Parallel Boolean Matrix Multiplication in Linear Time using Rectifying Memristors

Alvaro Velasquez
Department of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL
Email: velasquez@eecs.ucf.edu

Sumit Kumar Jha
Department of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL
Email: http://www.sumitkumarjha.com/contact.html

***Abstract*—Boolean matrix multiplication (BMM) is a fundamental problem with applications in graph theory, group testing, data compression, and digital signal processing (DSP). The search for efficient BMM algorithms has produced several fast, albeit impractical, algorithms with sub-cubic time complexity. In this paper, we propose a memristor-crossbar framework for computing BMM at the hardware level in linear time. Our design leverages the diode-like characteristics of recently studied rectifying memristors to resolve the pervasive sneak paths constraint that is ubiquitous in crossbar computing.**

I. Introduction

In the past few years, memristors [1], [2] have become highly regarded due to their diminutive size, great data retention rate, non-volatility, and power efficiency when compared to traditional transistors. Furthermore, these novel devices allow processing and memory access to take place on the same chip, thus ameliorating the memory transfer delay experienced in traditional von Neumann architectures [3]. These desirable traits have fueled a spark of interest in the research community that has led to significant effort towards exploiting these properties in the form of nanoscale memory arrays [4], neuromorphic computing paradigms [5], [6], [7], and digital logic operations [8], [9], [10] among others.

In most applications, memristors are arranged as a crossbar consisting of two sets of perpedicular wires with a memristor placed at each junction of wires. In this paper, we focus our attention on crossbars consisting of the recently studied rectifying memristors [11] and how they can be used to compute the product of Boolean matrices – a fundamental problem in computer science – in linear time. This problem is of interest for various reasons. It is an integral part of group testing [12] and its applications in genetics, data forensics, fault diagnosis, and on-chip sensing. Boolean matrix multiplication has also been studied in the context of matrix decomposition [13], cryptography [14], and CFG parsing [15], [16].

Multiplying two $n \times n$ Boolean matrices has been shown to be computable in $O(n^3/p + \log(p/n^2))$ time on a hypercube with $n^2 \leq p \leq n^3$ processors. This leads to the fastest runtime of $O(n)$ with n^2 processors [17]. Using this processor array model of parallel computation, we utilize n^2 processing elements in order to solve the Boolean matrix multiplication problem in linear time on a reconfigurable rectifying-memristor crossbar. In this case, each memristive device can be viewed as a processing element.

The paper is organized as follows. Section II provides background information on the memristor and the behavior of sneak currents in crossbar computing. The Boolean matrix multiplication problem is defined in Section III. We demonstrate how this problem can be mapped to a crossbar and computed in linear time in Section IV. We conclude in Section V with final remarks.

II. Memristors and Sneak Paths

In the literature, the typical memristor consists of a film containing a region doped with ions and an undoped region [2]. The doped region yields a low resistance R_{ON} while the undoped region poses a much higher resistance R_{OFF}. The boundary separating the two regions drifts in the presence of an electric field generated by applying a voltage bias. The direction of the drift depends on the direction of the current flowing through the memristor. Thus, a memristor can be seen as two resistors in series whose total resistance, or memristence R_{MEM}, is a function of the total charge applied across the memristor.

The normalized state parameter of a memristor, denoted by $x \in [0, 1]$, represents the ratio of the width of the doped region with respect to the entire width of the memristor. Consequently, the larger the state of a memristor, the lower its memristence is, and vice versa. In this paper, we deal with memristors in one of the two terminal states such that $x \in \{0, 1\}$ and $R_{MEM} \in \{R_{ON}, R_{OFF}\}$. Under these circumstances, memristors act as switches, where a turned-off memristor ($x = 0$) does not allow a significant amount of current to flow and a turned-on ($x = 1$) memristor does.

We base our designs on the rectifying memristors studied in [11], which have the capability to drastically suppress negative voltages smaller than some specified magnitude. As such, the flow of current through these devices is unidirectional. Similar diode-like structures has been utilized in RRAM circuits and programmable logic arrays (PLAs), but it has been shown that the rectifying capability of diodes deteriorates considerably at the nanoscale [18], thus necessitating a nanoscale diode alternative.

A pervasive issue that arises when memristors are placed in a crossbar is the emergence of sneak paths. Sneak paths are trajectories formed in a memristor crossbar due to the redirection of current caused by turned-on memristors. This

978-1-4799-5341-7/16/$31.00 ©2016 IEEE

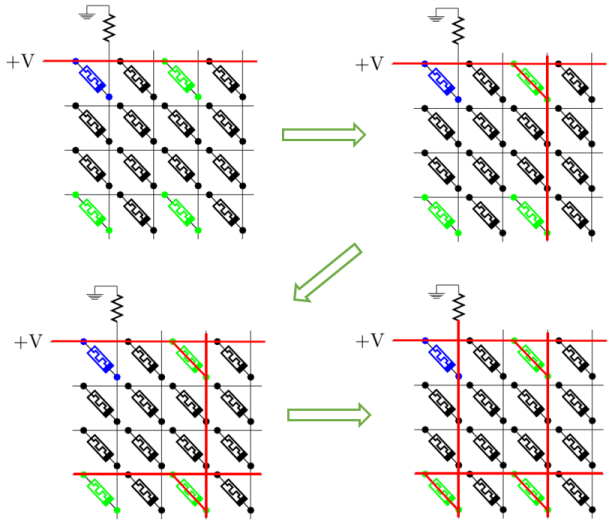

Fig. 1: Flow behavior of sneak paths, where green nodes denote turned-on memristors and black nodes represent turned-off memristors. The red bars represent flow of current and we wish to read the value of the blue memristor by applying a voltage at one terminal and grounding the other. It can be seen that a high voltage value will be read at the grounded terminal due to sneak paths regardless of the state of the blue memristor.

can lead to erroneous memristor-state readings due to the possibly many paths that a flow of current can travel through, see Fig. 1 for an example. The use of rectifying memristors to mitigate this crosstalk has been studied in [11], and can be used to resolve this sneak currents issue in applications such as ours by allowing current to flow in only one direction as opposed to the bidirectional flow of traditional memristors. We define these sneak path behaviors in Fig. 2.

The notion of a crossbar can be formalized as follows.

Definition 1. CROSSBAR *A crossbar is a 3-tuple* $\mathbb{C} = (\mathbb{M}, \mathbb{V}_r, \mathbb{V}_c)$ *where*

- $\mathbb{M} = \begin{pmatrix} M_{11} & M_{12} & \dots & M_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1} & M_{m2} & \dots & M_{mn} \end{pmatrix}$ *is a two-dimensional array of memristors with* m *rows and* n *columns, where* $M_{ij} \in \{0,1\}$ *denotes the state of the device connecting row* i *with column* j*;*
- $\mathbb{V}_r = \{V_{r_1}, \dots, V_{r_m}\}$ *is the set of horizontal nanowires such that wire* V_{r_i} *provides the same input voltage to every memristor in row* i*.*
- $\mathbb{V}_c = \{V_{c_1}, \dots, V_{c_n}\}$ *is the set of vertical nanowires such that wire* V_{c_j} *provides the same input voltage to every memristor in column* j*.*

We define a binary flow function $f : \mathbb{V}_r \cup \mathbb{V}_c \mapsto \{0,1\}$ such that $f(w) = 1$ denotes a significant flow of current on wire w and $f(w) = 0$ denotes negligible flow. Without loss of generality, we assume unidirectional flow of current from the column wires to the row wires due to the rectifying property of the crossbar's constituent devices. This idea is formalized in axiom 1, where $\wedge$ denotes the conjunction operator and $\implies$ represents logical implication. See Fig. 2 for a pictorial representation.

Axiom 1 (Unidirectional Flow)**.** *Let* $\mathbb{C} = (\mathbb{M}, \mathbb{V}_r, \mathbb{V}_c)$ *be an* $n \times n$ *crossbar. Then*

$$\forall i, j, 1 \le i, j \le n : (M_{ij} \wedge f(V_{c_j})) \implies f(V_{r_i}) \quad (1)$$

$$\forall i, \exists j, 1 \le i, j \le n : f(V_{r_i}) \implies (M_{ij} \wedge f(V_{c_j})) \quad (2)$$

The reconfiguration of each M_{ij} is made possible by the presence of a threshold voltage V_{TH} that must be exceeded in order to switch the device. In accordance to the rectifying memristor studied in [11], it is the case that the activation voltage threshold necessary to switch a node from the OFF to the ON state is less than the reset voltage threshold V'_{TH} required to switch back to the OFF state.

III. Problem Definition

Given two Boolean matrices $A = (a_{ij}) \in \{0,1\}^{n \times n}$ and $B = (b_{ij}) \in \{0,1\}^{n \times n}$, we wish to compute their product $C = (c_{ij}) = AB$, where $c_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$. For simplicity, we assume square matrices; however, the method described is applicable to matrices of arbitrary dimensions.

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nn} \end{pmatrix}$$

$$C = \begin{pmatrix} \bigvee_{i=1}^{n} (a_{1i} \wedge b_{i1}) & \bigvee_{i=1}^{n} (a_{1i} \wedge b_{i2}) & \dots & \bigvee_{i=1}^{n} (a_{1i} \wedge b_{in}) \\ \bigvee_{i=1}^{n} (a_{2i} \wedge b_{i1}) & \bigvee_{i=1}^{n} (a_{2i} \wedge b_{i2}) & \dots & \bigvee_{i=1}^{n} (a_{2i} \wedge b_{in}) \\ \vdots & \vdots & \ddots & \vdots \\ \bigvee_{i=1}^{n} (a_{ni} \wedge b_{i1}) & \bigvee_{i=1}^{n} (a_{ni} \wedge b_{i2}) & \dots & \bigvee_{i=1}^{n} (a_{ni} \wedge b_{in}) \end{pmatrix}$$

Efficient Boolean matrix multiplication algorithms have been well-studied in the literature, with several sub-cubic candidates proposed by Strassen and Coppersmith, among others [16]. However, the constants involved with these methods makes them inefficient in practice, with Strassens method requiring matrix sizes of over 100 before it is more efficient than the traditional cubic-time algorithm [16].

An interesting result that ties the BMM problem with grammar parsing was reported by Valiant [15]. In this work, Dr. Valiant claims that the asymptotically fastest Context-Free Grammar (CFG) parsing algorithm is entirely dependent on the runtime of a BMM algorithm. This relationship has been studied in detail in [16] and [15], and it is concluded that fast CFG parsing requires the existence of a fast BMM algorithm. We propose the latter in this paper by demonstrating how Boolean matrix multiplication can be computed at the hardware level in linear time. Such a result has the potential to improve the efficiency of CFG parsing and facilitate its application as an embedded system.

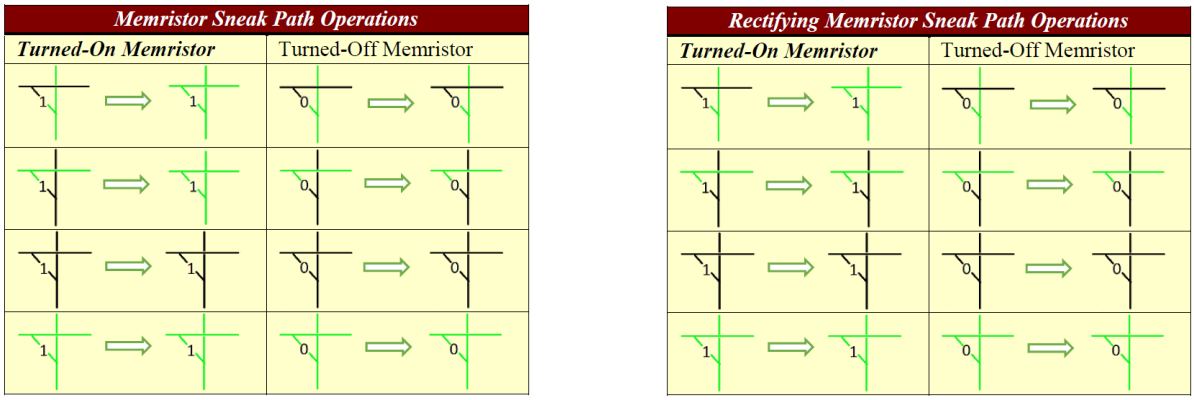

Fig. 2: Traditional memristive devices allow electric current to flow to and from either terminal. This bidirectional flow behavior is depicted on the left, where a 1 denotes a turned-on device and a 0 denotes a turned-off device. The green bars represent flow of current and the black bars denote a lack thereof. The unidirectional flow of rectifying memristors can be seen on the right. Note that current only flows from the terminal connected to the vertical wire to that of the horizontal wire.

IV. Methodology

In order to compute the product of Boolean matrices, we present a mapping of the Boolean matrix multiplication problem to a crossbar of rectifying memristors. This mapping necessitates a method to configure said crossbar to and from any state such that arbitrary problem formulations can be computed. We begin by demonstrating our configuration procedure, followed by the Boolean matrix multiplication mapping.

It is worth noting that rectifying memristors require a negative voltage drop of greater magnitude than V_P, the programming voltage used to switch a memristor from the OFF to the ON state, in order to turn off the device. Experiments conducted in [11], wherein $V_P = 2V$, demonstrate that applying a voltage drop equal to $-V_P$ across the rectifying memristor did not cause it to turn off. A significantly larger voltage of $-3.5V$ was required to switch the state of the memristor from the ON state to the OFF state. It follows that a rectifying memristor-crossbar can be set to an all-zeroes state (i.e. every memristor turned off) by grounding every column wire and applying a large voltage of approximately $V'_P = 2V_P$ on every row wire, where V'_P denotes the reset programming voltage used to switch a rectifying memristor from the ON state to the OFF state. A crossbar can also be programmed to a specified state by applying the method specified in the following lemma.

Lemma 1 (Crossbar Configuration)**.** *Let $\mathbb{C} = (\mathbb{M}, \mathbb{V}_r, \mathbb{V}_c)$ be an $n \times n$ crossbar. Then $\mathbb{M}$ can be configured to state $\mathbb{M}'$ in $n+1$ steps.*

Proof. Let V'_P, V_P, V_{LO}, and V_0 be a sequence of decreasing voltages such that $V_P - V_0 > V_{TH}$, $V_P - V_{LO} < V_{TH}$, $V_{LO} < V_{TH}$, and $|V'_P - V_0| > |V'_{TH}|$, where V_{TH} is the activation threshold voltage and V'_{TH} is the reset threshold voltage, which is often negative. We begin by setting all $V_{c_k} = V_0$ and $V_{r_i} = V'_P$, thereby resetting every M_{ij}. From this state, we can configure the j^{th} column vector of $\mathbb{M}$ by setting $V_{c_j} = V_P$ and $V_{r_i} = \begin{cases} V_0 & \text{if } \mathbb{M}'_{ij} = 1 \\ V_{LO} & \text{if } \mathbb{M}'_{ij} = 0 \end{cases}$. To preserve the state of the k^{th} column vector $\mathbb{M}_{:k}$, $k \neq j$, let $V_{c_k} = V_Z$, where V_Z is a high-impedance voltage. Repeating this n times (once for each column) yields $\mathbb{M}'$. □

Theorem 1 (Linear Time Boolean Matrix Multiplication)**.** *Let $A = (a_{ij}) \in \{0,1\}^{n\times n}$, $B = (b_{ij}) \in \{0,1\}^{n\times n}$, and $C = AB \in \{0,1\}^{n\times n}$. Then C can be computed using an $n \times n$ crossbar in $O(n)$ time steps.*

Proof. Let $\mathbb{C} = (\mathbb{M}, \mathbb{V}_r, \mathbb{V}_c)$ be an $n \times n$ crossbar. Note that $\mathbb{M}$ can be configured to state B^T in $n+1$ computations by Lemma 1. This implies that $M_{ij} = b_{ji}$. Let $V_{c_j}^{(i)} = \begin{cases} V_{LO} & \text{if } a_{ij} = 1 \\ V_Z & \text{if } a_{ij} = 0 \end{cases}$ and ground all $V_{r_k}^{(i)}$. It follows that $a_{ij} = f(V_{c_j}^{(i)})$. Thus, $(a_{ij} \wedge b_{jk}) \implies (f(V_{c_j}^{(i)}) \wedge M_{kj})$. From (1), we know that $(f(V_{c_j}^{(i)}) \wedge M_{kj}) \implies f(V_{r_k}^{(i)})$. By the transitive property, $(a_{ij} \wedge b_{jk}) \implies f(V_{r_k}^{(i)})$. From (2), it follows that $f(V_{r_k}^{(i)}) \implies \bigvee_{j=1}^{n}(f(V_{c_j}^{(i)}) \wedge M_{kj})$ and we know that $(M_{kj} \wedge f(V_{c_j}^{(i)})) \implies (b_{jk} \wedge a_{ij})$; thus, $f(V_{r_k}^{(i)}) \implies \bigvee_{j=1}^{n}(a_{ij} \wedge b_{jk})$. This results in a corollary of (1) and (2), where $f(V_{r_k}^{(i)}) = 1$ iff there exists some j, $1 \leq j \leq n$, such that $(a_{ij} \wedge b_{jk}) = 1$. It follows that $f(V_{r_k}^{(i)}) = c_{ik} = \bigvee_{j=1}^{n}(a_{ij} \wedge b_{jk})$. Thus, a row vector $c_{i:}$ of the product matrix C can be calculated at each time step, allowing us to calculate C in n time steps once the crossbar has been configured. We assume that a constant amount of time is required to store these values in memory. □

We demonstrate our approach by computing a sample problem. Let us define the following matrices:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \; B = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \; C = AB = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

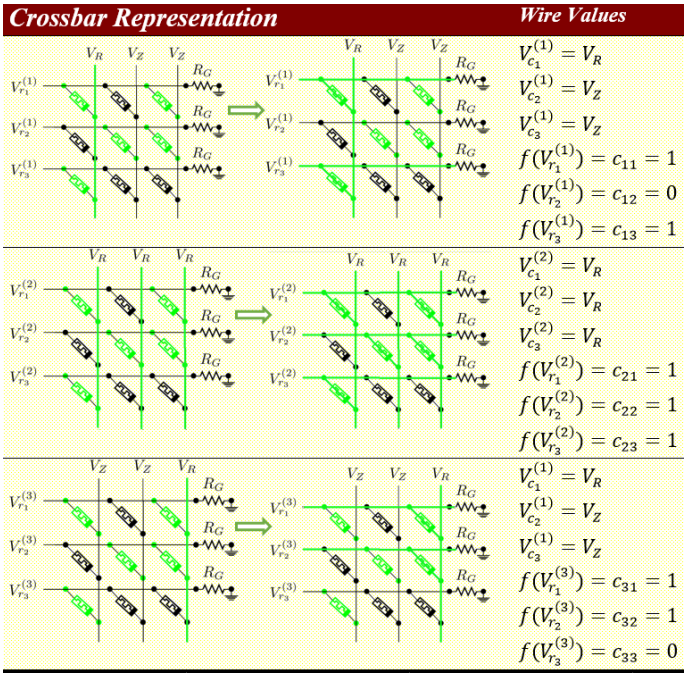

Fig. 3: Procedure for computing $C = AB$ in our example, where $V_R = V_{LO}$ denotes a read voltage. Entries c_{11}, c_{12}, and c_{13} are calculated at $t = 1$. Similarly, row vectors $c_{2:}$ and $c_{3:}$ are computed at $t = 2$ and $t = 3$, respectively.

Let $\mathbb{C} = (\mathbb{M}, \mathbb{V}_r, \mathbb{V}_c)$ denote a 3×3 crossbar such that $\mathbb{M}$ is in some arbitrary state. We first reset the crossbar to state $\mathbb{M}^{(0)}$ by letting all $V_{c_j} = V_0$ and $V_{r_i} = V_P'$. We wish to configure the crossbar such that $\mathbb{M} = B^T$. We configure the column vector $\mathbb{M}_{:1} = b_{1:}^T$ by letting $V_{c_1} = V_P$, $V_{r_1} = V_{r_3} = V_0$, $V_{r_2} = V_{LO}$, and $V_{c_2} = V_{c_3} = V_Z$. We can then configure $\mathbb{M}_{:2} = b_{2:}^T$ by letting $V_{c_2} = V_P$, $V_{r_2} = V_0$, $V_{r_1} = V_{r_3} = V_{LO}$, and $V_{c_1} = V_{c_3} = V_Z$. Finally, $\mathbb{M}_{:3} = b_{3:}^T$ can be configured by letting $V_{c_3} = V_P$, $V_{r_1} = V_{r_2} = V_0$, $V_{r_3} = V_{LO}$, and $V_{c_1} = V_{c_2} = V_Z$.

$$\mathbb{M}^{(0)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \mathbb{M}^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$

$$\mathbb{M}^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad \mathbb{M}^{(3)} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} = B^T$$

Once configured, the sequence of inputs described in Theorem 2 allows us to determine the values of the entries in the product matrix C in 3 time steps as shown in Fig. 3. We begin by grounding every V_{r_i} and applying V_{LO} on V_{c_1}, leading to $f(V_{r_1}) = c_{11} = f(V_{r_3}) = c_{13} = 1$ and $f(V_{r_2}) = c_{12} = 0$. The second row vector of C is computed by applying V_{LO} on V_{c_1}, V_{c_2}, and V_{c_3}. This causes $f(V_{r_1}) = c_{21} = f(V_{r_2}) = c_{22} = f(V_{r_3}) = c_{23} = 1$. Finally, we apply apply V_{LO} on V_{c_3}, which results in $f(V_{r_1}) = c_{31} = f(V_{r_2}) = c_{32} = 1$ and $f(V_{r_3}) = c_{33} = 0$. Thus, we have demonstrated how Boolean matrix multiplication can be computed on an $n \times n$ rectifying-memristor crossbar in $O(n)$ time.

V. Conclusion

We have demonstrated how the fundamental problem of Boolean matrix multiplication can be computed in a time- and space-efficient manner using a crossbar of rectifying memristors. The method proposed leverages the unidirectional flow of electric current throughout the crossbar in order to mitigate the infamous crosstalk problem that is so pervasive in crossbar memories. Consequently, we have shown how this allows us to compute the product of Boolean matrices in linear time using a quadratic number of processing elements.

References

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

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

[3] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Memristor-based material implication (imply) logic: Design principles and methodologies. 2013.

[4] Intel PR. Intel and micron produce breakthrough memory technology. `newsroom.intel.com/community/intelnewsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology`. Accessed: 28 July, 2015.

[5] Miao Hu, Hai Li, Yiran Chen, Qing Wu, Garrett S Rose, and Richard W Linderman. Memristor crossbar-based neuromorphic computing system: A case study.

[6] Chenchen Liu, Bonan Yan, Chaofei Yang, Linghao Song, Zheng Li, Beiye Liu, Yiran Chen, Hai Li, Qing Wu, and Hao Jiang. A spiking neuromorphic design with resistive crossbar. In *Proceedings of the 52nd Annual Design Automation Conference*, page 14. ACM, 2015.

[7] Zheng Li, Chenchen Liu, Yandan Wang, Bonan Yan, Chaofei Yang, Jianlei Yang, and Hai Li. An overview on memristor crossabr based neuromorphic circuit and architecture. In *Very Large Scale Integration (VLSI-SoC), 2015 IFIP/IEEE International Conference on*, pages 52–56. IEEE, 2015.

[8] Alvaro Velasquez and Sumit Kumar Jha. Fault-tolerant in-memory crossbar computing using quantified constraint solving. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 101–108. IEEE, 2015.

[9] Alvaro Velasquez and Sumit Kumar Jha. Automated synthesis of crossbars for nanoscale computing using formal methods. In *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pages 130–136. IEEE, 2015.

[10] Alvaro Velasquez and Sumit Kumar Jha. Parallel computing using memristive crossbar networks: Nullifying the processor-memory bottleneck. In *Design & Test Symposium (IDT), 2014 9th International*, pages 147–152. IEEE, 2014.

[11] Kuk-Hwan Kim, Sung Hyun Jo, Siddharth Gaba, and Wei Lu. Nanoscale resistive memory with intrinsic diode characteristics and long endurance. *Applied Physics Letters*, 96(5):053106, 2010.

[12] George K Atia and Venkatesh Saligrama. Boolean compressed sensing and noisy group testing. *Information Theory, IEEE Transactions on*, 58(3):1880–1901, 2012.

[13] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. Optimal boolean matrix decomposition: Application to role engineering. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 297–306. IEEE, 2008.

[14] Yeghisabet Alaverdyan and Gevorg Margarov. Fast asymmetric cryptosystem based on boolean product of matrices. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pages 392–395. IEEE, 2009.

[15] Leslie G Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, 10(2):308–315, 1975.

[16] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM (JACM)*, 49(1):1–15, 2002.

[17] Eliezer Dekel, David Nassimi, and Sartaj Sahni. Parallel matrix and graph algorithms. *SIAM Journal on computing*, 10(4):657–675, 1981.

[18] GDJ Smit, S Rogge, and TM Klapwijk. Scaling of nano-schottky-diodes. *arXiv preprint cond-mat/0202401*, 2002.