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 perpendicular 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 memristance \( R_{MEM} \), 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 memristance 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
Definition 1. A crossbar is a 3-tuple $\mathbb{C} = (M, V_r, V_c)$ where

- $M = \begin{pmatrix} M_{11} & M_{12} & \cdots & M_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1} & M_{m2} & \cdots & M_{mn} \end{pmatrix}$ 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 $j$;
- $V_r = \{V_{r1}, \ldots, V_{rm}\}$ is the set of horizontal nanowires such that wire $V_{ri}$ provides the same input voltage to every memristor in row $i$;
- $V_c = \{V_{c1}, \ldots, V_{cn}\}$ is the set of vertical nanowires such that wire $V_{cj}$ provides the same input voltage to every memristor in column $j$.

We define a binary flow function $f : V_r \cup V_c \mapsto \{0, 1\}$ 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 $\land$ denotes the conjunction operator and $\implies$ represents logical implication. See Fig. 2 for a pictorial representation.

Axiom 1 (Unidirectional Flow). Let $\mathbb{C} = (M, V_r, V_c)$ be an $n \times n$ crossbar. Then

$$\forall i, j, 1 \leq i, j \leq n : (M_{ij} \land f(V_{ci})) \implies f(V_{ri})$$

(1)

$$\forall i, \exists j, 1 \leq i, j \leq n : f(V_{ri}) \implies (M_{ij} \land f(V_{cj}))$$

(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 $c_{ij} = \bigvee_{k=1}^n (a_{ik} \land b_{kj})$. For simplicity, we assume square matrices; however, the method described is applicable to matrices of arbitrary dimensions.

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

$$C = \begin{pmatrix} \bigvee_{i=1}^n (a_{1i} \land b_{1i}) & \bigvee_{i=1}^n (a_{1i} \land b_{2i}) & \cdots & \bigvee_{i=1}^n (a_{1i} \land b_{ni}) \\ \vdots & \vdots & \ddots & \vdots \\ \bigvee_{i=1}^n (a_{ni} \land b_{1i}) & \bigvee_{i=1}^n (a_{ni} \land b_{2i}) & \cdots & \bigvee_{i=1}^n (a_{ni} \land b_{ni}) \end{pmatrix}$$

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.
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 $V_{cz} = V_0$ and $V_{ri} = V_p$, thereby resetting every $M_{ij}$. From this state, we can configure the $j^{th}$ column vector of $M$ by setting $V_{cz} = V_p$ and $V_{ri} = \begin{cases} V_0 & \text{if } M_{ij} = 1 \\ V_{LO} & \text{if } M_{ij} = 0 \end{cases}$. To preserve the state of the $k^{th}$ column vector $M_{k,j}$, $k \neq j$, let $V_{cz} = V_Z$, where $V_Z$ is a high-impedance voltage. Repeating this $n$ times (once for each column) yields $M'$.

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 $C = (M, V_r, V_e)$ be an $n \times n$ crossbar. Note that $M$ can be configured to state $B^T$ in $n + 1$ computations by Lemma 1. This implies that $M_{ij} = b_{ji}$. Let $V_{cz} = \begin{cases} V_{LO} & \text{if } a_{ij} = 1 \\ V_Z & \text{if } a_{ij} = 0 \end{cases}$ and ground all $V_{cz}^{(i)}$. It follows that $a_{ij} = f(V_{cz}^{(i)})$. Thus, $(a_{ij} \land b_{jk}) \implies (f(V_{cz}^{(i)}) \land M_{kj})$. From (1), we know that $\left( f(V_{cz}^{(i)}) \land M_{kj} \right) \implies f(V_{cz}^{(j)})$. By the transitive property, $(a_{ij} \land b_{jk}) \implies f(V_{cz}^{(j)})$. From (2), it follows that $f(V_{cz}^{(i)}) \implies \bigvee_{j=1}^{n} (f(V_{cz}^{(i)}) \land M_{kj})$ and we know that $(M_{kj} \land f(V_{cz}^{(i)})) \implies (b_{jk} \land a_{ij})$; thus, $f(V_{cz}^{(i)}) \implies \bigvee_{j=1}^{n} (a_{ij} \land b_{jk})$. This results in a corollary of (1) and (2), where $f(V_{cz}^{(i)}) = 1$ iff there exists some $j$, $1 \leq j \leq n$, such that $(a_{ij} \land b_{jk}) = 1$. It follows that $f(V_{cz}^{(i)}) = c_{ik} = \bigvee_{j=1}^{n} (a_{ij} \land b_{jk})$. 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:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \quad C = AB = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$
Let $C = (M_r, \mathcal{V}_r, \mathcal{V}_c)$ denote a $3 \times 3$ crossbar such that $M_r$ is in some arbitrary state. We first reset the crossbar to state $M_r^{(0)}$ by letting all $V_{r_1} = V_0$ and $V_{r_2} = V_p$. We wish to configure the crossbar such that $M_r = B^T$. We configure the column vector $M_{r:1} = b_{1}^T$ by letting $V_{c_1} = V_P$, $V_{r_1} = V_{r_2} = V_0$, $V_{r_2} = V_{LO}$, and $V_{c_2} = V_{c_3} = V_Z$. We can then configure $M_{r:2} = b_{2}^T$ by letting $V_{c_2} = V_P$, $V_{r_1} = V_{r_2} = V_0$, $V_{c_3} = V_{r_3} = V_{LO}$, and $V_{c_1} = V_{c_3} = V_Z$. Finally, $M_{r:3} = b_{3}^T$ can be configured by letting $V_{c_3} = V_P$, $V_{r_1} = V_{r_2} = V_0$, $V_{c_3} = V_{LO}$, and $V_{c_1} = V_{c_2} = V_Z$.

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 $V_{r_1}$ and applying $V_{LO}$ on $V_{c_1}$, leading to $f(V_{r_1}) = c_{11} = f(V_{r_2}) = c_{12} = 1$ and $f(V_{r_3}) = c_{13} = 0$. 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 $V_{LO}$ 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