

Enabling In-Memory Computation of Binary BLAS using ReRAM Crossbar Arrays

Debjyoti Bhattacharjee, Farhad Merchant, Anupam Chattopadhyay

School of Computer Science and Engineering, Nanyang Technological University, Singapore

{debjyoti001, mamirali, anupam}@ntu.edu.sg

Abstract—Memristive devices, such as ReRAMs, are fast gaining prominence for their low leakage power, high endurance and non-volatile storage capabilities. ReRAM crossbar arrays also found usage as platform for in-memory computing, particularly for data-intensive computations, due to its inherent capability to perform stateful logic operations. Binary matrix and vector operations arise in several applications that require close interaction with the storage, such as Error Correction Codes (ECC), approximate graph mining, and in general, diverse big data applications. In this paper, we explore for the first time, an efficient mapping of Binary Basic Linear Algebra Subprograms (BiBLAS) onto hybrid CMOS-ReRAM crossbar array. We investigate the impact of crossbar configurations on the delay, and area of BiBLAS operations for various vector sizes.

I. INTRODUCTION

Binary matrix is a matrix with entries from the Boolean domain $\mathcal{B}\{0,1\}$. Binary matrix computations are encountered across multiple application domains, such as, high-performance computing [10], algebraic cryptanalysis [5], combinatorics and finite geometry data [22], clustering [14] and in general large scale data analysis [23].

A notable use case is similarity search problem in the graph theory. For similarity search, given a query graph q and a data graph D , one needs to retrieve graphs in G that have similarity distances greater than a specified threshold t in the data graph D . Such a problem arises in several applications ranging from bio-informatics to identification compounds in the chemical database [10], [19]. Computationally intensive segment in the similarity search problem includes matrix operations, where matrices are essentially $\{0,1\}$ -valued adjacency matrices for the query and data graphs. Matrix computations are commonly grouped together in a set of programs termed, Basic Linear Algebra Subprograms (BLAS). BLAS is classified into three categories a) vector operations (Level-1), b) matrix-vector operations (Level-2), and c) matrix-matrix operations (Level-3). Vector operations are the most basic operations and the matrix-vector and matrix-matrix operations can be realized in-terms of vector operations. For realizing matrix-based operations, academic and industrial efforts have heavily focused on efficient implementation of BLAS [18], [12].

In-memory computing platforms are bringing in a paradigm shift by breaking away from traditional Von Neumann computing models. Multiple in-memory computation platforms have been proposed using ReRAMs. In [11], the authors

proposed a general purpose Programmable Logic-in-Memory (PLiM) architecture. Neuromorphic computing have also been realized using ReRAMs [7], [13], [15]. Furthermore, there have been multiple dedicated circuit proposals implemented using hybrid CMOS-ReRAM crossbar arrays [3], [2] along with design automation toolflow [6] to support in-memory computing using ReRAMs. A major advantage of in-memory computing is that it directly addresses the data bandwidth challenge, which is particularly prominent in data-intensive applications, such as those enabled by BLAS. Partly based on this motivation, in [17], integer matrix multiplication is presented, where the authors have proposed a distributed in-memory computing accelerator on ReRAM crossbar and reported significant performance benefits compared to CMOS-only realizations. However, given the prominent use-cases of binary matrix operations and the possibility to explore bit-level parallelism using ReRAM crossbar arrays, integer-level operations call for further optimization. Besides, authors in [17] essentially utilized the crossbar array as a Content-Addressable Memory (CAM) structure similar to those earlier proposed in [4] and did not utilize the stateful logic. To the best of our knowledge, there is no existing work that addresses the mapping of Binary BLAS (BiBLAS) using hybrid ReRAM crossbar arrays. In short, the major contributions of this paper are as follows.

- Efficient mapping of the Binary Level-1 BLAS on ReRAM crossbar array.
- Detailed analysis of the impact of crossbar and input vector dimensions on the overall performance is shown.
- We project speedup of 7x in terms of throughput compared to the recent GPGPU based realization of BiBLAS.

The rest of the paper is organized as follows. In section II, a brief introduction to Binary Level-1 BLAS and logic operation using ReRAM crossbar array is presented. In section III, efficient mapping schemes for multiple crossbar configurations are proposed and estimates of delay are obtained. In section IV, the proposed schemes are analyzed for the various crossbar sizes, with throughput and energy estimates. Finally, section V presents a conclusion to the paper.

II. PRELIMINARIES

A. Binary Level-1 BLAS Operations

For n -element vectors x and y , the inner product $p_{1,n}$ is defined as follows.

$$x = [a_1 \quad a_2 \quad \dots \quad a_n]$$

$$y = [b_1 \ b_2 \ \dots \ b_n]$$

$$p_{1,n} = x^T y = \sum_{i=1}^n a_i \cdot b_i \quad (1)$$

The vector elements are binary, i.e. $a_i, b_i \in \mathcal{B}\{0, 1\}$. Since we are operating on binary vectors, the multiplication is interpreted as Boolean AND and the sum is interpreted as Boolean XOR. Therefore, for binary inner product with n elements, n AND and $n - 1$ XOR operations are required. We represent Boolean AND, OR and XOR operations using \cdot , $+$ and \oplus symbols. We introduce intermediate variables, that are used during mapping onto ReRAM crossbar array. $p_{i,j}$ is used to represent the partial product of the elements i to j . $p_{1,n}$ is the inner product of two n -element vector.

$$c_{i,i+1} = a_i \cdot b_i \cdot (\overline{a_{i+1}} \cdot \overline{b_{i+1}}) \quad (2)$$

$$= (\overline{a_i} + \overline{b_i}) \cdot (\overline{a_{i+1}} + \overline{b_{i+1}}) \quad (3)$$

$$d_{i,i+1} = \overline{a_i \cdot b_i} \cdot a_{i+1} \cdot b_{i+1} \quad (4)$$

$$= (\overline{a_i} + \overline{b_i}) \cdot (\overline{a_{i+1}} + \overline{b_{i+1}}) \quad (5)$$

$$p_{i,j} = a_i \cdot b_i \oplus a_{i+1} \cdot b_{i+1} \oplus \dots \oplus a_j \cdot b_j \quad (6)$$

$$p_{i,j} = p_{i,k} \oplus p_{k+1,j} \quad (7)$$

$$p_{i,i+1} = c_{i,i+1} + d_{i,i+1} \quad (8)$$

B. Logic Operations using Crossbar Arrays

For logic operations, the ReRAM device can be seen as a Finite State Machine (FSM), as shown in Figure 1. The device has two input lines— wordline wl and bitline bl , and the internal state S stored in the device acts as the third input. The device computes $M_3(S, wl, \overline{bl})$, where M_3 is the 3-input Boolean majority function and this value is available stored as the next state of the device. Using this inherent function, we can realize the Boolean functions as shown below.

$$\overline{a} = M_3(0, 1, \overline{a}) \text{ // NOT}$$

$$a \cdot b = M_3(a, b, \overline{1}) \text{ // AND}$$

$$a + b = M_3(a, b, \overline{0}) \text{ // OR}$$

Since the bitline input is inverted in the inherent function

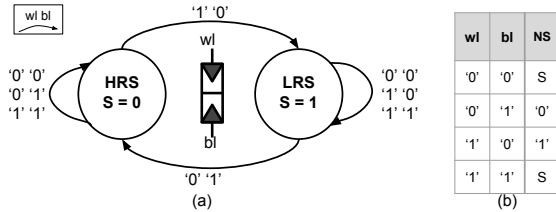


Fig. 1. ReRAM device logic operation realizing Majority. computed by ReRAM device, wordline and bitline inputs are not symmetric and hence cannot be interchanged. For further details of the device characteristics, kindly refer to [9], [8], [21]. Multiple devices can be organized as a crossbar array, which share common wordlines and bitlines. For example, the Figure 2 (a) shows a crossbar with a common wordline and four bitlines. We represent a 1×4

crossbar in Figure 2 (b), with 4 devices $C_{1,i}$, $i \in \{1, 2, 3, 4\}$ and one wordline wl and four bitlines bl_j , $j \in \{1, 2, 3, 4\}$. To perform logic operations, a CMOS control unit which coordinates and addresses the wordlines and bitlines, enables free communication across the lines, as shown in Figure 2 (c).

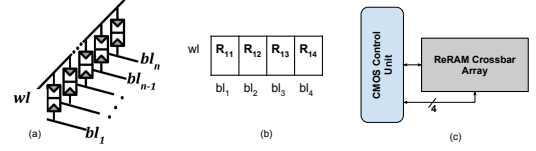


Fig. 2. ReRAM crossbar array with one wordline and four bitlines

In accordance with ReRAM device operations and state-of-the-art circuit design studies [9], [3], [11], we assume the following constraints related to logic operation using ReRAM crossbar array:

- C_1 : There is a single ReRAM crossbar array for logic operation. The operand vectors are stored in separate arrays, and the values are assumed to be available whenever required.
- C_2 : To read a device state, '1' and '0' has to be applied to the wordline and bitline of the device and the read out value is available in the next cycle.
- C_3 : In a single cycle, all devices with a common wordline can be read out simultaneously. Devices in that do not share wordlines cannot be read simultaneously.
- C_4 : A read out value can be applied to any wordline or bitline and it is also possible to apply it to multiple wordlines and bitlines simultaneously.

Constraint C_1 is imposed due to the fact that if the control unit has to coordinate and pass data between multiple crossbar arrays, then the control unit would itself significantly contribute to the performance overhead. If a value has to be read out and applied to a line in the same cycle, the control unit would need to operate at a higher frequency leading to high power consumption, and hence constraints C_2 is enforced. C_3 follow from the fact that there is only read port available per bitline and due to the array characteristics, enabling multiple wordlines for read simultaneously would result in correct read out values. Constraint C_4 is fundamental to enabling stateful logic using ReRAM crossbar arrays. Under these constraints, we design efficient mapping schemes for binary vector multiplication for various array configurations and subsequently study overall performance trade-offs.

III. BiBLAS MAPPING FOR IN-MEMORY COMPUTATION

A. Mapping on $(3k+1) \times 1$ crossbar

In this subsection, we explain the mapping of 4-element vector multiplication of a crossbar of dimensions 4×1 , i.e. $k = 1$. The steps have also been shown in Table I, for ease of understanding. We present the general case of mapping n -element vector multiplication, on crossbar of dimensions $(3k + 1) \times 1$, in Figure 3, by means of a flowchart.

Computing partial product $p_{1,2}$

S_1 : In the first step, the elements a_1 and a_2 are loaded in R_{11} and R_{21} respectively.

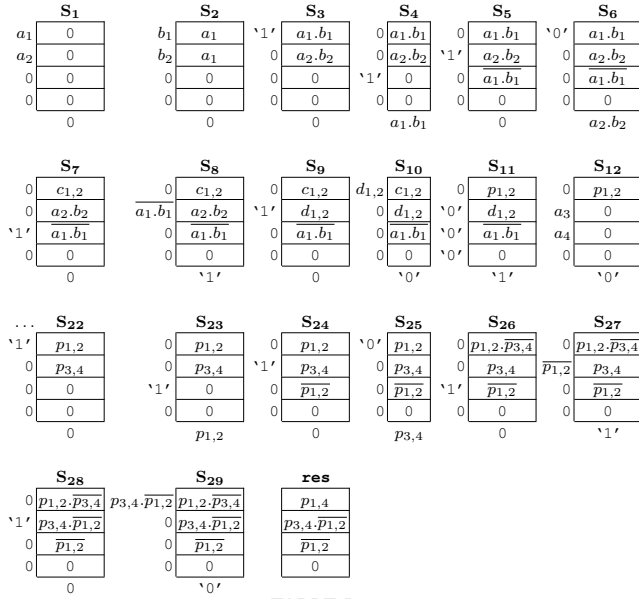


Fig. 3. Flowchart for n element vec. mul. on $(3k+1) \times 1$ crossbar.

i.e. $n = 2k$. The first two steps compute the AND of the $2k$ -vector elements. In the next, $2k$ steps, the inverted values of $a_i.b_i$ are stored, $i \in 1, 3, \dots, (2k-1)$. In the following $2k$ steps, the intermediate variable $d_{i,i+1}$ is computed, followed by computation of $c_{i,i+1}$ in the next $2k$ steps. Thereafter, another $2k$ steps are needed to OR $c_{i,i+1}$ with $d_{i,i+1}$ to compute the partial products $p_{i,i+1}$. One step is needed to reset the devices, that do not hold a partial product. Now, the crossbar has k partial products, which have to be XORed. From the instance of 4-element vector multiplication presented above, it can be seen that computing XOR of two partial products take 6 steps. Thus, it would take $6(k-1)$ steps to combine all the partial products, to obtain the partial product $p_{1,2k}$. If $n = 4k$, then we would have to reset the crossbar, except the device holding $p_{1,2k}$ and repeat all these steps again to obtain the partial product of the next $2k$ elements i.e. $p_{2k+1,4k}$, and XOR it with $p_{1,2k}$ to obtain the vector multiplication result— $p_{1,4k}$. Therefore, in general, the number of steps S that are required for multiplying n -element vector on $(3k+1) \times 1$ arrays is —

$$S = \lceil \frac{n}{2k} \rceil (14k - 2) + (\lceil \frac{n}{2k} \rceil - 1)6 \quad (9)$$

$2k$ devices are used for computation of $a_i.b_i$ and another k devices are necessary to store the $\overline{a_i.b_i}$ required for computation of $d_{i,i+1}$. The additional one device, is needed to hold the partial product from the previous round, when the computation of the current round is in progress. Due to the constraint C_3 , all the operations to compute $c_{i+1}, d_{i,i+1}$, XOR of partial products, have to be done sequentially, which contributes to the delay of the mapping solutions. In the following subsection, we propose a scheme of vector multiplication on $1 \times (2k+1)$ crossbar array, which has $2k+1$ devices sharing a common wordline that can be simultaneously read out, to mitigate the issue of sequential computation of $c_{i+1}, d_{i,i+1}$.

B. Mapping on $1 \times (2k+1)$ crossbar

Without loss of generality, we explain the steps of 4-element vector multiplication on a 1×3 crossbar array, i.e. $k = 1$. We present the generalized scheme for mapping n -element vector multiplication on $1 \times (2k+1)$ crossbar array, as flowchart in Figure 4.

Computing partial product $p_{1,2}$

S₁: Apply a_1 and a_2 to the bitlines of the device $R_{1,1}$ and $R_{1,2}$, to load the inverted values of the elements.

S₂ : Apply '1' to the wordline and b_1 and b_2 to the bitlines

TABLE I
BOOLEAN 4-ELEMENT VEC. MULT. ON $(3K+1) \times 1$, $K=1$ CROSSBAR ARRAY

S₂: a_i is ANDed with b_i , $i \in \{1, 2\}$.

S₃, S₄: $a_1.b_1$ is read out, and in the next step, applied to the bitline, to store the inverted value.

S₅, S₆: $a_2.b_2$ is read out, then applied to the bitline to compute the $a_1.b_1.a_2.b_2$, which is $c_{1,2}$.

S₇, S₈: $a_1.b_1$ is read out and ANDed with $a_2.b_2$ to compute $d_{1,2}$.

S₉, S₁₀: $d_{1,2}$ is read out and ORed with $c_{1,2}$. This is the result of partial product of multiplication of the first two elements of the vectors, and we represent it as $p_{1,2}$. **S₁₁ :** All the devices except the device holding $p_{1,2}$ is reset to 0, for computing the partial product of the next two elements of the vectors.

Computing partial product $p_{3,4}$

S₁₂ – S₂₁: Steps $S_1 - S_{11}$, are repeated, with a_i, b_i , $i \in \{3, 4\}$, using the devices $R_{21} - R_{51}$. At the end of these steps, $p_{3,4}$ is available in device R_{21} .

XORing partial products to get vector product

S₂₂, S₂₃: $p_{1,2}$ is read out, and in the next step, applied to the bitline, to store the inverted value.

S₂₄, S₂₅: $p_{3,4}$ is read out, then applied to the bitline, to compute $p_{1,2}.\overline{p_{3,4}}$ in device R_{11} .

S₂₆, S₂₇: $\overline{p_{1,2}}$ is read out and ANDed with $p_{3,4}$.

S₂₈, S₂₉ : In the final two steps, $\overline{p_{1,2}}.p_{3,4}$ is read out, and ORed with $p_{1,2}.\overline{p_{3,4}}$ to obtain the vector product $p_{1,4}$, available in device R_{11} .

Analysis of the mapping solution : Based on flowchart in Figure 3, the number of steps that are required by this scheme for multiplying n -element vector on $(3k+1) \times 1$ arrays can be computed. For sake of simplicity, we assume that n is a multiple of k . Otherwise, we can take the greatest integer k' for estimating the number of steps such that $k' \leq k$ and n is a multiple of k .

To multiply a $2k$ -element vector on a $(3k+1) \times 1$ crossbar,

of the device $R_{1,1}$ and $R_{1,2}$, to compute $\overline{a_i} + \overline{b_i}$.

S₃, S₄ : Read out the devices $R_{1,1}$ and $R_{1,2}$ and in the next clock cycle, apply '0' to the wordline with $\overline{a_1} + \overline{b_1}$ and $\overline{a_2} + \overline{b_2}$ to the bitlines of $R_{1,2}$ and $R_{1,1}$, to compute $c_{1,2}$ and $d_{1,2}$ respectively.

S₅, S₆ : Read out device $R_{1,2}$ which is holding $c_{1,2}$ and apply the read out value to the wordline and '0' to the bitline of device $R_{1,1}$ to compute the partial product $p_{1,2}$.

S₇ : Reset all devices to 0, other than $R_{1,2}$, which holds $p_{1,2}$.

Computing partial product $p_{3,4}$

S₈ – S₁₄ : Repeat the steps $S_1 - S_7$ with $a_i, b_i, i \in \{3, 4\}$, using the devices $R_{12} - R_{13}$. At the end of these steps, $p_{3,4}$ is available in device R_{12} .

XORing partial products to get vector product

S₁₅, S₁₆ : $p_{1,2}$ and $p_{3,4}$ is read out, and in the next step, '0' is applied to the wordline and $p_{1,2}$ and $p_{3,4}$ applied to the bitline of devices $R_{1,2}$ and $R_{3,4}$ respectively to compute the terms, $\overline{p_{1,2}} \cdot \overline{p_{3,4}}$ and $p_{1,2} \cdot \overline{p_{3,4}}$.

S₁₇, S₁₈ : In the final two steps, $\overline{p_{1,2}} \cdot \overline{p_{3,4}}$ is read out, and ORed with $p_{1,2} \cdot \overline{p_{3,4}}$ to obtain the vector product $p_{1,4}$, available in device R_{11} .

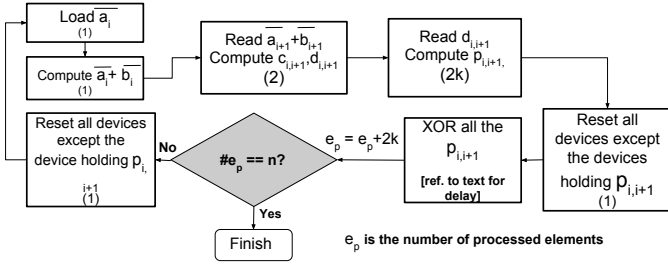


Fig. 4. Flowchart for n -element vec. mul. on $1 \times (2k+1)$ crossbar.

Analysis of the mapping solution : For $n = 2k$, in the first two steps, $\overline{a_i} + \overline{b_i}$ is computed for $2k$ -vector elements. In the next two steps, these values are read out and ANDed to obtain $c_{i,i+1}$ and $d_{i,i+1}$ terms. The computation of $c_{i,i+1}$ and $d_{i,i+1}$ terms is based on Equation 3 and 5 respectively. In the next $2k$ steps, $c_{i,i+1}$ is ORed with $d_{i,i+1}$ to compute partial product $p_{i,i+1}, i \in 1, 3, \dots, 2k-1$. In the next step, all the devices, other than the devices holding partial products, are reset.

Now, the crossbar has k partial products. In the next two steps, the adjacent partial products are read out and ANDed with each other. In the next $2k$ steps, $p_{i+2,i+3} \cdot \overline{p_{i,i+1}}$ is read out and ORed with $p_{i,i+1} \cdot \overline{p_{i+2,i+3}}$ to compute the partial product $p_{i,i+3}$. This set of steps have to be repeated, till we obtain the partial product $p_{1,2k}$. This computation is a XOR tree computation.

The above series of steps have to be repeated again for computing $p_{2k+1,4k}$. It has to be followed by XORing $p_{1,2k}$ and $p_{2k+1,4k}$ to obtain the final vector multiplication product $p_{1,4k}$. The XOR operation requires an additional 4 cycles, as evident from steps $S_{15} - S_{20}$.

For any n and k is a divisor of n , the number of steps S required for n element vector multiplication on $1 \times 2k+1$

crossbar array is —

$$S = \lceil \frac{n}{2k} \rceil (6 + \sum_{t=2, t'=t/2}^k (2 + 2t)) + (\lceil \frac{n}{2k} \rceil - 1)4 \quad (10)$$

In this scheme, $2k+1$ can be read out simultaneously, and hence improves the performance over the previous scheme for $(3k+1) \times 1$, both in terms of number of cycles and number of devices used. The reduction in k number of devices is due to the fact that the intermediate inverted values do not have to be stored in this case. The number of steps for XOR has also reduced from 6 to 4. However, even in this scheme, the operation for ORing $c_{i,i+1}$ and $d_{i,i+1}$, XOR operations required for combining the partial products have to be done sequentially. This can be averted by using a crossbar size of $2 \times (2k+1)$, for which the multiplication scheme is proposed in the next subsection.

C. Mapping on $2 \times (2k+1)$ crossbar

We demonstrate the mapping for multiplying 4-element vector using 2×5 crossbar array i.e. $k = 2$. The general scheme of mapping on $2 \times (2k+1)$ crossbar array is presented in Figure 5.

Computing partial product $p_{1,2}$ and $p_{3,4}$

S₁ : Apply $a_1 - a_4$ to the bitlines of the device $R_{1,1} - R_{1,4}$ respectively, to load the inverted values of the elements.

S₂ : Apply '1' to the wordline and $b_1 - b_4$ to the bitlines of the device $R_{1,1} - R_{1,4}$ respectively, to compute $\overline{a_i} + \overline{b_i}$.

S₃, S₄ : Read out the devices $R_{1,i}$ and in the next clock cycle, apply '0' to the wordline with content of $R_{1,i+1}$ to the bitline of device $R_{1,i}$ and content of $R_{1,i}$ to bitline of device of $R_{1,i+1}$, to compute $d_{i,i+1}$ and $c_{i,i+1}$ respectively, $i \in 1, 3$.

S₅, S₆ : Read out device $R_{1,j}$ which is holding $c_{j-1,j}$ and apply the read out value to the wordline and '0' to the bitline of device $R_{2,j}$ to compute the $\overline{c_{j-1,j}}, j \in 2, 4$.

S₇, S₈ : Read out device $R_{2,j}$ which is holding $c_{j-1,j}$ and apply the read out value to the bitline and '1' to the bitline of device $R_{1,j-1}$ to compute the $p_{j-1,j}, j \in 2, 4$.

XORing partial products to get vector product

S₉, S₁₀ : Read out $p_{1,2}$ and $p_{3,4}$, and in the next step, '0' is applied to the wordline and $p_{3,4}$ and $p_{1,2}$ applied to the bitline of devices $R_{1,1}$ and $R_{1,3}$ respectively to compute the terms, $p_{1,2} \cdot \overline{p_{3,4}}$ and $\overline{p_{1,2}} \cdot p_{3,4}$.

S₁₁, S₁₂ : In the final two steps, $\overline{p_{1,2}} \cdot p_{3,4}$ is read out, and ORed with $p_{1,2} \cdot \overline{p_{3,4}}$ to obtain the vector product $p_{1,4}$, available in device $R_{1,1}$.

Analysis of the mapping solution: Identical to the steps in the previous scheme, for $n = 2k$, in the first two steps, $\overline{a_i} + \overline{b_i}$ is computed for $2k$ -vector elements. In the next two steps, these values are read out and ANDed to obtain $c_{i,i+1}$ and $d_{i,i+1}$ terms. In the next 2 steps, $c_{i,i+1}$ is read out and stored in inverted form. This is followed by ORing with $d_{i,i+1}$ with $c_{i,i+1}$ by reading out $\overline{c_{i,i+1}}$ and applying it via the bitlines to compute partial product $p_{i,i+1}, i \in 1, 3, \dots, 2k-1$. In the next step, all the devices, other than the devices holding partial products, are reset.

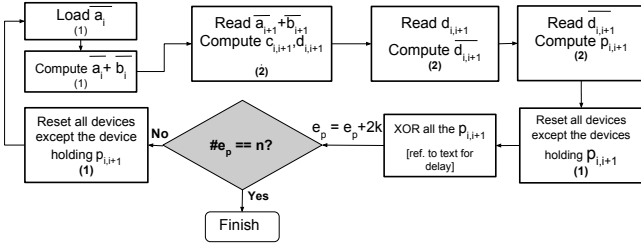
| | | | | | | |
|---|---|---|---|--|---|--|
| S_1 | S_2 | S_3 | S_4 | S_5 | S_6 | S_7 |
| $\begin{array}{ c c c } \hline 0 & 0 & 0 \\ \hline a_1 & a_2 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c } \hline \overline{a_1} & \overline{a_2} \\ \hline b_1 & b_1 \\ \hline 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline \overline{a_1+b_1} & \overline{a_2+b_2} & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline \overline{a_1+b_1} & \overline{a_2+b_2} & 0 \\ \hline \overline{a_2+b_2} & \overline{a_1+b_1} & 0 \\ \hline \end{array}$ | $\begin{array}{ c c } \hline d_{1,2} & c_{1,2} \\ \hline 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c } \hline d_{1,2} & c_{1,2} \\ \hline 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c } \hline p_{1,2} & c_{1,2} \\ \hline 0 & 0 \\ \hline \end{array}$ |
| S_8 | \dots | S_{14} | S_{15} | S_{16} | S_{17} | Res |
| $\begin{array}{ c c c } \hline p_{1,2} & 0 & 0 \\ \hline 0 & a_3 & a_4 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline p_{1,2} & p_{3,4} & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline p_{1,2} & p_{3,4} & 0 \\ \hline p_{3,4} & p_{1,2} & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline p_{1,2} \cdot \overline{p_{3,4}} & p_{3,4} \cdot \overline{p_{1,2}} & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline p_{3,4} \cdot \overline{p_{1,2}} & p_{1,2} \cdot \overline{p_{3,4}} & p_{3,4} \cdot \overline{p_{1,2}} \\ \hline 0 & 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline p_{1,2} \cdot \overline{p_{3,4}} & p_{3,4} \cdot \overline{p_{1,2}} & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c } \hline p_{1,4} & p_{3,4} \cdot \overline{p_{1,2}} & 0 \\ \hline \end{array}$ |

TABLE II

BOOLEAN 4-ELEMENT VECTOR MULTIPLICATION USING $1 \times (2K+1)$, $K=1$

| | | | | |
|---|---|---|---|---|
| S_1 | S_2 | S_3 | S_4 | S_5 |
| $\begin{array}{ c c c c } \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline a_1 & a_2 & a_2 & a_4 \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline \overline{a_1} & \overline{a_2} & \overline{a_3} & \overline{a_4} \\ \hline 0 & 0 & 0 & 0 \\ \hline b_1 & b_2 & b_3 & b_4 \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline \overline{a_1+b_1} & \overline{a_2+b_2} & \overline{a_3+b_3} & \overline{a_4+b_4} \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline \overline{a_1+b_1} & \overline{a_2+b_2} & \overline{a_3+b_3} & \overline{a_4+b_4} \\ \hline 0 & 0 & 0 & 0 \\ \hline \overline{a_2+b_2} & \overline{a_1+b_1} & \overline{a_4+b_4} & \overline{a_3+b_3} \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline d_{1,2} & c_{1,2} & d_{3,4} & c_{3,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$ |
| S_6 | S_7 | S_8 | S_9 | S_{10} |
| $\begin{array}{ c c c c } \hline d_{1,2} & c_{1,2} & d_{3,4} & c_{3,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & c_{1,2} & 0 & c_{3,4} \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline d_{1,2} & c_{1,2} & d_{3,4} & c_{3,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & c_{1,2} & 0 & c_{3,4} \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline d_{1,2} & c_{1,2} & d_{3,4} & c_{3,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline c_{1,2} & 0 & c_{3,4} & 0 \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline p_{1,2} & c_{1,2} & p_{3,4} & c_{3,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & c_{1,2} & 0 & c_{3,4} \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline p_{1,2} & c_{1,2} & p_{3,4} & c_{3,4} \\ \hline 0 & 0 & c_{1,2} & 0 \\ \hline 0 & c_{1,2} & 0 & c_{3,4} \\ \hline \end{array}$ |
| S_{11} | S_{12} | Res | | |
| $\begin{array}{ c c c c } \hline p_{1,2} \cdot \overline{p_{3,4}} & c_{1,2} & p_{3,4} \cdot \overline{p_{1,2}} & c_{3,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & c_{1,2} & 0 & c_{3,4} \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline p_{1,2} \cdot \overline{p_{3,4}} & c_{1,2} & p_{3,4} \cdot \overline{p_{1,2}} & c_{3,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & c_{1,2} & 0 & c_{3,4} \\ \hline \end{array}$ | $\begin{array}{ c c c c } \hline p_{1,2} \cdot \overline{p_{3,4}} & c_{1,2} & p_{3,4} \cdot \overline{p_{1,2}} & c_{3,4} \\ \hline 0 & c_{1,2} & 0 & c_{3,4} \\ \hline 0 & 0 & 0 & c_{3,4} \\ \hline \end{array}$ | | |

TABLE III

BOOLEAN 4-ELEMENT VECTOR MULTIPLICATION USING $2 \times (2K+1)$, $K=2$ Fig. 5. Flowchart for n -element vec. mul. on $2 \times (2k + 1)$ crossbar.

Now, the crossbar has k partial products. In the next two steps, the adjacent partial products are read out and ANDed with each other. In the next 2 steps, $p_{i+2,i+3} \cdot \overline{p_{i,i+1}}$ is read out and inverted by applying it via the bitlines. Thereafter, the inverted values $\overline{p_{i+2,i+3} \cdot \overline{p_{i,i+1}}}$ are read out and applied via the bitline to OR with $p_{i,i+1} \cdot \overline{p_{i+2,i+3}}$ to compute the partial product $p_{i,i+3}$. This set of steps have to be repeated, till we obtain the partial product $p_{1,2k}$. This computation is a XOR tree computation.

For $n = 4k$, the above series of steps have to be repeated again for computing $p_{2k+1,4k}$, after resetting the devices which do not hold the final partial product. It has to be followed by XORing $p_{1,2k}$ and $p_{2k+1,4k}$ to obtain the final vector multiplication product $p_{1,4k}$. The XOR operation requires an additional 4 cycles, as evident from steps $S_{15} - S_{20}$.

For any n and k is a divisor of n , the number of steps S required for n element vector multiplication on $2 \times 2k + 1$ crossbar array is —

$$S = \lceil \frac{n}{2k} \rceil \{10 + \sum_{t=2, t'=t/2}^{k/2} (7)\} + (\lceil \frac{n}{2k} \rceil - 1)4 \quad (11)$$

From equation 11, we can observe that due to the additional wordline available and using it to store the inverted values, the computation of OR step of multiple XOR operations can be done in parallel and in constant number of steps. This helps in substantial reduction in number of steps, compared to the previous schemes.

IV. EXPERIMENTAL RESULTS

We estimate the performance of Binary Level-1 BLAS on hybrid ReRAM crossbar array. In Figure 6, the impact on delay (in terms of number of steps) for a fixed array size ($k = 20$), on increasing the vector length is presented. As expected, with increase in vector size, the delay increases. In Figure 7, for a fixed vector length ($n = 1024$), with increase in the array sizes, the delay decreases. As expected from the analysis of the schemes, the vector multiplication using $2 \times (2k + 1)$ crossbar dimensions performs the best, in comparison to the other two schemes, independent of the vector size n . Due to the constraint C_3 , increasing the number of wordlines beyond would not help in reducing delay significantly, and hence we settle for $2 \times (2k + 1)$ configuration which has two wordline.

To estimate the throughput, we assume a mature ReRAM technology based on ITRS [1], with a read/write time of $1ns$. For large vector length of $n = 2^{20}$, the performance of the schemes for multiple values of k , in terms of throughput (Gbits/s) is shown in Figure 8. Since the scheme for $2 \times (3k + 1)$ crossbar does all the operations in parallel during operation, the throughput grows with increase in crossbar size, as expected. Beyond $k = 9$, the throughput saturates due to lack of sufficient data to process. For the other two schemes, due to multiple sequential operations, the throughput is limited, with increase in crossbar size. We estimate a maximum achievable throughput of **14.75 Gbit/s**. Direct implementation of BiBLAS has been not reported so far.

There are no known implementations of BiBLAS available to benchmark against our scheme. By representing binary number as single precision floating point number, it is estimated that GPGPUs can achieve a throughput of 2 Gbits/s [16][20]. This highlights that proposed scheme in-memory computing scheme promises a **7×** speed up in terms of throughput, compared to the BiBLAS operation on GPGPUs.

V. CONCLUSION

In this work, efficient mapping of BLAS subroutines for binary vector on ReRAM crossbar arrays is proposed. We explored multiple configurations for the crossbar array under

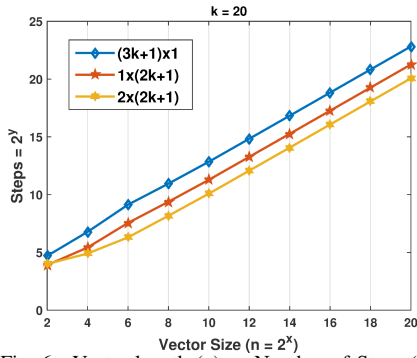


Fig. 6. Vector length (n) vs Number of Steps (S)

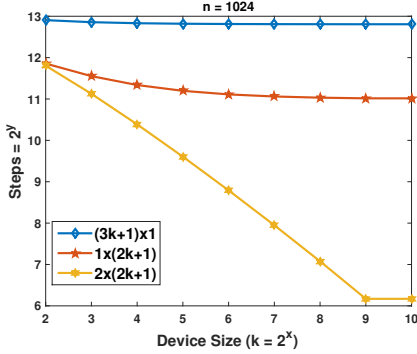


Fig. 7. Number of devices (k) vs Number of Steps (S)

practical design constraints and demonstrated performance trade-offs while varying the vector dimensions. The obtained performance results offer promising improvement over CMOS-only implementations. In future, we plan to extend this work in two major directions. First, we plan to look into level-2 and level-3 BLAS kernels. Second, the demonstration for a complete application flow using this kernels and benchmarking against comparable CMOS implementations is on our roadmap.

REFERENCES

- [1] Emerging Research Devices (ERD) report, International Technology Roadmap for Semiconductors (ITRS), 2013.
- [2] A. Siemon, S. Menzel, A. Chattopadhyay, R. Waser and E. Linn. In-memory adder functionality in 1s1r arrays. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pages 1338–1341, May 2015.
- [3] A. Siemon, S. Menzel, R. Waser and E. Linn. A complementary resistive switch-based crossbar array adder. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(1):64–74, March 2015.
- [4] F. Alibart, T. Sherwood, and D. B. Strukov. Hybrid cmos/nanodevice circuits for high throughput pattern matching applications. In *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, pages 279–286, 2011.
- [5] G. V. Bard. Achieving a log(n) speed up for boolean matrix operations and calculating the complexity of the dense linear algebra step of algebraic stream cipher attacks and of integer factorization methods. Cryptology ePrint Archive, Report 2006/163, 2006. <http://eprint.iacr.org/>.
- [6] D. Bhattacharjee and A. Chattopadhyay. Delay-optimal technology mapping for in-memory computing using ReRAM devices. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD, 2016*.
- [7] D. B. Strukov, D.R. Stewart, J. Borghetti, X. Li, M. Pickett, G. M. Ribeiro, W. Robinett, G. Snider, J. P. Strachan, W. Wu, Q. Xia, J. J. Yang and R. S. Williams. Hybrid cmos/memristor circuits. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1967–1970, 2010.

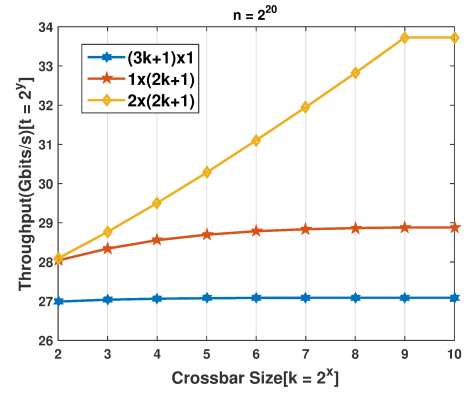


Fig. 8. Crossbar size (k) vs Throughput (Gbit/s)

- [8] E. Linn, R. Rosezin, C. Kügeler and R. Waser. Complementary resistive switches for passive nanocrossbar memories. *Nature Letters*, 9:403–406, 2010.
- [9] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger and R. Waser. Beyond von neumann-logic operations in passive crossbar arrays alongside memory operations. *Nanotechnology*, 23(30), 2012.
- [10] Z. Fan, B. Choi, Q. Chen, J. Xu, H. Hu, and S. S. Bhowmick. Structure-preserving subgraph query services. *IEEE Trans. Knowl. Data Eng.*, 27(8):2275–2290, 2015.
- [11] P. Gaillardon, L. Amaru, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. D. Micheli. The programmable logic-in-memory (plim) computer. In *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 427–432, 2016.
- [12] Intel. Intel Math Kernel Library (Intel MKL). <https://software.intel.com/en-us/intel-mkl>. [Online; accessed 05-May-2016].
- [13] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa and W. Lu. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano Letters*, 12(1):389–395, 2011.
- [14] T. Li. A general model for clustering binary data. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05*, pages 188–197, 2005.
- [15] M. P. Sah, H. Kim and L. O. Chua. Brains are made of memristors. *IEEE Circuits and Systems Magazine*, 14(1), 2014.
- [16] F. Merchant, A. Maity, M. Mahadurkar, K. Vattwani, I. Munje, M. K. C, N. Sivanandan, N. Gopalan, S. Raha, S. K. Nandy, and R. Narayan. Micro-architectural enhancements in distributed memory cgras for LU and QR factorizations. In *28th International Conference on VLSI Design, VLSID 2015, Bangalore, India, January 3-7, 2015*, pages 153–158, 2015.
- [17] L. Ni, Y. Wang, H. Yu, W. Yang, C. Weng, and J. Zhao. An energy-efficient matrix multiplication accelerator by distributed in-memory computing on binary rram crossbar. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 280–285, Jan 2016.
- [18] nVIDIA. cuBLAS. <https://developer.nvidia.com/cublas>. [Online; accessed 05-May-2016].
- [19] Y. Peng, Z. Fan, B. Choi, J. Xu, and S. S. Bhowmick. Authenticated subgraph similarity search in outsourced graph databases. *IEEE Trans. Knowl. Data Eng.*, 27(7):1838–1860, 2015.
- [20] Z. E. Rákossy, F. Merchant, A. A. Aponte, S. K. Nandy, and A. Chattopadhyay. Scalable and energy-efficient reconfigurable accelerator for column-wise givens rotation. In *22nd International Conference on Very Large Scale Integration, VLSI-SoC, Playa del Carmen, Mexico, October 6-8, 2014*, pages 1–6, 2014.
- [21] A. Siemon, S. Menzel, A. Marchewka, Y. Nishi, R. Waser, and E. Linn. Simulation of TaO_x-based complementary resistive switches by a physics-based memristive model. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 1420–1423. IEEE, 2014.
- [22] S. T. Xia, X. J. Liu, Y. Jiang, and H. T. Zheng. Deterministic constructions of binary measurement matrices from finite geometry. *IEEE Transactions on Signal Processing*, 63(4):1017–1029, Feb 2015.
- [23] Z. Zhang, T. Li, C. Ding, and X. Zhang. Binary matrix factorization with applications. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining, ICDM '07*, pages 391–400, 2007.