XMAP: Programming Memristor Crossbars for Analog Matrix-Vector Multiplication: Towards High Precision using Representable Matrices

Necati Uysal*, Baogang Zhang*, Sumit Kumar Jha†, and Rickard Ewetz*
*Department of Electrical and Computer Engineering University of Central Florida, Orlando FL, USA
†Department of Computer Science, University of Texas at San Antonio, San Antonio TX, USA
{baogang.zhang, necati}@knights.ucf.edu, rickard.ewetz@ucf.edu

***Abstract*—Linear transformations are the dominating computation within many important applications. The natural multiply and accumulate feature of memristor crossbar arrays promise unprecedented processing capabilities to resistive dot-product engines (DPEs), which can accelerate approximate matrix-vector multiplication (MVM). Unfortunately, the precision of the analog computation may be degraded by parasitics, non-linear device characteristics, and variations. In this paper, we propose a framework called XMAP for mapping an arbitrary matrix into appropriate memristor conductance values (or state variables for non-linear devices). The specified conductance values are next programmed to the memristor hardware using accurate closed-loop tuning. XMAP is based on formulating the mapping problem as a mathematical optimization problem, which can be elegantly minimized using the concept of representable matrices, i.e., the matrices that can be represented on a crossbar. Compared with the state-of-the-art conversion algorithm, the computational accuracy is improved with up to 3.29X at the expense of overhead in run-time. The precision improvements translate into noteworthy application level benefits within signal compression and neural network inference.**

***Index Terms*—In-memory computing, analog matrix-vector multiplication, non-volatile resistive technology, memristor.**

I. Introduction

Memristor crossbar arrays have attracted significant interest due to their natural ability of carrying out matrix-vector multiplication (MVM) in a single time-step, which is the dominating computational operation for many important applications [1]–[3]. By applying a vector of voltages to the rows of a crossbar array, multiplication with the memristors conductance values is performed using Ohm's law and summation of currents along the columns is performed using Kirchhoff's current law, i.e., MVM is performed in the analog domain. Recent hardware prototypes have shown that the analog computation is orders of magnitude more efficient than a highly optimized digital ASIC [1]. Moreover, the data movement on the system bus is reduced as the computation is performed in-memory.

The main challenge of utilizing memristor crossbars as resistive dot-product engines (DPEs) is that the computational accuracy may be degraded by parasitics, non-linear device characteristics, limited write accuracy, and environmental variations. In particular, the precision may be degraded by voltage IR-drop over non-zero input/output/array parasitics, which allows currents to flow from an input to an output using multiple alternative paths within a crossbar. This is related to but not equivalent to the *sneak path problem* for memory applications [4]. Moreover, the issue cannot be easily solved using selector devices or access transistors. Selector devices introduce unacceptable non-linearity and the access transistors can only be used to improve accuracy when programming the crossbar. Errors within analog computing paradigms are challenging because every error directly impacts the application-level functional correctness. In contrast, variations within digital computing systems mainly introduce timing violations, which can be alleviated by scaling down the clock frequency.

The handling of non-ideal effects within memristor crossbars has been the focus of many recent studies [5]–[17]. Hardware and software oriented training schemes for deep neural networks have been studied in [5]–[8]. The decomposition of large crossbars into smaller crossbars was explored in [9], [10]. The main limitation of these techniques is the undesirable software/hardware co-design. Architectural level studies have explored decomposing MVM operations though bit-slicing and reduction networks [11]–[14]. Nevertheless, such architectural level schemes are less energy-efficient because every matrix-vector operation is decomposed across multiple crossbars and time steps. Circuit level solutions mainly rely on representing each matrix element using a two (or multiple) memristors arranged in a differential pair configuration [15]. The use of multiple devices can in many instances reduce voltage IR-drop and variations at the expense of hardware overhead. The main body of work that we follow in this paper is focused on developing a software algorithm to convert an arbitrary matrix into appropriate memristor conductance values [9], [18]–[20]. This involves compensating for the IR-drop over the array parasitics when specifying the memristor's conductance values. The specified conductance values are next programmed to hardware using accurate closed-loop tuning. Many initial studies on matrix to conductance conversion only captured the output sensing resistance and omitted the array parasitics [18], [19]. More recently, mapping algorithms that account for all non-zero parasitics have been proposed [9], [18]. The difference between a target matrix and the conductance matrix

Research was supported by NSF awards CCF-1755825, CNS-1908471, and CNS-2008339.

realized by the crossbar was minimized using steepest gradient descent in [9]. Unfortunately, the computational accuracy of the resulting MVM operations is highly unpredictable due to that the bit-accuracy of the memristors was not considered. A mapping algorithm that can handle crossbars with large dimensions was proposed in [20]. This is the first algorithm capable of converging to mapping solutions of reasonable quality while accounting for non-zero parasitics, non-linear device characteristics, and variations. However, the method relies on assumptions that limit the attainable precision.

In this paper, we propose a framework called XMAP for converting an arbitrary target matrix A into a scaling factor and appropriate conductance values g. The scaling factor is used to decode the analog outputs into digital outputs. State variables s are specified when non-linear devices characteristics are considered. The main innovations within the XMAP framework are, as follows:

- A new method to unify the handling of linear and non-linear devices is proposed. The method is based on first converting all non-linear devices into linear devices. Later, the solution obtained with respect to the linear devices is converted into a solution for non-linear devices. The use of linear devices allows the matrix realized by a crossbar (A^r) to be computed analytically. Based on the matrix A^r, we define the notion of *value range errors*, *precision errors*, and *total errors*. These definitions allow the mapping problem to be formulated as minimizing the difference between A and A^r.
- We introduce a concept of representable matrices, which provides an understanding of the matrices that can be represented on a crossbar and the interplay between the scaling factor and the memristor conductance values g. The understanding enables the optimization problem to be elegantly minimized using a binary search and iteratively solving linear equations.
- The framework seamlessly supports each matrix element to be represented using one memristor or two memristors per matrix element. The algorithm essentially treats all memristors used to represent a single matrix element as a single memristor device.
- The peripheral circuitry imposes constraints on the maximum output current that can be measured from a bitline. Such constraints are easily handled by imposing an upper bound on the scaling factor .
- Three optimization techniques are proposed to speed-up the run-time of the XMAP. The first is based on utilizing hierarchical optimization framework. The last two are based on early termination features and avoiding redundant computation.

The experimental results demonstrates that XMAP is capable of mapping any arbitrary matrix to a crossbar. The run-time is only a few minutes for matrices with dimensions up to 256x256. Compared with the state-of-the-art conversion algorithm, the computational accuracy is improved with up to 3.29X. The improvements in computational accuracy translate into application level benefits. Moreover, XMAP obtains smooth trade-offs across every evaluated technology parameter.

II. Background

A. DPE friendly Applications

Applications that are i) dominated by MVM operations and ii) where the matrices are relatively constant and the input vectors change frequently, are candidates for acceleration using resistive DPEs. These two conditions hold for the applications below and many other applications.

Signal compression: from the time domain into the frequency domain is performed using a MVM operation, $c = Dx$, where D is the DCT matrix. x and c are respectively the time and frequency representation of a signal in vector form.

Neural network inference: involves classifying images into one of multiple output categories. The classification is performed by passing an input image (in a vector representation) to the first layer of the neurons in a neural network and recording the outputs from the last layer. The evaluation of each layer involves performing an MVM operation.

B. Memristor Crossbar Arrays

Memristor crossbar arrays have demonstrated great potential for both memory and in-memory computing applications [1], [4]. The interest in computing stems from that crossbars can perform MVM operations ($Ax = y$) in a single time step.

Computation within the paradigm is performed using an initialization phase and an evaluation phase. In the initialization phase, the target matrix is first converted in conductance values g using a mapping algorithm. Next, the memristors are programmed to the specified conductance values g using accurate closed-loop tuning. By utilizing access transistors, the memristors can be tuned with an accuracy of 5-8 bits [1], [21], which is called write bit-accuracy. The access transistors ensure that almost all currents flow through the devices being programmed. In the evaluation phase, analog matrix vector multiplication is performed by applying a vector of input voltages (v_{in}) to the wordlines and measuring the vector of output currents i_{out} from the bitlines, where $i_{out}^T = v_{in}^T G$ and G is the conductance matrix of the crossbar. When the crossbar is ideal (zero parasitics and linear devices), each element G_{ij} in the conductance matrix is equal to the cross-point conductance in the array, which is shown in Figure 1(a). Hence, a target matrix A can be linearly mapped into conductance values g. The input voltages are provided to the crossbar using digital-to-analog converters (DACs). The output currents are converted into voltages using transimpedance amplifiers (TIAs) and measured using analog-to-digital converters (ADCs).

Memristor devices are desired to have high-linearity and continuous conductance states. The crossbars are desired to have small input, output, and array parasitics. Nevertheless, when non-ideal crossbar effects are considered, it has been observed that a linear mapping results in very poor computational accuracy, which can be observed in Figure 1(b). While the figure shows that the state-of-the-art mapping algorithm improves the computational accuracy, there still exists a significant gap between the obtained and ideal accuracy [20].

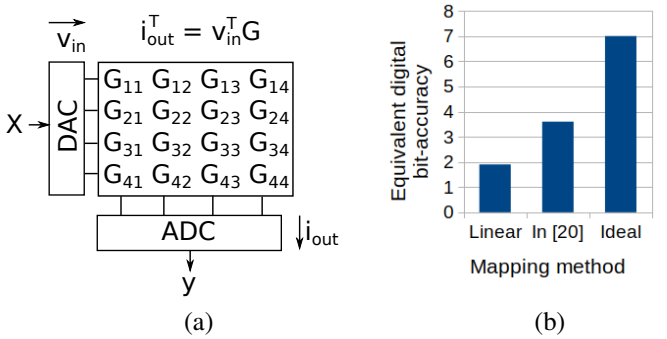

Fig. 1. (a) Ideal crossbar array. (b) Obtained equivalent bit-accuracy using linear mapping, mapping in [20], and ideal. The details of the experimental setup and equivalent bit-accuracy are provided in Section III-A and Section VIII

C. Previous work and its limitations

The first mapping algorithm capable of handling all non-ideal crossbar effects was proposed in [20]. The algorithm has been widely adopted and is considered the state-of-the-art.

The algorithm first determines a target current through each cross-point using a first order equation: $I_p = (aA + b)v_{cal}$, where a and b are technology dependent constants and v_{cal} is a calibration voltage. Next, the memristor conductance values g (or state variables s) are specified to deliver the target currents I_p through the cross-points while accounting for non-ideal crossbar effects using a built-in crossbar simulator. The built-in simulator accounts for that currents may flow from an input to an output using alternative paths in a crossbar.

The limitation of this approach is that the algorithm aims to deliver a current proportional to A_{ij} through each cross-point $(i; j)$, which is conceptually shown in Figure 2(a). In reality, the sum of the currents flowing on any path from row i to column j should be proportional to the matrix element A_{ij}, which is shown in Figure 2(b). The XMAP framework maps a matrix A into conductance values g with this objective, which explains why XMAP achieves superior computational accuracy compared with in [20]. Note that the built-in simulator in [20] accounts for that currents flow on alternative paths when specifying s (or g) with respect to I_p. However, the currents on the alternative paths are not considered when specifying I_p using the first order equation.

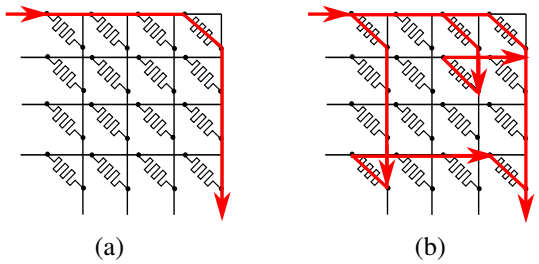

Fig. 2. Currents flowing from an input to an output (a) only using the memristor in the corresponding cross-point and (b) using multiple alternative paths in the crossbar. The access transistors in the crossbar are omitted in the figure because they are all on during each MVM operation.

III. Modeling and Simulation of Crossbars

We describe how crossbars with non-linear and linear devices are modeled and simulated in Section III-A and Section III-B, respectively. In Section III-C, we compute the matrix realized using one or two memristors per matrix. A non-ideal crossbar with non-linear and linear devices is shown in (a) and (b) of Figure 3, respectively.

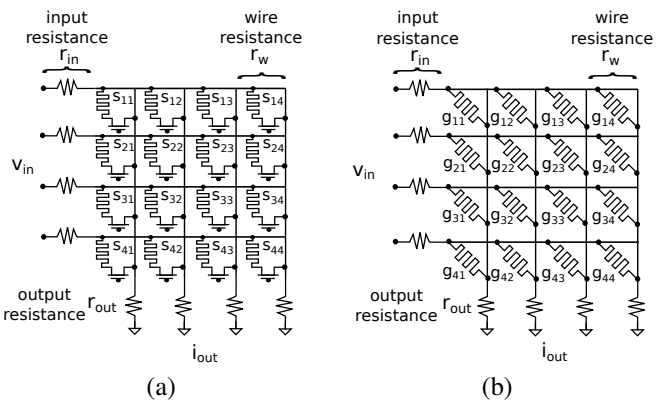

Fig. 3. Crossbar array with (a) non-linear devices and (b) linear devices.

A. Memristor crossbars with non-linear devices

A crossbar with non-zero parasitics and non-linear devices is shown in Figure 3(a). The non-zero parasitics are in the form of wire (r_w), input (r_{in}), and output (r_{out}) resistances. All the parasitics are modeled using standard resistances. The memristors and access transistors are modeled as non-linear devices. The current through the access transistors $i_t(v_d; v_s; v_g)$ is a non-linear function of the drain voltage (v_d), source voltage (v_s), and gate voltage (v_g). Similarly, the current through a memristor $i_m(v_m; s)$ is non-linear function of the voltage across the two terminals (v_m) and the state variable s. Two non-linear device models are evaluated in Section VIII. The write accuracy is modeled using 2^b linearly spaced states between s_{min} and s_{max}, where b is the memristor write bit-accuracy. s_{min} and s_{max} are lower and upper bounds on the state variable s, respectively.

The relation between a vector of input voltages (v_{in}) and the vector of output currents (i_{out}) for the circuit in Figure 3(a) can be captured using modified nodal analysis (MNA) [22]. This requires a system of $(3MN + M + N)$ non-linear equations to be formulated for a crossbar with M wordlines and N bitlines. The first $3MN$ are the KCL equations for the nodes in the crossbar. The next M equations are the input boundary conditions. The last N define the output currents. Given an input vector v_{in}, the output currents i_{out} are obtained by solving the system of equations using Newton's method. A detailed description of how to formulate and solve the non-linear model is provided in [23].

The outlined model captures the behaviour of the non-ideal crossbar with SPICE level accuracy. The main drawback of the accurate non-linear model is that it does not provide an intuitive understanding if the state variables s approximately realize the target matrix A. In contrast, if the memristors and access transistors are assumed to be linear, the matrix A^r realized by a crossbar can be computed analytically.

B. Crossbar with linear devices

In this section, we explain how the crossbar with non-zero parasitics and linear devices is modeled and simulated. When the memristors and access transistors in Figure 3(a) are assumed to be linear, each series connected memristor and access transistor can be replaced with an ideal memristor (or ideal conductor), which is shown in Figure 3(b). If the memristors have a programmable conductance range of $[g_{min}; g_{max}]$, the ideal memristors will have a slightly lower range of $[g_{lb}; g_{ub}]$ due to the access transistors. We let g and g_w respectively denote the conductance values of the ideal memristors without (with) the write-bit accuracy, i.e., g_w is obtained by quantizing g to 2^b linearly spaced states between g_{lb} and g_{ub}. As all the circuit elements (memristors, wires, input resistance, output resistance) are modeled as resistors, the crossbar can be modeled as a resistive network.

The relation between the input voltages (v_{in}) and the output currents (i_{out}) of the resistive network can be determined using MNA, as follows:

$$Y(g; r_w; r_{in}; r_{out}) \begin{bmatrix} v_{xbar} \\ v_{dac} \\ i_{out} \end{bmatrix} = \begin{bmatrix} 0 \\ v_{in} \\ 0 \end{bmatrix}; \quad (1)$$

where $Y(g; r_w; r_{in}; r_{out})$ is matrix with dimensions $(2NM + M + n)$x$(2NM + M + N)$. The detailed definition of $Y(g; r_w; r_{in}; r_{out})$ is provided in [9]. g are the ideal memristor conductance values; r_w, r_{in}, and r_{out} are respectively the wire, input, and output resistances; v_{xbar} and v_{dac} are the node voltages of the crossbar and the DACs, respectively. The linear system of equations is formulated in similar to in the previous section. However, the number of KCL equations are reduced from $3NM$ to $2NM$ because NM nodes are eliminated when the memristors and access transistors are combined into a single linear device.

As the system is linear, the output currents (i_{out}) can be expressed as a linear function of the input voltages (v_{in}) using a conductance matrix $G(g)$, i.e., $i_{out}^T = v_{in}^T G(g)$. The matrix $G(g)$ can be computed [24], as follows:

$$G(g) = SY^{-1}(g)B; \quad (2)$$

where $B = [0; I; 0]^T$ is a matrix with dimensions $(2NM + N + M)$x(M) and I is an MxM identity matrix. $S = [0; 0; I]$ is a selection matrix with dimensions (N)x$(2NM + N + M)$. The selection matrix is used to select the output currents from $Y(g; r_w; r_{in}; r_{out})^{-1}B$. In this paper, we refer to the conductance matrix as G or $G(g)$ based on if we want to emphasize that the matrix is a function of the memristor conductance values g. Note that $G(g)$ also is a function of the constant parameters r_w, r_{in}, and r_{out}.

C. Matrix realized by a crossbar with linear devices

Every crossbar with linear devices realizes a matrix A^r and accelerates the MVM operation $A^r x = y$, where x and y are the input and output vectors, respectively. The matrix A^r can be determined analytically based on the conductance matrix G and the arrangement of the peripheral circuitry. In this paper, we consider the case when each matrix element is realized by a single memristor or using two memristors arranged in a differential pair configuration, which is shown in (a) and (b) of Figure 4.

When each matrix element is represented using a single memristor, the target matrix is shifted to the positive domain before being mapped to a crossbar [25], i.e., $\widetilde{A} = (A - A_{shift})$ is the matrix mapped to the crossbar. The shift is required because memristors cannot be programmed to have negative conductance. Next, when MVM operations are performed, the outputs are post-processed to offset the matrix shift.

$$Ax = \underbrace{(A - A_{shift})}_{\text{Matrix } \widetilde{A} \text{ on crossbar}} x - \underbrace{sum(x) \cdot A_{shift}}_{\text{Post-processing}}; \quad (3)$$

where A_{shift} is equal to the smallest element in A. $sum(x)$ is the sum of the elements in the input vector x. For the remainder of the paper, we assume without loss of generality that each element in A (represented using a single memristor) is within $[0; 1]$. This allows the use of the $\widetilde{A}$ notation to be circumvented. Consequently, the matrix realized by the crossbar A^r can be computed, as follows:

$$A^r(\ ; g) = G(g)/\ ; \quad (4)$$

where $\ $ is a scaling factor used to decode the output currents from the crossbar into digital output vector.

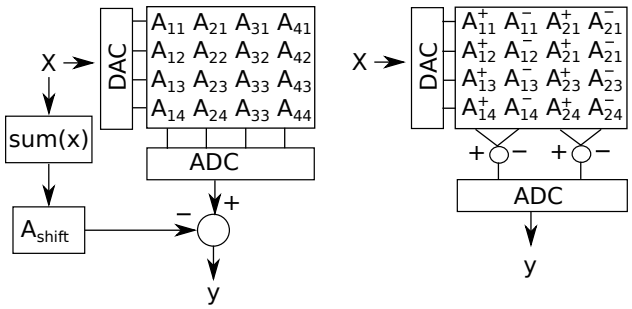

Fig. 4. (a) Traditional crossbar with each matrix element represented using one memristors. Note that a matrix is transposed when mapped to a crossbar. A_{ij} is the matrix element in row i and column j within A. (b) Crossbar with each matrix element represented using two memristors in a differential pair configuration. A_{ij}^+ and A_{ij}^- denote the positive and negative component of the matrix element A_{ij}.

When a crossbar with a differential pair configuration is used, ADCs are used to convert the difference in the output current between adjacent bitlines into digital values, which is illustrated in Figure 4(b). This implies that the positive and negative elements in a row of A is mapped to a pair of adjacent bitlines. Consequently, the effective matrix A^r realized by the crossbar is obtained, as follows:

$$A^r(\ ; g) = (G(g)_+ - G(g)^-)/\ ; \quad (5)$$

where $G(g)_+$ and $G(g)^-$ denotes the odd (positive) and even (negative) columns of $G(g)$. Again, $\ $ is a scaling factor used to decode the output currents into a digital output vector. For the remainder of the paper, when a matrix is represented using a differential pair configuration, we assume without loss of generality that each element in A is within $[-1; 1]$.

IV. Problem formulation

This paper addresses the problem of programming memristor crossbars to maximize the computational accuracy when accelerating MVM operations. This specifically involves mapping an arbitrary matrix A into a scaling factor and appropriate state variables s. The objective is to minimize difference between the ideal output $y = Ax$ and the non-ideal output from the crossbar $y(x; s;)_{xbar}$, as follows:

$$\begin{aligned} \min_{s;} \quad & E[||y - y(s; ; x)_{xbar}||_1]; \quad x \in X; \\ s.t \quad & ||i_{out}||_1 \leq i_{max} \end{aligned} \tag{6}$$

where $||:||_1$ is the L^1 norm. x is an input vector from the input vector space X. The constraint on the maximum output current is imposed by the peripheral circuitry [1]. $y(s; ; x)_{xbar}$ is obtained using the non-linear model described in Section III-A. The non-linear model captures the wire, input resistance, output resistance, the non-linear access transistors, and the non-linear memristors.

Minimizing the L^1 norm of a non-linear function over a vector space is difficult. Therefore, we define a related optimization problem that is expected to approximately minimize Eq (6). By assuming that the devices are linear, the matrix realized by a crossbar A^r can be computed, which allows the difference between A and A^r to be minimized. Next, we convert the linear solution with respect to the linear devices into an solution for non-linear devices.

We observe that there are two reasons for why each element in A^r is not exactly equal to the corresponding element in A after optimization: (i) the corresponding memristor cannot be tuned outside the lower and upper conductance bounds or (ii) the difference stems from the limited write bit-accuracy of the memristor devices. We denote these two types of errors as *value range errors* and *precision errors*, respectively. We also let the *total errors* be defined as the sum of the value range errors and precision errors. Next, we formally define the errors types and formulate and optimization problem that minimizes the total errors.

Definition 1 (Value range errors): Given a target matrix A, a scaling factor , and conductance values g, the value range errors ($_v$) are defined, as follows:

$$_v = ||A - A^r(; g)||^2; \tag{7}$$

where $A^r(; g)$ is the matrix realized by a crossbar, which is a function of g and . $||:||^2$ is the square of the L^2 norm. The errors are called value range errors because if g is specified optimally within $[g_{lb}; g_{ub}]$, the errors will stem from that g cannot be specified outside the value range $[g_{lb}; g_{ub}]$.

Definition 2 (Precision errors): Given a target matrix A, a scaling factor , and the conductance values with write errors (g_w), the precision errors ($_p$) are defined, as follows:

$$_p = ||A - A^r(; g_w)||^2 - \ _v; \tag{8}$$

where $A^r(; g_w)$ is matrix realized by the crossbar. Recall g_w is a discrete variable with 2^b states between $[g_{lb}; g_{ub}]$, where b is the write bit-accuracy of the memristors. As the value range errors are subtracted, the precision errors stem from the limited memristor bit-accuracy and the degree that the programmable conductance range is utilized.

Definition 3 (Total Errors): Given the value range errors ($_v$) and the precision errors ($_p$), the total errors ($_{tot}$) are defined, as follows:

$$_{tot} = \ _p + \ _v; \tag{9}$$

Using the aforementioned definitions, the problem of minimizing the total errors is formulated, as follows:

$$\begin{aligned} \min_{;g} \quad & _p + \ _v \\ & _v = ||(A - A^r(; g)||^2; \\ & _p = ||(A - A^r(; g_w)||^2 - \ _v; \\ & \leq \ _{max} \end{aligned} \tag{10}$$

where g are continuous memristor conductance values within $g_{lb} \leq g \leq g_{ub}$. g_w is the discrete version of the variable g. The constraint on the maximum output current is transformed into an upper bound on .

After the conductance values g that minimize Eq (10) have been specified, they are translated into state variables s to approximately minimize Eq (6) using optimization.

V. Insights and Motivations

In this section, we provide the high level insights and motivations behind the XMAP framework. We first introduce the concept of representable matrices in Section V-A. Optimization insights are provided in Section V-B.

A. Representable matrices

In this section, the concept of representable matrices $\mathcal{A}( )$ is introduced to define the matrices that can be represented on a crossbar, which is illustrated in Figure 5. More specifically, the concept provides an understanding of interplay between the parameters and g.

Definition 4 (Representable Matrices): Given the properties of a crossbar and a matrix A, the representable matrix $\mathcal{A}( )$ defines the *precision* and *value range* for each matrix element based on the *crossbar location* and the scaling factor . The value range for a matrix element is a lower and upper bound on the value that can be realized. The precision of a matrix element refers to the number of distinguishable states between the lower and upper bound. Qualitatively, matrix elements outside their respective value ranges introduce *value range errors* and matrix elements within the value range errors introduce *precision errors*.

The dependency of the *value ranges* and the *precision* on the *crossbar location* is shown in Figure 5(a). Intuitively, the value ranges are the worst (most restrictive) in the far-end of the crossbar because the array parasitics makes it impossible to realize both small and large values. On the other hand, the precision of each matrix element is higher because of the reduced voltage drop across each memristor devices, i.e., a large change of a memristor's conductance value is necessary to make a small change to the output current.

The dependency of the *value ranges* and the *precision* on is shown in (b) and (c) of Figure 5. Each matrix element

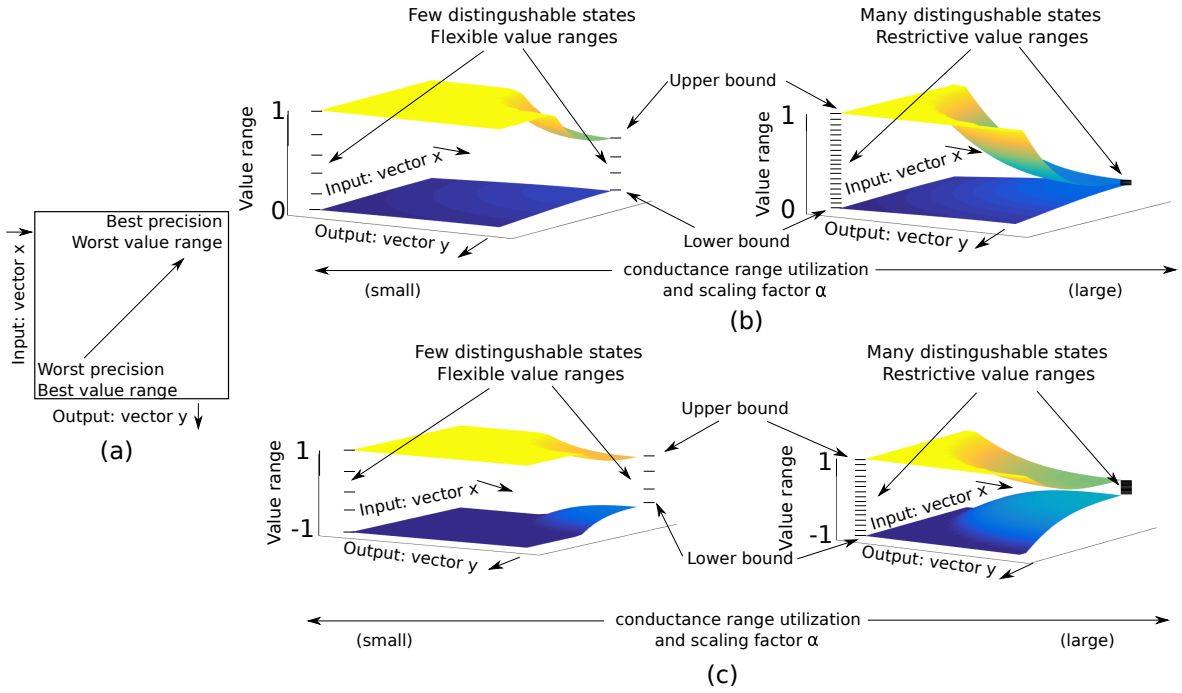

Fig. 5. (a) The value range and precision of a matrix element based on the crossbar location. Trade-off between value range and precision based on for a matrix element represented using (b) one memristor per matrix element and (c) two memristors per matrix element. The input (output) arrows indicate the direction of the wordlines (bitlines) within the crossbars.

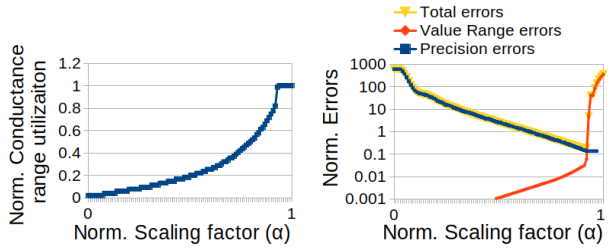

Fig. 6. (a) Memristor conductance range utilization vs. scaling factor . (b) Total errors, value range errors, precision errors vs. scaling factor . The results are obtained for a 128x128 crossbar.

is represented using one and two memristors in Figure 5(b) and Figure 5(c), respectively. The scaling factor regulates the utilization of the conductance range, i.e., a smaller (larger) results in that a smaller (larger) portion of the range is utilized. It can be observed that the least conductive portion of the range is always utilized in Figure 6(a).

When a small is used, the voltage IR-drop across the parasitics are minor, which results in very flexible value ranges. On the other hand, there are only few distinguishable states between the lower and upper bounds, which is illustrated to the left within (b) and (c) of Figure 5. For example, if only half of the conductance range is utilized, the write bit-accuracy is effectively reduced by one. When a larger is used, the value ranges become more restricted but there are many states between the bounds, which is shown to the right within (b) and (c) in Figure 5. The trends are slightly different when each element is represented using one and two memristors because the differential pair configuration can cancel out some errors. Moreover, as the differential pair configuration has more flexible ranges and distinguishable states, the computational accuracy is expected to be higher.

The mapping problem for a matrix A becomes that of selecting such that the total errors are minimized. If is set to small, large *precision errors* are introduced because there are few states between the upper and lower bounds. On the other hand, if is selected too large, large *value range errors* will be introduced because elements in A are outside the lower and upper bounds. Consequently, the key is to select such that *precision errors* are balanced with the *value range errors*, which is illustrated in Figure 6(b). The figure shows that the total errors are close to a minimum when the value range errors are equal to the precision errors.

B. Optimization insights

In this section, we outline our three main insights for minimizing the problem in Eq (10).

1) Decoupling the optimization of g and : Based on the discussion in the previous section, it is intuitive to decouple the optimization of g and . The scaling factor is used to optimize the trade-off between the precision errors and the value range errors. Given , the conductance values g are next optimized such that value range errors only occur for elements outside their respective value ranges.

2) Optimization of : The parameter regulates a trade-off between precision errors ($_p$) and the value range errors ($_v$). We observed in Figure 6(b) that total errors ($_{tot}$) are close to minimum when $_p \approx _v$. Consequently, the problem of minimizing the total errors can be cast into the problem of specifying such that $_p = _v$. By observing that the value range errors (precision errors) are increasing (decreasing) monotonically with respect to , it can be concluded that can be optimized using a *binary search*. We also observe that it is easy to maintain a lower bound on the total errors, which allows the binary search to be terminated when the gap between the best observed solution and the lower bound is less than a threshold.

3) Optimization of g: While optimizing the conductance values, we propose to relax the discrete variables g_w into continuous variables g. It is well known that it is easier to optimize problems with continuous variables than discrete variables. As there are many discrete states between g_{lb} and g_{ub}, predictable precision errors will be introduced when the continues variables are rounded (or snapped) to the closest discrete state after optimization.

We also propose to specify the conductance of each memristor g_{ij} with the objective of minimizing the element-wise difference $||A_{ij} - A^r_{ij}||^2$. In contrast, previous work attempted to specify all conductance values g to minimize $||A - A^r||^2$ as a whole [9]. This requires the use of computationally expensive steepest gradient descent, which results in long run-times. The decoupling allows the objective function to be minimized by iteratively updating the conductance values g with a conductance adjustments $\triangle g$. The adjustments are obtained from the matrix error $\triangle A = (A - A^r)$ by solving a linear equation.

VI. The XMAP Framework

An overview of the XMAP framework is shown in Figure 7. The input is a matrix A and the properties of the crossbar. The output is a crossbar programmed with state variables s and a specified scaling factor . The framework consists of five main steps. The fist step is to convert the non-linear devices into linear devices. The second step is to optimize the scaling factor with respect to A. The third step is to specify the conductance values g with respect to A and . The fourth step is to convert the specified conductance variables g into state variable s. The last step is to program the state variables to the hardware using accurate closed-loop tuning [1], [21].

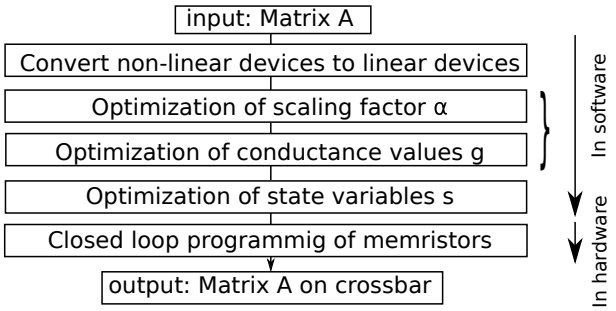

Fig. 7. Flow of XMAP framework.

A. Convert non-linear devices to linear devices

The first step is to convert the non-linear memristors and access transistor to linear (or ideal) memristors with a conductance of g, i.e., converting the circuit in Figure 3(a) to the circuit in Figure 3(b). The objective is to specify the bounds on the conductance range $[g_{lb}; g_{ub}]$ such that any set of conductance values g within the range can later be converted into state variables s within $[s_{min}; s_{max}]$. This can be performed by conservatively specifying the range $[g_{lb}; g_{ub}]$ with respect to $[s_{min}; s_{max}]$ and the non-linear equations describing the memristors $i_m(s; v_m)$ and access transistors $i_t(v_s; v_d; v_g)$. This is performed by simply plugging in the worst case state variable and node voltages into the non-linear equations.

B. Optimization of

In this section, we explain how the scaling factor is specified using the function *ComputeScalingFactor(A)* in Algorithm 1. The value range $[0; \ _{max}]$ for is first determined based on the constraint on the maximum output current from a bitline. Next, the variable is specified using a binary search.

The maximum scaling factor $_{max}$ is determined by solving for the scaling factor that results in that the maximum output current i_{max} is delivered through a bitline, as follows:

$$_{max} = \frac{i_{max}}{||Av_{max}||_1}; \tag{11}$$

where A is the matrix and i_{max} is the maximum output current from a bitline. v_{max} is a vector with the maximum voltage provided by the DACs. If a differential pair configuration is used, the matrix A is split into a positive and negative component before formulating the equation.

Given $_{max}$, the scaling factor that minimizes the total errors $_{tot}$ is determined using the binary search outlined in Algorithm 1. In iteration k of the binary search, the *ComputeConductance(A, $_k$)* function is invoked to specify the conductance values g with respect to A and $_k$. The details of the *ComputeConductance(A, $_k$)* function are provided in Section VI-C. Based on the specified conductance values g and $_k$, the value range errors $_v$ and precision errors $_p$ are computed using Eq (7) and Eq (8), respectively. Next, $_{k+1}$ is updated to be larger or smaller based on the ratio of the precision errors to the value range errors. More specifically, $_k$ is updated to be larger (smaller) if the precision errors c_p are larger (smaller) than the value range errors $_v$, which is shown on line 16 to line 21.

The binary search is continued until the gap between the best observed solution $_{best}$ and the lower bound $(lb_v + lb_p)$ is less than 5%, which is regulated using . $_{best}$ is the best observed solution in terms of total errors. When the value range errors are larger than the precision errors, we observe that the current precision errors can be used as a lower bound on the attainable precision errors (lb_p). Similarly, when the precision errors are larger than the value range errors, the current value range errors can be used as a lower bound on the attainable value range errors (lb_v). This allows us to formulate the termination condition on line 6.

In our implementation, we only update the lower bounds if the magnitude difference between $_p$ and $_v$ is less than $10X$. This is due to that for small (large) 's, we have empirically observed that precision (value range) errors may not be increasing (decreasing) monotonically.

C. Optimization of g

In this section, we explain how the conductance values g are specified with respect to a matrix A and scaling factor . We explain the algorithm for the case when each matrix element is represented using one memristor in Section VI-C1. The algorithm is extended to crossbars with a differential pair configuration in Section VI-C2.

Algorithm 1: ComputeScalingFactor(A).

Input: Target matrix A.
Output: Scaling factor .
$_{max} = \frac{i_{max}}{\|A v_{max}\|_1}$;
$_0 = _{max}$=2; $_{step} = _0$=2;
$lb_w = 0$; $lb_v = 0$; $_{best} = \infty$; $k = 0$; $c_{end} = 0.95$;
while $lb_p + lb_v <$ $_{best}$ **do**
  Compute g using ComputeConductance(A, $_k$)
  Compute $_v$ and $_p$ using g, $_k$, Eq (7), and Eq (8)
  $_{tot} = _p + _v$
  //save best solution in terms of total errors
  if $_{tot} <$ $_{best}$ **then**
    $=$ $_k$
    $_{best} =$ $_{tot}$
  end
  //update alpha within the binary search
  if $_p >$ $_v$ **then**
    $_{k+1} =$ $_k +$ $_{step}$
  else
    $_{k+1} =$ $_k -$ $_{step}$
  end
  $_{step} = \frac{_{step}}{2}$; $k = k + 1$;
  //update one of the lower bounds
  if $_p <$ $_v$ **then**
    $lb_v =$ $_v$
  else
    $lb_p =$ $_p$
  end
end
return ;

Algorithm 2: ComputeConductance(A,).

Input: Target matrix A and scaling factor .
Output: Memristor conductance values g.
$g_0 =$ A;
$^0_v = \infty$; = 0:99; $k = 1$;
Compute A^r and ^k_v using Eq (4) and Eq (7)
while $^k_v <$ $\cdot$ ^k_v ^{1} **do**
  Compute matrix error $\triangle A = (A - A^r)$
  Compute current correction $\triangle i =$ $\triangle A$
  $\triangle g = \frac{4\,i}{4\,v}$
  $g_{k+1} = g_k + \triangle g$
  Bound elements in g within $[g_{lb}; g_{ub}]$
  $k = k + 1$
  Compute A^r and ^k_v using Eq (4) and Eq (7)
end
return g;

1) Basic conductance mapping: A matrix A and scaling factor is mapped into conductance values g using the *ComputeConductance(A,)* function in Algorithm 2.

The algorithm is based on first linearly mapping the matrix A into conductance values $g_0 =$ A. The value range errors are next minimized by iteratively updating g_k to g_{k+1} using conductance adjustments $\triangle g$. The process is performed until the value range errors stop decreasing, which is shown on line 6 to line 14 of Algorithm 2.

The conductance adjustment $\triangle g$ is computed by first calculating the difference $\triangle A = (A - A^r)$. Using the approximation in Figure 2(a), the matrix error is translated into a current correction $\triangle i =$ $\triangle A$. The current corrections are next converted into a conductance adjustments $\triangle g = \triangle i$=$\triangle v$, where $\triangle v$ is the voltage drop across each device when computing G using Eq (2). G is anyways required to be calculated in order to compute A^r using Eq (4). After each update $g_{k+1} = g_k + \triangle g$, any conductance value that is updated outside $[g_{lb}; g_{ub}]$ is set g_{lb} and g_{ub}, respectively.

2) Conductance mapping using differential pair: In this section, we extend the conductance mapping in Algorithm 2 to handle crossbars with memristors arranged in a differential pair configuration. The extension is performed with the following two objectives:

The difference between A^r and A is minimized without explicitly decomposing A^r and A into a positive and negative component.

To minimize $||A - A^r||^2$ using Algorithm 2, only the conductance of one memristors in each differential pair is updated in each iteration. Moreover, one memristor in each pair should have the minimum conductance g_{lb}.

The motivation for avoiding an explicit decomposition is that it significantly degrades the achievable computational accuracy. This stems from that it is impossible to realize small elements at certain locations in a crossbar, which is illustrated with an example in Figure 8.

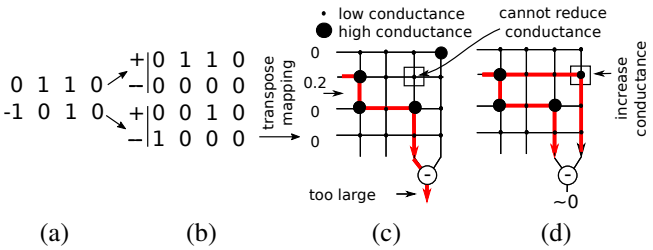

Fig. 8. (a) 2x4 matrix. (b) Differential pair decomposition into a 4x4 matrix and transpose mapping to crossbar. (c) The net output current from column three and four is too large with respect to the input vector $[0; 0.2; 0; 0]$. The current cannot be reduced by tuning the memristor at (2,3) to be less conductive. (d) XMAP reduces the current by tuning the memristor at (2,4) to be more conductive.

It is straightforward to understand that an error in $\triangle A$ can be eliminated by tuning either one of the memristors in a differential pair. A positive error can be eliminated by tuning the memristor representing the positive (negative) component to be less (more) conductive. Similarly, a negative error can be eliminated by tuning the positive (negative) memristor to be more (less) conductive. To minimize the voltage IR-drop in the crossbar, we propose always eliminate errors by tuning the appropriate memristor to be less conductive. However, if a memristor already has the minimum conductance g_{lb}, the error is eliminated by tuning the other memristor to be more conductive. This method ensures that one memristor in each pair will have the minimum conductance.

The outlined method only requires modifications on line 3 and line 8 in Algorithm 2. On line 3, the matrix A is required to be split into a positive and negative component before being mapped into conductance values, which is shown in Figure 4(b). On line 8, the matrix error $\triangle A$ is converted into current corrections. By translating the errors $\triangle A$ into current corrections using the flow in Figure 9, it is ensured that only one memristor is updated and that one memristor in each pair will have the conductance g_{lb}.

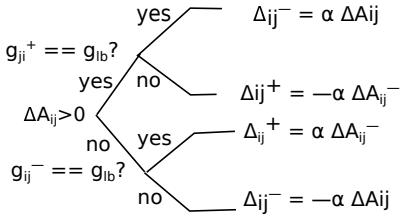

Fig. 9. Flow for updating $\triangle A$ into current correction $\triangle i$. Let an element in $\triangle A$ be denoted $\triangle A_{ij}$. The current correction for the two memristors representing that element are denoted $\triangle_{ij}^{+}$ and $\triangle_{ij}$, respectively. All current corrections are first set to zero. Next, half of the corrections are updated based on $\triangle A$.

D. Specification of state variables

In this section, the state variables of the memristors are determined from the conductance values g and a calibration signal v_{cal}, which is illustrated in Figure 10. The results will be different for different selections of the calibration signal. We set v_{cal} to v_{max}=2, where v_{max} is the maximum input voltage based on an empirical study. First, the node voltages on the wordlines (v_{word}) and bitlines (v_{bit}) and the currents through the conductors i_g are computed using g and v_{cal}, which is shown in Figure 10(a). Next, the state variables s are computed using v_{word}, v_{bit}, and i_g, which is illustrated in Figure 10(b).

Computation of v_{word}, v_{bit}, i_g: First, the node voltages of the wordlines (v_{word}) and the bitlines (v_{bit}) in the crossbar are computed with respect to g and a calibration signal v_{cal} using Eq (1). Next, the currents through each conductor g is obtained using $i_g = g \cdot (v_{word} - v_{bit})$.

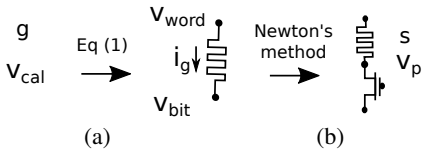

Fig. 10. Specification of s from g and v_{cal}.

Computation of state variables s: The state variables s are found by solving a non-linear system of two equations, as follows:

$$X = \begin{matrix} s \\ v_p \end{matrix} \; ; \quad F(X) = i_g - \begin{matrix} i_m(s; v_{word}\text{-}v_p) \\ i_t(v_p; v_{bit}; v_g) \end{matrix} \; ; \qquad (12)$$

where v_p is the node voltages between the memristors and the access transistors. Next, Newton's method is used to solve for $F(X) = 0$, as follows:

$$X_{k+1} = X_k - J^{-1} F(X_k); \qquad (13)$$

where J^{-1} is the inverse of the Jacobian of F. Note that Newtons method can be applied independently for each memristor and access transistor pair.

VII. XMAP SPEED-UP TECHNIQUES

In this section, we propose three speed-up techniques for the XMAP algorithm. We analyze the run-time bottlenecks of XMAP in Section VII-A. The three speed-up techniques to solve the bottlenecks are proposed in Section VII-B.

A. Run-time bottleneck analysis

In this section, we analyze the run-time bottlenecks of XMAP. The flow of the algorithm is shown in Figure 11. The flow consists of two loops. The scaling factor is optimized in the left loop and the conductance values g are optimized in the right loop. Based on a breakdown of the run-time shown in Figure 12(a), the edges in Figure 11 are annotated with the qualitative and quantative number of executions and run-time per execution for a 256x256 matrix. The left loop is only executed a few times and the run-time is short but it calls the right loop. The right loop is takes *long* run-time and is performed *many* times. Consequently, the overall run-time is **many* long**. The total mapping time is dominated by the computation of the conductance matrix G, which is shown in Figure 12(a). Clearly, the computation of G is the bottleneck of the mapping algorithm. We further breakdown the run-time of computing G with respect to the matrix size in Figure 12(b).

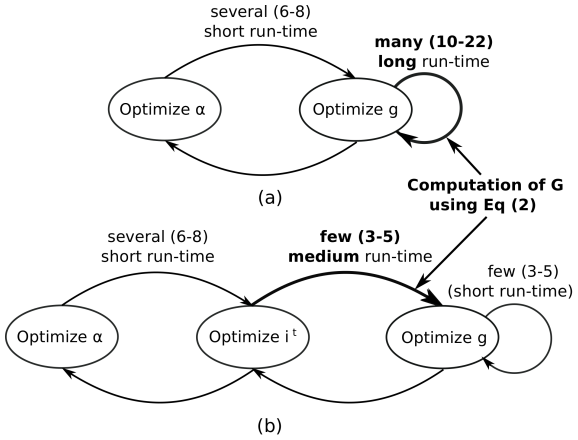

Fig. 11. (a) Flow of XMAP baseline and (b) flow of XMAP with speed-up. The annotated numbers are with respect to a 256x256 matrix.

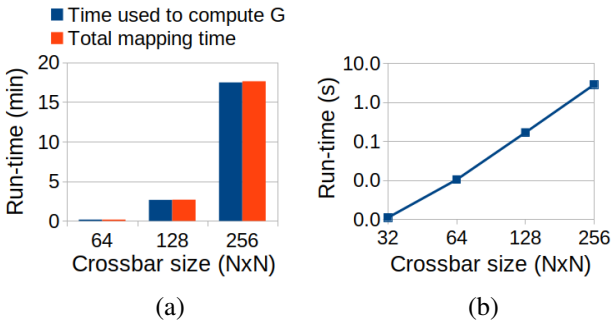

Fig. 12. (a) Breakdown of run-time of XMAP. (b) Runtime to compute G using Eq (2) for crossbars of different dimensions.

We propose to speed-up XMAP using a i) hierarchical algorithm, ii) an early termination feature, and iii) speed-up techniques for computing G. Instead of optimizing the conductance values g with respect to $_v$, the hierarchical

algorithm optimizes the conductance values with respect to a target current i^t through each memristor device. Next, the target currents i^t are calibrated with respect to $_v$, the details are provided in Section VII-B1. An early termination feature to Algorithm 2 is proposed in Section VII-B2. Together, the first two techniques reduce the number of times G is computed from **many** to **several** in Figure 11. A technique to speed-up the calculation of G itself is proposed in Section VII-B3. This explains why the computation of G in Figure 11(b) is labeled with **medium** run-time (instead of long). Consequently, the overall run-time is reduced from **many*****long** to **several*****medium**.

B. Speed-up techniques

1) Hierarchical approach: The hierarchical flow is implemented using Algorithm 3 and by modifying Algorithm 2.

The main idea of the hierarchical algorithm is to optimize the conductance values g with respect to a target current i_t and an uniform input calibration vector v_{cal}. Next, the target currents are optimized to minimize the value range errors $_v$. The original approach in Algorithm 2 was based on directly optimizing g with respect to $_v$.

The relation between the node voltages in a crossbar v_{xbar}, the currents through the memristors i_k, and an voltage calibration vector v_{cal} can be formulated, as follows:

$$\overline{Y}(r_w; r_{in}; r_{out}) \begin{matrix} v_{xbar} \\ v_{dac} \end{matrix} = \begin{matrix} 2 \\ 4 \\ \\ v_{cal} \end{matrix} \begin{matrix} -i^t \\ i^t \\ \\ 3 \\ 5 \end{matrix}; \quad (14)$$

where $\overline{Y}$ is a matrix with dimension $(2NM + M)$x$(2NM + M)$. The system matrix $\overline{Y}$ is formulated using KCL in similar to Y in Eq (1). However, the system matrix $\overline{Y}$ is much more sparse than Y because it is not a function of g.

The crossbar node voltages v_{xbar} can be solved for using Eq (14). Next, the voltage across each memristor v_m is computed as the voltage difference between two node voltages in v_{xbar}. Finally, the conductance values g are obtained by dividing the target currents by v_m elementwise. The iterative approach in Algorithm 3 is proposed to ensure that the conductance values g are specified within the bounds $[g_{lb}; g_{ub}]$.

Now we turn our attention to how the target currents are updated to minimizing the value range errors $_v$. To simplify the optimization process, we introduce an ideal conductance matrix G_k^{target}. The matrix directly defines the target currents i_t. Next, Algorithm 2 is modified to initialize and update the ideal conductance matrix G_k^{target} on line 3 and line 10 of Algorithm 2. The variable G_k^{target} basically replaces the variable g_k on line 3. Given G_k^{target} and an uniform calibration vector v_{cal}, a target current i^t through each memristor is specified using the approximation in Figure 2(a). The conductance values g are subsequently specified to deliver the target current through each device using *ComputeConductanceCurrent(i^t,v_{cal})* in Algorithm 3. Next, line 10 in Algorithm 2 is modified to update G_k^{target} into G_{k+1}^{target} using $G_{k+1}^{target} = G_k^{target} + \quad (A - A^r)$.

2) Early termination: In this section, we propose to speed-up the optimization of g using an early termination feature. The memristor conductance values g are iteratively updated until the value range errors converge using Algorithm 2. We observe that if the value range errors are significantly smaller than the precision errors, there is no need to continue optimizing the value range errors. The total errors would be dominated by the precision errors. Consequently, the optimization in Algorithm 2 can be terminated early when the described situation occurs. In particular, we terminate the while loop within Algorithm 2 if $_v \cdot c_t < lb_p$, where c_t is set to 1000. Moreover, we initialize lb_p using the ideal bit-accuracy of the memristors. This ensures that the lower bound is always non-zero, which saves substantial amount of run-time.

Algorithm 3: ComputeConductanceCurrent(i^t, v_{cal}).

```
1  Input: Target current i^t and calibration vector v_cal.
2  Output: Memristor conductance values g.
3  i_0 = i^t;
4  Compute g and v_m using Eq (14)
5  while ¬g ∈ [g_lb; g_ub] do
6  |   Bound g within [g_lb; g_ub];
7  |   i_{k+1} = v_m · g;
8  |   Compute g and v_m using Eq (14)
9  end
10 return g;
```

3) Computation of G: In this section, we propose to speed-up the calculation of G to reduce the overall run-time of XMAP. The conductance matrix G is computed by solving Eq (2) using sparse LU factorization with pivoting, as follows:

$$G = SQU \ ^1L \ ^1PB \quad (15)$$

where P and Q are permutation matrices. L and U are respectively a lower and upper triangular matrices. The expression is evaluated from right-to-left. First, $T_1 = L \ ^1(PB)$ is computed using forward substitution. Next, $T_2 = U \ ^1T_1$ is solved using backward substitution. Lastly, G is computed using $G = SQT_2$.

We propose to reduce the run-time by (i) permuting the order of linear algebra operations and (ii) avoiding redundant computation. The first speed-up is obtained by computing SQT_2 from left-to-right instead of right-to-left. This reduces the run-time significantly because S has smaller dimensions than T_2. The reordering results in that Q permutes the N non-zero elements in S to form a new selection matrix $\hat{S} = SQ$. Next, $\hat{S}$ is used to select N of the rows in T_2. We now turn our attention to avoiding redundant computation by exploiting the structure of $\hat{S}$. $\hat{S}$ is an (N)x$(2NM + N + M)$ matrix with only N non-zero elements. Let r be the number of columns starting from the first in $\hat{S}$ that only contain zeros. It can be observed that the top r rows of T_2 are not used, which is illustrated in Figure 13(a). Consequently, the top r rows in T_2 are not required to be computed when solving $T_2 = U \ ^1T_1$, which is illustrated in Figure 13(b). Therefore, the matrices $\hat{S}$, U, T_1, and T_2 can be reduced to $\hat{S}_r$ U_r, T_{1r}, and T_{2r} by removing rows and columns as illustrated in (a) and (b) of Figure 13. Next, G can be computed more efficiently, as follows:

$$G = \hat{S}_r(U_r \ ^1T_{1r}) \quad (16)$$

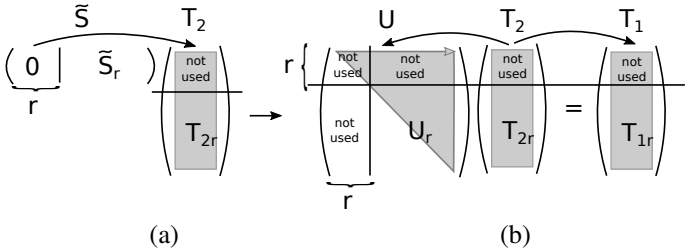

Fig. 13. (a) Exploiting the structure of S to reduce T_2 into T_{2r}. (b) Exploiting the structure of T_{2r} to reduce U and T_1 into U_r and T_{1r}, respectively.

VIII. Experimental evaluation

The experimental results are obtained using a quad core 3.4 GHz Linux machine with 32GB of memory. XMAP is implemented in MATLAB. Memristor crossbars with dimensions 32x32 to 256x256 are used in the evaluation. The parasitic properties of the crossbars are provided in Table I. We also evaluate the sensitivity to two non-linear device models. The static version of the memristor model used by HP labs is formulated [26], as follows:

$$i_m = v_m : * (s \cdot G_m + (1 - s) \cdot a \cdot exp(b \cdot \sqrt{|v_m|})); \quad (17)$$

where $a = 7{:}2e-9$, $b = 4{:}7$, and $G_m = 2{:}5e-3$. s is the state variable of the memristor. We also evaluate the performance using the model provided in [27], as follows:

$$i_m = I_0 \cdot exp(-\frac{s}{d_0}) \cdot sinh(\frac{v_m}{v_0}); \quad (18)$$

where $d_0 = 0{:}25nm$, $i_0 = 1mA$, and $v_0 = 0{:}25V$. For this model, the maximum programmable conductance is set to $10k$ due to a constraint on s within the model.

The computational accuracy of the analog matrix-vector multiplication performed by a DPE is evaluated using the circuit modeling described in Section III. The accuracy is therefore equivalent to that of SPICE. The parameters (, g) or (, s) are obtained using XMAP as described in Section VI.

Using the outlined setup, we compare XMAP with a naive linear mapping and the state-of-the-art mapping in [20]. We evaluate three different versions of XMAP to clearly demonstrate the benefits from the different optimization techniques. XMAP+SGD [9] is the flow in Section VI with Algorithm 2 replaced with the steepest gradient descent (SGD) algorithm in [9]. XMAP-B is the baseline flow of XMAP in Section VI. XMAP is XMAP-B with the speed-up technique in Section VII.

The performance on the DPE level is evaluated in Section VIII-A. The capability of XMAP to accelerate signal compression and the inference of neural networks is evaluated in Section VIII-B.

A. DPE level evaluation

In this section, we evaluate the XMAP framework on the DPE level. We first evaluate the optimization of and g within XMAP. Next, we compare XMAP with previous works and perform a sensitivity analysis to different crossbar parameters.

We evaluate the optimization of using total errors in Figure 14(a). The figure shows that the gap between the lower and upper bound on the total errors converges swiftly using the binary search in Algorithm 1. We evaluate the optimization of g in terms of value range errors in Figure 14(b). The figure shows that the value range errors quickly converge to a minimum using Algorithm 2. The smooth optimization of both parameters demonstrates the effectiveness of XMAP.

TABLE I
Crossbar parameters used in evaluation.

Property	Value
Array block resistance	2
Input resistance	100
Output resistance	100
Programmable resistance range	$[2k; 3M]$
Max input voltage	$0{:}25V$
Max output current	$1{:}0\ mA$
ReRAM bit-accuracy	6 bits
DAC/ADC bit-accuracy	8 bits
Transistor model	JETMOS v1

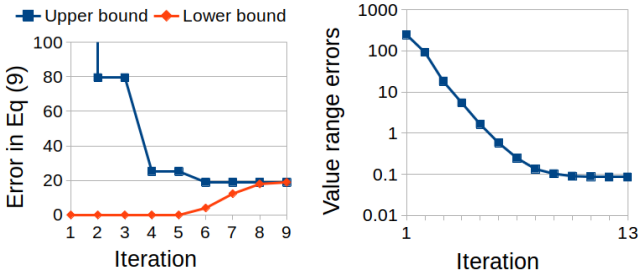

Fig. 14. Optimization of (a) in Algorithm 1 and (b) g in Algorithm 2.

We compare XMAP with previous work in Figure 15. The comparison is performed with respect to the objective in Eq (6), the objective in Eq (10), and run-time. Crossbars with a differential pair configuration and a dimension of 128x128 are used for the evaluation. The figure shows that the performance in terms of errors is similar for the different versions of the XMAP framework, i.e. optimizing the objective element-wise and the speed-up techniques do not degrade the performance in terms of errors. Compared with the mapping in [20] and linear mapping, the errors in Eq (10) are reduced with 48X and 1600X, respectively. Therefore, it is not surprising that the errors in Eq (6) are reduced with 3:29X and 17:10X, respectively. XMAP is 3:8X and 42X faster than XMAP-B and XMAP-SGD, respectively. The shorter run-time stems from the proposed speed-up techniques in Section VII and avoiding

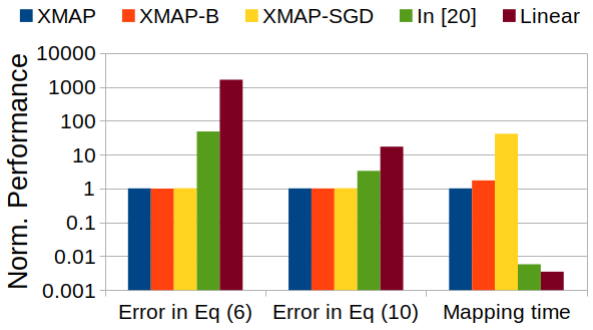

Fig. 15. Comparison of XMAP with the mapping in [20] and linear mapping in terms of errors in Eq (10), errors in Eq (6), and run-time.

optimization using steepest gradient descent. Compared with the mapping in [20] and linear mapping, the XMAP framework is 173X and 287X slower, respectively. Nevertheless, we deem the slower run-time acceptable for the improved computational accuracy. When a single memristor is used per matrix element, the run-time of XMAP is 0:04 min, 0:4 min, and 7:7 min for crossbars with dimensions 64x64, 128x128, and 256x256, respectively. Using a crossbar with differential pair configuration, the run-time is 0:02 min, 0:28 min, and 2:72 min for 64x64, 128x128, and 256x256 crossbars, respectively.

We evaluate the sensitivity of XMAP to the crossbar parameters in Figure 16. We evaluate the accuracy in terms of equivalent digital accuracy. The equivalent digital accuracy is obtained through regression with respect to a fixed-point multiplier (where the matrix is represented using one to ten bits). Specifically, the equivalent bit-accuracy is obtained by performing matrix-vector multiplications using matrices with different fixed-point precision (between one and ten bits). Next, we fit a two-term power series to the data using regression. Next, an error can be translated into an equivalent bit-accuracy using the power series. Here, we simulated $10,000$ input vectors both when building the model and evaluating the equivalent bit-accuracy of a mapping scheme. We evaluate the sensitivity of representing each matrix element using one or two memristors for different crossbar sizes in Figure 16(a). The differential pair configuration results in 1-2 bits higher computational accuracy due to the more flexible value ranges in Figure 5. The upper bound is seven bits, i.e., two memristors with six bits each. The figure shows that the equivalent digital accuracy is gracefully degraded when the dimensions of the crossbar is scaled-up. It is very promising that the differential pair configuration improves the digital bit-accuracy with more than one bit on the average, which validates the effectiveness of the pair-wise update in Figure 9. The sensitivity to the non-linear device models with respect to memristors with different write bit-accuracy in Figure 16(b). The figure shows that the device model [26] is almost linear up to 8 bits, which is more than the memristor write bit-accuracy. Using the non-linear model in [27], it can be observed that the equivalent digital accuracy saturates at 6:5 bits. The errors are almost negligible for a write bit-accuracy of 5 bits or lower. The sensitivity to the wire resistance is evaluated in Figure 16(c). The figure shows that the computational accuracy is gracefully degraded from 6:3 bits to 2:3 bits when the wire resistance is increased from 0:1 to 10 . The sensitivity to the minimum programmable resistance is shown in Figure 16(d). It can be observed that the computational accuracy is gracefully degraded with the inverse of the minimum programmable resistance. This stems from that it is advantageous to have the programmable states in a more resistive range, as less voltage IR-drop is introduced over the parasitics. The equivalent accuracy in Figure 16(d) is slightly higher than expected because we increased the minimum programmable resistance to $10k$ due to a constraint on the maximum current in Eq (18). The smooth trade-offs across all the evaluated crossbar parameters indicate that XMAP is in the vicinity of an optimal solution.

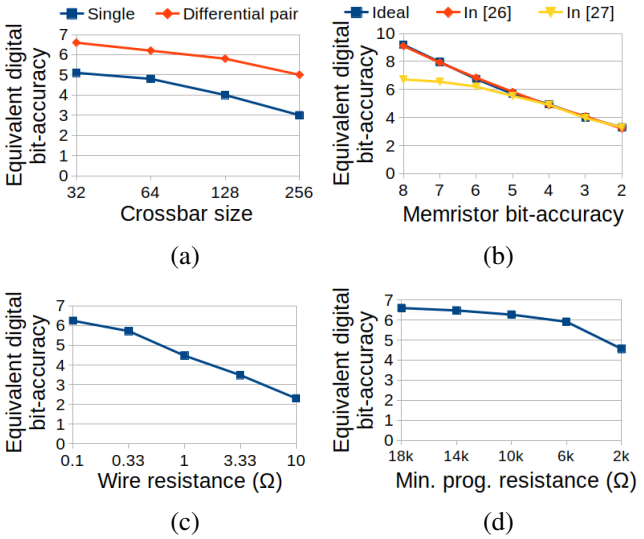

Fig. 16. Sensitivity of the computational accuracy with respect to various crossbar parameters. The computational accuracy is measured in terms of equivalent digital bit-accuracy.

B. Application level evaluation

In this section, we evaluate the impact of XMAP on the application level performance. The devices are set to be linear to enable evaluation with reasonable run-times. Crossbars with a differential pair configuration are used in the evaluation.

1) Signal compression: In this section, we evaluate the application level performance when a DPE is used to perform signal compression using 1D-DCT.

The signal compression is performed by first programming a memristor crossbar with the DCT matrix D [28]. Next, the signals in Table II are converted from the time domain to the frequency domain using a matrix-vector multiplication operation. Decompression is performed using digital hardware as it is assumed to be performed on a cloud server. The compression is evaluated by comparing the original signal with the decompressed signal. The signal quality is evaluated in terms of mean square error (MSE) and peak signal to noise ratio (PSNR). The degree of compression is evaluated in terms of bits per sample (BPS). Definitions of MSE, PSNR, and BPS are provided in [29].

TABLE II
PROPERTIES OF INPUT SIGNALS.

Id	Equation	Samples	BPS (num)
1	$\sin(x)$	128	8
2	$\sin(2x)$	128	8
3	$\sin(2x) + \cos(x)$	128	8
4	$\sin(x) + \sin(2x) + \sin(3x)$	128	8
5	$\sin(2x) + 0.2\sin(x)$	128	8

We compare X-MAP and X-MAP+Q with linear mapping and the state-of-the-art mapping in [20]. In the discussion of the results, we mainly compare the state-of-the-art algorithm with XMAP and XMAP-Q (page limit). X-MAP+Q is X-MAP extended with quantization of the frequency components, where the quantization level is configured such that the degree of compression is better than in [20]. Quantization is a standard technique within signal and image processing [28].

For the signal compression, we discard frequency components such that only 99% of the signal energy is preserved.

The mapping performance is evaluated using errors in Eq (10) and mapping time in Table III. A 128x256 crossbar is used for the signal compression. The table shows that XMAP reduces the total errors with between 59X and 1728X. The errors reductions come at the expense of longer mapping time.

TABLE III
EVALUATION OF DCT MATRIX MAPPING FOR SIGNAL COMPRESSION.

Method	Signal compression	
	Error in Eq (10)	Time (s)
Linear	7468.2	1.3
In [20]	1240.2	2.7
XMAP	21.1	63.9

The application level performance of the signal compression is evaluated in Table IV. Compared within [20], it can be observed that XMAP respectively improves MSE and PSNR with 2.6X and 4.2X on the average. However, the BPS is 6% higher when XMAP is used. This may stem from that some frequency components of small magnitude are more accurately captured, which requires additional bits. When quantization is applied, we can observe that XMAP+Q improves both the signal quality and the degree of compression compared with the mapping algorithm in [20]. Compared with performing the compression using a digital ASIC [1], the speed and efficiency are 3.8X and 64.5X higher, respectively.

TABLE IV
EVALUATION OF SIGNAL COMPRESSION.

Id	Method	Signal quality		Degree of compression (BPS)
		(MSE)	(PSNR)	
1	Linear	3990.7	-23.9	0.7
	In [20]	203.0	2.0	1.4
	XMAP	91.3	9.0	1.4
	XMAP+Q	144.9	4.9	0.9
2	Linear	4109.1	-24.1	1.7
	In [20]	271.2	-0.5	1.5
	XMAP	82.5	9.8	1.6
	XMAP+Q	150.2	4.6	1.1
3	Linear	4173.0	-24.2	0.8
	In [20]	214.6	1.5	1.4
	XMAP	77.2	10.4	1.5
	XMAP+Q	130.4	5.9	1.0
4	Linear	1888.9	-17.4	2.3
	In [20]	127.6	6.0	2.7
	XMAP	39.7	16.2	2.8
	XMAP+Q	219.8	1.3	1.2
5	Linear	3233.4	-22.0	0.9
	In [20]	166.8	3.7	1.5
	XMAP	92.5	8.8	1.5
	XMAP+Q	141.7	5.1	1.0
Norm.	Linear	17.69	-8.71	0.78
	In [20]	1.00	1.00	1.00
	XMAP	**0.39**	**4.23**	1.06
	XMAP+Q	0.80	1.71	**0.61**

2) Neural network inference: In this section, we evaluate the application level performance when DPEs are used to accelerate the inference of five different neural networks. Three multilayer perception (MLP) networks and two convolutional neural networks (CNNs) with a VGG structure are trained in software on the MNIST and CIFAR-10 datasets using TensorFlow, respectively. An overview of the properties of the different networks is shown in Table V. The software classification accuracy is the probability of an input being correctly classified. The MLPs only consists of fully-connected (FC) layers and the CNNs consists of convolution (Conv) layers, FC layers, max pooling layers, and normalization layers. The number after MLP or VGG indicates the number of layers of neurons.

TABLE V
PROPERTIES OF EVALUATED NEURAL NETWORKS.

Name	Dataset	Software accuracy (%)	#Conv. layers	#FC layers	#Max pooling layers	#Norm layers	#Train. params (M)
MLP-3	MNIST	98.25	0	2	0	0	0.4
MLP-4	MNIST	98.35	0	3	0	0	0.5
MLP-5	MNIST	98.41	0	4	0	0	0.6
VGG-7	CIFAR-10	85.94	4	2	2	6	2.2
VGG-13	CIFAR-10	93.26	10	2	5	12	9.7

Next, we map the neural networks to a crossbar based platform using XMAP to evaluate the classification accuracy in hardware. Each FC and Conv layer can be formulated as a MVM operation. The dimensions of the matrices are shown in Table VI. We use the kernel to crossbar decomposition described in [30] to map each FC layer or Conv layer to one or multiple crossbars. If a weight matrix is smaller than 128x128, smaller crossbars are used to save power [14]. If a weight matrix is larger than 128x128, the weight matrix is partitioned to multiple crossbars of size 128x128. Next, XMAP is used to map each weight matrix to each crossbar. Note that the results from the separate crossbars are summed using adders in the digital domain.

The mapping algorithms are compared on the application level in Table VII. The comparison is performed using the total errors in Eq (10), classification accuracy, and mapping time, which are reported in the columns labeled 'Total errors', 'Mapping time', and 'Classification accuracy', respectively. The classification accuracy is shown both with and without the errors introduced by the 8-bit DACs and ADCs. For each network, the software classification accuracy is shown using a row labeled 'Software'. We compare XMAP with linear mapping and the state-of-the-art mapping in [20].

Linear mapping performs poorly for all the neural networks. The mapping algorithm in [20] attains close to software level accuracy on the MLPs trained on the MNIST data set. However, there is a 12.2% and 10.4% loss in accuracy on CIFAR-10 using CNN-7 and CNN-13, respectively. The runtime is less than half an hour for all networks. Compared with the mapping in [20], XMAP achieves 98% of software level accuracy on the average. There is 7.7% accuracy loss for CNN-7 on CIFAR-10. The accuracy improvements are not surprising because the total errors are 75X smaller on the average. The improved accuracy comes at the expense of longer mapping times. Nevertheless, the mapping time is less than 12.1 hours for all networks. Compared with a digital approach [1], all data movement is eliminated and 256x256 crossbars have respectively 500X and 4X higher speed and power than digital ASICs.

TABLE VI
DIMENSIONS OF THE WEIGHT MATRICES IN THE EVALUATED NEURAL NETWORKS.

Name	Layer											
	1	2	3	4	5	6	7	8	9	10	11	12
MLP-3	784x500	500x10	'-'	'-'	'-'	'-'	'-'	'-'	'-'	'-'	'-'	'-'
MLP-4	784x500	500x300	300x10	'-'	'-'	'-'	'-'	'-'	'-'	'-'	'-'	'-'
MLP-5	784x500	500x300	300x200	200x10	'-'	'-'	'-'	'-'	'-'	'-'	'-'	'-'
VGG-7	27x32	288x32	288x64	576x64	2304x512	512x10	'-'	'-'	'-'	'-'	'-'	'-'
VGG-13	27x64	576x64	576x128	1152x128	1152x256	2304x256	2304x512	4608x512	4608x512	4608x512	512x512	512x10

TABLE VII
EVALUATION OF NEURAL NETWORK INFERENCE.

Network	Method	Error in Eq (10)	Mapping time (h)	Classification accuracy	
				analog (%)	+DAC/ADC (%)
MLP-3	Software	'-'	'-'	98.2	98.2
	Linear	18726	0.0	8.9	8.9
	In [20]	959	0.0	98.2	98.2
	XMAP	229	1.3	97.9	98.0
MLP-4	Software	'-'	'-'	98.2	98.2
	Linear	24033	0.0	8.9	8.9
	In [20]	1422	0.0	98.2	98.2
	XMAP	264	1.4	98.4	98.4
MLP-5	Software	'-'	'-'	98.5	98.5
	Linear	38638	0.0	8.9	8.9
	In [20]	2533	0.1	98.1	98.1
	XMAP	388	1.9	98.5	98.5
CNN-7	Software	'-'	'-'	82.8	82.8
	Linear	39873	0.1	9.0	9.0
	In [20]	1636	0.1	71.3	70.6
	XMAP	907	2.1	75.9	75.1
CNN-13	Software	'-'	'-'	92.5	92.5
	Linear	189650	0.2	11.2	11.2
	In [20]	8803	0.4	81.9	82.1
	XMAP	3222	12.1	92.1	92.0
Norm.	Software	'-'	'-'	**1.00**	**1.00**
	Linear	75.07	0.02	0.10	0.10
	In [20]	4.13	0.03	0.95	0.95
	XMAP	**1.00**	**1.00**	0.98	0.98

IX. SUMMARY AND FUTURE WORK

In this paper, we have proposed XMAP for mapping arbitrary matrices to memristor crossbar arrays. Compared with the state-of-the-art mapping algorithm, the accuracy is improved with up to 3:29X, which translates into application level performance improvements. In the future, we plan to extend XMAP to handle larger crossbars, resistance drift, device aging, device-to-device variations, and stuck-at-fault defects. The main challenge is to further scale-up the crossbar dimensions without severely elongating mapping time.

REFERENCES

[1] M. Hu *et al.*, "Memristor-based analog computation and neural network classification with a DPE," *Adv. Materials*, vol. 30, 2018.

[2] R. S. Williams, "What's next? [the end of moore's law]," *Computing in Science Engineering*, vol. 19, no. 2, pp. 7–13, 2017.

[3] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH News*, vol. 23, no. 1, pp. 20–24, 1995.

[4] C. Xu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," HPCA'15, pp. 476–488, 2015.

[5] B. Liu *et al.*, "Vortex: Variation-aware training for memristor X-bar," DAC'15, pp. 1–6, 2015.

[6] M. Prezioso *et al.*, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *CoRR*, vol. abs/1412.0611, 2014.

[7] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "Rxnn: A framework for evaluating deep neural networks on resistive crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–14, 2020.

[8] Z. He *et al.*, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," DAC'19, pp. 57:1–57:6, 2019.

[9] B. Liu *et al.*, "Reduction and IR-drop compensations techniques for reliable neuromorphic systems," ICCAD'14, pp. 63–70, 2014.

[10] Y. Wang, W. Wen, B. Liu, D. Chiarulli, and H. H. Li, "Group scissor: Scaling neuromorphic computing design to large neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, (New York, NY, USA), pp. 85:1–85:6, ACM, 2017.

[11] A. Shafiee *et al.*, "ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars," ISCA'16, pp. 14–26, 2016.

[12] P. Chi *et al.*, "PRIME: a novel processing-in-memory architecture for neural network computation in reram-based main memory," ISCA'16, pp. 27–39, 2016.

[13] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," HPCA'16, pp. 1–13, 2016.

[14] B. Feinberg *et al.*, "Enabling scientific computing on memristive accelerators," ISCA'18, pp. 367–382, 2018.

[15] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "Rram-based analog approximate computing," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1905–1917, 2015.

[16] S. Agarwal, R. L. Schiek, and M. J. Marinella, "Compensating for parasitic voltage drops in resistive memory arrays," in *2017 IEEE International Memory Workshop (IMW)*, pp. 1–4, 2017.

[17] Y. Jeong, M. A. Zidan, and W. D. Lu, "Parasitic effect analysis in memristor-array-based neuromorphic systems," *IEEE Transactions on Nanotechnology*, vol. 17, no. 1, pp. 184–193, 2018.

[18] M. Hu *et al.*, "Memristor crossbar-based neuromorphic computing system: A case study," *NN. and Learning Sys., IEEE Tran. on*, vol. 25, pp. 1864–1878, 2014.

[19] L. Xia *et al.*, "Technological exploration of rram crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, 2016.

[20] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," DAC'16, pp. 1–6, 2016.

[21] F. Alibart *et al.*, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[22] C.-W. Ho *et al.*, "The modified nodal approach to network analysis," *IEEE Tran. on Cir. and Sys.*, vol. 22, pp. 504–509, June 1975.

[23] F. Zhang and M. Hu, "Cccs: Customized spice-level crossbar-array circuit simulator for in-memory computing," ICCAD '20, 2020.

[24] J. Rommes *et al.*, "Efficient methods for large resistor networks," *IEEE Tran. on CAD of ICs Cir. and Sys.*, vol. 29, no. 1, pp. 28–39, 2010.

[25] T. Ngoc *et al.*, "New memristor-based crossbar array architecture with 50-% area reduction and 48-% power saving for matrix-vector multiplication of analog computing," vol. 14, pp. 356–363, 06 2014.

[26] J. P. Strachan, A. C. Torrezan, F. Miao, M. D. Pickett, J. J. Yang, W. Yi, G. Medeiros-Ribeiro, and R. S. Williams, "State dynamics and modeling of tantalum oxide memristors," *IEEE Transactions on Electron Devices*, vol. 60, no. 7, pp. 2194–2202, 2013.

[27] S. Yu *et al.*, "A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation," *Advanced Materials*, vol. 25, no. 12, pp. 1774–1779, 2013.

[28] C. Li *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, 2017.

[29] U. Jayasankar, V. Thirumal, and D. Ponnurangam, "A survey on data compression techniques: Data quality, coding schemes, data type and applications," *Computer and Information Sciences*, vol. 33, no. 2, pp. 119–140, 2021.

[30] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," HPCA'17, pp. 541–552, 2017.