The Power of Computation-in-Memory Based on Memristive Devices

Jintao Yu, Muath Abu Lebdeh, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui

Computer Engineering Lab
Delft University of Technology, Delft, The Netherlands
E-mail: {J.Yu-1,M.F.M.AbuLebdeh,H.A.DuNguyen,M.Taouil,S.Hamdioui}@tudelft.nl

Abstract— Conventional computing architectures and the CMOS technology that they are based on are facing major challenges such as the memory bottleneck making the memory access for data transfer a major killer of energy and performance. Computation-in-memory (CIM) paradigm is seen as a potential alternative that could alleviate such problems by adding computational resources to the memory, and significantly reducing the communication. Memristive devices are promising enablers of a such CIM paradigm, as they are able to support both storage and computing. This paper shows the power of memristive device based CIM paradigm in enabling new efficient application-specific architectures as well as efficient implementations of some known domain-specific architectures. In addition, the paper discusses the potential applications that could benefit from such paradigm and highlights the major challenges.

I. Introduction

Data-intensive applications are becoming more important in various domains such as artificial intelligence, health-care, and business in the recent years. They demand larger storage, more computing power, and higher energy efficiency from the IT infrastructure [1]. However, CMOS technology is facing significant challenges that make them inefficient in delivering the higher required performance as CMOS scaling is slowing down due to three walls [2]: (1) the reliability wall as further technology scaling leads to increased failure rate and reduced device lifetime; (2) the leakage wall as static power dominates and may even exceeds the dynamic power due to volatile technology; (3) the cost wall as the cost per transistor via pure geometric scaling of process technology saturated. On the other hand, today's computer architectures also face three well-known walls [3]: (1) the memory wall due to the growing gap between processor and memory speed, and the limited memory bandwidth; (2) the Instruction-Level parallelism (ILP) wall due to the difficulty of extracting sufficient parallelism to fully exploit all the cores; (3) the power wall as the CPU clock frequency has reached the practical maximum value that is limited by cooling. In order for computing systems to continue delivering the required performance given the economical power constraints, novel computer architectures in the light of emerging non-volatile (practically no leakage) device technologies have to be explored.

Many alternatives architectures and technologies are under investigations. Resistive computing [4], quantum computing [5], and neuromorphic computing [6] are couple of alternative computing notions, while memristive devices, quantum dots, spin-wave devices are couple of emerging device technologies [7]. Memristive device is seen as a very a promising candidate to complement and/or replace traditional CMOS (at least in some applications) due to many advantages including CMOS process compatibility [8], zero standby power, great scalability, and high density, etc. Note that the memristive device (or memristor) refers to a two terminal non-volatile memory device, which could, for example, be a spin-transfer-torque magnetic random access memory (STT-MRAM) device, phase-change random access memory (PCRAM) device, or resistive random access memory (RRAM) device. Due to its properties, memristive device has the potential to simultaneously implement high density memories [9] as well as different computing styles [10], [11], enabling new computing paradigms, namely, *Computation-in-memory* (CIM); CIM can alleviate the memory wall as data can be processed locally inside the memory and hence, no needed to be fetched and computed by the processors. In order to study its potentials, intensive research has been done on the concept of CIM. From the circuit design perspective, several primitive operators were developed, such as IMPLY (executing material implication logic function) [12], Scouting (executing OR, NOR and XOR functions) [13], DPE (executing vector-matrix multiplication kernel) [14], etc. Other work cascaded these primitive operators to realize more complex functions such as arithmetic adders [12], [15] and multipliers [16], [17]. Many researchers also used one or multiple of these circuits to propose application-specific (and even general-purpose) architectures such as MPU [18], PLiM [19], Pinatubo [11], ISAAC [20], etc. Given this interest in memristive device based CIM, it is of great importance to explore the potential of such architectures and the application domains that could benefit from them.

This paper presents a classification of CIM architectures and shows the strong dependency between the targeted application, the CIM architecture as well as its circuit design. This is illustrated by targeting three different applications: query select for a data-base, automata processor, and compressed sensing. The paper will not only show the superiority of CIM based on memristive device for such applications, but also will present the design considerations during the development process.

The rest of this paper is structured as follows. Section II presents a classification of CIM architectures. Then, Section III describes a methodology for developing a CIM architecture for a given application, provides a list of potential applications that could make use of each class, and highlights the selected applications for this paper. Thereafter, Section IV and V apply this methodology to the three case studies at the circuit and system level respectively. In Section VI, we highlight the potential and challenges of memristive device based CIM architectures. Finally, Section VII concludes the paper.

978-1-7281-4123-7/20/$31.00 ©2020 IEEE

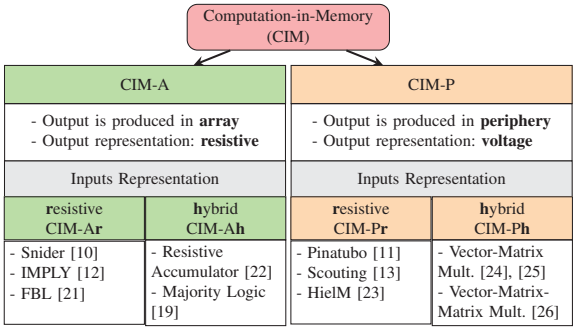


Fig. 1. CIM circuit classification [27].

II. CIM Classification

A memory core, including memristive device based cores, consists of a *memory array* and its *peripheral circuits*. Each circuit design that aims at implementing a logic or arithmetic operation inside the memory core produces the computing result either within the *array* or within the *periphery*. Hence, depending on *where* the result of the computation is produced, the CIM architecture can be divided into two classes as shown in Fig. 1.

- *CIM-Array* (CIM-A): the computing result is generated within the memory array. In other words, the output is stored in a memristive device in the array, e.g., in form of a resistance state.
- *CIM-Periphery* (CIM-P): the computing result is generated within the peripheral circuitry. Given the fact that memory periphery is based on CMOS technology, the output is physically generated as voltage.

As shown in Fig. 1, we further divide the CIM-A and CIM-P classes into two subcategories. In the first category, *all operands* of the operation are stored in the array, e.g., in the form of resistance. In the second category, *only* part of the operands is stored in the array and the other part is received via the memory port(s). Hence, the logic values of the second category are *hybrid*, e.g., resistive and voltage. If none of the operands is stored in the array, then CIM concept is not applicable as the data is not stored in the same physical location as the computation will take place. The above classification results into four sub-categories as indicated in the figure: CIM-Ar, CIM-Ah, CIM-Pr and CIM-Ph; the additional letters 'r' and 'h' indicate the nature of the inputs (operands), namely resistive and hybrid, respectively. The bottom part of the figure shows the existing circuit designs for each of the classes. In the next sections, some of these designs will be discussed in the case studies.

III. Designing CIM Architectures for Domain-Specific Applications

Developing an appropriate CIM architecture and its design is strongly dependent on the targeted application domain; different applications with different requirements result in different designs. Next we will show this at the circuit level as well as at the system level. Then we will provide a list of potential applications that could benefit, and highlight the three cases studies for this paper.

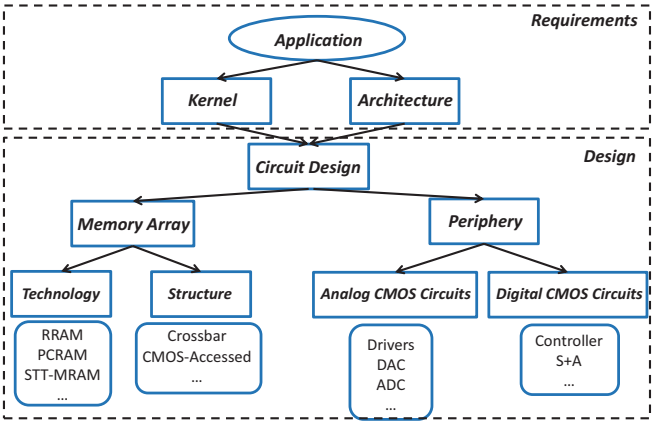


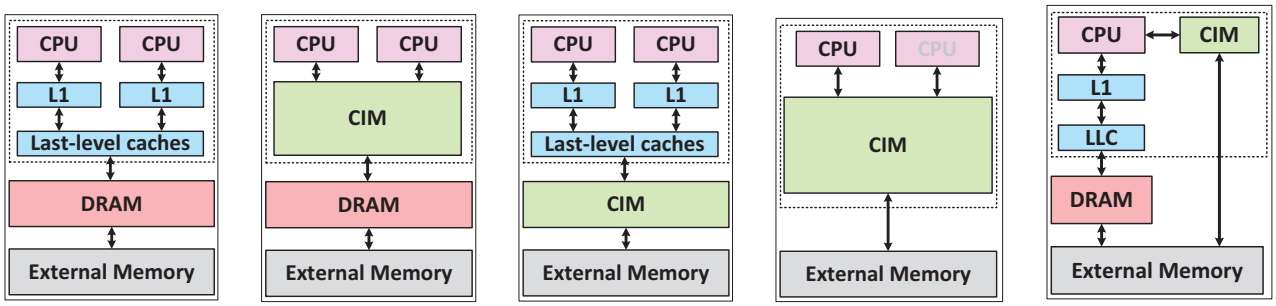
Fig. 2. CIM design flow.

A. Circuit-level design

CIM architectures have a great potential to improve the overall performance and energy of some emerging applications. These applications may have different requirements that may lead to different circuit designs; i.e., the circuit design of the array and the periphery. Therefore, to design an efficient CIM circuit, two pieces of information must be extracted from the targeted application: (1) the kernel(s) to accelerate, and (2) the CIM architecture.

- Kernel(s): the kernel is the most time/energy consuming function in the targeted application. It dictates the size of the operands, i.e., whether one-bit or multiple-bit numbers. For example, database applications require bit-wise logic functions, while compressed sensing requires arithmetic vector-matrix multiplication.
- Architecture: the architecture is mainly related to the location and type of inputs and outputs of the kernel; i.e., the architecture can be CIM-Ar, CIM-Ah, CIM-Pr or CIM-Ph. For example, database applications extract information from a database (stored in the memory) using queries [28]; hence it requires CIM-Pr (or CIM-Ar) architecture. Compressed sensing application converts a sensory signal (i.e., the first voltage input) using many pre-defined weights (i.e., the second resistive input) to another signal (i.e., a voltage output); hence, it requires e.g., CIM-Ph architecture [28].

After analyzing the kernel and suited architecture, we can start the designing the CIM circuit as shown in Fig. 2. A CIM circuit can be roughly divided into two parts, i.e., the memory array and the periphery. For the memory array, a suitable memristive technology such as RRAM, PCRAM, or STT-MRAM, should be selected based on the requirements of the endurance, resistance variation, etc. Thereafter, the structure of the array should be determined. It could be a crossbar containing only memristive devices or one with additional CMOS transistors that control the access, e.g., the one-transistor-one-memristor (1T1R) structure. For the periphery, the analog components including drivers, digital-analog converters (DACs) and analog-digital converters (ADCs) must be designed based on the needed functionality. In some cases, digital components such as controllers and shift-and-add (S+A) are required as well.



(a) Conventional (b) CIMC (c) CIMM (d) CIMU (e) CIMX

Fig. 3. Conventional architecture and potential CIM architectures.

B. System-level design

Fig. 3b, 3c, 3d, and 3e present four possible ways to embed a circuit design aiming at realizing CIM (denoted by *CIM*) into the conventional computer architecture of Fig. 3a; they are, respectively, CIM used as cache, as main memory, as complete memory hierarchy, and as accelerator [29]. They are discussed next.

Fig. 3b shows the CIM cache (CIMC) architecture where the memristive circuit replaces one or more cache levels. Besides the normal cache functionality, the memristive circuit can be used to perform in-memory operations as well. The in-memory operations are performed in the cache and prevent moving data towards processors. Hence, it reduces the cost (i.e., latency and power) of moving data between caches and processors, and vice-versa. As cache is more frequently accessed than main memory, the limited endurance and high read/write latency of resistive devices are the main bottlenecks of this approach [30]. Hence, this architecture makes sense only if the endurance is high enough ($> 10^{16}$).

Fig. 3c shows the "CIM main" (CIMM) where the memristive circuit replaces the main memory. The trade-off between capacity and performance is similar as in the approach as CIMC. Although a memristive device is naturally to be used as memory, a lot of efforts are still required for build CIMM. The challenges include high write latency, endurance limitation, and memory coherence management and complexity [31].

Fig. 3d shows the CIM universal (CIMU) where the memristive circuit replaces the entire memory hierarchy. One or multiple processors are used to execute the conventional instructions and fetch data directly from memristive devices; in-memory instructions can still be used to accelerate memory-intensive parts of an application. This reduces the pressure on the registers. However, to be competitive in terms of performance, CIMU must have a low write and read latency, and the endurance needs to be high enough to service write accesses.

Fig. 3e shows the CIM accelerator (CIMX) where the memristive circuit is used as an on-chip data-centric accelerator. CIM executes parts of the program that cannot be efficiently handled by conventional processors, e.g., a loop that consists of simple operations on a huge data set. This CIM accelerator differs from a traditional accelerator such as an FPGA or GPU in the amount of data that can be stored in the accelerator. This advantage alleviates the latency and energy cost caused by frequent data movement. Compared with an ALU operation, loading a word from on-chip SRAM or off-chip DRAM costs $50\times$ and $6400\times$ energy, respectively [32].

TABLE I
EXAMPLES OF POTENTIAL APPLICATIONS [27]

Kernels (operations)	CIM arch.	Applications
OR	Ar, Ah	
	Pr	**database** (bitmap indices, bitWeaving) [33]
AND	Ar, Ah	
	Pr	**database** (bitmap indices, bitWeaving), hyper-dimensional computing, language recognition, biosignal processing [34]
	Ph	
XOR	Ar	
	Pr	database (bitmap indices, bitWeaving), encryption, hyper-dimensional computing: language recognition, biosignal processing, k-mean clustering [35]
	Ph	CAM[36]
IMPLY, Majority	Ar, Ah	
Addition	Ar	
	Ah	temporal correlation, factorization [37]
	Pr	
Multiplication	Ar	
	Ph	
Vector-Matrix Multip.	Ph	**automata processor, compressed sensing and recovery**, image and signal processing, feature extraction, filtering, neural networks, pattern recognition, convolutional neural networks, recurrent neural networks, compressed sampling, image compression[25], [20], [28]
Vector-Matrix-Matrix Multip.	Ph	transitive closure[38]

C. Potential and targeted applications

Table I shows the different kernels (primitive operations) that can be implemented using memristive devices, the type of CIM architecture they enable, and some potential applications that make use of the kernels [27]. Note that not all kernels have been covered by this table and that a kernel may be implemented with different CIM circuit types. For example, an OR logic function can be implemented based on CIM-Ar (IMPLY), CIM-Ah (Majority logic), or CIM-Pr (Scouting logic); see also Fig. 1. The third column of the table lists different applications that could make use of the corresponding kernel. The three applications given in bold font will be the focus of this paper; these include database, compressed sensing and recovery, and automata processor.

- Database: This requires *bit-wise operations* such as OR, AND, and XOR. The CIM-Pr as the right architecture for this applications as it is less demanding from design and technology point of view [29], [39].
- Automata processor: Implementing such processor using memristive device based CIM requires the *binary vector-matrix multiplication*. Also here the CIM-Ph is the most suitable for this design [25].
- Compressed sensing and recovery: This requires *multi-level vector-matrix multiplication*. Here CIM-Ph is the right architecture due to the nature of the application and the implementation constraints [37].

In the rest of this paper we will first show the circuit implementation of the above three kernels. Thereafter, show the system design of the three CIM-P architectures together with their potential.

IV. CIM Circuit

Several CIM circuits based on memristive devices have been proposed, as shown in Fig. 1. Among them, we are particularly interested in CIM-P as these architectures do not reduce the lifetime of memristive devices. Next the implementation of the three kernels needed for the targetd applications in this paper will be discussed.

A. Bitwise logical operations

Scouting Logic [13] is of the schemes that fits in the CIM-Pr type, i.e., the inputs are resistive and the output is voltage. Fig. 4(a) shows which parts of the memory are used in Scouting Logic; the figure contains two memristive devices (M_1 and M_2) and a sense amplifier for a single column. Normally, when a cell (e.g., M_1) is read, a read voltage V_r is applied to its row and the switch S_1 connects. Subsequently, a current I_{in} flows through the bit line to the sense amplifier (SA). This current is compared to the reference current I_{ref}. If I_{in} is greater than I_{ref} (i.e., when M_1 has low resistance R_L), the output of the SA is logic 1. Similarly, when M_1 has high resistance R_H, the output is logic 0.

Instead of reading a single device at a time, performing Scouting Logic requires reading two devices simultaneously (e.g., M_1 and M_2 in Fig. 4a). As a result, the input current to the sense amplifier is determined by the equivalent input resistance. This resistance results in one of the three values: $\frac{R_L}{2}$, $\frac{R_H}{2}$ or $R_L//R_H \approx R_L$. By changing the value of I_{ref}, we can implement different gates. For an OR gate, I_{ref} should be set between $\frac{2V_r}{R_H}$ and $\frac{V_r}{R_L}$ as depicted in Fig. 4c. As a result, only when $R_1//R_2 = \frac{R_H}{2}$ the output is 0. Similarly, to implement an AND operation, I_{ref} should be set between $\frac{2V_r}{R_L}$ and $\frac{V_r}{R_L}$. The XOR operation requires two references and only when $R_1//R_2 \approx R_L$, the output is logic 1. Note that it is also possible to support multiple fan-in logic gates by setting proper reference currents.

SPICE simulation shows that Scouting Logic operates faster than other logic styles such as Resistive Boolean Logic [40] and Material Implication Logic [41]. In addition, the states of the memristive devices are kept unchanged during the execution of the logical operations. Therefore, their lifetime is not reduced. This is an important feature for today's immature memristive devices that suffer from low endurance.

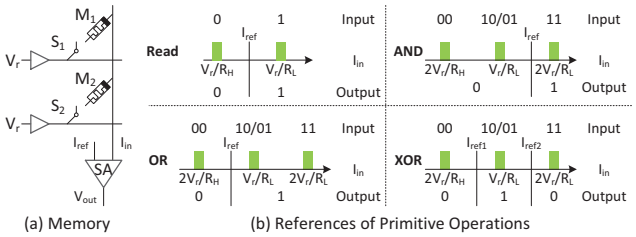


Fig. 4. Main idea of Scouting Logic [13].

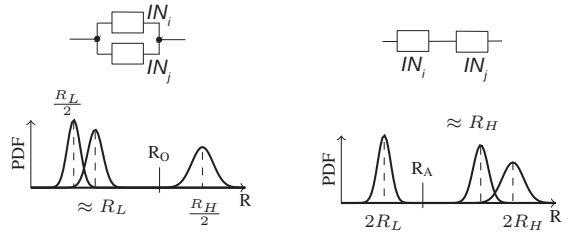


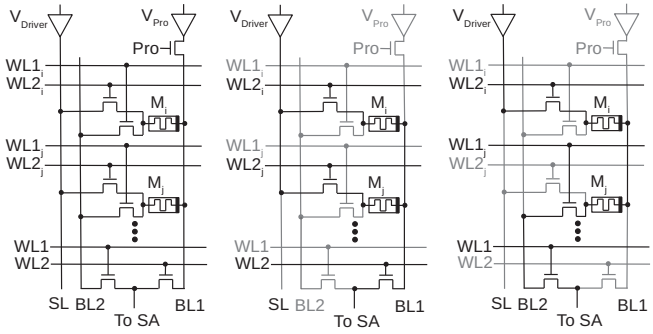
Fig. 5. Resistance distribution of two memristive devices in different configurations

However, and due to variations, the high and low resistance of memristive devices have a distribution rather than a single value; this may lead to incorrect operations. E.g., the resistance range of $\frac{R_L}{2}$ and $R_L//R_H \approx R_L$ overlap in Fig. 5a; hence, it is impossible to set a proper reference to implement an AND gate.

To solve this problem, we can change the connection style of the memristive devices being read during the logic operations in such way that they can be figured in series or in parallel depending on the operation to be performed (AND versus OR). Fig. 5b shows how the reading devices should be configured during the AND operation in order to reduce the impact of the variations; the equivalent resistance of two read elements being both in logic 1 will be $R_L + R_L = 2R_L$ while it will be $R_L + R_H \approx R_H$ when reading two elements one in logic 1 and one in logic 0. This solution is known as Enhanced Scouting Logic (ESL) [42]. The implementation of ESL as shown in Fig. 6a; each memristive device is connected to two transistors which control its connection style. In addition, another bit line is added to each column. For OR operations, the input memristive devices are connected in parallel as shown in Fig. 6b, which is similar to Scouting Logic. However, for AND operations, the input memristive devices are connected in series as shown in Fig. 6c. The robustness of ESL has been verified with Monte Carlo simulations. Even when the input memristive devices have a large resistance variation, ESL can still guarantee the operation correctness.

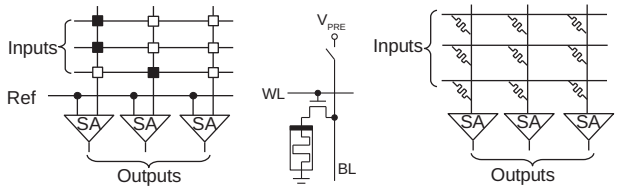
B. Binary vector-matrix multiplication

A memristive array can be used to compute a binary vector-matrix product [25], as illustrated in Fig. 7a; the figure consists of multiple bit lines each connected to a sense amplifier (SA). Each box at the crossing point of a bit line and a word line denotes a memristive cell as shown in Fig. 7b. One end of the cell is connected to the ground, and the other end to a transistor that act as a switch. The transistor connects the bit line (BL) and the memristive device, and it is controlled by



(a) Circuit (b) OR operation (c) AND operation

Fig. 6. ESL circuit and its working status [42].



(a) Binary VMM (b) A cell (c) Multi-level VMM

Fig. 7. Binary and multi-level vector-matrix multiplication (VMM) circuit [25], [28].

the word line (WL). The memristive device can be configured as a binary value. Low and high resistance represent logic 1 and 0, respectively. Therefore, the whole array represents a binary matrix.

The vector-matrix product operation is performed by special read instructions. During a read operation, the bit lines are first precharged to a high voltage. Subsequently, a binary vector is applied as input to the word lines; note that multiple word lines are activated simultaneously. Each column computes the inner product of the word line vector and a column vector. If at least one memristive cell is configured as a low resistance (logic 1) and its word line is active (representing logic 1), then the bit line discharges to a low voltage. In this case, the dot-product of the word line vector and the column vector results into logic 1. Otherwise the bit line remains high and the dot-product results into logic 0. Therefore, by comparing the voltage of the bit lines with a fixed reference voltage, the SAs can generate the product of an input vector and the configuration matrix.

Compared to similar binary vector-matrix multiplication circuits based on other memory technologies such as SRAM, SPICE simulations shows that the memristive device based one achieves shorter discharge time (hence faster) and lower energy consumption [25]. It is mainly because that the memristive device based circuit has less devices, especially less transistors, and hence smaller intrinsic capacitance. Note that the area of the memristive device based circuit is much smaller than the other ones as well.

C. Multi-level vector-matrix multiplication

Memristive arrays can also be used for multi-level vector-matrix multiplication. Fig. 7(c) shows how this can be implemented based memristive devices/crossbar. It consists of multiple rows and columns, with memristive devices placed at the crosspoints. Each row and column is connected to a driver and a sense amplifier, both supporting multi-level precision as each memristive device has multiple resistance levels. The matrix used in the computation is stored in the array as resistances while the vector is applied to rows in the form of voltages. The computation result is produced by the sense amplifiers by measuring the total current flowing into each of them.

	Dist.	Size	Year
A	55	Large	2016
B	23	Medium	2014
C	43	Small	2015
D	60	Medium	2016
E	25	Medium	2000
F	34	Medium	2001
G	18	Small	2012
H	30	Small	2011

(a) Original dataset

	A	B	C	D	E	F	G	H
Far	1	0	1	1	0	0	0	0
Near	0	1	0	0	1	1	1	1
Large	1	0	0	0	0	0	0	0
Medium	0	1	0	1	1	1	0	0
Small	0	0	1	0	0	0	1	1
New	1	0	0	1	0	0	0	0
Old	0	1	1	0	1	1	1	1
OR	1	0	1	1	0	0	0	0
AND	0	0	0	1	0	0	0	0

(b) Bitmap indexing

Fig. 8. Map a dataset to a bitmap [28].

Compared to the circuit presented in Section IV-B, this circuit supports a more complex computation. However, due to the resistance variation of memristive devices, the computing result may not be accurate enough; hence, suitable for specific domains such as approximate computing.

V. CIM-based System Design

The circuits presented in Section IV can be used for implementing various applications. In this section, we present three case studies, where memristive device based CIM enables novel architectures (the cases of database analytics and compressed sensing) and improves the implementation of an existing architecture (the case of automata processor).

A. CIM for database analytics

Database queries often involve many logical operations. Therefore, we can develop a CIM architecture using Scouting Logic that supports such bitwise logical operations. As the queries also contain other types of operations, we adopt the CIMX architecture that is shown in Fig. 3e where the CPU is used for processing such operations.

We use a bitmap indexing scheme to implement database queries. Fig. 8 shows an example; Fig. 8a contains the original database containing information related to planets and Fig. 8b its corresponding bitmap. Each column in the dataset (i.e., distance (dist.), size and the year of discovery) can be characterized using binary values. For example, the distance is considered to be *far* if it is longer than 40; otherwise, it is *near*. Note that the bitmap shown in Fig. 8b is transposed. Typical database queries consist of searching for specific data patterns, e.g., searching for entries that are of *medium* size and *new*. These queries are carried out by performing bitwise operations on the bitmaps; e.g., the above query can be acquired with an AND operation marked by the red boxes in Fig. 8b.

To evaluate the CIM solution, two analytical models are developed for conventional architecture and CIMX, respectively [28]. The conventional architecture uses Intel Xeon E5-2680 multicore as a baseline with 4 cores (each with a frequency of 2.5GHz, L1 cache of 32KB and L2 cache of 256KB) sharing 4GB DRAM memory. On the other hand,

CIMX contains a host processor (same as an individual core with 1GB DRAM) and a CIM unit with a size of equivalent to a size of 3GB DRAM.

The evaluation results show that the speedup of CIMX over the multi-core system reaches up to 35× for a problem size of 32 GB. In case that only 30% of the instructions are accelerated using the memristive circuit, the multi-core system consumes 6× more energy compared with the CIMX architecture.

B. Automata processor

Automata Processing (AP) is widely used in diverse fields such as network security, computational biology, and data mining; they offer significant advantages over the traditional von Neumann architectures as they as they enable computation-in-memory [43], [44]. An AP can be represented using a 5-tuple: $(Q, \Sigma, \delta, q0, C)$. Q denotes a set of states, Σ is a set of possible input symbols, δ is a function describing the set of possible transitions among the states, $q0$ is one of the states from Q and presents the *start state*, and C is a subset of Q and contains the *accepting states*. An automaton processes an input symbol sequence and produces a Boolean value A that indicates whether the input sequence is *accepted*. The processing is done by altering the set of active states P based on the input symbol and δ. If $P \cap C \neq \emptyset$, then the input sequence is accepted.

Fig. 9 shows the block diagram of RRAM-AP [25]. In every clock cycle, an input symbol I is processed using three major steps:

① **Input symbol matching.** All the states that have incoming transitions occurring on I are identified in this step. The N states are presented by column vectors called state-transit elements (STEs) which are pre-configured based on the targeted automaton. The decoder activates one of the word lines according to the input symbol I. If an STE has an incoming transition occurring on I, its corresponding output is logic 1; otherwise, it is logic 0. The outputs of all STEs are mapped to a vector called Symbol Vector $\mathbf{s}$.

② **Active state processing.** It generates all the possible states that can be reached from the currently active states (stored in Active Vector $\mathbf{a}$) based on the transition function (stored in the switching network), and stores the result in the Follow Vector $\mathbf{f}$.

③ **Output identification.** Accept Vector $\mathbf{c}$ is pre-configured based on the automaton's accepting states. This step checks the intersection of $\mathbf{a}$ and $\mathbf{c}$ to decide whether the input sequence is accepted.

As the STE matrix is huge, it is fragmented across the entire chip and we refer to each fragment as a tile. To determine the next states, RRAM-AP uses a hierarchical switching network that consists of global and local switches as shown in Fig. 10a. If the communication takes place inside a tile, only local routing is used; otherwise, global routing is used as well. The Active Vector $\mathbf{a}$ is divided into several groups each containing several signals entering global switches (represented by the box *G*). The outputs of the global switches combined with the initial vector $\mathbf{a}$ forms a *Global Vector* $\mathbf{g}$ and is used as the input to the local switches (represented by boxes *L1*, *L2*, and *L3*). The outputs of local switches form the Follow Vector $\mathbf{f}$.

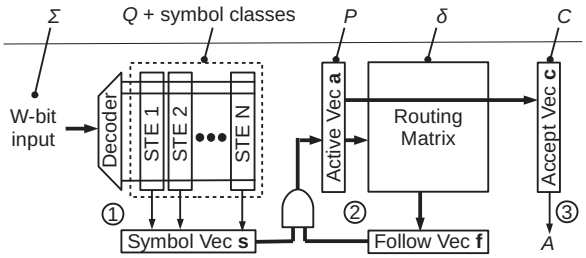
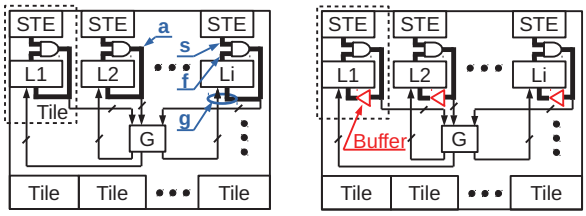


Fig. 9. Architecture of RRAM-AP [25].



(a) RRAM-AP (b) TDM-AP

Fig. 10. Switching network in RRAM-AP and TDM-AP [45].

The STE arrays, global, and local switches all conduct *binary vector-matrix product* operations, and hence they can be implemented using the memristive device based circuit presented in Section IV-B [25]. In fact, the simulation results reported in [25] even reveals that the RRAM-AP outperforms the two known hardware implementation of AP, namely APs based on SDRAMs [43] and SRAMs [44], in terms of speed, area and energy consumption.

By inserting buffers between the global and local switches as shown in Fig. 10b, we can change the switching network into two pipeline stages and further improve the working frequency of the chip. However, due to data dependency, the automata states can no longer be updated within a cycle. To grantee the processing correctness and fully utilize the hardware, multiple input streams enter the chip in a time-division multiplexing (TDM) manner. We refer to this design as TDM-AP [45]. A multiplexer and a demultiplexer are added to process the input and output signals. They can be control by the same selection signal with two additional buffers. Although these components increase the chip area by 2.8%, they raise the working frequency by 86%. With this improvement, TPM-AP can process input streams with a throughput of 24.0 Gbps, which is the highest reported number among all automata accelerators [45].

C. Compressed sensing and recovery

Compressing (and reconstruction) of a sparse high-dimensional signals is a common practice in in many applications such as image processing and data compression. Assume the original signal is $x_0 \in \mathbb{R}^N$, the compression process can be expressed as

$$y = Ax_0 + w$$

where $A \in \mathbb{R}^{M \times N}$ is a known measurement matrix, $w \in \mathbb{R}^M$ represents the measurement noise, and $y \in \mathbb{R}^M$ is the compression result. M is much smaller than N, and hence, the compression can reduce the bandwidth for data transfers.

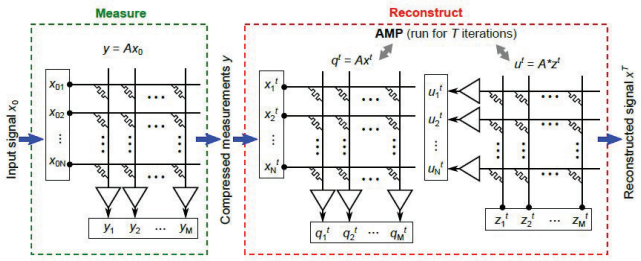


Fig. 11. CIM based compressed sensing with AMP recovery [47].

After data transferring, the original signal can be recovered using y and A. We use a first order approximate message passing (AMP) technique [46] that can be represented as

$$z^t = y - Ax^t + \frac{N}{M} z^{t-1} \langle \eta'_{t-1}(A^* z^{t-1} + x^{t-1}) \rangle$$

$$x^{t+1} = \eta_t(A^* z^t + x^t)$$

where $x^t \in \mathbb{R}^N$ represents the current estimate of x_0 at iteration t, $z^t \in \mathbb{R}^M$ the current residual, A^* the transpose of A, $\eta_t(\cdot)$ a scalar threshold function, $\eta'_t(\cdot)$ its derivative, $\langle \cdot \rangle$ the mean, and $x^0 = 0$. The final value of x^t provides the estimate of x_0. The AMP algorithm requires only multiplications and additions, making it suitable for a CIM implementation.

Inspecting the AMP algorithm reveals that the main cost comes from the vector-matrix multiplications (i.e., Ax^t and $A^* z^t$), assuming that $\eta_t(\cdot)$ and $\eta'_t(\cdot)$ involve only O(N) or less operations. Each vector-matrix multiplication requires O(MN) operations for dense matrix A. The remaining operations in the AMP algorithm are vector additions and multiplications that require O(N) operations. If we use the circuit presented in Section IV-C for vector-matrix multiplication, the complexity of AMP can be easily reduced from O(MN) to O(N). Note that the expectation is that in an memristive crossbar, matrix-vector multiplications can be performed with constant time complexity O(γ), where γ is independent of the crossbar size. The precise value of γ depends on settling time of the read current and the sending time of the peripheral circuitry. As a result, larger crossbars may eventually lead to higher γ since some of the readout circuitry may need to be shared across columns/rows and multiplexed. Nevertheless, significant speedup is expected when using normal memristive crossbar sizes.

Fig. 11 illustrates how the memristor based multi-level vector matrix multiplication (of Fig. 7(c)) can be used to implement the compress and recovery processes. The elements of A are mapped to conductance values on memristive devices. One possible mapping method is an iterative program-and-verify procedure [47]. After the matrix A is programmed in the crossbar array, the measurements y can obtained through a vector-matrix multiplication as illustrated in Fig. 11. When the AMP algorithm is performed, q^t=Ax^t and u^t=$A^* z^t$ is conducted using the (same) memristive crossbar. The vector q^t is computed by applying x^t as voltages to the rows and acquiring the result through the column sense amplifiers, and u^t by applying z^t as voltages to the columns and acquiring the resulting currents through the row sense amplifiers.

VI. Potential and Challenges

In general, memristive device based computing, if successful, will be able to significantly reduce the power consumption and enable massive parallelism; hence, increase computing energy and area efficiency by orders of magnitudes. This may enable new (economically affordable) computing paradigms such as Neuromorphic computing, Artificial neural networks, Bio-inspired neural networks, etc [6]. As memristive device based computing enables computing at the edge (e.g., at the sensors), a lot of application domains can strongly benefit from this computation; examples are IoT devices, wearable devices, wireless sensors, automotive, avionics, etc [48]. In short, if successful, memristive device based computing will enable the computation of currently (economically) infeasible applications, fuelling important societal changes.

Research on memsristive device based computing is still in its infancy stage, and the challenges are substantial at all levels, including material/technology, circuit and architecture, and tools and compilers.

- Materials/Technology: At these stage, there are still many open questions and aspects where the technology can help in making memristive device based computing a reality. Examples are device endurance [8], high resistance ratio between the off and on state of the devices [42], multi-level storage, precision of analog weight representation, resistance drift, inherent device-to-device and cycle-to-cycle variations, yield issues, etc.
- Circuit/Architecture: Analog Computation-in-Memory comes with new challenges to the design of peripheral circuits. Examples are high precision programming of memory elements, relatively stochastic process of analog programming, complexity of signal conversion circuit (digital to analog and analog-to-digital converters), accuracy of measuring (e.g., the current as a metric of the output), scalability of the crossbars and their impact on the accuracy of computing, etc.
- Tools/Compilers: Design automation is still an open question. Profiling and simulation tools can help the user to a) identify the kernels that can be accelerated on memristive device based computing and estimate the benefit, b) perform design exploration to select appropriate device technology/ architecture/ design/ etc.

As of today, most of the work in the public domain is based on simulations and/or small circuit designs. It is not clear yet when the technology will be mature enough to start commercialization for the first killing applications. Nevertheless, some start-ups on memristor technologies and their application are already emerging; examples are Crossbar, KNOWM, BioInspired, and GrAI One.

VII. Conclusion

This paper showed the potential of memristive devices in enabling new energy efficient Computation-In-Memory (CIM) paradigm through two cases studies (database analytics and compressed sensing), and in enabling efficient and cost effective implementation of some known architectures such as automata processing. The paper also highlighted some major challenges that have to be solved at different levels before CIM becomes a reality.

REFERENCES

[1] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Information Sciences*, vol. 275, pp. 314 – 347, 2014.

[2] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs *et al.*, "Memristor for computing: Myth or reality?" in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 722–731.

[3] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Amsterdam, Netherlands: Elsevier, Nov. 2017.

[4] S. Hamdioui, L. Xie, H. A. D. Nguyen *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015.

[5] X. Fu, M. A. Rol, C. C. Bultink *et al.*, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 813–825.

[6] C. D. James, J. B. Aimone, N. E. Miner *et al.*, "A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications," *Biologically Inspired Cognitive Architectures*, vol. 19, pp. 49 – 64, 2017.

[7] D. E. Nikonov and I. A. Young, "Overview of beyond-cmos devices and a uniform methodology for their benchmarking," *Proceedings of the IEEE*, vol. 101, no. 12, pp. 2498–2533, Dec 2013.

[8] S. Yu and P. Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, Spring 2016.

[9] M. Chang, A. Lee, P. Chen *et al.*, "Challenges and circuit techniques for energy-efficient on-chip nonvolatile memory using memristive devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, pp. 183–193, June 2015.

[10] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, no. 6, pp. 1165–1172, 2005.

[11] S. Li, C. Xu, Q. Zou *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 173:1–173:6.

[12] S. Kvatinsky, G. Satat, N. Wald *et al.*, "Memristor-based material implication (imply) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, Oct 2014.

[13] L. Xie, H. A. D. Nguyen, J. Yu *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2017.

[14] M. Hu, J. P. Strachan, Z. Li *et al.*, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Proceedings of the 53rd annual design automation conference*. ACM, 2016, p. 19.

[15] H. A. D. Nguyen, L. Xie, M. Taouil *et al.*, "On the implementation of computation-in-memory parallel adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2206–2219, Aug 2017.

[16] L. E. Guckert *et al.*, "Memristor-based arithmetic units," Ph.D. dissertation, 2016.

[17] A. Haj-Ali, R. Ben-Hur, N. Wald *et al.*, "Efficient algorithms for in-memory fixed point multiplication using magic," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[18] R. B. Hur and S. Kvatinsky, "Memory processing unit for in-memory processing," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2016, pp. 171–172.

[19] P. E. Gaillardon, L. Amarú, A. Siemon *et al.*, "The programmable logic-in-memory (plim) computer," in *Design, Automation & Test in Europe (DATE) Conference*, 2016, pp. 427–432.

[20] A. Shafiee, A. Nag, N. Muralimanohar *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, 2016.

[21] L. Xie, H. A. D. Nguyen, M. Taouil *et al.*, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, Oct 2015, pp. 335–342.

[22] S. R. Ovshinsky and B. Pashmakov, "Innovation providing new multiple functions in phase-change materials to achieve cognitive computing," *MRS Online Proceedings Library Archive*, vol. 803, 2003.

[23] F. Parveen, Z. He, S. Angizi *et al.*, "Hielm: Highly flexible in-memory computing using stt mram," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2018, pp. 361–366.

[24] A. Velasquez and S. K. Jha, "Parallel boolean matrix multiplication in linear time using rectifying memristors," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016.

[25] J. Yu, H. A. Du Nguyen, L. Xie *et al.*, "Memristive devices for computation-in-memory," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, March 2018, pp. 1646–1651.

[26] A. Velasquez and S. K. Jha, "Computation of boolean matrix chain products in 3d reram," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[27] M. Abu Lebdeh, U. Reinsalu†, H. A. D. Nguyen *et al.*, "Memristive device based circuits for computation-in-memory architectures," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019.

[28] S. Hamdioui, A. Sebstian, and S. Pande et al., "Applications of computation-in-memory architectures based on memristive devices," in *Design, Automation & Test in Europe (DATE) Conference*, 2019.

[29] H. A. Du Nguyen, J. Yu, M. Abu Lebdeh *et al.*, "A computation-in-memory accelerator based on resistive devices," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. Washington, DC, USA: ACM, Oct. 2019.

[30] J. Wang, X. Dong, Y. Xie *et al.*, "Endurance-aware cache line management for non-volatile caches," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 1, p. 4, 2014.

[31] Z. Pajouhi, X. Fong, and K. Roy, "Device/circuit/architecture co-design of reliable stt-mram," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015.

[32] A. Pedram, S. Richardson, M. Horowitz *et al.*, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, no. 2, pp. 39–50, 2017.

[33] V. Seshadri, D. Lee, T. Mullins *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *IEEE/ACM International Symposium on Microarchitecture*, 2017.

[34] A. Rahimi, P. Kanerva, J. d. R. Millán *et al.*, "Hyperdimensional computing for noninvasive brain-computer interfaces: Blind and one-shot classification of eeg error-related potentials," *International Conference on Bio-inspired Information and Communications Technologies*, 2017.

[35] Y. K. Rupesh, P. Behnam, G. R. Pandla *et al.*, "Accelerating k-medians clustering using a novel 4t-4r rram cell," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, Dec 2018.

[36] W. Chen, X. Yang, and F. Z. Wang, "Memristor content addressable memory," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2014, pp. 83–87.

[37] A. Sebastian, T. Tuma, N. Papandreou *et al.*, "Temporal correlation detection using computational phase-change memory," *Nature Communications*, vol. 8, no. 1, p. 1115, 2017.

[38] A. Velasquez and S. Jha, "Brief announcement: Parallel transitive closure within 3d crosspoint memory," 2018, pp. 95–98.

[39] H. A. Du Nguyen, J. Yu, M. A. Lebdeh *et al.*, "A classification of memory-centric computing," *ACM Emerging Technologies in Computing (JETC)*, 2019.

[40] L. Xie, H. A. D. Nguyen, M. Taouil *et al.*, "Boolean logic gate exploration for memristor crossbar," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, April 2016, pp. 1–6.

[41] J. Borghetti, G. S. Snider, P. J. Kuekes *et al.*, "'memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[42] J. Yu, H. A. Du Nguyen, M. Abu Lebdeh *et al.*, "Enhanced scouting logic: a robust memristive logic design scheme," in *2019 IEEE/ACM International Symposium on Nanoscale Architectures*, July 2019.

[43] P. Dlugosch, D. Brown, P. Glendenning *et al.*, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3088–3098, Dec 2014.

[44] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian *et al.*, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017.

[45] J. Yu, H. A. Du Nguyen, M. Abu Lebdeh *et al.*, "Time-division multiplexing automata processor," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, March 2019.

[46] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, pp. 18 914–18 919, 2009.

[47] M. Le Gallo, A. Sebastian, G. Cherubini *et al.*, "Compressed sensing with approximate message passing using in-memory computing," *IEEE Transactions on Electron Devices*, vol. 65, no. 10, pp. 4304–4312, 2018.

[48] K. Ma, X. Li, S. Li *et al.*, "Nonvolatile processor architecture exploration for energy-harvesting applications," *IEEE Micro*, vol. 35, no. 5, pp. 32–40, 2015.