Memristive Device Based Circuits for Computation-in-Memory Architectures

Abu Lebdeh, Muath; Reinsalu, Uljana; Du Nguyen, Hoang Anh; Wong, Stephan; Hamdioui, Said

DOI
10.1109/ISCAS.2019.8702542

Publication date
2019

Document Version
Accepted author manuscript

Published in
2019 IEEE International Symposium on Circuits and Systems (ISCAS)

Memristive Device Based Circuits for Computation-in-Memory Architectures

Muath Abu Lebdeh*, Uljana Reinsalu†, Hoang Anh Du Nguyen*, Stephan Wong* and Said Hamdioui*

*Computer Engineering Lab, Delft University of Technology, Delft, The Netherlands
†Department of Computer Engineering, Tallinn University of Technology, Tallinn, Estonia
Email: {m.f.m.abulebdeh,h.a.dunguyen,j.s.s.m.wong,s.hamdioui}@tudelft.nl uljana@ati.ttu.ee

***Abstract*—Emerging computing applications (such as big-data and Internet-of-things) are extremely demanding in terms of storage, energy and computational efficiency, while todays architectures and device technologies are facing major challenges making them incapable to meet these demands. Computation-in-Memory (CIM) architecture based on memristive devices is one of the alternative computing architectures being explored to address these limitations. Enabling such architectures relies on the development of efficient memristive circuits being able to perform logic and arithmetic operations within the non-volatile memory core. This paper addresses memristive circuit designs for CIM architectures. It gives a complete overview of all designs, both for logic as well as arithmetic operations, and presents the most popular designs in details. In addition, it analyzes and classifies them, shows how they result in different CIM flavours and how these architectures distinguish themselves from traditional ones. The paper also presents different potential applications that could significantly benefit from CIM architectures, based on their kernel that could be accelerated.**

I. Introduction

The modern-day data explosion is triggered by many factors and applications; e.g., content creation by consumers, genomics, Internet-of-things, (video) surveillance, and autonomous driving cars. This results in a certain pervasiveness of (large amounts of) data surrounding us in our daily lives, which needs processing to extract meaningful information to enhance our lives. The data ubiquity directly contradicts the large-scale utilization of a von Neumann architecture as it would require the continuous transportation of data towards and from central processing units. In particular, high energy consumption, limited bandwidth, guaranteed response times are obstacles that prove to be increasingly difficult to overcome [1], [2]. On a smaller scale (between CPUs and memory), we are talking about the memory, ILP, and power walls [1]. Consequently, researchers have been looking at computation-in-memory (CIM) architectures based on memristive devices [3]–[6] as a possible solution to the aforementioned problems. The CIM approach keeps data at the location of creation or consumption, and avoids any unnecessary transfer of data. Furthermore, all needed processing is "brought" to and performed at the data location, i.e., the memory or storage. Enabling such architecture requires the design of memristive based circuits being able to perform both logic as well as arithmetic operations in an efficient manner within the memory core.

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

Many memristive based circuits have been proposed in the last decade to enable the implementation of some primitive functions. Most of this work addressed logic bitwise operations such as NAND, AND, OR, NOR and XOR [7]–[12]. Recent work has focused more on some (limited) arithmetic operations such as vector-matrix multiplication [13]–[15], addition [16]–[19] and multiplication [16], [20], [21]. In addition, there is some limited work aiming at reviewing and analyzing such circuits from different perspectives. Maan et al. [22] focused on the memristive threshold logic circuits and their different implementations. Vourkas et al. [23] presented a general overview of the memristive logic circuits. Reuben et al. [24] discussed a framework for the stateful logic circuits, which are suitable for enabling CIM architectures. Zidan et al. [25] overviewed the potential of memristive devices in embedded memory design, biologically inspired computing, and CIM circuits. Ielmini et al. [26] discussed the emerging resistive devices to implement digital and analog CIM circuits. Although all this work tries to analyze the memristive circuit designs from different perspectives, none of them is able to extract the impact of these designs on CIM architecture and the potential applications they could enable. Inspecting these circuit designs reveals that they are not "generic" designs; i.e., they impose constrains on the kind of the CIM architecture they support. For example, Scouting logic [12] requires both inputs (operands) to be stored and aligned in the crossbar memory, whereas the vector-matrix multiplication [13] requires the matrix to be stored in the memory array and the vector to be provided via the the memory port. Understanding such constrains and their impact on the CIM architecture is the key towards defining both the details of the architecture and the applications that could be targeted.

This paper presents an overview of the memristive circuit designs (for executing primitive logic and arithmetic operations), classifies them to present the different types of CIM architectures they enable, and discusses some potential applications that could benefit (depending on the available kernels and their designs).

The rest of the paper is organized as follows. Section II classifies the CIM architectures based on the location where the memristive-based circuit design of a kernel (primitive

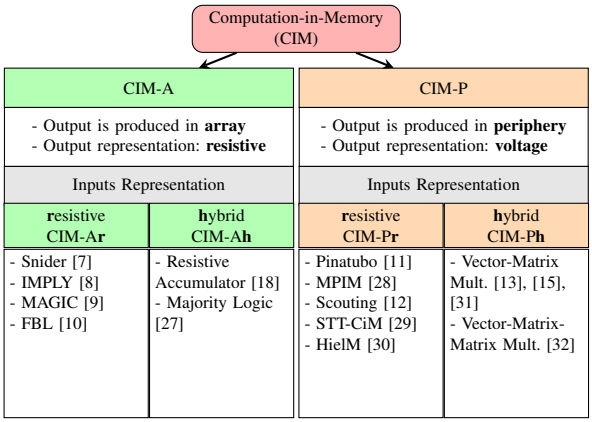


Fig. 1: CIM Classification

function) produces its results; either in the memory *A*rray (CIM-A) or in the *P*eripheral circuit of the memory (CIM-P). Section III discusses CIM-A class and reveals the details of two popular designs. Section IV presents the same, but then for CIM-P class. Section V shows the available kernels, their CIM architectures and the applications they could make use of. Finally, section VI concludes the paper.

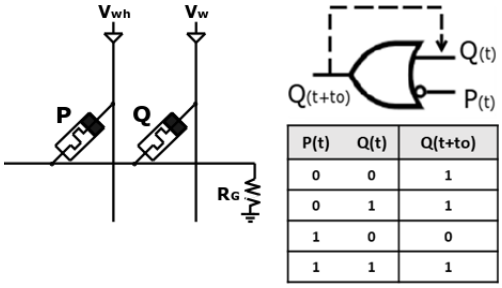
II. Classification

Computation-in-memory (CIM) aims at integrating processing within the memory itself; i.e., the computation takes place *within* the memory core. In addition, it aims at using the non-volatile memristive device technology (e.g., Resistive RAM), as it has practically no leakage and its nature enables both storage and computing capabilities.

As it is known, any memory core (including memristive memory cores) consists of a *memory array* and its *peripheral circuits*. Each memrsitive circuit design (aiming at implementing any logic or arithmetic operation in 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 Figure 1.

- *CIM-Array (CIM-A):* the computing result is produced within the memory array. Hence, the output should be stored in a memristive device in the array in form of a resistance state.
- *CIM-Periphery (CIM-P):* the computing result is produced within the peripheral circuitry. Given the fact that memory periphery is based on CMOS technology, the nature of the produced output is voltage.

The memristive circuit designs that enable CIM architecture (by implementing logic and arithmetic operations) are confined to hold at least one of its inputs (operands) in the array. In other words, the operator being executed within the memory needs to have *all operands* stored in the array (hence their logic values are *resistive*) or *only* part of the operands is stored in the array and the other part is received via the memory port(s) (hence their logic values are *hybrid*, i.e., resistive and voltage). This results into four sub-classes as shown in Figure 1: CIM-Ar, CIM-Ah, CIM-Pr and CIM-Ph; the additional letters 'r' and 'h' denote the nature of the inputs (operands), namely resistive and hybrid, respectively. The figure maps the existing memristive circuit designs into the classification. In the next section, some of these designs will be discussed as examples.



P(t)	Q(t)	Q(t+to)
0	0	1
0	1	1
1	0	0
1	1	1

Fig. 2: IMPLY gate

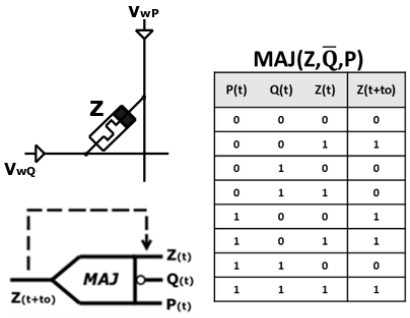
III. Memristive Circuit Design for CIM-A Architecture

CIM-A architecture uses the memristive CIM-Ar and CIM-Ah circuit designs to perform primitive logic (e.g., NOR) and arithmetic (e.g., addition) operations. The existing CIM-Ar circuit designs include Snider logic [7], implication logic [8], memristor aided logic (MAGIC) [9], and fast boolean logic (FBL) [10]. These designs implement primitively bitwise logic operations such as material implication, NOR, and NOT functions. On the other hand, the existing CIM-Ah circuit designs include majority logic [27] and resistive accumulator (nanoscale abacus) [18]. These two designs implement primitively logic operations such as majority function, and arithmetic operations such as addition.

The following subsections discuss two popular design examples in details: implication logic [8] as an example of CIM-Ar design, and majority logic [27] as an example of CIM-Ah design. Snider, MAGIC and FBL are quite similar to implication logic, while the resistive accumulator is similar to majority logic.

A. Implication Logic (CIM-Ar design example)

The implication logic (IMPLY) design executes the material implication as a universal logic function. An example of IMPLY gate is shown in Fig. 2, where the memristive devices P and Q present the operands. The design requires $R_{on}<<R_G<<R_{off}$ (R_{on} is the low Ohmic state and presents logic 1, and R_{off} is the high Ohmic state of the memristive device and presents logic 0). In this design, the device Q acts also as an output. Assuming that the devices P and Q are already programmed, the execution of IMPLY is done by applying two control voltages V_{wh} and V_w as shown in Fig. 2; note that the design requires $V_w > V_{th} > V_{wh}$, where V_{th} is the threshold voltage switching of the device. The magnitude of V_w across a device is sufficient to switch the state of the device from R_{off} to R_{on}, whereas $V_w - V_{wh}$ is not sufficient enough to switch the device state. Hence, after execution, and depending on the values of P and Q, the state of Q



MAJ(Z,$\overline{Q}$,P)

P(t)	Q(t)	Z(t)	Z(t+to)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Fig. 3: Majority logic gate

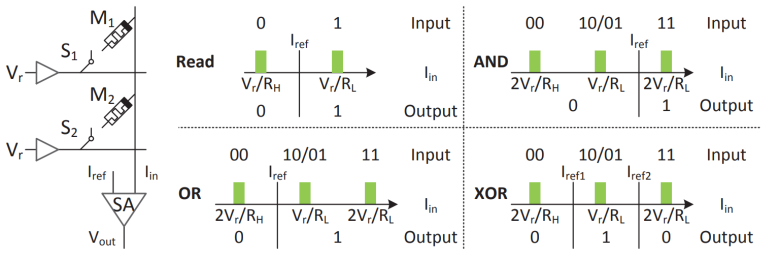


Fig. 4: Scouting logic gates

may or may not switch. E.g., if P=Q=0, the state of Q will switch from R_{off} to R_{on}. Therefore, this logic design is input destructive. More complex logic functions can be implemented by cascading IMPLY sequentially.

B. Majority Logic (CIM-Ah design example)

The majority logic design executes the majority logic function. An example of such gate is shown in Fig. 3; the circuit has two *voltage* inputs (P and $\overline{Q}$), and one resistive input Z that acts also as an output. Assuming that the device Z is already programmed, the execution of the function is done by applying the voltages V_{wP} and V_{wQ} to the top and bottom electrodes of the memristive device, respectively, as shown in Fig. 3. Depending on the state of the device Z and the applied voltages, the device may or may not switch. E.g., if initially Z=0 (R_{off}), and we apply V_{wP}>V_{th} and $V_{wQ} = 0$, then Z will switch to R_{on}. Therefore, this operation is destructive for input Z. This majority gate can be used to implement other CIM-Ah logic functions, such as OR and AND, by fixing one of the voltage inputs (P or Q) to a specific logic value. Other than the majority logic design, the resistive accumulator is another CIM-Ah circuit that utilizes multiple resistance storage of some memristive devices to implement addition (accumulation) operation primitively [4], [18].

C. Common Aspects

In summary, the existing memristive circuit designs that enable CIM-A architecture share the following aspects:

- Provide maximum level of parallelism within the memory, as execution is independent from sense amplifiers.
- Allow cascading of operations without being fed back to the array.
- Affect the memory endurance, as execution requires changing the state of the memristive devices.
- Require large drivers for computing, as execution requires high voltages to be applied to the memory array (high power consumption).
- Require redesigning the memory array to support computing, as the conventional optimized structure of memory array (including bitlines, wordlines and memory cells) may not allow for correct computation-in-memory operations.

IV. Memristive Circuit Design for CIM-P Architecture

The CIM-P architecture uses the memristive CIM-Pr and CIM-Ph circuit designs to perform primitive logic (e.g., OR) and arithmetic (e.g., vector-matrix multiplication) operations. The existing CIM-Pr circuit designs include Pinatubo [11], MPIM [28], Scouting [12], STT-CiM [29], and HielM [30]. These designs implement primitive logic operations such as OR, AND and XOR functions. On the other hand, the existing CIM-Ph circuit designs execute logic and arithmetic operations; including vector-matrix multiplication [13]–[15], [31] and vector-matrix-matrix multiplication [32].

The following subsections discuss two design examples in details: scouting logic [12] as an example of CIM-Pr design, and vector-matrix multiplication [15] as an example of CIM-Ph design. Pinatubo, MPIM, STT-CiM and Hielm are similar to scouting logic. Similarly, the other existing CIM-Ph designs (including designs described in [13], [31], [32]) are quite similar to the discussed example.

A. Scouting Logic (CIM-Pr design example)

Scouting logic design executes bit-wise OR, AND and XOR logic functions. The concept of scouting logic is shown in Fig. 4, where devices M_1 and M_2 represent the resistive operands. The output is produced as voltage by the sense amplifier. The execution of scouting logic operations is based on reading multiple rows simultaneously, and activating the appropriate reference current of the desired bitwise operation. The value of the reference current depends on the executed operation as shown in Fig. 4. The reference current of OR function must be selected between $\frac{2V_r}{R_{OFF}}$ and $\frac{V_r}{R_{ON}}$ (where V_r is the read voltage), whereas the reference current of the AND function must be selected between $\frac{V_r}{R_{ON}}$ and $\frac{2V_r}{R_{ON}}$ [12]. The XOR function requires two reference currents. The design of scouting logic is based on read operations; hence, it does not destruct input data (states) neither impact the endurance.

B. Vector-Matrix Multiplication (CIM-Ph design)

The vector-matrix multiplication design described in [15] is implemented using 1T1R (1 transistor and 1 resistive device) array as shown in Fig. 5; the inputs consist of the binary vector $\mathbf{a}$=$[a_1, a_2, a_3]$ provided via the wordlines of the array, and the binary matrix $\mathbf{B}$=$[b_{ij}]$ stored in the array. Each bit line will provide one element of the output vector. In this design, the

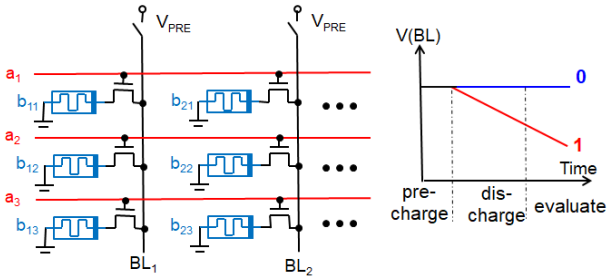


Fig. 5: Vector-Matrix multiplication in 1T1R array

multiplication is performed as AND function, and the addition as OR function. The execution starts by precharging the bitlines of the array, and then applying the vector elements to the wordlines. A precharged bitline will discharge (resulting in logic 1) only if at least one low ohmic path is created as a result of applying the vector elements; see Fig. 5. E.g., BL_1 will result in output 1 if a_1=1 and b_{11}=1 (low ohmic state). As it is based on the read operations, this design does not destruct the state of the matrix elements, neither reduce their endurance. More complex designs of vector-matrix multiplication (e.g., Dot Product Engine [13], and ISAAC [14]) utilizes analog circuits in the periphery (e.g., ADC and DAC) to perform arithmetic multiplications and additions using ohm's law and kirchhoff's current law, respectively.

C. Common Aspects

The existing memristive circuit designs that enable CIM-P architecture share the following aspects:

- Do not affect the memory endurance, as execution does not change the states of the memristive devices.
- Require small drivers, as execution requires low voltages to be applied to the memory array (low power consumption).
- Provide limited parallelism within the memory, as execution is dependent on the sense amplifiers. In the extreme case, each column may have a sense amplifier, resulting in a maximum parallelism at the cost area of overhead.
- Limit cascading of operations without feeding (or writing) the intermediate results back to the memory array, as the output is produced in the periphery.
- Entail less impact on the structure of memory array, as their designs focus on modifying the peripheral circuits to realize logic or arithmetic operations.

V. Potential Applications

The CIM architecture has a great potential to improve the overall performance and power consumption of at least some of emerging applications. Table I shows the different kernels (primitive operations) that can be implemented using memristive devices, the class of CIM architecture they enable, and some potential applications which make use of such kernels; hence, they could be accelerated. The kernels include bitwise operations (e.g., OR, XOR, and implication material functions) and arithmetic operations (e.g., addition, multiplication and vector-matrix multiplication operations).

TABLE I: Examples of potential applications

Kernels (operations)	CIM	Applications
OR	Ar, Ah	
	Pr	database (bitmap indices, bitWeaving) [33]
AND	Ar, Ah	
	Pr	database (bitmap indices, bitWeaving), hyper-dimentional computing, language recognition, biosignal processing [34]
	Ph	
XOR	Ar	
	Pr	database (bitmap indices, bitWeaving), encryption, hyper-dimentional computing: language recognition, biosignal processing, k-mean clustering [35]
	Ph	CAM [36]
IMPLY, Majority	Ar, Ah	
Addition	Ar	
	Ah	temporal correlation, factorization [4]
	Pr	
Multiplication	Ar	
	Ph	
Vector-Matrix Multip.	Ph	automata processor, image and signal processing, feature extraction, filtering, neural networks, pattern recognition, convolutional neural networks, recurrent neural networks, compressed sampling, image compression [14], [15], [37]–[41]
Vector-Matrix-Matrix multip.	Ph	transitive closure [42]

Note that a kernel may be implemented with different designs resulting in different CIM architectures. For example, OR logic function can be implemented using IMPLY, Majority logic, or Scouting logic, resulting in CIM-Ar, CIM-Ah and CIM-Pr, respectively. Note also that this is not applicable to all kernels, at least as of today. The third column of the table shows different applications that could make use of of the corresponding kernel/architecture. For example, the temporal correlation utilizes the addition kernel implemented in CIM-Ah architecture [4]; the one-time-pad cryptography encryption is accelerated using the XOR kernel implemented in CIM-Pr architecture [33]; the convolutional neural networks can be implemented more efficiently using vector-matrix multiplication in CIM-Ph architecture [14].

It is worth noting that most of the applications found in the literature investigate the potential of vector-matrix multiplication kernel in CIM-Ph architecture to boost the efficiency of computation. In addition, and as Table I clearly shows, it is not clear yet which applications could make use of some kernels and its associated architectures, as it is the case for OR kernel in CIM-Ar and CIM-Ah architectures (first row in the table).

VI. Conclusion

This paper has shown the importance of understanding and analyzing the nature of each memristive device based circuit design for logic or arithmetic operations. It does not allow only for selection of the right computation-in-memory architectures, but also for definition of the right applications that could significantly benefit from them.

REFERENCES

[1] D. A. Patterson, "Future of computer architecture," in *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*, 2006.

[2] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs, L. Xie, N. Wald, S. Joshi, H. M. Elsayed, H. Corporaal, and K. Bertels, "Memristor for computing: Myth or reality?" in *Design, Automation & Test in Europe (DATE) Conference*, 2017, pp. 722–731.

[3] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Design, Automation & Test in Europe (DATE) Conference*, 2015, pp. 1718–1725.

[4] A. Sebastian, T. Tuma, N. Papandreou, M. Le Gallo, L. Kull, T. Parnell, and E. Eleftheriou, "Temporal correlation detection using computational phase-change memory," *Nature Communications*, vol. 8, no. 1, p. 1115, 2017.

[5] H. A. Du Nguyen, J. Yu, L. Xie, M. Taouil, S. Hamdioui, and D. Fey, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017, pp. 1–10.

[6] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, p. 333, 2018.

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

[8] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, p. 873, 2010.

[9] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic - memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[10] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Fast boolean logic mapped on memristor crossbar," in *IEEE International Conference on Computer Design (ICCD)*, 2015, pp. 335–342.

[11] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[12] L. Xie, H. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, "Scouting logic: A novel memristor-based logic design for resistive computing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017, pp. 176–181.

[13] M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of bsb recall function using memristor crossbar arrays," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2012, pp. 498–503.

[14] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[15] J. Yu, H. A. Du Nguyen, L. Xie, M. Taouil, and S. Hamdioui, "Memristive devices for computation-in-memory," in *Design, Automation & Test in Europe (DATE) Conference*, 2018, pp. 1646–1651.

[16] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 1–14.

[17] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[18] 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.

[19] H. A. Du Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels, "Computation-in-memory based parallel adder," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2015, pp. 57–62.

[20] A. Haj-Ali, R. Ben-Hur, N. Wald, R. Ronen, and S. Kvatinsky, "Imaging-in-memory algorithms for image processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1–14, 2018.

[21] A. Haron, J. Yu, R. Nane, M. Taouil, S. Hamdioui, and K. Bertels, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *International Conference on High Performance Computing & Simulation (HPCS)*, 2016, pp. 759–766.

[22] A. K. Maan, D. A. Jayadevi, and A. P. James, "A survey of memristive threshold logic circuits," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 8, pp. 1734–1746, 2017.

[23] I. Vourkas and G. C. Sirakoulis, "Emerging memristor-based logic circuit design approaches: A review," *IEEE Circuits and Systems Magazine*, vol. 16, no. 3, pp. 15–30, 2016.

[24] J. Reuben, R. Ben-Hur, N. Wald, N. Talati, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky, "Memristive logic: A framework for evaluation and comparison," in *International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[25] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electronics*, vol. 1, no. 1, p. 22, 2018.

[26] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, p. 333, 2018.

[27] P. E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (plim) computer," in *Design, Automation & Test in Europe (DATE) Conference*, 2016, pp. 427–432.

[28] M. Imani, Y. Kim, and T. Rosing, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 757–763.

[29] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 3, pp. 470–483, 2018.

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

[31] A. Velasquez and S. K. Jha, "Parallel boolean matrix multiplication in linear time using rectifying memristors," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 1874–1877.

[32] 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.

[33] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 273–287.

[34] A. Rahimi, P. Kanerva, J. d. R. Milln, and J. M. Rabaey, "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, M. Miryala, and M. Nazm Bojnordi, "Accelerating k-medians clustering using a novel 4t-4r rram cell," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2709–2722, 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] M. Nourazar, V. Rashtchi, F. Merrikh-Bayat, and A. Azarpeyvand, "Towards memristor-based approximate accelerator: application to complex-valued fir filter bank," *Analog Integrated Circuits and Signal Processing*, vol. 96, no. 3, pp. 577–588, Sep 2018.

[38] Y. Long, T. Na, and S. Mukhopadhyay, "Reram-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2781–2794, Dec 2018.

[39] F. Qian, Y. Gong, G. Huang, M. Anwar, and L. Wang, "Exploiting memristors for compressive sampling of sensory signals," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2737–2748, Dec 2018.

[40] Y. Halawani, B. Mohammad, M. Al-Qutayri, and S. F. Al-Sarawi, "Memristor-based hardware accelerator for image compression," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2749–2758, Dec 2018.

[41] 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, inpress.

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