Matrix-to-Vector Multiplication Leveraging RRAM Crossbar

Matheus Farias[1]
[1]Harvard School of Engineering & Applied Sciences (SEAS), Cambridge, MA, USA.
matheusfarias@g.harvard.edu

***Abstract*—** Matrix-to-Vector multiplications (MVM) are basis for several modern applications such as neural networks, computer vision and digital signal processing. In this work, I design, with XFAB 130nm PDK technology, an accelerator for 3x3 MVM based on the Resistive RAM (RRAM) crossbar architecture. Here we provide an explanation on how to program RRAM to perform MVM**.** This work enables room for simulating the architecture and evaluate future benefits and advances on it.

I. Introduction

After many years of the proposal of Von Neumann's architecture for computers [1], the technology used for designing processors had a huge improvement, however, memories did not follow the same trend for performance purposes [2]. In this architecture, the processor must acquire data from the memory which is physically located outside the processor's chip. Thus, even though the processors are fast, as we always need to acquire data from the memory, we face an overhead called Von Neumann's bottleneck. See in Fig. 1 a comparison of different operations for a CPU designed with 45nm CMOS technology. To overcome this situation, researchers are studying specific-domain architectures that can leverage processing in memory to fasten operations and decrease the energy consumption such as the Resistive RAM (RRAM) crossbar architecture [4]. In this work I study how to use this architecture to perform MVM in the analog domain as a faster alternative way compared to CPU.

II. Background and Methodology

Here I consider a specific application of multiplying a vector $a \in \{0,1\}_{1x3}$ to a matrix $B \in R_{3x3}$. To perform this multiplication in a RRAM crossbar architecture, we need to define multiply-and-accumulate operations, which in this case are done by using Kirchhoff and Ohm's laws of circuits analysis theory as shown in Fig. 2. First, I consider a case that each cell in the crossbar corresponds to a single and fixed resistor to shown that the architecture works as expected, we call this arrangement as 1R (see Fig. 3). Then, a next step is to replicate the model now using cells composed of one transistor and one resistor (1T1R cells). This way we can control the current that flows through the cells by activating/deactivating the transistor depending on the difference between the gate and source voltage. Finally, I use cells with 1T1R in parallel with the RRAM alongside with a transistor to activate/deactivate the current to this path. RRAM is a non-volatile memory based on memristors (resistors that have memory properties). We can control the resistance of these resistors by sending voltage pulses to the cell (see Fig. 4 and 5).

III. Model and Simulation

To design these three circuits, I used XFAB 130nm PDK (xr013). For the 1R cells, I used resistors of 1 kΩ, 2 kΩ and 3 kΩ (ranging from 500 nm to 1.2 µm of width/length). For the 1T1R cells, I considered NMOS transistors with total width of 4 µm and length 120 nm. Instead of using only one resistor for each cell, I considered a model of 4-bit digital resistors, which means that we can sweep over 16 possible resistance configurations by turning the transistors ON/OFF for each group of resistors (see Fig. 6). This was made so that we could perform MVM with different matrix *B* and knowing exactly what is going to be the resistance of each cell. Finally, I considered the model attaching a RRAM (see Fig. 7 for the parameters of the RRAM model used). The simulations were made in Spectre and Python.

III. Key Results

For the 1R cell architecture, I considered the MVM in Fig. 8a. To get results for the digital resistor 1T1R architecture, I used two cycles, the first is to get the resistance for each wire, considering the fixed resistor of 2 kΩ and the transistor (see Fig. 8b). For the RRAM we have two steps: program and multiplication. For the programming, we have to activate the transistor to enable the wire that has the RRAM, then we send a triangular pulse to trigger the lower resistance state. The amount of time we let that wire activated with a constant input voltage, after the triangular pulse, determines the resistance (Fig. 5). After this, we turn off the wire and do the same procedure for all the cells, finishing the programming. Now, we just need to input the voltages we are considering for the input vector, just making sure they are not so big that can influence the resistance we configured previously. The layout for the design considering the last cell (with RRAM in parallel to 1T1R with digital resistances) is shown in Fig. 9.

IV. Conclusion

This work demonstrated how it is possible to perform MVM in the analog domain. The code and simulation is provided in a GitHub repository [6] so that future researchers can use it. These processing in memory architectures have a bottleneck of analog to digital conversion and I expect this work to support the scientific community to simulate new strategies to overcome this overhead.

Acknowledgments

I would like to thank Prof. Gage Hills for providing the RRAM model as well as for our insightful discussions.

References

[1] J. von Neumann, "First draft of a report on the EDVAC," in IEEE Annals of the History of Computing.
[2] Mutlu, O. et al. (2020). A Modern Primer on Processing in Memory.
[3] Mark Horowitz. Energy table for 45nm process, Stanford VLSI wiki.
[4] I. Chakraborty et al., "Resistive Crossbars as Approximate Hardware Building Blocks for Machine Learning: Opportunities and Challenges," in Proceedings of the IEEE, vol. 108, no. 12, pp. 2276-2310, Dec. 2020.
[5] Work repository. https://github.com/matheussfarias/cs248

Operation (32 bits)	Energy [pJ]	Relative Cost
int ADD	0.1	1
float ADD	0.9	9
Register File	1	10
int MULT	3.1	31
float MULT	3.7	37
SRAM Cache Access	5	50
DRAM Memory Access	**640**	**6400**

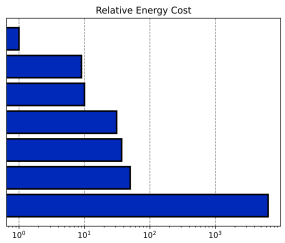


Figure 1 (up): Comparison between operations in a CMOS 45 nm technology-based computer [3]

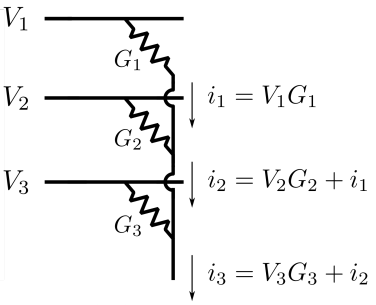


Figure 2 (up): The current is calculated by Ohm's Law, and then it is summed after each node by Kirchhoff's Law.

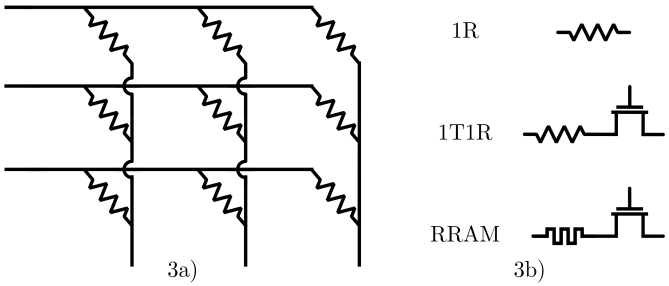


Figure 3 (up): a) Crossbar architecture b) Elements in each cell for different configurations

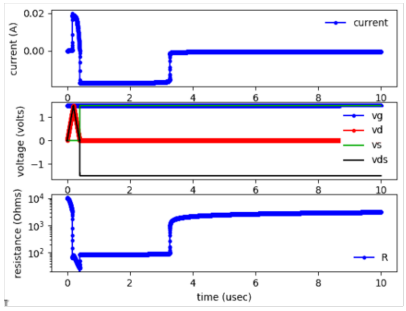


Figure 5 (up): Depending on the input voltage of the pulse and also the pulse width, we can select the exact resistance we want to choose for our memristor.

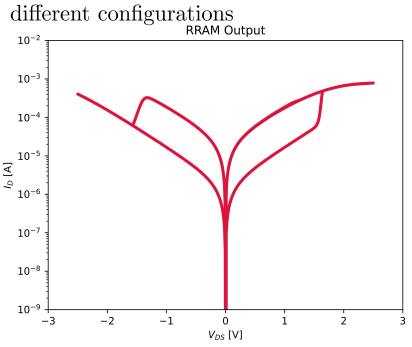


Figure 4 (left): By sweeping the input voltage, we can get across orders of magnitude for the output current. This happens because internally we are changing the resistance in our memristor. Note that the RRAM behavior has two bottom legs that correspond to the region of high resistances and two upper legs that correspond to the region of low resistances.

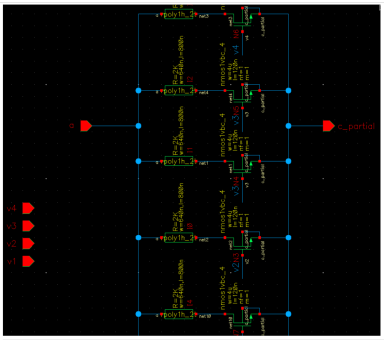


Figure 6 (left): The digital resistor model, note that the screenshot only shows 5 pairs of transistor and resistor, however the whole model has 15 so that we can represent 4 bits. See file digital_res_4bits_nmos in the repository for more details.

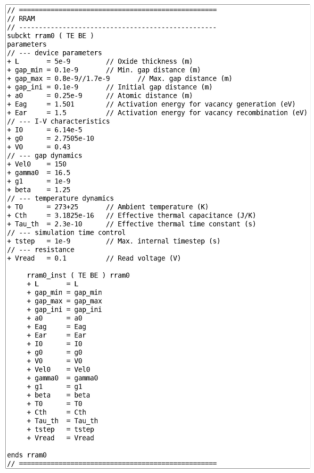
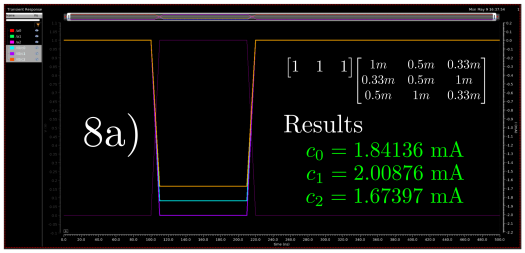
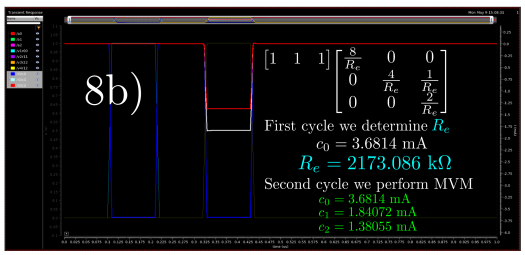


Figure 7 (up): The RRAM model used for simulation

Figure 8 (bottom left): Simulation for a) 1R and b) 1T1R with digital resistor archit.ectures





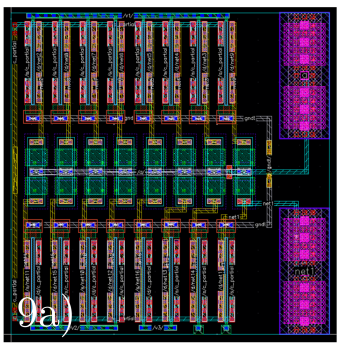


Figure 9 (up): a) The layout for a single cell in the 3x3 crossbar grid (15.45 um x 15.29 um) b) The layout for the full crossbar architecture with top-layer access for RRAM to be attached (48.35 um x 52.21 um)