System Simulation of Memristor Based Computation In Memory Platforms

Ali BanaGozar[1]✉, Kanishkan Vadivel[1], Joonas Multanen[2], Pekka Jääskeläinen[2], Sander Stuijk[1], and Henk Corporaal[1]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
[2] Tampere University, Tampere, Finland
{initial.last name}@tue.nl {first name.last name}@tuni.fi

Abstract. Processors based on the von Neumann architecture show inefficient performance on many emerging data-intensive workloads. Computation in-memory (CIM) tries to address this challenge by performing the computation on the data location. To realize CIM, memristors, that are deployed in a crossbar structure, are a promising candidate. Even though extensive research has been carried out on memristors at device/circuit-level, the implications of their integration as accelerators (CIM units) in a full-blown system are not studied extensively. To study that, we developed a simulator for memristor crossbar and its analog peripheries.This paper evaluates a complete system consisting of a TTA based host core integrating one or more CIM units. This evaluation is based on a cycle-accurate simulation. For this purpose we designed a simulator which a) includes the memristor crossbar operations as well as its surrounding analog drivers, b) provides the required interface to the co-processing digital elements, and c) presents a micro-instruction set architecture (micro-ISA) that controls and operates both analog and digital components. It is used to assess the effectiveness of the CIM unit in terms of performance, energy, and area in a full-blown system. It is shown, for example, that the EDAP for the deep learning application, LeNet, is reduced by 84% in a full-blown system deploying memristor based crossbars.

Keywords: Computation In Memory · Non-Volatile Memory · Memristor · Simulator · non-Von Neumann Architectures · TTA

1 Introduction

Optimization of speed and power consumption in conventional processor architectures, i.e. von Neumann based processors, is difficult due to the fundamental design choice to place the memory unit apart from the processing unit (Fig. 1(a)). For this configuration to operate, data is required to be transferred between the two units back and forth many times. These data movements not only elongate the process-time but also consume a significant amount of energy. Adding levels to the memory hierarchy and introducing small-sized caches, close to the processing unit, have alleviated the problem (Fig. 1(b)). Nonetheless, such techniques

fail to meet ever-increasing performance and energy requirements of emerging data-intensive applications, e.g deep learning networks.
Computation in-memory, on the other hand, suggests processing the data in the very same site where it resides [15, 17, 19]. Eliminating a huge part of the data transfers, CIM reduces the data bandwidth constraints, decreases energy budget requirements, and improves performance. Reference [17] realizes CIM by placing complex computational elements on the logic layer below the 3D memory stack (Fig. 1(c)). Reference [19] proposes exploiting the analog attribute of DRAM technology to perform bulk bitwise operations right inside a DRAM. Doing so a DRAM is used at its full internal bandwidth.
Another approach to realize CIM is to exploit emerging non-volatile memories, e.g., resistive random access memory (ReRAM), phase change memory (PCM), and spin-transfer torque magnetic random access memory (STT-MRAM) [22] (Fig. 1(d)). These devices offer promising features like compact realizations, ultra-low static power –hardly any leakage, and non-volatility –eliminating the required energy to refresh cells– which makes them favorable candidates to realize CIM. Being organized in a crossbar structure, they can operate as extremely dense memory units. Employing the very same structure, several operations such as vector matrix multiplication (VMM), and bulk Boolean bitwise operation (BBB), can be executed on them exploiting their analog behavior. These operations are explained in more details in Section 2. To enjoy data locality and reuse, CIM is connected to the DRAM and not directly to the DISK. Although memristor based CIM is considered one of the most promising options for the next generation of processors, the implications associated with their deployment at the computer architecture level are rarely studied. The number of studies that are exploring memristors at the device level, as well as the required driving mixed-signal circuitry, are quite considerable, however. The contributions of this paper are as follows:

- An in depth system level analysis of the efficacy of deploying memristor crossbar based CIM accelerators at system-level. Based on the study, us-

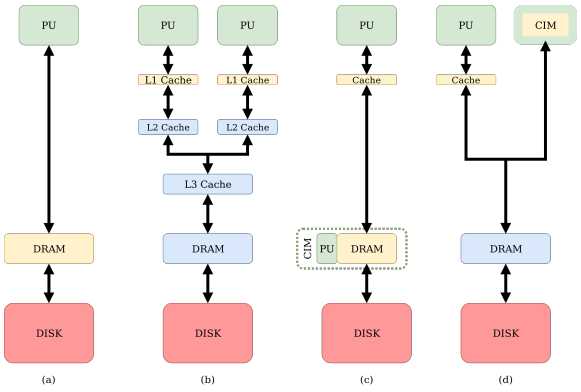


Fig. 1: a) early computers, b) Multi-processor with multi-level caches, c) computation in memory (DRAM), d) computation in memory (memristor) [21]

ing memristor crossbars the EDAP is reduced by 84% for the LeNet kernel (Section 5).
- A novel C++ based CIM unit simulator that includes the memristor crossbar, its surrounding analog drivers, and the required interface to communicate to the co-processing digital part. Furthermore, a micro-instruction set architecture (micro-ISA) for memristor crossbar based platforms, which is independent of technology and capable of execution various memristor crossbar operations, is presented (Section 3).
- Integration of CIM unit(s) into a TTA, a class of exposed data path architectures. Using TTA-based Co-design Environment (TCE), an open-source tool-set that enables the designer to freely customize the TTA, energy, area, and performance figures at system-level are calculated (Section 4).

2 Background and Related Work

For a good understanding a proper background is needed. This section presents the basics of non-volatile memories, their organization in a crossbar structure, and memristor crossbar based CIM processors are presented.

2.1 Memristors and Memristor Crossbars

Device engineers have been seeking for alternative technology for the post-CMOS era since the device scaling is reaching the atomic realm. Non-volatile memories, e.g. phase change memories (PCM), resistive RAMs (ReRAM), are considered a promising instance of such devices. Information maps to the conductance states of a device. The device retains its value until it is (RE)SET by a voltage higher than its threshold voltage.
Being organized in an area-efficient crossbar structure, they can be exploited as dense blocks of memories. To further increase the density –bits per area– multi-level cells are used. The non-linear I-V characteristics of the memristors allow them to hold multiple conductance states. In multi-level cells, multiple bits of information can be stored on a single device with multi-conductance states. To fine-tune the conductance level, in a write-verify scheme, pulses with different width (amplitude) are applied across the target device.
However, the non-linear characteristics of the device itself, and the sneak path current in a crossbar –an undesired current that flows through the memory cells parallel to the desired one– make it hard to use the device in practice. To make the programming stage more controllable, the 1T1R structure (1-Transistor 1-Resistor) is proposed [27] (Fig. 2.a). In the 1T1R structure, a transistor is placed in series with a memristor. Depending on the direction in which the gates of the transistors are connected, row/column-wise, different functionalities can be achieved. For instance, if the gates are connected horizontally, bulk bit-wise operations (BBB) can be executed, but not hyper-dimensional computation (HDC) [9]. Whereas, if the gates are connected vertically, the platform is suitable for HDC and not BBB.

2.2 Memristor Crossbars Functions

Memristors, both individually or in a crossbar layout, have been employed for various use cases. They have been used as physically unclonable functions [11], radio frequency switches [18], dot product engines [10], and memory blocks [8]. In this section, we focus on the memristor crossbar based use-cases.

The most appealing feature of memristors is that they can perform vector-matrix multiplication (VMM), which is the core operation of many applications especially neural networks, in a single shot. To perform a VMM ($Y_{1\times n} = W_{1\times m} \times X_{m\times n}$), where W, X, and Y are weight matrix, input vector, and output vector, respectively, several measures should be taken. First, the elements of the weight matrix are mapped to the conductance states of the crossbar cells. Then, elements of the input vector are encoded to the amplitude or the length of pulses to be applied to the rows. Note that the amplitude of the input voltages should not exceed the threshold voltage of the device, as this destructs the stored information. Next is to drive the input voltages into the word-lines while the bit-lines are virtually grounded. According to Ohm's law, a current equal to the product of input voltage and the cell conductance flows through the device ($I_{ij} = V_i \times G_{ij}$). The currents, that are resulted of all element-by-element multiplications, are accumulated and sensed simultaneously at the end of active columns. Thanks to this attribute, memristor crossbars are considered one of the most promising platforms to realize neuromorphic computing architectures [3,10,20,23]. It is worth mentioning that if the required analog peripheries –which drive the crossbar in the reverse direction– are available, it is possible to do VMM on the transpose matrix using the same crossbar [6].

The other use-case, in which memristor crossbars have shown promising results, is in-memory bulk Boolean bit-wise operations [14]. BBB is frequently performed in database queries and DNA sequence alignment [19]. To perform a bit-wise operation, reference [25] programs every single bit of an operand to cells in a row. Then, by applying a read voltage to two desired rows, where targeted operands are stored, currents flow in bit-lines. Currents are sensed using scouting logic. The reference current of the sense amplifier is determined based on the operation, i.e. AND, OR, XOR to be performed. Comparing the sensed current with the reference current(s), different Boolean operations can be realized.

2.3 Architectures and Simulators

Although memristors are broadly examined at the device and circuit levels higher abstraction levels are barely studied. Architectural simulators like [7, 8, 20, 24] present designs that put memristor crossbar models into practice at the architecture abstraction level. In [20], for example, the memristor crossbars are arranged in a pipelined structure that targets the efficient execution of convolutional neural networks (CNNs). Similarly, in [8] and [24] architectural simulators that are designed for performing neural network implementation are presented.

Ankit et al. propose a "programmable ultra-efficient memristor-based accelerator" (PUMA) that adds general-purpose execution units to enhance memristor

crossbars for machine learning purposes [2]. PUMA presents a specialized ISA that offers programmability without degrading the efficiency of in-memory computing. The instructions, however, are considerably long which performs inefficiently if subtle changes are required to be made. PUMA also is an accelearator which again bears the above mentioned problem.
Zidan et al. propose a "field programmable crossbar array" (FPCA) which unites various combinations of memristor crossbar use cases, namely memory, digital and analog computing in one basic core that can be reconfigured dynamically [28]. The low write endurance of the memristors, however, limits the number of times a unit can be reprogrammed [26]. Consequently, the reconfiguration, which was the core idea of FPCA, seems impractical.
To the best of our knowledge, our work is the first system level analysis of memristor based CIMs integration into a fully programmable architecture. Our research in some aspects is inline with PUMA, i.e. enhancing the memristor crossbars by adding general-purpose execution units. While PUMA focuses on presenting an accelerator we integrate our CIM unit into a fully programmable architecture. Doing so, we investigate and address the possible challenges such an integration brings. Additionally, by introducing a configuration register file, we significantly simplify our instruction set architecture. Hence, it can easily be expanded if new operations are developed for memristor crossbars.

3 CIM Unit

As memristor technologies are hardly available, to develop an architecture for processors based on memristor crossbars we designed a simulator that replicates the functional behavior of memristor crossbars. The simulator also covers the operation of the peripheral mixed-signal circuitry. We call this mixed-signal part "Calculator" (see Fig. 2.b). For the Calculator to communicate with other processing elements, we have expanded the simulator, by adding the necessary digital components, e.g. buffers, register files, controller. The added digital components all together comprise the "Micro-engine". In the remainder of this section, first, the architecture of the CIM unit is illustrated; then, the proposed micro-ISA is introduced; and in the end, the tasks of the controller are described.

3.1 CIM Architecture

CIM tile, a cycle-accurate simulator that is developed in C++, replicates the function of a memristor crossbar, the driving mixed-signal circuits, and necessary digital elements. The tile includes two different domains, mixed-signal, the Calculator, and digital, the Micro-engine (Fig. 2.b).
In the Calculator, various modules are defined to imitate the behavior of every single analog component, i.e. the crossbar, analog/digital converters (ADC), digital input modulators (DIM), sample and holds (S&H). This modular design

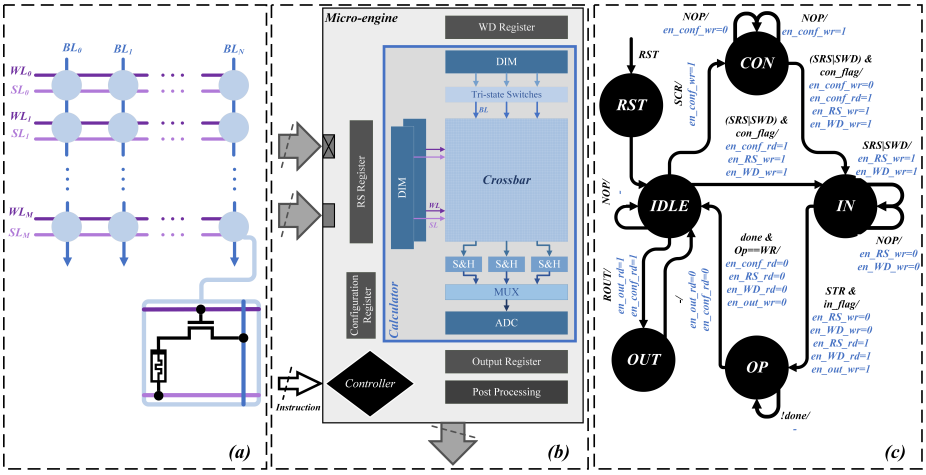


Fig. 2: Overall look of the CIM unit. a) 1T1R non-volatile memory crossbar , b) the tile structure, c) The controller. (only memory units enable/disable signals are presented in the FSM)

enables a designer to change the configuration of the Calculator, e.g. crossbar size, number of ADC/DIMs, or add new modules with a negligible effort.
DIMs and ADCs, at the edges of the Calculator, are connected to digital buffers, i.e. RS/WD/Output. DIMs convert the raw data into amplitude/width of pulses that are to drive the crossbar. Crossbar results, if operation produces any, are converted to digital values by ADCs and stored in the output buffer. Simple digital logics, present inside the Micro-engine, carry out simple operations like weighted sum if desired. To instruct the CIM unit to carry out different operations, we propose a micro-instruction set architecture (Table 1). The micro-instructions promote the nano-instructions that are presented in [5]. Although the nano-instructions offer full control over the tile, there is quite some room to enhance them. For example, the process of fetching raw data into buffers requires the whole buffer to be overwritten, even if only a few entries are supposed to change. The full control offered by nano-architecture is not desirable as it comes with many dependencies between instructions. This not only make the compilation complex, long, and inefficient but may lead to unreliable code. Therefore we designed a micro-ISA to void interfering with any pre/post-processing stage inside the CIM unit, thus eliminating the chance of external error. The controller reads the operation parameters that are written into the configuration register via SCR instruction and manage the whole operation based on these registers.

3.2 Micro-ISA and Micro-engine

To avoid very long instructions, that include several execution parameters, a configuration register is introduced to hold these parameters (Table 2). The controller, a mealy machine (FSM), ensures the correct execution of instructions, e.g. filling buffers, enabling/disabling the analog elements, etc (Table 1). To

Table 1: Micro instructions. Simplified instruction set that can easily be extended.

Class	Mnemonic	Description	Operands
Initialization	SCR	Set Configuration Register	address, data
	SRS	Set Row Select Buffer	data
	SWD	Set Write Data Buffer	data
Compute	STR	Start operation, e.g. Write, VMM, BBB	-
Read	ROUT	Read result Out	-

Table 2: Configuration Registers. Controller conducts an operation based on the content of configuration register

Register index	0	1	2	3	4	5	6	7	8	9-15
Register content	Start Row	Start Col	Number of Rows	Number of Cols	Input Precision	Weight Precision	Output Precision	Truncate Bits	Operation	Reserved

carry out an operation, first, the configuration register should be filled using the set configuration register (SCR) instructions. Based on the contents of the configuration register, the FSM calculates the address of the buffers, aligns the data, triggers the operation, collects the crossbar output, and post-processes the output (if needed). In the end, the FSM controls the process of sending the final results out. As mentioned, the controller is a mealy machine that issues the control signal based on the received micro-instructions, configuration register contents, the state of the FSM, and the internal flags (Fig. 2.c). The digital buffers, the configuration registers, and the controller, all together, constitute the Micro-engine.

Without loss of generality, we use PCM technology to demonstrate our work. According to a prototype developed by IBM [13], to write a phase change memory (PCM), or to perform a VMM takes respectively 2.5μs and 1μs. Considering that these delays are considerable, we attempted to schedule some tasks into these very long time slots. The data that must be processed on the CIM unit is a vector. Therefore we propose to add an extra set of buffers in the CIM unit (double buffering (DB)) to fetch the $(n+1)^{th}$ vector while vector n^{th} is being processed (Fig. 3.b). To ensure that no data is lost, we add an extra level to the controller that supervises the correct redirection of the control signals. This approach of having a top-layer in the controller enables us to perform the operations that are targeted in [9].

4 TTA-CIM

We integrate our CIM unit into *Low-power TTA* (LoTTA) –a processor design based on Transport Triggered Architecture (TTA)– developed for energy-efficient execution of always-on application [16]. This section start with introducing TTAs. It is followed by a description of the TTA Co-design Environment

(TCE), an open-source tool-set that allows adding special functional units like the CIM unit to the architecture. Lastly, the details of the CIM special functional unit (CIM-SFU) integration into TTA architecture are explained.

4.1 TTA and TCE

Transport Triggered Architectures are a class of Exposed Data Path Architectures (EDPAs), where the data path of the processor is exposed to the programmer. Fine-grain control over the data path allows compile-time bypassing of data between processing-elements, i.e. software bypassing, without dependency checking hardware circuitry, which improves energy-efficiency. Furthermore, static scheduling of instructions on EDPAs opens up new optimization possibilities on the software as well as on the hardware side. Compared to traditional operation triggered architectures, where operation triggers data-path activity, the TTA instruction represents the data transports (TTA data-move) and the computations are triggered as a side effect of the data-move. Functional Units of TTA comprise one or more operand ports and optional result ports that can communicate data over the interconnect network. In TTAs, one of the operation ports of FUs is designated as a special port named "trigger-port". A data move to the trigger port starts the execution of an operation on FUs. Fig. 4 (bottom) depicts an example of a TTA processor instance with three communication resources (bus network) and one CIM unit as an added special functional unit (CIM-SFU). The trigger port of the FUs are marked with a cross mark. The CIM-SFU is the functional model of the CIM unit with CIM-ISA defined in Section 3. The Fig. 4 (top) shows a TTA program for simple increment operation (i.e. $a = a + 5$) on the processor instance [16] to illustrate a programming model of the TTAs. Three buses in the instance imply that three data transports can happen in parallel during each clock cycle. Therefore, both operands move (a and 4) can happen in parallel for the considered example and the whole operation takes two cycles in total as shown by the highlighted data moves. The data move $4 \rightarrow ALU1.add.in1t$ triggers execution of an add operation in the ALU unit with operands on input ports $in1t$ and $in2$ at cycle-1. At cycle-2, the result of the add operation (assuming add latency is one cycle) is written back to RF. A mature open-source tool-set, TTA Co-design Environment (TCE), enables users to design and freely customize TTAs for their purpose. It includes a retargetable instruction-set compiler, cycle-accurate ISA simulator, RTL generator, and support for adding special compute units for dedicated functions. TTAs

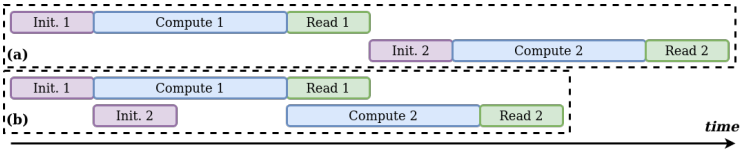


Fig. 3: CIM unit execution flow, a) without DB, b) with DB (Init., Compute and Read are three classes of micro instructions in Table 1)

are an ideal candidate for energy-efficient application-specific instruction-set processors (ASIPs), and a sensible choice for prototyping experimental platforms such as our CIM unit.
Low-power TTA (LoTTA) [16] is a processor design aimed for always-on processing and efficient execution of both signal processing and control-oriented programs. The core used for evaluations in this paper is a variant of the original work. Functional units of the core and its interconnection network are presented in Fig. 4 (top). Considering that LoTTA is specifically optimized for energy efficiency it is a suitable core to start with.

4.2 CIM Unit Integration (CIM-SFU)

As pointed out in Section 3, the latency of the CIM computations (VMM, Write, etc.) is much higher than the regular operations on the traditional FUs such as ALU, MUL, etc. This gives rise to two main requirements for the CIM-SFU design, 1) A multi-cycle FU model to hide the latency of the CIM-SFU from other units, and 2) Pipelined FU model to separate the CIM compute unit (Calculator) from the TTA interface (operand and result ports) to hide data-latency with the double-buffering concept (Fig. 3.b). The semi-virtual time latching model of the FUs in the TCE allows for a multi-cycle, pipelined TTA-SFU model in the TCE tool-set. Fig. 2.b depicts the designed CIM-SFU. The CIM-SFU is modelled as a three-stage pipeline with the first-stage fetching the input data, the second stage covering the core CIM mixed-signal computation logic (*Calculator*), and the third stage sending the results out (Fig. 3.a). The pipeline of the SFU is controlled by a trigger move. i.e. when the trigger move happens, the operands ($input_{1t}$ and optional $input_{2-n}$) are latched to the Micro-engine and the opcode is issued to the controller of the Micro-engine. The FSM, then, issues necessary control signals for the rest of the components. The Calculator, which is modeled cycle-accurately, produces the result on the output buffer once the operation latency is elapsed for the triggered operation. The data on the output buffer is then serialized to the output port via an explicit trigger commands to controller.

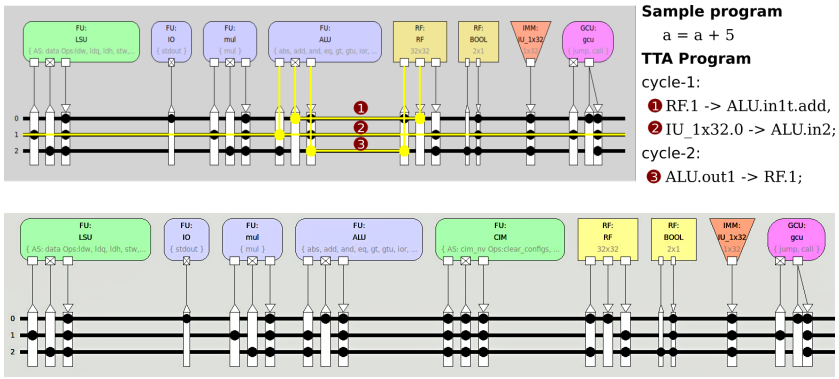


Fig. 4: TTA based processors overview. LoTTA(Top). LoTTA+1×CIM(Bottom)

Table 3: CIM unit characteristics

Crossbar Parameters	**Value**	
Memristor Technology	PCM (from IBM)	
Cell precision	8-bit (implemented by 2×(4-bit) PCMs)	
Compute and Write Latency/8-bit	1 μs and 2.5 μs	
Compute Energy/8-bit	200 fJ (2×100 fJ/4-bit PCM)	
Write Energy/8-bit	200 pJ (2×100 pJ/4-bit PCM)	
Area (128×128)	50 μm^2	
Peripheral Circuitry	**Energy**	**Area**
Mixed Signal	2.1 nJ/cycle (@1.2GHz)	1252 μm^2
Micro-engine (Digital)	64.8 pJ/byte	865 μm^2

The architecture template of the TCE requires that each operation in the FU to have a deterministic latency such that the resulting read for the operation can be scheduled at compile time.

5 Experiments

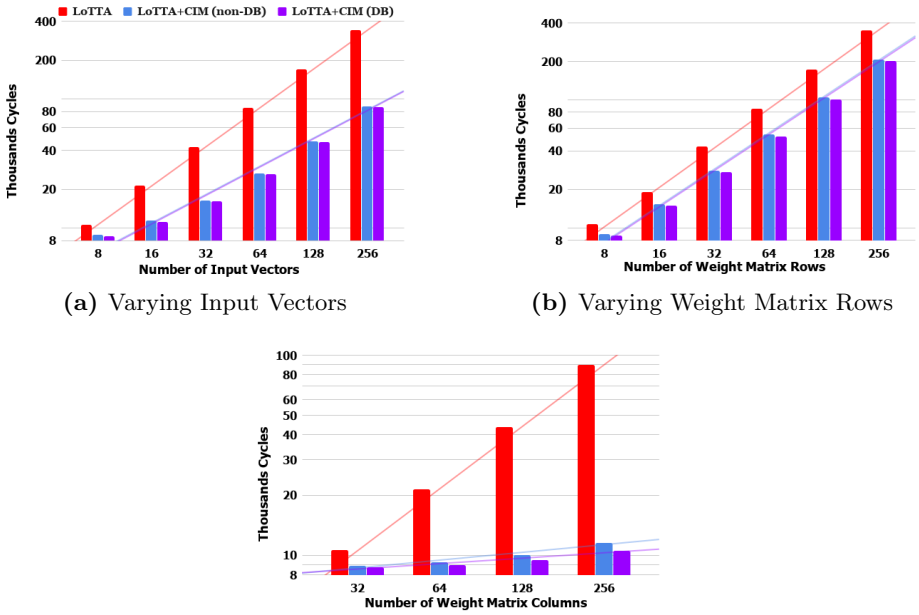
In this section, we present the effects of adding the CIM unit in a quantitative manner. To do so, we employ LoTTA, without a CIM-SFU, as a base setup. Then, CIM-SFU(s) added to LoTTA to explore its effect on various parameters such as performance, energy, and area. For evaluation, we used gemm as well as deep learning LeNet kernels.

5.1 Experimental Setup

The energy and area estimates for the LoTTA core are obtained after synthesis with Synopsys Design Compiler, version 2016.12. A 28 nm process is used at 0.95 V operating voltage and 25°C temperature process corner. For power consumption analysis, switching activity information files (SAIFs) are generated with ModelSim 10.5. The weights to be mapped on the memristor crossbar are 8-bit values (8-bit precision is enough to obtain a reasonable accuracy) which are mapped on IBM's 4-bit PCM [12], as it has shown promising results [12]. To mimic an 8-bit weight with 4-bit cells, two columns are used, one for four MSBs and the other for four LSBs. The final result is computed by a weighted sum of MSB and LSB columns in the Micro-engine. The models for the CIM unit components are from [12] and [20].

5.2 Evaluation

As mentioned earlier, we evaluated gemm and LeNet kernels. For gemm, we studied the impact of different input and matrix sizes. For LeNet, we evaluated performance, accuracy, energy, as well as area for different crossbar sizes.
gemm: To evaluate how memristor crossbars perform on VMM, we implemented the gemm kernel on a crossbar of size 256×256. Fig. 5.a shows that by increasing the number of input vectors the speedup increases from 1.2X to 3.9X for



(a) Varying Input Vectors

(b) Varying Weight Matrix Rows

(c) Varying Weight Matrix Columns

Fig. 5: Performance for gemm kernel (in thousands cycles).

basic CIM unit without double buffering. This was expected since the computation dominates the initial overhead of programming the crossbar. Increasing the number of the rows of the weight matrix, i.e. the columns of the input matrix, it is observed that although the speed-up increases, the improvement rate is less significant compared to the previous case (Fig. 5.b). This happens as the number of cycles required to program the crossbar dominates the overall execution time. Considering these two experiments, and the fact that memristors still suffer from low endurance, the read/write ratio shall be taken into account to assess if it is reasonable to use a memristor crossbar or not. Lastly, in Fig. 5.c we observe that increasing the number of weight matrix columns increases the speed-up rate of LoTTA+CIM over LoTTA since the size of the vector to be programmed grows while the required time for programming the crossbar just slightly increases. **Double Buffering:** Fig. 5.a shows that although by deploying double buffering performance improves compared to non-DB, the relative speedup goes down from 1.02X to 1.01X. This happens since the size of input vector is too small, thus, a limited number of instructions can be scheduled to a VMM execution period (Compute stage in Fig. 3.) However, in Fig. 5.c we observe that as the number of columns of weight matrix increases the relative speedup caused by deployment of DB goes from 1.02X to 1.10X. This happens since the size of the vector to be loaded and programmed on the crossbar grows; therefore, more instructions can be scheduled to a write execution period. **LeNet:** To assess the suitability of memristor based CIMs for deep learning applications, we implemented the LeNet architecture on LoTTA and LoTTA+CIM unit(s). The LeNet

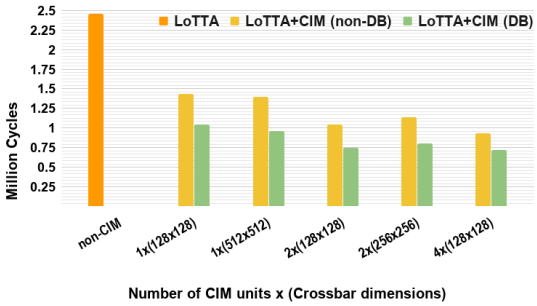
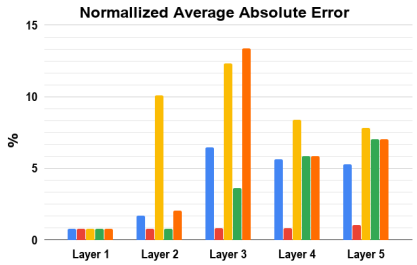
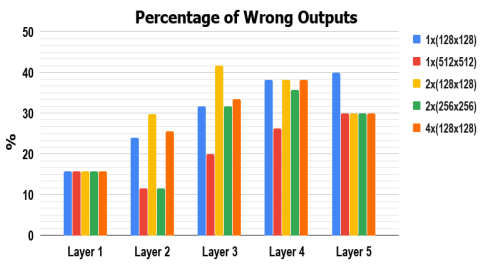


Fig. 6: Performance comparison of architectures for LeNet deep neural network

architecture comprises both convolutional as well as fully connected neural network layers that makes it a perfect data-intensive application instance to study with respect to implications associated with deploying memristor crossbars in a full-blown system. One of the challenges that memristor crossbars should address is the possibility that the weight matrix exceeds the memristor crossbar in size, i.e., either in the number of columns or rows. If the number of columns of the weight matrix is bigger than that of the actual memristor crossbar, the only measure that should be taken is to divide the weight matrix over M CIM units, where $M = \lceil \frac{WeighMatrix_{col}}{MemristorCrossbar_{col}} \rceil$, or to divide the task over time and use a CIM unit M times. The second solution is not quite desirable due to the low endurance of memristors. Obviously the RS buffer in each and every CIM unit has to hold the exact same data. Since columns results are independent they can be handled without any dependency. In case the number of rows of the weight matrix is bigger than that of of the actual memristor crossbar, like in the previous case, either N CIM units are required, where $N = \lceil \frac{WeighMatrix_{row}}{MemristorCrossbar_{row}} \rceil$, or the operation should be carried out in N time steps with one CIM unit. Unlike the previous case the results of each part are only partial results and should be accumulated to produce the final result. To study this, we assume various sizes for the memristor crossbar. The biggest memristor crossbar has 512 rows and the smallest has 128. The number of columns of all the layers of LeNet are always smaller than the matrix size. Considering that the weight matrix of the second and the third layers of LeNet after unrolling are 150 and 400 rows, respectively,



(a) Sum of absolute errors normalized by the maximum value



(b) Percentage of faulty outputs

Fig. 7: Errors in results using basic weight without retraining

these layers should be either distributed temporally, if the resources are limited, or spatially, if enough CIM units are available. Fig. 6 shows the results of various implementations. Using only one CIM unit, even one that is big enough to map a whole layer to the crossbar, performance is worse than any other implementation with multiple CIM units due to parallelism. Distributing weights matrix amongst several CIM units saves quite a few cycles while programming the crossbar, since this is the most time consuming step of performing an operation on a memristor crossbar. The reason that the implementation with two units of size 256×256 yields worse results, in terms of speed and energy, compared to the one with two crossbars of size 128×128 is that we map a whole layer to one crossbar if it was possible. This is important as splitting weight degrades the accuracy (Fig. 7). As can be seen in Fig. 6, we have mapped the architecture for the DB version as well. The best performance is achieved when the four CIM units that are deployed in the design, do the calculation in parallel.

Accuracy: The error sources that we have spotted are two folds; 1) MSB and LSB columns are truncated by the DC offset of the ADC before being summed up together; therefore, the expected carry bit from the addition of the lower bits is lost, 2) The saturation voltage of ADC clips the high/low voltages. In case a crossbar is distributed amongst several CIM units or it is multiplexed in time, a partial sum may exceed the saturation voltage while other partial sums do not reach the saturation voltage. If all parts were together this would be a saturated result while in the distributed case partial sums produce a different result. In a mathematical terminology $\Sigma F(x_i^j) \neq F(\Sigma x_i^j)$, where F is a non-linear function – characteristics of ADC – and x_i^j is the output of i^{th} column of j^{th} CIM tile. Looking at Fig. 7, we observe that splitting the weight matrix in different manners results in different errors. As an instance, for crossbar(s) of size 128×128, although the inputs are the same, just by splitting the second layer weight differently different number of faulty errors with different averages are reported (see blue, yellow, and orange bars in the figure). Although these error degrades the final result, due to the resilient nature of LeNet the input images are still classified correctly. To address the degradation of the results, it is required to retrain the network with crossbar size being taken into account. The retraining process is out of the scope of this paper.

Energy and Area: One of the most important concerns in the deployment of CIM units is their analog nature which requires data conversion that can be costly. Although digital/analog converters (DACs) are relatively cheap in terms of energy/area, analog/digital converters (ADCs) can be extremely costly – more than 500× more power hungry and more than 7000× bulkier compared to DACs [20]. One of the techniques that is commonly used to reduce the en-

Table 4: Energy and area results for different CIM unit configurations

Core	LoTTA	1×(128 × 128)		1×(512 × 512)		2×(128 × 128)		2×(256 × 256)		4×(128 × 128)	
		non-DB	DB	non-DB	DB	non-DB	DB	non-DB	DB	non-DB	DB
Energy (mJ)	1.54	0.92	0.68	0.89	0.62	0.68	0.49	0.74	0.52	0.61	0.48
Area (μm^2)	10009	12175	12460	17534	18674	13761	14331	16934	18074	16934	18074
EDAP (10^9)	13.25	5.66	7.61	3.43	4.97	3.36	3.07	3.88	1.86	2.65	2.18

ergy/area overhead is to share an ADC amongst several columns. To do so, sample and hold circuits (S&Hs) are introduced between the ADC and crossbar. With such modifications the required energy budget falls into a reasonable scale. Table 4 shows that in the best case up to 69% energy reduction can be achieved, while area is increased by 80%. Looking at EDAP –energy, delay, area, product– we observe that in almost all cases, except for a crossbar of size 512×512, the double buffered version performs better than all other implementation in the basic CIM unit. Also, it is spotted that in the DB version the $2 \times (128 \times 128)$ yields the best EDAP, while in the basic version the best EDAP is obtained by $4 \times (128 \times 128)$. The reason why the DB version consumes less energy compared to the non-DB version is that the total energy consumed by other FUs reduces as the number of total required cycles declines.

6 Conclusion

In this paper, we proposed a cycle-accurate simulator for memristor based CIM accelerators to scrutinize its challenges. The simulator offers a micro-ISA that allows a memristor crossbar to be integrated into a transport triggered architecture. The integration not only enhances the crossbar with general-purpose functional units to execute complex kernels but also enables us to study the challenges that are associated with a memristor crossbar deployment in a full-blown system.

Deploying CIM unit(s) shows huge improvements in terms of performance and energy, up to 3.9X speedup and 69% energy reduction. However, including CIM units has a price. The extra units increase the overall area. In our examples, the area increases between 21% and 86%. In addition, accuracy of the results may degrade since the architecture is not taken into account while the networks are trained. Hence, training should take the architecture properties into account. Of course it would elongate the training process [4]. The huge reduction of EDAP (up to 84%) is an extremely motivating point, though.

To follow up on this research, first, one can improve the CIM unit model by introducing non-ideal characteristics of both the memristor crossbars, such as IR drop, and noise, as well as the surrounding driving circuits, like process variation. This would certainly affect the final result and should be compensated by CIM unit characteristics aware training. Another interesting topic is to study the data reuse and local memory size on reducing the number of accesses to the global memory, which can further improve the energy and performance figures.

Acknowledgment. This research is supported by EC Horizon 2020 Research and Innovation Program through MNEMOSENE project under Grant 780215. The work also recevied support from the FitOptiVis project [1] funded by the ECSEL Joint Undertaking under grant number H2020-ECSEL-2017-2-783162.

References

1. Al-Ars, Z., Basten, T., de Beer, A., Geilen, M., Goswami, D., Jääskeläinen, P., Kadlec, J., de Alejandro, M.M., Palumbo, F., Peeren, G., et al.: The FitOptiVis ECSEL project: Highly efficient distributed embedded image/video processing in cyber-physical systems. In: Proceedings of the 16th ACM International Conference on Computing Frontiers. p. 333–338. CF '19 (2019)
2. Ankit, A., Hajj, I.E., Chalamalasetti, S.R., Ndu, G., Foltin, M., Williams, R.S., Faraboschi, P., Hwu, W.m.W., Strachan, J.P., Roy, K., et al.: Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. p. 715–731. ASPLOS '19, Association for Computing Machinery, New York, NY, USA (2019)
3. Ansari, M., Fayyazi, A., Banagozar, A., Maleki, M.A., Kamal, M., Afzali-Kusha, A., Pedram, M.: Phax: Physical characteristics awareex-situtraining framework for inverter-based memristive neuromorphic circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(8), 1602–1613 (2017)
4. BanaGozar, A., Maleki, M.A., Kamal, M., Afzali-Kusha, A., Pedram, M.: Robust neuromorphic computing in the presence of process variation. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017. pp. 440–445 (2017)
5. BanaGozar, A., Vadivel, K., Stuijk, S., Corporaal, H., Wong, S., Lebdeh, M.A., Yu, J., Hamdioui, S.: Cim-sim: computation in memory simuiator. In: Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems. pp. 1–4. ACM (2019)
6. Cai, F., Correll, J.M., Lee, S.H., Lim, Y., Bothra, V., Zhang, Z., Flynn, M.P., Lu, W.D.: A fully integrated reprogrammable memristor–cmos system for efficient multiply–accumulate operations. Nature Electronics **2**(7), 290–299 (2019)
7. Chen, P.Y., Peng, X., Yu, S.: Neurosim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures. In: 2017 IEEE International Electron Devices Meeting (IEDM). pp. 6–1. IEEE (2017)
8. Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., Xie, Y.: Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In: Proceedings of the 43rd International Symposium on Computer Architecture. pp. 27–39. ISCA '16, IEEE Press, Piscataway, NJ, USA (2016)
9. Hamdioui, S., Du Nguyen, H.A., Taouil, M., Sebastian, A., Gallo, M.L., Pande, S., Schaafsma, S., Catthoor, F., Das, S., Redondo, F.G., Karunaratne, G., Rahimi, A., Benini, L.: Applications of computation-in-memory architectures based on memristive devices. In: 2019 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 486–491 (2019)
10. Hu, M., Strachan, J.P., Li, Z., Stanley, R., et al.: Dot-product engine as computing memory to accelerate machine learning algorithms. In: 2016 17th International Symposium on Quality Electronic Design (ISQED). pp. 374–379. IEEE (2016)
11. Jiang, H., Belkin, D., Savel'ev, S.E., Lin, S., Wang, Z., Li, Y., Joshi, S., Midya, R., Li, C., Rao, M., et al.: A novel true random number generator based on a stochastic diffusive memristor. Nature communications **8**(1), 882 (2017)
12. Le Gallo, M., Sebastian, A., Cherubini, G., Giefers, H., Eleftheriou, E.: Compressed sensing with approximate message passing using in-memory computing. IEEE Transactions on Electron Devices **65**(10), 4304–4312 (2018)
13. Le Gallo, M., Sebastian, A., Mathis, R., Manica, M., Giefers, H., Tuma, T., Bekas, C., Curioni, A., Eleftheriou, E.: Mixed-precision in-memory computing. Nature Electronics **1**(4), 246 (2018)

14. Li, S., Xu, C., Zou, Q., Zhao, J., Lu, Y., Xie, Y.: Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–6 (June 2016)
15. Mittal, S.: A survey of reram-based architectures for processing-in-memory and neural networks. Machine learning and knowledge extraction **1**(1), 75–114 (2018)
16. Multanen, J., Kultala, H., Jääskeläinen, P., Viitanen, T., Tervo, A., Takala, J.: Lotta: Energy-efficient processor for always-on applications. In: 2018 IEEE International Workshop on Signal Processing Systems (SiPS). pp. 193–198. IEEE (2018)
17. Nair, R., Antao, S.F., Bertolli, C., Bose, P., Brunheroto, J.R., Chen, T., Cher, C.Y., Costa, C.H., Doi, J., Evangelinos, C., et al.: Active memory cube: A processing-in-memory architecture for exascale systems. IBM Journal of Research and Development **59**(2/3), 17–1 (2015)
18. Pi, S., Ghadiri-Sadrabadi, M., Bardin, J.C., Xia, Q.: Nanoscale memristive radiofrequency switches. Nature Communications **6**, 7519 (2015)
19. Seshadri, V., Lee, D., Mullins, T., Hassan, H., Boroumand, A., Kim, J., Kozuch, M.A., Mutlu, O., Gibbons, P.B., Mowry, T.C.: Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 273–287. ACM (2017)
20. Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P., Hu, M., Williams, R.S., Srikumar, V.: Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: Proceedings of the 43rd International Symposium on Computer Architecture. pp. 14–26. IEEE Press, Piscataway, NJ, USA (2016)
21. Singh, G., Chelini, L., Corda, S., Awan, A.J., Stuijk, S., Jordans, R., Corporaal, H., Boonstra, A.J.: A review of near-memory computing architectures: Opportunities and challenges. In: 2018 21st Euromicro Conference on Digital System Design (DSD). pp. 608–617. IEEE (2018)
22. Upadhyay, N.K., Jiang, H., Wang, Z., Asapu, S., Xia, Q., Joshua Yang, J.: Emerging memory devices for neuromorphic computing. Advanced Materials Technologies **4**(4), 1800589 (2019)
23. Wang, Z., Joshi, S., Savel'ev, S., Song, W., Midya, R., Li, Y., Rao, M., Yan, P., Asapu, S., Zhuo, Y., et al.: Fully memristive neural networks for pattern classification with unsupervised learning. Nature Electronics **1**(2), 137 (2018)
24. Xia, L., Li, B., Tang, T., Gu, P., Yin, X., Huangfu, W., Chen, P., Yu, S., Cao, Y., Wang, Y., Xie, Y., Yang, H.: Mnsim: Simulation platform for memristor-based neuromorphic computing system. In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 469–474 (2016)
25. Xie, L., Du Nguyen, H.A., Yu, J., Kaichouhi, A., Taouil, M., AlFailakawi, M., Hamdioui, S.: Scouting logic: A novel memristor-based logic design for resistive computing. In: 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp. 176–181. IEEE (2017)
26. Yang, J.J., Strukov, D.B., Stewart, D.R.: Memristive devices for computing. Nature nanotechnology **8**(1), 13 (2013)
27. Zangeneh, M., Joshi, A.: Performance and energy models for memristor-based 1t1r rram cell. In: Proceedings of the Great Lakes Symposium on VLSI. p. 9–14. GLSVLSI '12, Association for Computing Machinery, New York, NY, USA (2012)
28. Zidan, M.A., Jeong, Y., Shin, J.H., Du, C., Zhang, Z., Lu, W.D.: Field-programmable crossbar array (fpca) for reconfigurable computing. IEEE Transactions on Multi-Scale Computing Systems **4**(4), 698–710 (Oct 2018)