System Design for Computation-in-Memory
From Primitive to Complex Functions

Zahedi, Mahdi; Shahroodi, Taha; Custers, Geert ; Singh, Abhairaj; Wong, Stephan ; Hamdioui, Said

DOI
10.1109/VLSI-SoC54400.2022.9939571

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the 2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)

System Design for Computation-in-Memory: From Primitive to Complex Functions

Mahdi Zahedi, Taha Shahroodi, Geert Custers, Abhairaj Singh, Stephan Wong, Said Hamdioui
Department of Quantum and Computer Engineering, Delft University of Technology, Delft, The Netherlands
Email: {m.z.zahedi, T.Shahroodi, G.A.J.Custers, A.Singh-5, j.s.s.m.wong, s.hamdioui}@tudelft.nl

2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC) | 978-1-6654-9005-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/VLSI-SoC54400.2022.9939571

***Abstract*—In recent years, we are witnessing a trend moving away from conventional computer architectures towards Computation-In-Memory (CIM) based on emerging memristor devices. This is due to the fact that the performance and energy efficiency of traditional computer architectures can no longer be increased at the same pace as before. The main barriers which limit the performance and energy improvement are the memory and power walls. Thus far, the main effort from researchers is toward enabling CIM as an accelerator for specific applications. Consequently, this current application-specific nature/approach has put less emphasis on the potential general-purpose applicability of CIM, i.e., merging several accelerators into one that is less than the sum of the parts. In this paper, we demonstrate the CIM concept using a broader and generalized model. Considering this model, the state-of-the-art CIM-based logic and arithmetic primitive functions, which can be the building blocks for complex functions, are investigated. Besides, we present potential applications of CIM which provides insights into the challenges and opportunities of a generic CIM system design. Finally, we highlight the future directions regarding the construction of CIM-based systems.**

I. Introduction

Over the past decades, (1) improvements in transistor downscaling and (2) enhancements in processor (micro)architectures led to significant improvements in reducing energy consumption and increasing performance. However, both advances also led to their own set of issues. For example, transistor downscaling led to issues as reliability, leakage power, and the cost wall [1], [2]. On the other hand, microprocessors were (are still) confronted with (1) the memory wall and (2) the instruction-level parallelism (ILP) wall. The memory wall describes the performance gap between the processor and the main memory leading to increasingly longer data access times (latency) relative to the processor clock. Besides latency, the relative energy consumption of data accesses compared to basic processor operations also grew. The ILP wall made it increasingly harder to extract parallelism from applications due to their inherent sequential nature [3]. Consequently, in order to achieve the "next" major energy-consumption and performance improvements, alternative technologies as well as architectures are needed.

Recently, Computation-in-Memory (CIM) is being researched as a promising alternative to conventional Von-Neumann architectures. The motivation is to reduce the number of transactions between storage and compute units by enabling the storage unit to perform some primitive functions rather than solely relying on the compute units [4]. Until now, the main points of attention from the researchers were enabling more primitive functions in the memory while some accelerator-based designs were also proposed targeting a certain application or specific kernels. From the technology perspective, emerging non-volatile memories called memristors [5], [6] are the promising candidates for computational storage units. This is due to their special characteristics such as great scalability, high density, near-zero standby power, and non-volatility [5]. Current designs that exploit these technologies rely on an (analog) memristive crossbar array accompanied with analog and digital peripheries

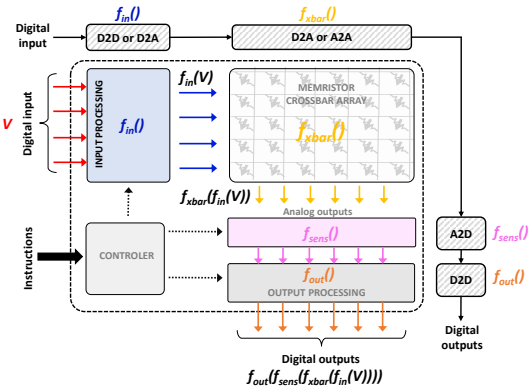

Fig. 1: Generic illustration of CIM-tile comprising four steps: 1) input processing 2) crossbar array 3) sensing 4) output processing

- collectively called a CIM-tile. The memory and compute functionalities are embedded in such a CIM-tile. Until now, many works focus on further improving the characteristics of these devices. In parallel, other works propose novel circuit designs based on memristive devices to enable more primitive functions inside the storage unit. However, these functions and their associated circuits have different requirements and implementations (e.g., different crossbar structures or voltage levels). As a few examples, some logic bitwise operations are proposed based on non-stateful logic where inputs served as voltage level while the output is presented as device resistance level [7]. On the other hand, MAGIC and IMPLY demonstrate a new approach to implement some stateful logic operations where the state of device used for both input and output [8], [9]. Similar to MAGIC, more bitwise logical operations are covered in [10] by modifying the MAGIC design. Alternatively, scouting logic [11] implements some bitwise logic operations by changing the reference of the Sense Amplifier (SA) and generating the result in the periphery rather than within the crossbar. Apart from logical operations, some accelerators are proposed [12]–[14] to exploit Vector-Matrix-Multiplication (VMM) operation inside the memristor crossbar by employing an Analog-to-Digital Converter (ADC). Despite the promising results provided by these designs, they are limited to the specific applications they were tailored for. Hence, there is a need to contemplate how to enable a CIM-based approach for general-purpose systems (e.g., servers) to make it accessible for as many as possible applications.

In an attempt to potentially define a generic CIM-tile design, one should first establish a better understanding of all potential capabilities of memristor-based circuits as well as a deep understanding of the data flows within a CIM-tile for a wide range of applications. More precisely, it is not our goal to define an

978-1-6654-9005-4/22/$31.00 ©2022 IEEE

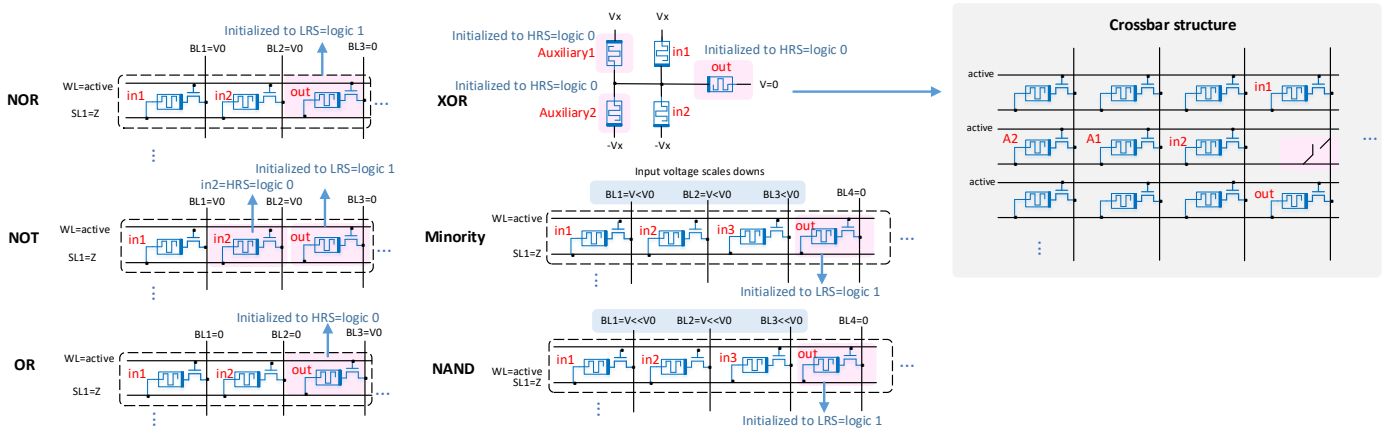

Fig. 2: Primitive functions supported inside the crossbar (f_{Xbar})

all-encompassing general-purpose CIM-tile. Instead, we explore the potential merger of several such designs into a more FU-like (function-unit-like) CIM-tile. In this paper, we:

1) Present a logical breakdown of the data flow within an abstract CIM-tile and its potential to create complex functions.
2) Investigate the state-of-the-art memristor CIM-based logic and arithmetic functions.
3) Demonstrate an overview of applications that have the opportunity to be executed inside the memory and discuss some of their important aspects concerning generalization.
4) Highlight the future direction regarding the construction of CIM computers.

We strongly believe that flexibly adapting the sequencing of the functionalities within a CIM-tile can lead to new and improved CIM-tile functions. We aim to:

- Inspire circuit-level designers to support new functionalities within the CIM-tile as well as to design a unified circuit that can support a wide range of functionalities rather than a one-to-one mapping of implementations and functions.
- Help architecture and application designers to find out the challenges and open research directions concerning a generalized CIM-tile.

II. Overview of Supported Functionalities in CIM with a Glance on Generalization

Memristor devices are non-volatile memories where their resistance levels can represent data. The device resistance level can be changed (programming the device) by applying proper voltage or current pulses. Similarly, in order to read the device without any disturbance, smaller voltage or current pulses are applied and the current passing through the device is sensed. By placing the memristor devices in a crossbar structure, not only they can be used as a storage unit, but some functionalities can be enabled in the memory as well. A CIM-tile comprises a memristor crossbar and its digital as well as analog peripheries. Figure 1 depicts a generic CIM-tile where the tile receives digital data as well as instructions [15] (or control signals) as inputs. The controller inside the tile is responsible to orchestrate all the circuits according to the instructions. The data may pass through four steps, 1) Input processing f_{in} 2) Crossbar array f_{Xbar} 3) Sensing f_{sens} 4) Output processing f_{out}.

Until now, some novel circuit designs were proposed to expand the number of functionalities that can be supported in each of these steps. Based on this knowledge, when an application is intended to be executed using CIM-tiles, the designer selects a certain function and its associate circuit for each of those steps. Hence, we will end up with an accelerator designed just for a certain application. Until now, the main focus of researchers regarding memristor-based CIM was toward designing accelerators that can only benefit specific applications. However, in order to have an as general as possible design, a wide range of functionalities should be supported in each of these steps at the same time, rather than having a different circuit for each function. In the following, we will describe what primitive functionalities are already supported in these four steps. Identifying these functions also enable us to build more complex functionalities in the CIM-tile.

A. Primitive functions in the crossbar (f_{Xbar})

Functions inside the crossbar can be classified into two main categories: 1) non-stateful logic where the inputs are applied as voltage to the both sides of the device and the resistance level can be switched according to the initial state of the device [7]. 2) stateful logic where the inputs and output are presented as device resistance level. Here, we focus on stateful logic since cascading primitive functions to implementing more complex ones is more straightforward.

NOR and NOT [8]: As illustrated in Figure 2, in order to perform the *NOR* function, two memristors are used to represent the two input operands and the result is written to a third memristor. At first, the output memristor should be initialized to Low Resistance State (LRS) representing a logic 1. Afterward, voltage level *V0* is connected to the bit-lines of the input memristors while the output memristor is connected to the ground. When one or two inputs are a logic 1, the voltage level *V0* should be sufficient enough to be able to change the resistance of the output memristor without switching the resistance of input memristors. By programming one input of the NOR gate to a logic 1, the NOT function can be implemented as well.

OR [10]: The schematic of an OR function is similar to the NOR function. However, the output memristor is initialized to the High resistance State (HRS). In addition, despite the NOR operation, the bit-lines of input memristors are grounded while the bit-line of the output memristor is connected to voltage V0. Similar to a NOR, V0 should be determined to fulfill this functionality without being destructive.

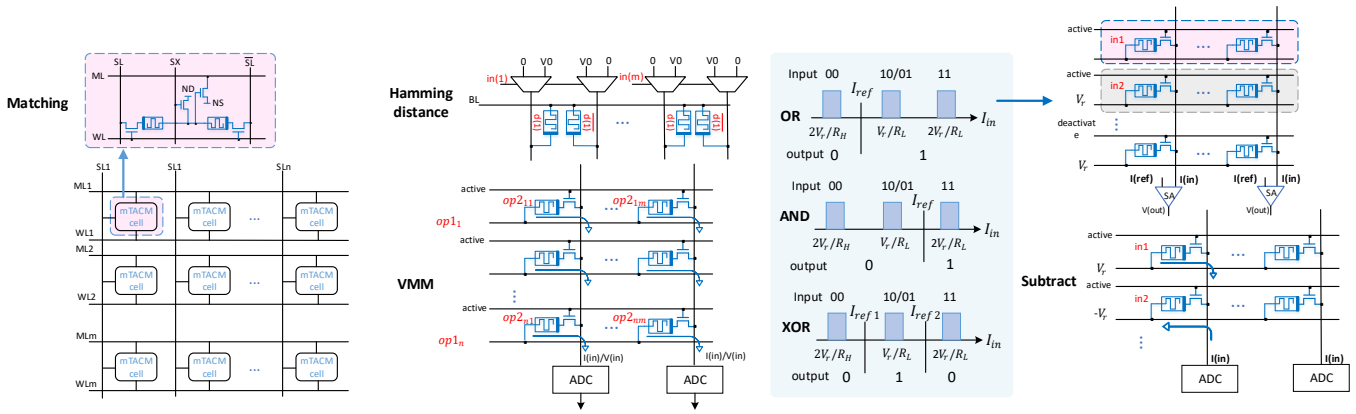

Fig. 3: Primitive functions supported inside the sensing step (f_{sens})

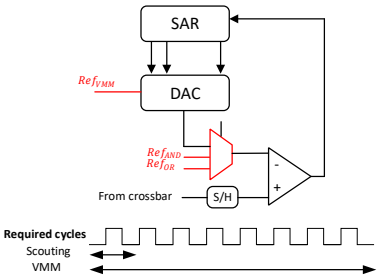

Fig. 4: A potential design to have a unified circuit supporting VMM and Scouting logic

XOR [16]: This function requires five memristor devices among which two of them are auxiliary memristors. Figure 2 depicts how these memristors should be connected and initialized. In the case memristor devices, representing input operands, are both either ON or OFF, the common node between memristor devices is virtually ground, keeping the output memristor at its original state. Otherwise, the voltage on the common node is Vx which switches the output memristor to the LRS state with the help of auxiliary memristors. The crossbar structure and the way data mapped to it to support this function is also demonstrated in Figure 2. More information can be found in [16].

Minority and NAND [10]: Considering the NOR function, having one of the input memristors as a logic 1 is enough to switch the resistance of the output memristor. Accordingly, by reducing the voltage of V0, we can determine the minimum number of ON resistances to be able to switch the output device. This can effectively implement the minority function. In addition, by further reducing the voltage V0, at one point, all the memristor inputs should be ON to be able to switch the output memristor, which can implement the NAND function.

By cascading the aforementioned functions inside the crossbar, more complex functions such as arithmetic addition or multiplication can be implemented. However, as demonstrated before, different functions require different voltages and crossbar structures. This becomes even more challenging when the sensing step and its functions are taken into consideration. Therefore, finding a unique solution that can cover most of these functions with minimum cost is a valuable step toward a generalized memristor-based CIM-tile. In addition, having a smart tile controller to flexibly deal with different execution flows and patterns of data mapping is an essential part of this system.

B. Primitive functions in the sensing step (f_{sens})

Despite the functions mentioned before, here the results of the functions are generated by using sense amplifier (SA) or analog-to-digital converters (ADC). In other words, the current generated by the crossbar is assigned to different logical values which in turn can implement different functions.

Hamming Distance [36] and Matching [23]: Using a CIM-tile, the Hamming Distance function can be implemented. To calculate the distance between two vectors, one vector and its complementary value have to be programmed to the crossbar. As depicted in Figure 3, the second vector is given to the select signals of the pair of multiplexers placed on top of the crossbar. This provides data as well as its complementary value to the memristors holding the first vector and its complement, respectively. Subsequently, based on the similarity of the two vectors, different current levels flow into the bit-line, which is sensed to determine the distance of the two vectors. Similarly, a Match function can be performed by just using a SA. A potential crossbar structure for this function [23] is depicted in Figure 3.

Vector-Matrix Multiplication [12]: Vector-Matrix Multiplication is the main functionality for which CIM-tile is intended to be employed. To perform this function, first, the matrix has to be programmed to the crossbar. Subsequently, the vector is applied to the select line of the crossbar. Finally, all the elements of the output vector are obtained in one shot after converting analog current/voltage levels to their digital value using ADCs.

Scouting Logic (AND-OR-XOR) [11]: Logical operations can be implemented in the sensing step. In this approach, both operands as vectors are programmed to the crossbar. As illustrated in Figure 3, depends on where to put a reference(s) of SA, logical OR, AND, and XOR functions can be implemented. The main advantage of this approach compared to implementing these functions inside the crossbar is reducing the number of device programming. This is important due to the endurance problem and high programming energy of memristor devices.

Subtraction [38]: Element-wise subtraction can be performed in the crossbar. The proper voltage level has to be provided to the row corresponding to the negative operand to be able to drain the current from the bit-line. Hence, the remaining current on the bit-line represents the result of this function.

TABLE I: List of some potential applications/algorithms/kernels that can be executed using memristor-based CIM

Domain		Applications/Algorithms/Kernels				
Network		Automata processor for network security [17]	Packet classification [18]	SAMCRA [19]	IP-routing	-
	operation	AND	Matching	VMM	Matching	-
Security		Security primitives (PUF-RNG) [20]	Encrypted communication [21]	AES [22]	Regular expression matching [23]	Salsa20/ChaCha20
	operation	Analogue properties	Analogue properties	Add-XOR-Multiply	Matching	Addition - XOR
Approximate computing		Image property calculation [24]	Approximate matrix multiplication	Quantum simulations [25]	-	-
	operation	Addition - Subtraction	VMM	VMM	-	-
Mathematics		Partial differential equation solver [26]	Eigenvector calculation [27]	Markov chain	Vector-cosine similarity	-
	operation	VMM	VMM	VMM	VMM	-
Signal and image processing		Compressed sensing [28]	Discrete Fourier Transform [29]	Convolutional Image filtering	Huffman encoding	GLCM feature extraction
	operation	VMM	VMM	VMM	Matching	Addition
Classification and Prediction		Recurrent neural network [30]	Convolutional neural network [12]	Hyperdimensional computing [31]	Reservoir computing	Auto-regressive models
	operation	VMM	VMM	AND - OR - VMM	Matching	VMM
Database		Query-06 of the TPC-H [32]	Pattern matching in databases	Transitive closure	Bitmap indices	BitWeaving
	operation	Bitwise XOR	Matching	VMM	OR - AND - XOR	OR - AND - XOR
Bioinformatics		DNA sequencing or allignment [33]	Genome Base-calling [34]	Genome Profiling [35]	-	-
	operation	XOR - VMM	VMM	VMM	-	-
Graph processing		Graph Clustering [36]	Breadth first search [37]	VLSI routing	PageRank	All-pair-shortest-path
	operation	VMM - NOR - HD	VMM	VMM	VMM	VMM

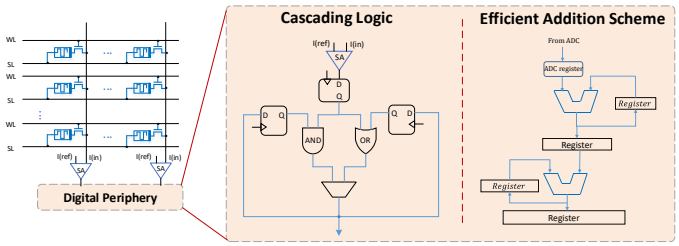

Fig. 5: Examples of digital periphery circuits designed to deliver certain functionality (f_{out})

Based on the example of functions in the sensing step, when a designer intends to perform an application using one of those functions, having the competence to do other functions becomes challenging due to the necessity of different array structures, circuits, and constraints. As an example, Scouting logic just requires SA with two determined references with a constraint on activating two rows at a time. However, in order to generalize the design for other functions like VMM, ADCs have to be placed and more rows potentially should be activated. In order to have a unified circuits performing both of the functions, Figure 4 depicts a simple solution on how to modify SAR ADC to be able to use it as a SA for Scouting logic. Therefore, based on the function decides to be performed, proper reference voltage is selected and required cycles are considered.

C. Prevalent functions in the digital periphery (f_{out})

Due to the complexity of the application execution flow and the limitation of functionalities in CIM, some parts of the application should be performed by the host. However, researchers try to place customized circuit in digital peripheries to expand the functions can be done in the memory and gain more energy/performance improvement rather than simply assuming unsupported parts of an application are assigned to the host processor.

Many circuits have been considered in the digital periphery for different applications. Figure 5 shows two examples: 1) Cascading logic circuit [39] is customized digital periphery to perform sequential AND and OR logic in the periphery targeting database applications. 2) Efficient Addition Scheme [40] is a novel structure to replace conventional *shift and add unit* targeting matrix-matrix multiplication kernel. Based on the applications, some designers also placed subtractor, sigmoid function, vector processing unit, and register files in the digital periphery. Due to the flexibility that can be provided by digital design compared to the crossbar and sensing step, more attention should be given to this step for a generalized CIM-based design. One example of periphery design that can support a wide range of functionalities is employing Lookup Table (LUT) [38], [41]. However, still more research has to be conducted to have a smart and general solution with a detailed execution model for a wide range of applications.

D. Examples of function in the input processing step (f_{in})

Despite other steps, this step has not been investigated much yet in the literature. However, some digital circuits were designed in [15] to carry out some functions. First, a buffer using parallel-in-serial-out registers is considered to provide a clear separation of tile from outside and facilitate data communication considering datatype size and limited resolutions for Digital to Analog Converters (DAC). Second, a digital circuit is placed to be able to flexibly support different patterns of crossbar rows selection. Other examples for this step could be data quantization, downscaling, and alignment.

III. Exploring Potential Applications

Besides the existing operations, a designer should have a good insight into the applications that have the potential to be executed using a memristor-based CIM system. Table I presents the list of some applications/algorithms/kernels where a considerable part of the program can be performed inside the memristor-based memory and potentially gain performance and energy from the concept of CIM. The examples in the table cover a wide range of domains. This helps the designer to expand the generalization of the design by finding out the corner cases regarding application requirements and constraints. In the following, some important aspects to be considered for a generic CIM system design

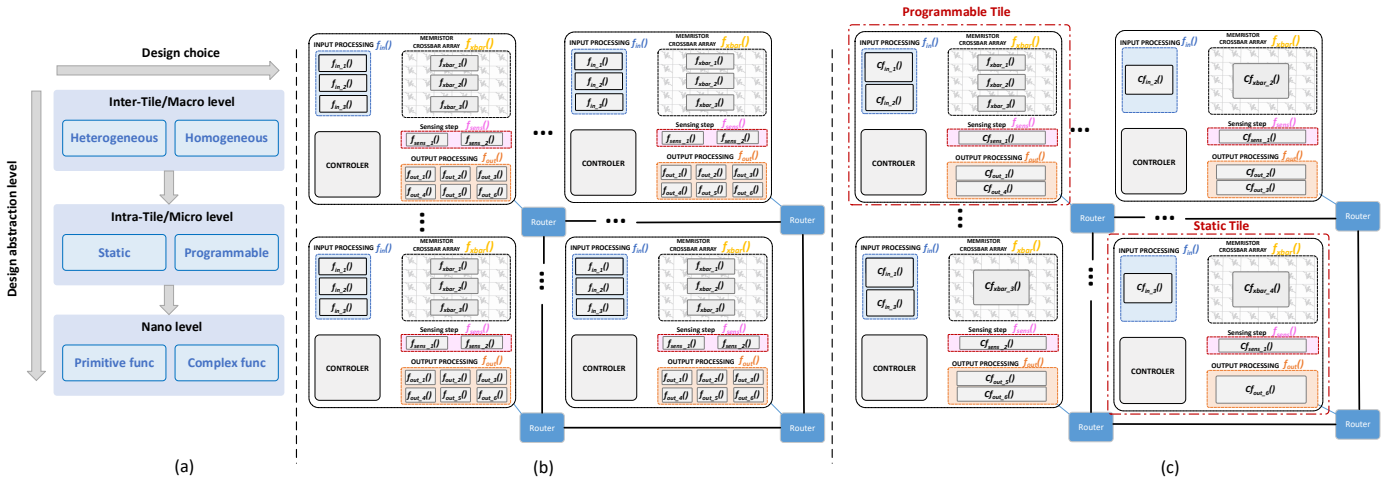

Fig. 6: (a) Possible design choices in different abstraction levels (b) homogeneous design where each tile is capable and generalized enough to support a wide range of functionalities vs (c) heterogeneous design where tiles are customized to execute different kernels efficiently

targeting wide range of applications are listed:

❶ Different applications require computation over different data types and data type sizes. A generalized design should enable computation for some basic **data types**. Besides, a design that can flexibly and efficiently support computation for different datatype **sizes** is valuable since it can potentially lead to energy/performance improvement.

❷ As shown in Table I, different applications consist of different **operations**. However, according to what was discussed in Section II, a designer should realize a system to cover more functionalities. This results in offloading fewer parts of the program into the host and in turn less communication between host and storage unit.

❸ Based on the execution flow of applications, they may require different **patterns** on crossbar rows and columns selection. In other words, during the execution time, an application may operate on different locations of a crossbar. Therefore, this is important that a generalized system provides this flexibility and accessibility to a programmer.

❹ Memristor crossbars have limited dimensions which is usually much less than what is required for an application. Considering that, the application should be mapped to several tiles. Hence, a detailed implementation of **tile communication** which can flexibly cope with different data flows is essential.

❺ Variability is one of the challenging aspects of memristor devices, which can end to **accuracy** reduction. Some applications can tolerate this inaccuracy while others need precise computation. Therefore, supporting algorithms to dynamically adjust the accuracy of computations based on the application requirements can be another research direction for a generalized system.

❻ Although it is desire to offload more parts of an application on CIM-tiles and reduced data movement more, still there might be some parts that have to be executed on the host side. Accordingly, this indicates the necessity of **communication** and synchronization of CIM-tiles with the host. Based on the existing works in the literature, a detailed design regarding this aspect is missing and requires more attention from the community.

IV. Future Direction of CIM-Tile and Conclusion

As discussed before, despite application-specific designs, in order to have a memristor-based CIM-tile system capable of executing a wide range of applications, more complexity and requirements are raised. To move toward such a system, different strategies can be applied with respect to different abstraction levels. As illustrated in Figure 6 (a), the system can be broken into at least three levels. Starting from the lowest level, Nano-level, a CIM-tile can comprise just a primitive function in one of the steps while no major functionalities supported in other steps in the tile. An example of this can be MAGIC [8] or Scouting [11] where just primitive functions supported in the crossbar f_{Xbar} or sensing steps f_{sens}, respectively. On the other hand, different functionalities can be supported at different steps of the tile in order to end up with a complex function. In the second level of abstraction, Intra-Tile/Micro level, a Static or Programmable CIM-tile can be targeted. In the case of Static, each step of the tile is designed (and **C**ustomized) to provide a specific functionality while in the case of programmable tile, more than one function are considered for the tile's steps. Regarding the design choices in the highest abstraction level and considering a spectrum, on one side, a designer can aim at a homogeneous CIM-tiles system where the tiles are the same and each can support a wide range of functionalities and requirements. On the other side of this spectrum, we will have a heterogeneous design where CIM-tiles are customized for different purposes and they all together can provide this generality for the programmer. Finally, in the middle, we also may have a hybrid solution where limited heterogeneity is taken into account while CIM-tiles are pushed to support different functions as much as possible.

Figure 6 (b and c) illustrates the design choices for inter-tile abstraction level. Considering a homogeneous design, in order to execute an application, more resources are available since the tiles are generalized and can cope with different functions. However, this might come at cost of an area, energy, and latency overhead. In addition, due to the complexity of the tile, an advanced controller has to be employed. Rather, considering heterogeneous design, while fewer resources might be available for a certain application, the tiles might have higher

energy, area, and performance efficiency. Nonetheless, due to the heterogeneity, a more complex task offloading scheme might require. Finally, the communication between tiles may impose more overhead when the tiles selected for an application are located far from each other. The design choice heavily depends on how flexible the circuit in CIM tile will be concerning the applications' functionalities and requirements. No matter which approach is chosen, flexibly controlling this storage unit both in the granularity of inter- and intra-tiles for different execution flows is a crucial aspect. This can be achieved either all in hardware or partially at the software level with the help of a compiler (or scheduler) [42]. This becomes more important for heterogeneous designs when different workloads have to be mapped and scheduled for specific CIM-tiles. In conclusion, 1) expanding the functionality of a CIM-tile with a unified circuit, 2) flexibly interfacing the CIM-tiles to each other as well as the host, and 3) an advanced controlling system, able to flexibly cope with different scenarios, are the key aspects toward a generalized CIM-tile design.

REFERENCES

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

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

[3] H. Esmaeilzadeh *et al.*, "Dark Silicon and the End of Multicore Scaling," *IEEE Micro*, vol. 32, no. 3, pp. 122–134, 2012.

[4] S. Hamdioui *et al.*, "Applications of Computation-in-Memory Architectures Based on Memristive Devices," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 486–491.

[5] B. Govoreanu *et al.*, "10*10nm2 Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *2011 International Electron Devices Meeting*, Dec 2011, pp. 31.6.1–31.6.4.

[6] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, pp. 1–16, 2020.

[7] L. Xu *et al.*, "Memristor-Based Efficient In-Memory Logic for Cryptologic and Arithmetic Applications," *Advanced Materials Technologies*, vol. 4, no. 7, p. 1900212, 2019.

[8] S. Kvatinsky *et al.*, "MAGIC—Memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

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

[10] S. Gupta *et al.*, "FELIX: Fast and Energy-Efficient Logic in Memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[11] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 176–181.

[12] A. Shafiee *et al.*, "ISAAC: A convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[13] A. Ankit *et al.*, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 715–731.

[14] G. Karunaratne *et al.*, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[15] M. Zahedi *et al.*, "Tile Architecture and Hardware Implementation for Computation-in-Memory," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 216–221.

[16] N. TaheriNejad, "Sixor: Single-cycle in-memristor xor," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 5, pp. 925–935, 2021.

[17] J. Yu, *et al.*, "Time-division multiplexing automata processor," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 794–799.

[18] Y.-D. Kim, *et al.*, "A high-speed range-matching tcam for storage-efficient packet classification," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 6, pp. 1221–1230, 2009.

[19] P. Van Mieghem and F. A. Kuipers, "Concepts of exact QoS routing algorithms," *IEEE/ACM Transactions on networking*, vol. 12, no. 5, pp. 851–864, 2004.

[20] H. Nili *et al.*, "Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors," *Nature Electronics*, vol. 1, no. 3, pp. 197–202, 2018.

[21] B. Cambou *et al.*, "Cryptography with analog scheme using memristors," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 4, pp. 1–30, 2020.

[22] M. Masoumi, "Novel Hybrid CMOS/Memristor Implementation of the AES Algorithm Robust Against Differential Power Analysis Attack," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 7, pp. 1314–1318, 2020.

[23] C. E. Graves, *et al.*, "In-Memory Computing with Memristor Content Addressable Memories for Pattern Matching," *Advanced Materials*, vol. 32, no. 37, p. 2003437, 2020.

[24] S. Muthulakshmi, *et al.*, "Memristor augmented approximate adders and subtractors for image processing applications: An approach," *AEU-International Journal of Electronics and Communications*, vol. 91, pp. 91–102, 2018.

[25] I.-A. Fyrigos *et al.*, "Memristor Hardware Accelerator of Quantum Computations," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 799–802.

[26] M. A. Zidan *et al.*, "A general memristor-based partial differential equation solver," *Nature Electronics*, vol. 1, no. 7, pp. 411–420, 2018.

[27] Z. Sun *et al.*, "In-Memory Eigenvector Computation in Time O (1)," *Advanced Intelligent Systems*, vol. 2, no. 8, p. 2000042, 2020.

[28] M. Le Gallo *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.

[29] R. Cai *et al.*, "Memristor-based discrete fourier transform for improving performance and energy efficiency," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 643–648.

[30] H. Tsai *et al.*, "Inference of Long-Short Term Memory networks at software-equivalent accuracy using 2.5 M analog Phase Change Memory devices," in *2019 Symposium on VLSI Technology*. IEEE, 2019, pp. T82–T83.

[31] T. Shahroodi *et al.*, "Demeter: A Fast and Energy-Efficient Food Profiler using Hyperdimensional Computing in Memory," *IEEE Access*, 2022.

[32] H. A. D. Nguyen *et al.*, "A computation-in-memory accelerator based on resistive devices," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 19–32.

[33] F. Zokaee *et al.*, "Finder: Accelerating fm-index-based exact pattern matching in genomic sequences through reram technology," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2019, pp. 284–295.

[34] Q. Lou *et al.*, "Helix: Algorithm/Architecture Co-design for Accelerating Nanopore Genome Base-calling," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 293–304.

[35] T. Shahroodi *et al.*, "KrakenOnMem: A Memristor-Augmented HW/SW Framework for Taxonomic Profiling," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–14.

[36] M. Imani *et al.*, "DUAL: Acceleration of Clustering Algorithms Using Digital-based Processing In-Memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 356–371.

[37] L. Song *et al.*, "GraphR: Accelerating graph processing using ReRAM," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 531–543.

[38] D. Fujiki *et al.*, "In-memory data parallel processor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.

[39] I. Giannopoulos *et al.*, "In-Memory Database Query," *Advanced Intelligent Systems*, vol. 2, no. 12, p. 2000141, 2020.

[40] M. Zahedi *et al.*, "Efficient Organization of Digital Periphery to Support Integer Datatype for Memristor-based CIM," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 216–221.

[41] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 541–552.

[42] M. Zahedi *et al.*, "MNEMOSENE: Tile Architecture and Simulator for Memristor-based Computation-in-memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, pp. 1–24, 2022.