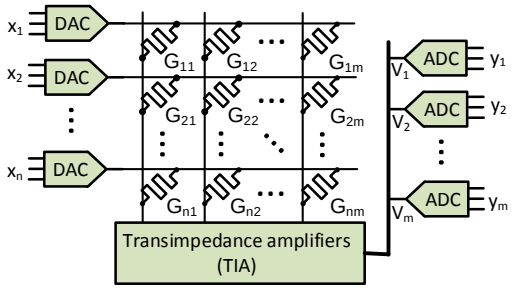

Fig. 1.5 Traditional analog-fashion memristor crossbar with ADC and DAC

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.

1.4.1 Memristor Device and Crossbar

Memristor 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 memristor cell, one can apply a read voltage V_r to the memristor. The V_r and V_w follow

$$V_w > V_{th} > V_w/2 > V_r, \tag{1.10}$$

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

Because of the high density of memristor device, one can build a crossbar structure as the array of memristor [17, 16, 7, 10, 34, 41]. 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 memristor cell on i^{th} row, j^{th} column.

1.4.2 Traditional Analog Memristor 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 memristor resistances. As shown in Fig. 1.5, by configuring Φ into the memristor crossbar, analog computation $y = \Phi x$ by memristor crossbar can be achieved.

However, such analog memristor-crossbar has two major drawbacks. Firstly, the programming of continuous-valued memristor resistance is practically challenging due to large memristor process variation. Specifically, the memristor resistance is determined by the integral of current flowing through, which leads to a switching curve as shown in Fig. 1.6 (a). With the process variation, the curve may shift and leave intermediate values very unreliable to program, as shown in Fig. 1.6 (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. 1.7.

1.4.3 Proposed Digitalized Memristor Crossbar

To overcome the aforementioned issues, we propose a full-digitalized memristor-crossbar for matrix-vector multiplication. Firstly, as ON-state and OFF-state are much more reliable than intermediate values shown in Fig. 1.6, only binary values of memristor are allowed to reduce the inaccuracy of memristor programming. Secondly, we deploy a pure digital interface without A/D conversion.

In memristor 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

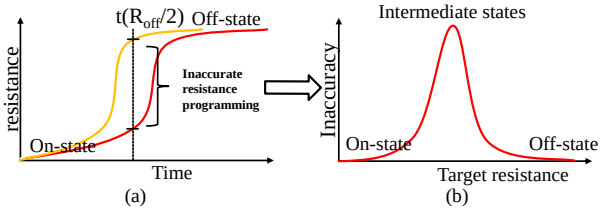

Fig. 1.6 (a) Switching curve of memristor under device variations (b) Programing inaccuracy for different memristor target resistances

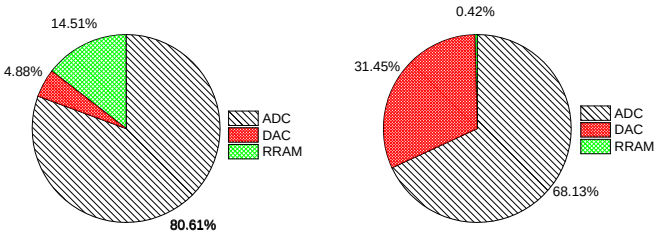

Fig. 1.7 (a) Power consumption of analog-fashion memristor crossbar (b) Area consumption of analog-fashion memristor crossbar

$$V_{bl}^{j} = \sum_{i=1}^{m} g_{ij} V_{wl}^{i} R_s \tag{1.11}$$

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. 1.2,

$$O_j = \begin{cases} 1, \ if \ V_{bl}^{j} \geq V_{th}^{j} \\ 0, \ if \ V_{bl}^{j} < V_{th}^{j} \end{cases} \tag{1.12}$$

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.

1.4.4 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 memristor-crossbar. Here, a three-step memristor-crossbar based binary matrix-vector multiplier is proposed, in which both the input and output of the memristor-crossbar are binary data without the need of ADC. The three memristor-crossbar step: parallel digitizing, XOR and encoding are presented in details as follows. As the output of a memristor-crossbar array can be connected to the input of another memristor-crossbar array, we can use multiple memristor 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

$$0 \leq s \leq N, \tag{1.13}$$

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:

$$[00101011] \times [10111110]^{\mathrm{T}} = 3 \tag{1.14}$$

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

1.4.4.1 Parallel Digitizing

The first step is called parallel digitizing, which requires $N \times N$ memristor 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 memristor crossbar. For each crossbar, as shown in Fig. 1.8, 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 memristors with high impedance are insignificant, so that the voltages on BLs approximately equal to $kV_r g_{on} R_s$ according to Eq. (1.11) where k is the number of memristor 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,

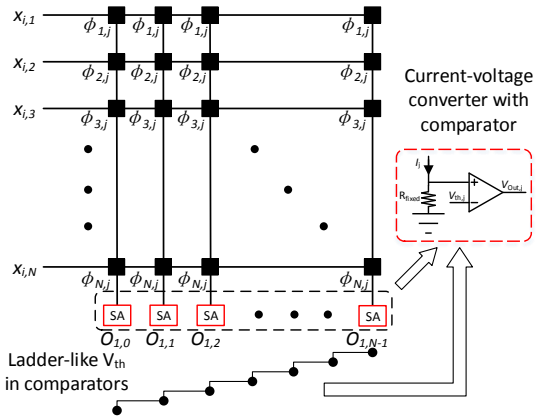

Fig. 1.8 Parallel digitizing step of memristor crossbar in matrix multiplication

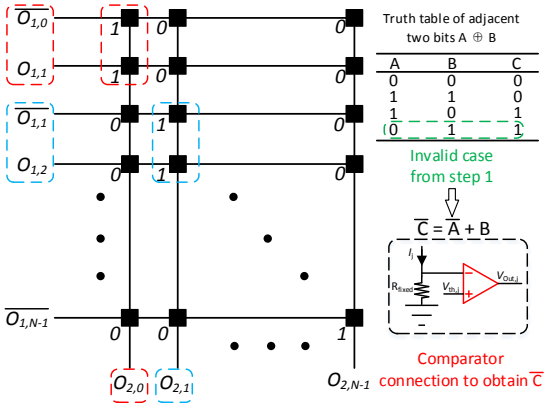

Fig. 1.9 XOR step of memristor crossbar in matrix multiplication

$$V_{th,j} = \frac{(2j+1)V_r g_{on} R_s}{2}, \tag{1.15}$$

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 memristor crossbar step i after sensing. Therefore, for the output we have

$$O_{1,j} = \begin{cases} 1, & j \leq s \\ 0, & j > s, \end{cases} \tag{1.16}$$

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 $O_1 = [11100000]$.

1.4.4.2 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 memristor crossbar is shown in Fig. 1.9. 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

$$\overline{O_{2,j}} = \begin{cases} \overline{O}_{1,j} + O_{1,j+1}, & j < N-1 \\ \overline{O}_{1,j}, & j = N-1. \end{cases} \tag{1.17}$$

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

$$V_{th,j} = \frac{V_r g_{on} R_s}{2} \tag{1.18}$$

Eqs. (1.17) and (1.18) 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 $N \times (2N-1)$.

1.4.4.3 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. 1.10. Encoding step needs $N \times n$ memristors, 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. 1.18 as well.

For activation function in (1.3), the exponentiation and division operations can be implemented by look-up table (LUT). The output of XOR layer is the index of **preH**. Therefore, the encoding layer performs as a LUT mapping process from **preH** to **H**. For example, if we want to map **preH** = 3 to **H** = 0.953, we can make the memristors 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.

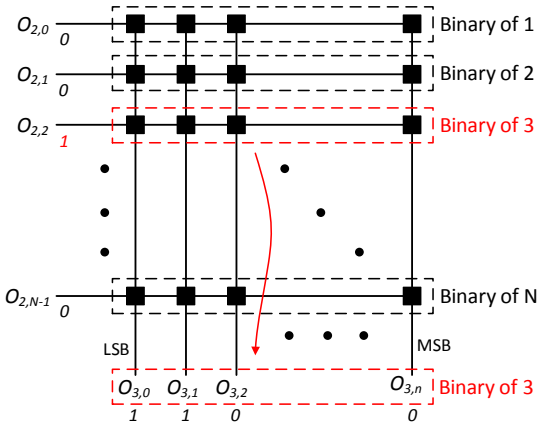

Fig. 1.10 Encoding step of memristor crossbar in matrix multiplication

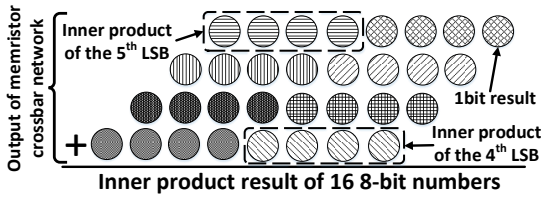

Fig. 1.11 Memristor-based inner-product operation

1.4.4.4 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. 1.11. 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 memristor-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 1.5.

1.5 Performance Evaluation

1.5.1 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, CMOS-based ASIC, non-distributed and distributed in-memory computing based on digitalized memristor crossbar respectively. For the memristor-crossbar design evaluation, the resistance of memristor is set as 1kΩ and 1MΩ as on-state and off-state resistance respectively according to [24].

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 memristor-crossbar based solution, we verify the function in circuit level with SPICE tool NVMSPICE [48]. By analyzing the machine learning algorithm, we obtain the basic operations and the number of memristor-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 memristor 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 memristor-crossbar is performed for comparison based on design in [33].

In the followings, we will show the performance of matrix-vector multiplication on memristor-crossbar first. A scalability study is introduced to show the area, energy and computation delay with different matrix sizes. Afterwards, the evaluation of face recognition on in-memory architecture is presented. Finally, we will illustrate the object classification on 3D CMOS-memristor architecture. Performance of different bid-width configurations will also be shown. In addition, the 3D CMOS-memristor solution will be compared with CMOS-ASIC as well as GPU implementation.

1.5.2 Performance Comparison of Multiplication

To evaluate the performance of binary memristor-crossbar for matrix-vector multiplication, we use the proposed architecture to accelerate dimension reduction of fingerprint images. 1,000 fingerprint images selected from [37] 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 1.2 with numerical results including energy consumption and delay obtained for one image on average of 1,000 images. For the digitied XIMA implementation, we need to compute the area of memristor 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 memristor 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

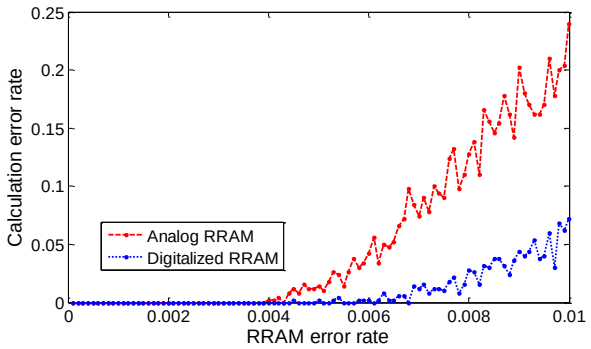

Fig. 1.12 Calculation error comparison between multi-leveled and binary memristor

Table 1.2 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 memristors) + $0.088mm^2$ + $128\mu m^2$	$0.05mm^2$ (12 MBit memristors) + $0.088mm^2$ + 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	$84W$	$34.938W$	Memristor: $4.71W$ Control-bus: $100\mu W$	Memristor: $4.71W$ Control-bus: $6.4mW$	Memristor: $1.28W$ Control-bus: $6.4mW$
Energy	$0.1424J$	$2.4457mJ$	Memristor: $23.17\mu J$ Control-bus: $0.131\mu J$	Memristor: $23.17\mu J$ Control-bus: $0.131\mu J$	Memristor: $2.1\mu J$ Memristor: $0.131\mu J$

fewer CMOS control bus but large data communication overhead on a single-layer crossbar compared to distributed memristor crossbar. Although distributed analog memristor crossbar can achieve the best in energy perspective but has larger area compared to the digitalized one. Shown in Table 1.2, memristor crossbar in analog fashion only consumes $2.1\mu J$ for one vector multiplication while the proposed architecture requires $23.17\mu J$ because most of power consumption comes from memristor in computing instead of ADCs. However, ADCs need more area so that memristor crossbar with ana-

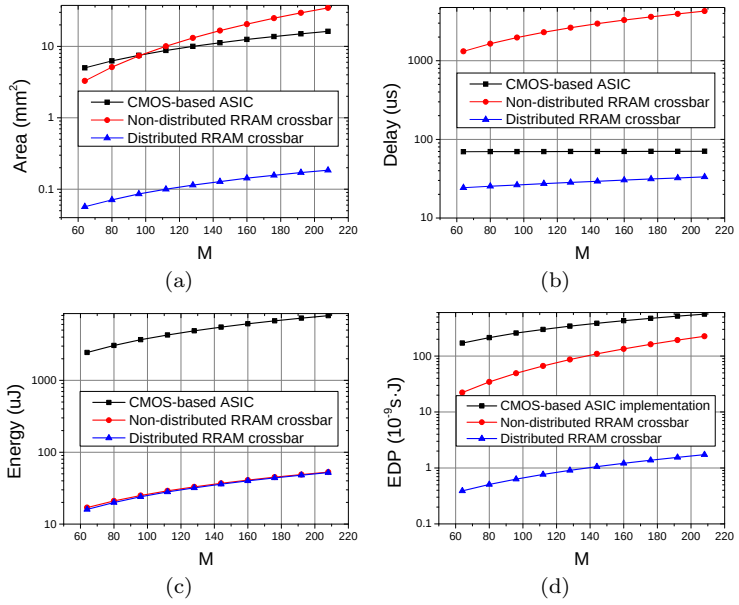

Fig. 1.13 Hardware Performance Scalability under Different Reduced Dimension for (a) area; (b) delay; (c) energy (d) EDP

Table 1.3 Face recognition performance comparison under among software and hardware implementation

Implementation	General purpose processor (MatLab)				CMOS-ASIC				Distributed in-memory memristor-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	Input layer	L2-norm	Output layer	PCA	Input layer	L2-norm	Output layer	PCA	Input layer	L2-norm	Output layer
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	-				$16.11\times$ ($7991.4mJ$: $496.17mJ$)				$210.69\times$ ($7991.4mJ$: $37.93mJ$)			

log fashion is $8.32mm^2$ while the proposed one is only $0.15mm^2$ because of the high density of memristor crossbar.

Calculation error of analog and digitalized memristor crossbar are compared in Fig. 1.12, where M and N are both set as 256. Calculation error is very low when memristor error rate is smaller than 0.004 for both analog and digitalized fashion memristor. However, when memristor error rate reaches 0.01, calculation error rate of analog memristor 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 memristor crossbar.

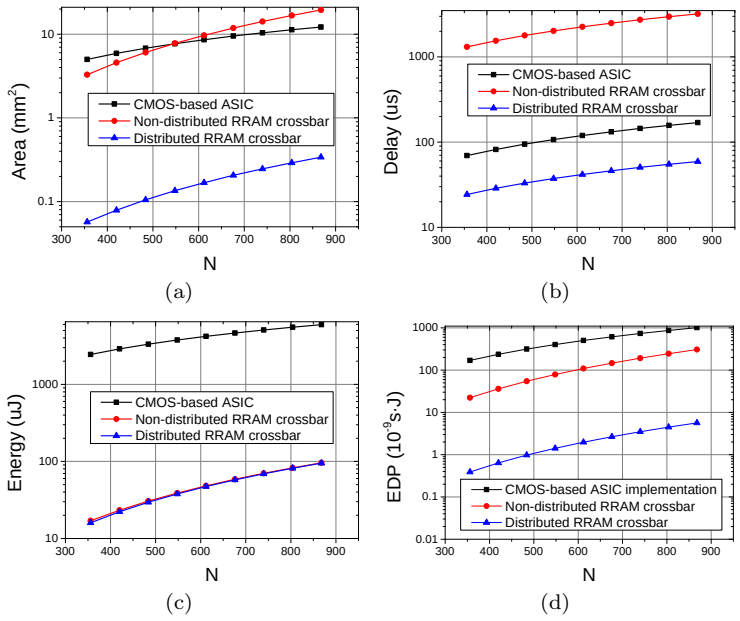

Fig. 1.14 Hardware Performance Scalability under Different Original Dimension for (a) area; (b) delay; (c) energy (d) EDP

1.5.3 Scalability Study

Hardware performance comparison among CMOS-based ASIC, non-distributed and distributed XIMA with varying M is shown in Fig. 1.13. From area consumption perspective shown in Fig. 13(a), distributed memristor-crossbar is much better than the 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 memristor crossbar becomes the worst one when $M > 96$. From delay perspective shown in Fig. 13(b), non-distributed memristor crossbar is the worst because it has only one control bus and takes too much time on preparing of computing. Delay of non-distributed memristor crossbar grows rapidly while distributed memristor 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 memristor 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 $0.3\times10^{-9}s\cdot J$ to $2\times10^{-9}s\cdot J$, which is 60x better than non-distributed memristor crossbar and 100x better than CMOS-based ASIC.

What is more, hardware performance comparison with varying N is shown in Fig. 1.14. Area and energy consumption trend is similar to Fig. 1.13. 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.

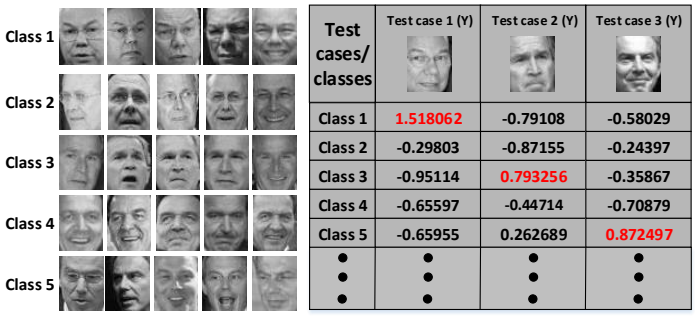

Fig. 1.15 Training samples and prediction value $\mathbf{Y}$ (1.6) for face recognitions

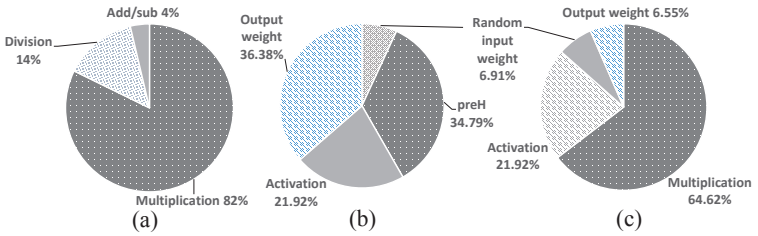

Fig. 1.16 (a) Time consumption breakdown for output weight calculation (b) neural network training computation effort analysis (c) Multiplication analysis for neural network training ($N = 200$, $n = 128$, $L = 160$ and $m = 13$)

1.5.4 Performance of In-memory Architecture

In this work, we implement the face recognition application on the in-memory architecture. We will analyze the computation complexity of face recognition first, and then evaluate the performance.

In the experiment, 200 face images of 13 people are selected from [15], 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. 1.16. 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. 1.16(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. 1.16(c).

We implement the face recognition in the distributed in-memory architecture. In Table 1.3, general performance comparisons among MatLab, CMOS-ASIC and memristor-crossbar accelerator are introduced, and the acceleration of each procedure as the formula described in Section 1.2 is also addressed. Among three implementations, memristor-crossbar architecture performs the best in area, energy and speed. Compared to MatLab implementation, it achieves 32.84× speed-up, 210.69× energy-saving and almost four-magnitude area-saving. We also design a CMOS-ASIC implementation with similar structure as memristor-crossbar with better performance compared to MatLab. memristor-crossbar architecture is 4.34× speed-up, 13.08× energy-saving and 51.3× area-saving compared to CMOS-ASIC. The performance comparison is quite different from Table 1.2, because we applied different designs (memristor-crossbar size) is this two experiments according to the dimension of matrices.

The result of face recognition is shown in Fig. 1.15. 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.

1.5.5 Performance of 3D CMOS-memristor Architecture

We implement the object classification in the 3D CMOS-memristor Architecture. Table 1.4 shows the testing accuracy under different datasets [26, 21] and configurations for machine learning of support vector machine (SVM) and single layer feed-forward neuron network (SLFN). It shows that accuracy of classification is not very sensitive to the memristor configuration bits. For example, the accuracy of Iris dataset is working with negligible accuracy at 5 memristor bit-width. When the memristor bit-width increased to 6, it performs the same as 32 bit-width configurations. Similar observation is found in [2] by truncating algorithms with limited precision for better energy efficiency. Please note that training data and weight related parameters are quantized to perform matrix-vector multiplication on memristor crossbar accelerator.

Fig. 1.17 shows the energy comparisons under different bit-width configurations for CMOS and memristor under the same accuracy requirements. An average of 4.5× energy saving can be achieved for the same number of bit-

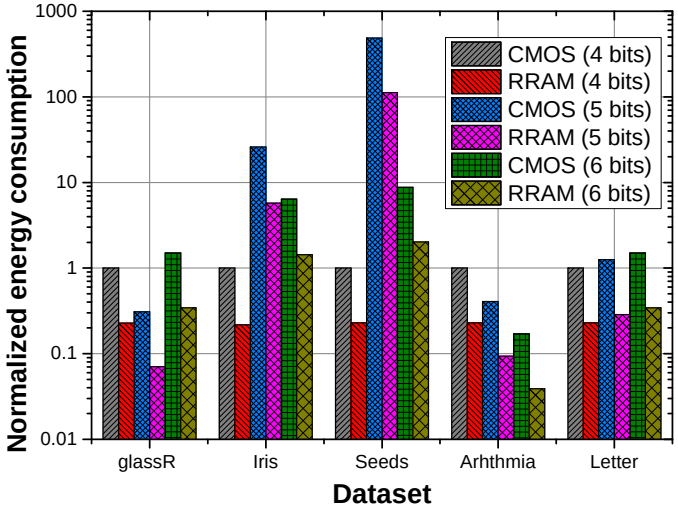

Fig. 1.17 Energy-saving comparison under different bit-width of CMOS and memristor with the same accuracy requirement

width configurations. The energy consumption is normalized by the CMOS 4 bit-width configuration. Furthermore, we can observe that not always smaller number of bits achieves better energy saving. Fewer number of bit-width may require much larger neuron network to perform required classification accuracy. As a result, its energy consumption increases.

Table 1.4 Testing accuracy of ML techniques under different dataset and configurations (normalized to all 32 bits)

Datasets	Size	Feat.	Cl.	4 bit Acc.(%)		5 bit Acc.(%)		6 bit Acc.(%)	
				SVM‡	&SLFN	SVM‡	&SLFN	SVM‡	&SLFN
Glass	214	9	6	100.07	100.00	93.88	99.30	99.82	99.30
Iris	150	4	3	98.44	94.12	100.00	94.18	100.00	100.00
Seeds	210	7	3	97.98	82.59	99.00	91.05	99.00	98.51
Arrhythmia	179	13	3	96.77	97.67	99.18	98.83	99.24	100.00
Letter	20,000	16	7	97.26	53.28	98.29	89.73	99.55	96.13
CIFAR-10	60,000	1600†	10	98.75	95.71	99.31	97.06	99.31	99.33

† 1600 features extracted from $60,000$ 32×32 color images with 10 classes. ‡ Least-square SVM is used for comparison.

1.5.6 Performance of Machine Learning based Face Recognition

Figure 1.18 shows the classification values in (1.6) on image data [21] with an example of 5 classes. As mentioned in the Section III, the index with maximum values (highlighted in red) is selected to indicate the class of test case. A few sample images are selected. Please note that $50,000$ and $10,000$ images are used for training and testing with 10 classes.

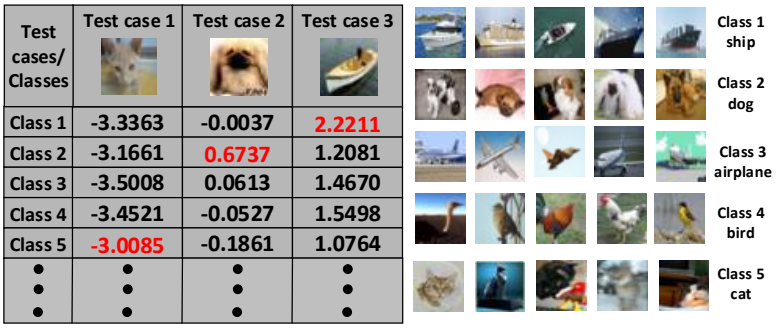

Fig. 1.18 On-line machine-learning for image recognition on the proposed 3D multi-layer CMOS-memristor accelerator using benchmark CIFAR-10

Table 1.5 Performance comparison under different software and hardware implementations

Implementation	**General purpose processor (Matlab)**						**3D-CMOS-ASIC †**						**3D CMOS-memristor Architecture †**					
Frequency	3.46GHz						1.0GHz						200MHz					
Area	$240mm^2$ (Intel Xeon X5690)						$1.86mm^2$ (65nm Global Foundry)						$1.46mm^2$ (65nm memristor and CMOS)					
Power (W)	130						39.41						13.1					
Computation	Feature Extraction		$IL^{\ddagger}$	ℓ_2 norm		OL^*	Feature Extraction		$IL^{\ddagger}$	ℓ_2 norm		OL^*	Feature Extraction		$IL^{\ddagger}$	ℓ_2 norm		OL^*
Operation	Sort	Mul.	-	Div.	Mul.	-	Sort	Mul.	-	Div.	Mul.	-	Sort	Mul.	-	Div.	Mul.	-
Time (s)	1473.2	736.6	729.79	294.34	1667.95	750.3	216.65	97.43	96.53	43.29	220.62	99.25	216.65	22.43	22.22	43.29	50.79	22.85
Energy (KJ)	191.52	95.76	94.87	38.26	216.83	97.54	0.078	3.84	3.80	0.015	8.69	3.91	0.078	0.293	0.291	0.015	0.67	0.30
Speed-up	-						7.30× (5651.88s : 773.77s)						14.94× (5651.88s : 378.22s)					
Energy-saving	-						36.12× (734.78KJ : 20.34KJ)						447.17× (734.78KJ : 1.643KJ)					

†6 bit-width configuration is implemented for both CMOS and memristor. $IL^{\ddagger}$ is for input layer and OL^* is for output layer.

In Table 1.5, performance comparisons among Matlab, 3D-CMOS-ASIC and 3D multi-layer CMOS-memristor accelerator are presented, and the acceleration of each procedure based on the formula described in Section 1.2 is also addressed. Among the three implementations, 3D multi-layer CMOS-memristor accelerator performs the best in area, energy and speed. Compared to Matlab implementation, it achieves 14.94× speed-up, 447.17× energy-saving and 164.38× area-saving. We also design a 3D-CMOS-ASIC implementation with similar structure as 3D multi-layer CMOS-memristor accelerator with better performance compared to Matlab. The proposed 3D multi-layer CMOS-memristor 3D accelerator is 2.05× speed-up, 12.38× energy-saving and 1.28× area-saving compared to 3D-CMOS-ASIC. To compare the performance with GPU, we also implemented the same code using Matlab GPU parallel toolbox. It takes 1163.42s for training benchmark CIFAR-10, which is 4.858× faster than CPU. Comparing to our proposed 3D multi-layer CMOS-memristor architecture, our work is 3.07× speed-up and 162.86× energy saving (267.59KJ :1.643KJ). Detailed comparisons of each step is not shown due to the limited space of table.

1.6 Conclusion

In this chapter, we have 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. In addition, 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× speedup and 105.6× power saving can be achieved when compared to the CMOS-ASIC. What is more, as for the machine learn-

ing based face recognition, 4.34× speedup and 13.08× power saving can be also achieved when compared to the CMOS-ASIC.

Acknowledgment

The work of H. Yu was supported in part by Singapore NRF-CRP (NRF2011NRF-CRP002-014) and MOE Tier-2 (MOE2015-T2-2-013).

References

[1] AkinagaH., ShimaH. (2010) Resistive random access memory (reram) based on metal oxides. Proceedings of the IEEE 98(12):2237–2251
[2] ChenP. Y., et al. (2015) Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip. In: IEEE DATE
[3] ChenY.-C., WangW., LiH., ZhangW. (2012) Non-volatile 3d stacking rram-based fpga. In: 22nd International Conference on Field Programmable Logic and Applications (FPL), IEEE, pp. 367–372
[4] ChuaL. O. (1971) Memristor-the missing circuit element. Circuit Theory, IEEE Transactions on 18(5):507–519
[5] CoatesA., NgA. Y., LeeH. (2011) An analysis of single-layer networks in unsupervised feature learning. In: International conference on artificial intelligence and statistics, pp. 215–223
[6] CongJ., XiaoB. (2014) Minimizing computation in convolutional neural networks. In: International Conference on Artificial Neural Networks, Springer, pp. 281–290
[7] FanD., SharadM., RoyK. (2014) Design and synthesis of ultralow energy spin-memristor threshold logic. Nanotechnology, IEEE Transactions on 13(3):574–583
[8] FeiW., YuH., ZhangW., YeoK. S. (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):1012–1025
[9] GlorotX., BengioY. (2010) Understanding the difficulty of training deep feedforward neural networks. In: International conference on artificial intelligence and statistics, pp. 249–256
[10] GuP., LiB., TangT., YuS., CaoY., WangY., YangH. (2015) Technological exploration of rram crossbar array for matrix-vector multiplication. In: Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific, IEEE, pp. 106–111

[11] HaykinS. S., HaykinS. S., HaykinS. S., HaykinS. S. (2009) Neural networks and learning machines, vol 3. Pearson Education Upper Saddle River

[12] HighamN. J. (2009) Cholesky factorization. Wiley Interdisciplinary Reviews: Computational Statistics 1(2):251–254, DOI 10.1002/wics.18, URL http://dx.doi.org/10.1002/wics.18

[13] HintonG. E., OsinderoS., TehY.-W. (2006) A fast learning algorithm for deep belief nets. Neural computation 18(7):1527–1554

[14] HuangG.-B., ZhuQ.-Y., SiewC.-K. (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1):489–501

[15] HuangG. B., RameshM., BergT., Learned-MillerE. (2007) Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Tech. rep., Technical Report 07-49, University of Massachusetts, Amherst

[16] KangJ., GaoB., ChenB., HuangP.-Y., ZhangF., DengY., LiuL., LiuX., ChenH.-Y., JiangZ., et al (2014) 3d rram: Design and optimization. In: Solid-State and Integrated Circuit Technology (ICSICT), 2014 12th IEEE International Conference on, IEEE, pp. 1–4

[17] KimK.-H., GabaS., WheelerD., Cruz-AlbrechtJ. M., HussainT., SrinivasaN., LuW. (2011) A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. Nano letters 12(1):389–395

[18] KimY., ZhangY., LiP. (2012) A digital neuromorphic vlsi architecture with memristor crossbar synaptic array for machine learning. In: SOC Conference (SOCC), 2012 IEEE International, IEEE, pp. 328–333

[19] KouzesR. T., AndersonG. A., ElbertS. T., GortonI., GracioD. K. (2009) The changing paradigm of data-intensive computing. Computer (1):26–34

[20] KrishnamoorthyA., MenonD. (2011) Matrix inversion using cholesky decomposition. arXiv preprint arXiv:11114144

[21] KrizhevskyA., HintonG. (2009) Learning multiple layers of features from tiny images

[22] KumarV., SharmaR., UzunlarE., ZhengL., BashirullahR., KohlP., BakirM. S., NaeemiA. (2014) Airgap interconnects: Modeling, optimization, and benchmarking for backplane, pcb, and interposer applications. Components, Packaging and Manufacturing Technology, IEEE Transactions on 4(8):1335–1346

[23] LeCunY. A., BottouL., OrrG. B., MüllerK.-R. (2012) Efficient backprop. In: Neural networks: Tricks of the trade, Springer, pp. 9–48

[24] LeeH., CheP., WuT., CheY., WanC., TzenP., LinC., ChenF., LienC., TsaiM. (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, pp. 1–4

[25] LiauwY. Y., ZhangZ., KimW., El GamalA., WongS. S. (2012) Nonvolatile 3d-fpga with monolithically stacked rram-based configuration

memory. In: 2012 IEEE International Solid-State Circuits Conference, IEEE, pp. 406–408

[26] LichmanM. (2013) UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`

[27] LiuX., MaoM., LiuB., LiH., ChenY., LiB., WangY., JiangH., BarnellM., WuQ., et al (2015) Reno: a high-efficient reconfigurable neuromorphic computing accelerator design. In: Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE, IEEE, pp. 1–6

[28] LuW., KimK.-H., ChangT., GabaS. (2011) Two-terminal resistive switches (memristors) for memory and logic applications. In: Design Automation Conference (ASP-DAC)

[29] MatsunagaS., HayakawaJ., IkedaS., MiuraK., EndohT., OhnoH., HanyuT. (2009) Mtj-based nonvolatile logic-in-memory circuit, future prospects and issues. In: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, pp. 433–435

[30] MüllerK.-R., TangermannM., DornhegeG., KrauledatM., CurioG., BlankertzB. (2008) Machine learning for real-time single-trial eeg-analysis: from brain–computer interfacing to mental state monitoring. Journal of neuroscience methods 167(1):82–90

[31] ParkS., QaziM., PehL.-S., ChandrakasanA. P. (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, pp. 1637–1642

[32] ShangY., FeiW., YuH. (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):1906–1918

[33] SinghP. N., KumarA., DebnathC., MalikR. (2007) 20mw, 125 msps, 10 bit pipelined adc in 65nm standard digital cmos process. In: Custom Integrated Circuits Conference, 2007. CICC'07. IEEE, IEEE, pp. 189–192

[34] SrimaniT., MannaB., MukhopadhyayA. K., RoyK., SharadM. (2015) Energy efficient and high performance current-mode neural network circuit using memristors and digitally assisted analog cmos neurons. arXiv preprint arXiv:151109085

[35] StrukovD. B., SniderG. S., StewartD. R., WilliamsR. S. (2008) The missing memristor found. nature 453(7191):80–83

[36] SuykensJ. A., VandewalleJ. (1999) Least squares support vector machine classifiers. Neural processing letters 9(3):293–300

[37] TanT., SunZ. (2010) CASIA-FingerprintV5. URL `http://biometrics.idealtest.org/`

[38] TopalogluR. O. (2015) More than moore technologies for next generation computer design. Springer

[39] VaidyanathanS., VolosC. (2016) Advances and Applications in Chaotic Systems, vol 636. Springer

[40] VaidyanathanS., VolosC. (2016) Advances and Applications in Nonlinear Control Systems, vol 635. Springer

[41] WangY., YuH., ZhangW. (2014) Nonvolatile cbram-crossbar-based 3-d-integrated hybrid memory for data retention. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 22(5):957–970

[42] WangY., YuH., NiL., HuangG.-B., YanM., WengC., YangW., ZhaoJ. (2015) An energy-efficient nonvolatile in-memory computing architecture for extreme learning machine by domain-wall nanowire devices. Nanotechnology, IEEE Transactions on 14(6):998–1012

[43] WerbosP. J. (1990) Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10):1550–1560

[44] WilliamsS. R. (2008) How we found the missing memristor. Spectrum, IEEE 45(12):28–35

[45] WoldS., EsbensenK., GeladiP. (1987) Principal component analysis. Chemometrics and intelligent laboratory systems 2(1-3):37–52

[46] WolpertD. H. (1996) The lack of a priori distinctions between learning algorithms. Neural computation 8(7):1341–1390

[47] WrightJ., YangA. Y., GaneshA., SastryS. S., MaY. (2009) Robust face recognition via sparse representation. Pattern Analysis and Machine Intelligence, IEEE Transactions on 31(2):210–227

[48] YuH., WangY. (2014) Design Exploration of Emerging Nano-scale Non-volatile Memory. Springer