SySCIM: SystemC-AMS Simulation of Memristive Computation In-Memory

Seyed Hossein Hashemi Shadmehri[1*], Ali BanaGozar[2*], Mehdi Kamal[1],
Sander Stuijk[2], Ali Afzali-Kusha[1], Massoud Pedram[3], Henk Corporaal[2]
[1] School of Electrical and Computer Engineering, University of Tehran
[2] Electronic Systems Group, Eindhoven University of Technology
[3] Department of Electrical Engineering, University of Southern California
{seyed.hashemi.ho, mehdikamal, afzali}@ut.ac.ir, {a.banagozar, s.stuijk, h.corporaal}@tue.nl, pedram@usc.edu

***Abstract*—Computation-in-memory (CIM) is one of the most appealing computing paradigms, especially for implementing artificial neural networks. Non-volatile memories like ReRAMs, PCMs, etc., have proven to be promising candidates for the realization of CIM processors. However, these devices and their driving circuits are subject to non-idealities. This paper presents a comprehensive platform, named SysCIM, for simulating memristor-based CIM systems. SySCIM considers the impact of the non-idealities of the CIM components, including memristor device, memristor crossbar (interconnects), analog-to-digital converter, and transimpedance amplifier, on the vector-matrix multiplication performed by the CIM unit. The CIM modules are described in SystemC and SystemC-AMS to reach a higher simulation speed while maintaining high simulation accuracy. Experiments under different crossbar sizes show SySCIM performs simulations up to 117× faster than HSPICE with less than 4% accuracy loss. The modular design of SySCIM provides researchers with an easy design-space exploration tool to investigate the effects of various non-idealities.**

***Index Terms*—simulation, computation-in-memory, memristor, reliability**

I. Introduction

The growing interest in artificial neural networks (ANNs), along with their deployments in many applications on computing platforms with energy constraints, have made the energy optimization of their implementations a vital design objective. Conventional processors based on the von Neumann architecture are not efficient enough for such data-intensive applications due to the significant cost of data movement between separated computing and memory units. Computation in-memory (CIM) alleviates this problem by bringing the computation to the data, and reducing the costly data movement. This makes computation-in-memory a very appealing paradigm, especially for implementing ANNs. CIM can be realized by slightly modifying the existing DRAM, and SRAM by exploiting 3D-stacked memories [1], or by employing emerging non volatile memory technologies such as memristors [2]. Our focus in this work is on memristor-based CIM, one of the most promising approaches for CIM implementation. By organizing memristors in a crossbar structure, one may realize vector-matrix multiplication (VMM), the core operation of ANNs, in a highly efficient fashion on memristor-based CIM. These devices and their driving circuits, however, are subject to non-idealities. More specifically, memristive devices, suffer from non-idealities like Device-to-Device (D2D), Cycle-to-Cycle (C2C) variation, etc. Such non-idealities severely degrade the computation accuracy of these platforms. In addition, the impacts of the process variations (PV) on the peripheral circuitry, required for driving memristor crossbars, and parasitic resistances further degrade the system accuracy. Therefore, for the design of these systems, one may need to study these impacts thoroughly. There have been various attempts at devising a high-level simulation framework for memristive computation in memory [3]–[7]. As mentioned in [4], this task faces many challenges. The circuit-level design has not yet reached a point where there is a single preferable choice and designers need to conduct experiments at this level. Therefore, frameworks should accommodate new designs effortlessly. In addition, estimating computation accuracy, which is the most critical aspect of such frameworks requires capturing the true behavior of a complex system, incorporating many non-ideal factors. This has been our motivation in this work to devise the CIM simulator called SySCIM. It provides the ability to explore the vast design space at several abstraction levels. For instance, while one can use SySCIM to easily investigate various memristor technologies, others can study different driving units and the corresponding non-idealities. Moreover, the design of SySCIM allows its integration into a data-path aware architecture, like the Transport Triggered Architecture (TTA) in [3], or as an accelerator in more commercialized architectures, like RISC-V in [8]. In this work, like in [3], we have integrated SySCIM into the TTA [9], using the TTA-based Co-Design Environment (TCE) [10]. Considering that for the simulation of a memristive CIM platform both analog and digital domains shall be taken into account, we employ the combination of SystemC and SystemC-AMS for developing SySCIM. The key contributions of this work are as follows:

- Presenting SySCIM, a comprehensive CIM simulator that is developed in SystemC and SystemC-AMS. It considers non-idealities for various components, required for accurate memristive CIM design space exploration (DSE).
- Integrating SySCIM into a TTA, which enables detailed simulation of CIM platforms, at the system level. This not only allows designers to observe the effects of non-idealities at higher abstraction levels, but it also enables them to study the interaction of the CIM unit with other modules, like arithmetic-logic unit (ALU), direct memory access unit (DMA), etc.
- Studying the efficacy of SySCIM in comparison to HSPICE in terms of simulation speed and accuracy. Based on the experiments, SySCIM reduces the simulation time

* equal contribution

TABLE I: Summary of the features of memristor modeling and simulation platforms.

Reference	**MNSIM** [4]	**Neurosim** [5]	**DNN+Neurosim** [6]	**MemTorch** [7]	**SySCIM**
Device Model	✓	✓	✓	✓	✓
Parasitics	✓	✓	✓	✓	✓
Variations	✓	✓	✓	✓	✓
Faulty Devices	-	-	-	✓	✓
Data Retention	-	-	✓	-	-
Non-ideal ADC	-	-	-	-	✓
TIA Gain Error	-	-	-	-	✓
Full Fledged System	-	-	-	-	✓
Development Language	unstated	C++	Python, C++	Python	SystemC, SystemC-AMS

by 117× while maintaining the simulation accuracy within 96% of HSPICE accuarcy.

- Employing SySCIM to demonstrate the impact of several non-idealities, including device variations, parasitic resistances, transimpedance amplifier (TIA) fluctuations, etc., on the VMM accuracy performed by the CIM unit.

In the rest of the paper, we first review the platforms developed to study memristive CIMs (Section II). In Section III, we recap on the basics of memristor-based CIM and introduce SySCIM in detail. In Section IV, the results for some DSEs using SySCIM are illustrated. Section V concludes this paper.

II. Related Work

In this section, the prior works on the high-level modeling of memrisitve CIM are briefly reviewed. MNSIM [4], which proposes a coarse-grained accuracy estimation model for memristor crossbars, addresses the need for reconfigurability by abstracting design choices into parameters stored in a configuration file. Although architectural simulators like PRIME [11] and ISAAC [12], are powerful tools to study higher-level implications of memristive CIM, they lack the ability to consider various non-idealities mentioned in the previous section. Neurosim [5] addresses the limitations of architectural simulators and also by considering the output error of different parts of a neuromorphic system and not only the weighted sum, it provides more accurate simulation results than MNSIM. Using a python wrapper, the framework proposed in [6] improves the effectiveness of Neurosim by interfacing it with some well-known machine learning frameworks, such as Tensorflow and PyTorch. MemTorch [7] is similar to [6], with greater emphasis on the modeling of random non-ideal behavior. Although the implications of memristor crossbar behavior have been widely studied at circuit and device levels, the effects on a full-fledged system need more attention [3]. Thus, in [3], the authors supplemented a TTA processor with a memristive CIM unit, which is developed in C++ leading to the possibility of simulating the whole system. However, this description is abstract and does not consider non-ideal effects. Table I summarizes the important features of the related works, and illustrates how SySCIM differentiates from them.

III. SySCIM

Memristors store the information as their conductance state. Arranging them at cross-points in a crossbar structure, makes it possible to implement highly demanded operations, like VMM, in a hugely efficient and significantly parallel fashion. This is performed by mapping the vector and the matrix values in a VMM to voltages ($\vec{V}$) and the conductance states (conductance matrix M), respectively. Leveraging Ohm's law and Kirchhoff's current summation law, VMM is performed by applying V to the Source-Lines (SL), causing the currents to flow through the Bit-Lines (BL) of the crossbar producing the products of the multiplications. The general structure of the considered CIM unit in the proposed simulator is illustrated in Fig. 1 with the described crossbar at its center. For writing data on the memristor crossbar, the data is first stored in the Write Data (WD) register. Then, using the Digital Input Modulators (DIMs), all the memristors in a single row are programmed. The Row Select (RS) register enables the target row while disabling all others. This procedure is repeated until all necessary cells in the crossbar are written. SySCIM supports two common writing methods, namely open-loop and write-and-verify [13], [14]. In the computation phase, first, the input vector is stored in the RS register. Then, it is applied to the crossbar via the DIMs. After the VMM operation is done in the analog domain, TIAs convert the accumulated currents of the columns to voltages which are then sampled by sample and hold (S&H) units. S&H units then feed the voltages to shared ADC(s), sequentially. After the proper shift and add operations, the results are ready to be written to the output register. In both phases, programming the crossbar and computation, the controller orchestrates the flow of the data (see Fig. 1) and issues the necessary commands based on the instruction received from the host and its internal state. To configure the specifications of SySCIM, like weight/input/output width, number of ADCs, etc., a configuration header file is provided. The controller reads the configuration file and manages the operation accordingly. For example, to simulate the execution of an 8-bit VMM on 4-bit memristors (the case considered in this work), the controller splits 8-bit data into two 4-bit values and stores 4-MSBs and 4-LSBs on adjacent columns. In order to implement

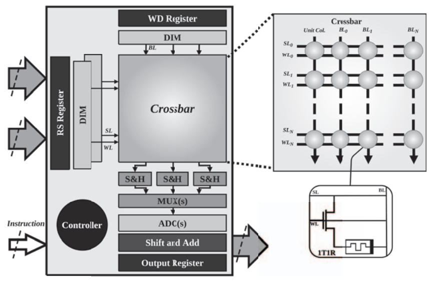


Fig. 1: A view of the employed CIM unit, the crossbar structure, and 1-transistor-1-memristor schematic [3].

the signed arithmetic (similar to [12]), the least significant 4-bit part is considered as an unsigned integer and stored in its corresponding memristor. However, the most significant 4-bit part is considered as a signed integer (ranging from -8 to +7) and added with a bias of 8, then stored in its corresponding memristor. In the SySCIM crossbar, one additional column is considered in which memristors have equal values of 8 (in unsigned interpretation). This Column (referred to as the unit column, see Fig. 1) calculates the bias added to the columns containing the most significant 4-bits, and thus its result is subtracted from them. In this structure, the inputs are injected to the crossbar one bit at a time, and thus, shift and subtract operation is performed on the MSB instead of the usual shift and add operation [12]. In this way the need for a digital-to-analog converter (DAC) is avoided. Hence, an N$\times$2N SySCIM crossbar could store an N$\times$N matrix of 8-bit entries. More information regarding the controller can be found in [3]. The rest of this section delineates how each part of the CIM unit is developed and how its non-idealities are incorporated.

A. Memristor

Several device models for capturing the behaviors of memristors with decent accuracy have been presented in the literature. The memristor generalized model of [15] has 11 fitting parameters to mimic the behavior of fabricated devices. The considered CIM structure comprises analog and digital parts. The existing memristor models, however, are developed for continuous-time simulations; thus, they are not compatible with the event-driven simulations used for the digital parts. To address this, we implement the memristor generalized model using the SystemC-AMS timed data flow (TDF) model. This allows us to perform continuous-time simulations for the crossbar and link the results to the event-driven simulation for the rest of the system. Although the electrical-linear network (ELN) model of computation can be used as well, we decided not to use it since it performs poorly in terms of simulation speed.

Memristors suffer from D2D and C2C variations. While the C2C variation impacts the function of the device only during the programming phase, D2D variation affects the device both during the read and write processes [16], [17]. To study the impact of different PVs on the memristor devices, the user of SySCIM can define random and systematic variations for the memristor fitting parameters. In this work, for the random variation, the user can define a specific distribution formula for modeling the uncertainty of each fitting parameter. For the systematic variation, we implement it based on the multi-level spatial quad-tree method introduced in [18]. In this method, the area of the crossbar is divided into regions based on a multi-level quad-tree partitioning. Each level of the quad-tree consists of 4^{LN} squares (LN is the level number) and each square is associated with a random number from a normal distribution with zero mean and user-defined standard deviation. Ultimately, the sum of the numbers of the squares at various levels covering a single device determines its variation. Since the adjacent devices are placed in more common regions at the different levels, their variations have more correlation as it should be for systematic variation.

Among the 11 fitting parameters of the employed generalized memristor, two of them (i.e., Ap and An) impact the precision of the written data in the memristor more than the others. These two parameters control how fast the conductance of the device changes when a SET or RESET pulse is applied. Thus, applying uncertainty to these parameters has a substantial impact on the accuracy of the computations and should therefore be captured more precisely in SySCIM. To use SySCIM without a thorough knowledge of the device properties, we also develop a behavioral simulation model (in addition to the TDF model). For this, the memristor is modeled at a higher abstraction level where the user only defines the maximum and the minimum conductance values plus the number of the conductance states. Unlike the TDF model, programming this model does not require set and reset pulses, and values are passed as integers and stored as conductance values. As one would expect, the simulation speed is higher in this case. Finally, it is likely for some of the memristors in the crossbar to be unresponsive to programming pulses and get stuck at the high(low) resistance state, denoted by HRS(LRS) [16]. To study the impact of the stuck devices, the user could define a “fault-map” to determine the probability of each memristor being faulty, or let SySCIM generate a random one. Since the faulty memristors are determined at crossbar level in the simulator, we will detail it in the following subsection.

B. Crossbar

The crossbar is formed by the interconnection of memristors. Our SPICE simulations show that the access device variations have a negligible impact on the imprecision of the crossbar computation result. Hence, in SySCIM, we consider 1T1R cells with ideal switches as access devices [2]. To model the parasitics, we use the approach of [19], where crossbars are considered with perfect interconnect, and the parasitics resistances are taken into account by multiplying the current of each column with a coefficient that is specific to that column. The coefficient is proportional to the distance of a column from the point where the input voltages are applied. To simulate the stuck devices, SySCIM generates a random matrix of the size of the crossbar with normally distributed numbers, $N(\mu = 0.5, \sigma^2 = 0.0225)$. The devices whose corresponding figure in the random matrix is lower(higher) than a predefined low(high) threshold are considered stuck at HRS(LRS).

C. Peripheries

The employed peripheral units for the considered CIM structure (Fig. 1) are similar to those of other works in the literature, e.g. [3], [12], [20], [21]. As mentioned before, to eliminate the need for DACs, the input voltages are applied one bit at a time. TIAs are employed at the end of each column to convert the produced currents to a set of voltages. We performed HSPICE simulations which accounted for op-amp non-idealities, e.g., input resistance and offset voltage. The simulations showed that these non-idealities (within their reasonable magnitudes) do not significantly affect the computation accuracy. However, variations in the feedback resistors used with the op-amps to form the TIAs proved to be highly detrimental to the accuracy

TABLE II: SySCIM configuration

Parameter	Default Value
Memristor model Parameters	Based on device in [22]
Source-Line Resistance	5Ω
Bit-Line Resistance	5Ω
ADC resolution	10-bit (Ideal)

by changing the gain of the TIA. Hence, SySCIM can consider variation of the TIA gains. ADC(s), convert the analog voltages (stored by S&H circuits) to digital values. The required ADC resolution for converting the output voltage of each column of the crossbar was obtained by [12]:

$$Resolution = \begin{cases} \log(R) + v + w, & v > 1,\, w > 1 \\ \log(R) + v + w - 1, & \text{Otherwise} \end{cases} \quad (1)$$

where R is the number of rows, v is the number of input bits and w is the number of bits stored in memristors. ADC accuracy is vital for the accurate operation of memristive CIM. In ideal conditions, the ADC only adds quantization noise to its input, but in reality the adverse effects could be worse. To simulate the non-ideal transfer function of the ADC, SySCIM injects uncertainty to the ideal transition points in the ADC I/O relation. Transition points are values of the analog input voltage, for which a change in the output digital code occurs. To do this, the transition points are shifted by normally distributed random numbers with a zero mean and a variance that is a fraction of V_{LSB}[1] (for example, one-tenth). The static specification differential non-linearity (DNL) indicates the significance of the non-ideal behavior. DNLs show the deviation of code width (distance between transition points) from V_{LSB} for each output digital code. Since we are applying the inputs one bit at a time, shifts and adds are performed after each step. At the end, outputs are written to the output registers, ready to be read by the host processor. Finally, note that due to the higher robustness of the CMOS digital circuits in the presence of PV and the existence of many PV mitigation approaches for digital circuits, SySCIM does not consider the impact of PV on the digital circuits and on the output accuracy of the CIM structure.

IV. Results and Discussion

In this section, we present a number of experiments using SySCIM to showcase its capabilities in analysing different scenarios. Table II shows the default values for important parameters in SySCIM in the following experiments.

A. *SySCIM Speedup and Accuracy*

SySCIM simulates the behavior of memristor-based CIM at a higher abstraction level than that of HSPICE. Thus, the simulation speed is higher while the accuracy lower. By exploiting the models described in Section III, we managed to keep the accuracy loss of SySCIM negligible. To measure the speedup of SySCIM versus HSPICE and compare their accuracy, we use SySCIM and HSPICE to investigate the effect of A_p and A_n as well as parasitic resistance variations on the VMM results. For this study, we utilize the open-loop writing method described earlier; hence, there is no way of compensating for A_p and A_n variations. Using a multi-level quad-tree, we also implement the spatial correlation of A_p and A_n variations (systematic variation). We perform this study for crossbars with the sizes of 8×8 to 128×128. Also, we analyze the effect of variations by running the simulation 1,000 times for each crossbar size. Then, we repeat these simulations with HSPICE using 45nm CMOS technology for the access devices and a realistic op-amp model for the TIAs. Fig. 2 shows the difference between run-times of our work and those of HSPICE. As shown, the simulation time difference surges up to approximately 117x. In each run, 4 different vectors were sequentially applied to the crossbar. Considering the output values for columns obtained by HSPICE simulations as accurate data (y) and output values for columns obtained by SySCIM simulations as estimated data ($\bar{y}$), one may calculate normalized root mean squared error as

[1] $V_{LSB} = \frac{V_{Full\ Scale}}{2^{Number\ of\ Bits}}$

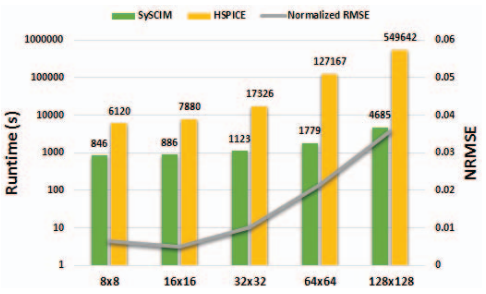


Fig. 2: Comparison of simulation run-times and NRMSE for SySCIM and HSPICE when analyzing the effects of device PV and parasitic resistances for different crossbar sizes.

$$NRMSE = \frac{\sqrt{\frac{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}{N}}}{y_{max} - y_{min}} \quad (2)$$

where N is the number of outputs generated and y_{max} and y_{min} are the maximum and minimum values obtained from HSPICE simulations, respectively. The NRMSE values for different crossbar sizes are shown in Fig. 2. As it can be seen in the figure, the error, in the worst case, is less than 4%.

B. *Non-idealities impact*

In this part, we employ SySCIM to study the impact of different non-idealities on the accuracy of the operation performed by the CIM. We consider a 64×64 memristor crossbar for all experiments in this part. The impact of every non-ideality on the output results of the columns (i.e., the dot product) is measured for 1000 simulations.

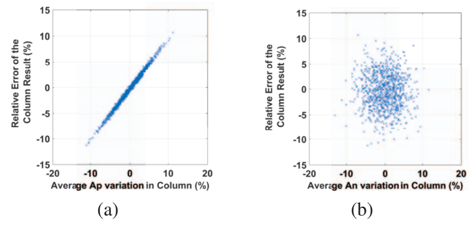


Fig. 3: Relative Error in a random column: a) versus the average of A_p variations in that column; b) versus the average of A_n variations in that column.

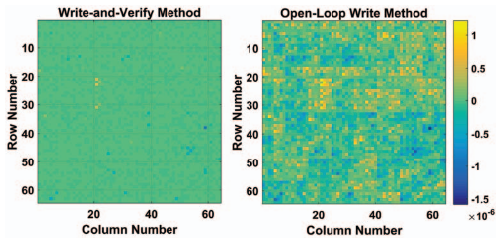
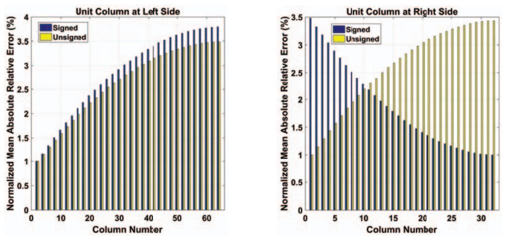


Fig. 4: The difference between achieved conductances and ideal ones for different write methods under device variation.

1) Writing Variation: Here, the impact of the A_p and A_n variations on the imprecision of the stored values on the memristor is studied. To do so, we consider normal systematic and random variations on the A_p and A_n parameters. To write the data on the memristor an open-loop scheme was employed. Fig. 3.a and 3.b show the distribution of the relative output error of a column in the crossbar with respect to the Ap and An variations in that column, respectively. As the figure indicates, there is a linear relationship between the A_p variation and the loss of accuracy. In contrast, the A_n variation does not appear to be correlated to the output error. This is due to the fact that we have used an open-loop writing approach and the A_p (A_n) controls the response of the device to the SET (RESET) pulse. At the beginning of the programming phase, a long RESET pulse is applied to put the selected device in the highest resistance state. Then, based on the data magnitude a number of SET pulses are applied for writing the data. Since the RESET pulse is long enough to ensure the device reaches its highest resistance state the A_n variation effect is insignificant. On the other hand, since there are several SET pulses, the variation of A_p affects the written value considerably. As mentioned before, SySCIM supports both open-loop and write-and-verify writing methods. Since the latter scheme compensates for the writing error by getting feedback from the devices, it is more accurate, but more time and energy-consuming. The writing error in the case of the write-and-verify approach under the Ap and An variations is considerably lower. Fig. 4 illustrates the difference between actual conductances and ideal values after programming with these two methods. Obviously, the error is smaller for the write-and-verify case. Fig. 4 supports this.

2) IR drop: To demonstrate the impact of IR drop of the crossbar interconnects on the dot product operation of the columns, we apply a variation with normal distribution on the SL/BL resistances. In addition, we set the conductances of all memristors in the crossbar to approximately 4.5μS (representing 9 in the unsigned columns and 1 in the signed ones), except the unit column where the conductances of the memristors are set to 4μS (representing a bias equal to 8). Since the result of the unit column is subtracted from all the *signed* columns, a small error of the dot-product operation in this column induces a large error on the final result of the crossbar. In addition, since memristors in different positions are subject to different wire resistances, the current passing through them depends on their position in the crossbar. Therefore, in this subsection, we assumed that the unit column placement could be either at the left side of the crossbar (closest to the DIMs as in Fig. 1) or at the right side (furthest from DIMs).



(a) (b)

Fig. 5: Mean absolute relative error for each column in the presence of parasitic resistance.

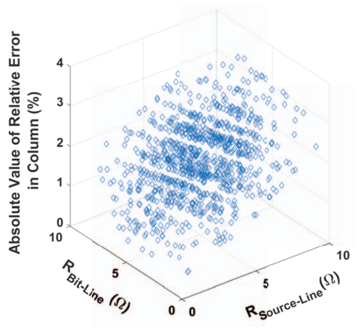


Fig. 6: Relative error in a selected column vs SL and BL resistances.

The average of the absolute values of the relative errors of each column after 1,000 simulation iterations are illustrated in Fig. 5. As expected, by moving away from the DIMs, the errors of the unsigned columns increased independent from the unit column position. This is because the number of parasitic resistances in the current paths to the memristors are increased and consequently their impact on current reduction is accumulated. On the other hand, in the case of the signed column, distance from the unit column determines the error. This originates from the fact that the wire resistance impact on the current of the columns closer to the unit column is almost similar to that of the unit column. Thus, the computation errors of signed columns, closer to the unit column, are lower. Fig. 6 depicts the relative error of a selected column versus the SL and BL resistances to further illustrate the impact of the BL and SL parasitics. It can be observed that the SL resistance has a more significant role than the BL resistance in the accuracy reduction. This is attributed to the fact that each BL resistance only affects its corresponding column while the SL resistances degrade the voltage that reaches the next column and thus their effect is accumulated. Note that the errors were separately normalized for the signed and unsigned columns in the Fig. 5.

3) Faulty devices: Here, we demonstrate the impact of the existence of faulty devices on the output accuracy of the dot-product computation. In doing so, for each simulation, we consider an equal probability for the memristor cells to be faulty as explained in Section III. Fig. 7 shows the average accuracy degradation of the columns vs the percentage of devices that were stuck at LRS and HRS. This results show that the impact of faulty devices is considerably larger than the non-idealities

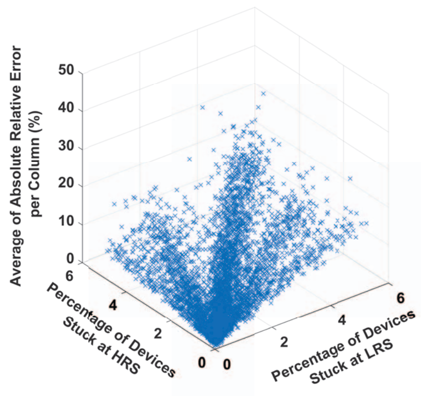


Fig. 7: Average of absolute value of relative error per column (%) vs the percentage of stuck devices at LRS or HRS.

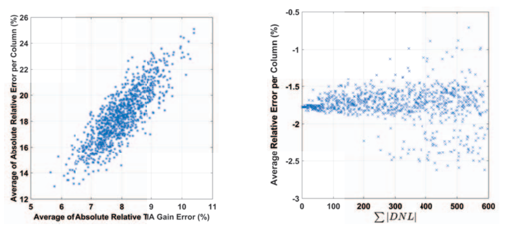
of other CIM components.

4) TIA gain error: For this study, we consider normal variation on the TIA gain. Fig. 8.a illustrates the effects of TIA gain variation on the accuracy of the computations of the columns. As shown, when the average of the absolute values of relative gain errors is high the output error is high as well. Furthermore, it appears that a straight line best fits the samples.

5) ADC non-ideality: Since we consider a 64x64 crossbar in this study, the required ADC resolution is 10 bits. To study the ADC non-idealities, we apply normal variation to the transition points of the ADC I/O relation. The average error per column versus the sum of the absolute values of DNLs of ADC output bits are presented in Fig. 8.b. The sum of absolute values of the DNLs could be considered as a metric showing the overall mismatch between the actual ADC I/O relation and its ideal one. As it is observed, by increasing the sum of absolute values of DNLs, i.e., increasing the mismatch between the analog values and their corresponding digital ones, generally the errors of the computations in the columns increases.

V. Conclusion

In this paper, we presented a comprehensive memristor-based CIM simulator developed in SystemC and SystemC-AMS, called SySCIM. The non-idealities of memristor device, memristor crossbar and the driving circuits are modeled in SySCIM. Our studies showed that SySCIM reduces the simulation time up to 117× at the cost of negligible accuracy



(a) (b)

Fig. 8: a) The average of absolute value of relative error per column (%) vs the average of absolute value of relative gain errors (%). b) Average error per column vs the sum of the absolute values of the DNLs.

loss compared to the HSPICE simulation. We believe SySCIM helps designers foresee the impacts of different sources of non-idealities and adjust their design choices accordingly. As for future work, we suggest using SySCIM to examine various reliability improvement schemes as suggested in the literature.

Acknowledgment

This research is supported by EC Horizon 2020 Research and Innovation Program through MNEMOSENE project under Grant 780215.

References

[1] O. Mutlu, “Processing data where it makes sense in modern computing systems: Enabling in-memory computation,” in *MECO 2018*, pp. 8–9.
[2] M. Hu *et al.*, “Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication,” in *DAC 2016*, pp. 1–6.
[3] A. BanaGozar, K. Vadivel, J. Multanen, P. Jääskeläinen, S. Stuijk, and H. Corporaal, “System simulation of memristor based computation in memory platforms,” in *International Conference on Embedded Computer Systems*. Springer, 2020, pp. 152–168.
[4] L. Xia *et al.*, “MNSIM: Simulation platform for memristor-based neuromorphic computing system,” in *DATE 2016*, 2016, pp. 469–474.
[5] P.-Y. Chen, X. Peng, and S. Yu, “NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning,” *IEEE TCAD*, vol. 37, no. 12, pp. 3067–3080, 2018.
[6] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, “DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies,” in *IEDM 2019*, pp. 32.5.1–32.5.4.
[7] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, “MemTorch: An open-source simulation framework for memristive deep learning systems,” *arXiv preprint:2004.10971*, 2020.
[8] M. Galicia *et al.*, “Neurovp: A system-level virtual platform for integration of neuromorphic accelerators,” in *International Conference on Communications and Technology*. IEEE SOCC, 2021.
[9] J. Multanen, H. Kultala, P. Jääskeläinen, T. Viitanen, A. Tervo, and J. Takala, “Lotta: Energy-efficient processor for always-on applications,” in *IEEE SiPS 2018*. IEEE, pp. 193–198.
[10] C. P. C. C. group at the Tampere University. Tta-based co-design environment (tce). [Online]. Available: http://openasip.org/
[11] P. Chi *et al.*, “PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory,” in *2016 ACM/IEEE 43rd ISCA*, 2016, pp. 27–39.
[12] A. Shafiee *et al.*, “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.
[13] H. Jiang *et al.*, “Sub-10 nm ta channel responsible for superior performance of a hfo 2 memristor,” *Scientific reports*, vol. 6, p. 28525, 2016.
[14] N. Papandreou *et al.*, “Programming algorithms for multilevel phase-change memory,” in *ISCAS 2011*, 2011, pp. 329–332.
[15] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, “Generalized memristive device spice model and its application in circuit design,” *IEEE TCAD*, vol. 32, no. 8, pp. 1201–1214, 2013.
[16] D. Joksas *et al.*, “Committee machines—a universal method to deal with non-idealities in memristor-based neural networks,” *Nature communications*, vol. 11, no. 1, pp. 1–10, 2020.
[17] I. Chakraborty *et al.*, “Resistive crossbars as approximate hardware building blocks for machine learning: Opportunities and challenges,” *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2276–2310, 2020.
[18] A. Agarwal *et al.*, “Path-based statistical timing analysis considering inter-and intra-die correlations,” in *Proc. TAU*, 2002, pp. 16–21.
[19] 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.
[20] A. Banagozar *et al.*, “CIM-SIM: Computation in memory simulator,” 05 2019, pp. 1–4.
[21] A. Ankit *et al.*, “PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference,” in *Proceedings of the 24th ASPLOS*, 2019, pp. 715–731.
[22] C. Yakopcic, R. Hasan, T. M. Taha, M. R. McLean, and D. Palmer, “Efficacy of memristive crossbars for neuromorphic processors,” in *IJCNN 2014*, pp. 15–20.