MNEMOSENE

Tile Architecture and Simulator for Memristor-based Computation-in-memory

Zahedi, Mahdi; Lebdeh, Muah Abu; Bengel, Christopher; Wouters, Dirk; Menzel, Stephan; Le Gallo, Manuel; Sebastian, Abu; Wong, Stephan; Hamdioui, Said

DOI
10.1145/3485824

Publication date
2022

Document Version
Final published version

Published in
ACM Journal on Emerging Technologies in Computing Systems

Citation (APA)
Zahedi, M., Lebdeh, M. A., Bengel, C., Wouters, D., Menzel, S., Le Gallo, M., Sebastian, A., Wong, S., & Hamdioui, S. (2022). MNEMOSENE: Tile Architecture and Simulator for Memristor-based Computation-in-memory. *ACM Journal on Emerging Technologies in Computing Systems*, *18*(3), Article 44. https://doi.org/10.1145/3485824

MNEMOSENE: Tile Architecture and Simulator for Memristor-based Computation-in-memory

MAHDI ZAHEDI and MUAH ABU LEBDEH, Delft University of Technology, The Netherlands
CHRISTOPHER BENGEL and DIRK WOUTERS, RWTH Aachen University, Germany
STEPHAN MENZEL, Peter-Grünberg-Institut, Germany
MANUEL LE GALLO and ABU SEBASTIAN, IBM Research–Zurich, Switzerland
STEPHAN WONG and SAID HAMDIOUI, Delft University of Technology, The Netherlands

In recent years, we are witnessing a trend toward in-memory computing for future generations of computers that differs from traditional von-Neumann architecture in which there is a clear distinction between computing and memory units. Considering that data movements between the central processing unit (CPU) and memory consume several orders of magnitude more energy compared to simple arithmetic operations in the CPU, in-memory computing will lead to huge energy savings as data no longer needs to be moved around between these units. In an initial step toward this goal, new non-volatile memory technologies, e.g., resistive RAM (ReRAM) and phase-change memory (PCM), are being explored. This has led to a large body of research that mainly focuses on the design of the memory array and its peripheral circuitry. In this article, we mainly focus on the tile architecture (comprising a memory array and peripheral circuitry) in which storage and compute operations are performed in the (analog) memory array and the results are produced in the (digital) periphery. Such an architecture is termed compute-in-memory-periphery (CIM-P). More precisely, we derive an abstract CIM-tile architecture and define its main building blocks. To bridge the gap between higher-level programming languages and the underlying (analog) circuit designs, an instruction-set architecture is defined that is intended to control and, in turn, sequence the operations within this CIM tile to perform higher-level more complex operations. Moreover, we define a procedure to pipeline the CIM-tile operations to further improve the performance. To simulate the tile and perform design space exploration considering different technologies and parameters, we introduce the fully parameterized first-of-its-kind CIM tile simulator and compiler. Furthermore, the compiler is technology-aware when scheduling the CIM-tile instructions. Finally, using the simulator, we perform several preliminary design space explorations regarding the three competing technologies, ReRAM, PCM, and STT-MRAM concerning CIM-tile parameters, e.g., the number of ADCs. Additionally, we investigate the effect of pipelining in relation to the clock speeds of the digital periphery assuming the three technologies. In the end, we demonstrate that our simulator is also capable of reporting energy consumption for each building block within the CIM tile after the execution of in-memory kernels

The results presented in this article have been obtained in the framework of the project "Computation-in-memory architecture based on resistive devices" (MNEMOSENE), which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780215.

Authors' addresses: M. Zahedi, Delft University of Technology, Mekelweg 4; email: M.Z.Zahedi@tudelft.nl; M. A. Lebdeh, Delft University of Technology, Mekelweg 4; email: M.F.M.AbuLebdeh@tudelft.nl; C. Bengel, RWTH Aachen University; email: bengel@iwe.rwth-aachen.de; D. Wouters, RWTH Aachen University; email: wouters@iwe.rwth-aachen.de; S. Menzel, Peter-Grünberg-Institut; email: st.menzel@fz-juelich.de; M. Le Gallo, IBM Research–Zurich; email: ANU@zurich.ibm.com; A. Sebastian, IBM Research–Zurich; email: ASE@zurich.ibm.com; S. Wong, Delft University of Technology, Mekelweg 4; email: J.S.S.M.Wong@tudelft.nl; S. Hamdioui, Delft University of Technology, Mekelweg 4; email: S.Hamdioui@tudelft.nl.

1550-4832/2022/01-ART44 $15.00
https://doi.org/10.1145/3485824

considering the data-dependency on the energy consumption of the memory array. All the source codes are publicly available.

CCS Concepts: • **Computer systems organization** → **Parallel architectures**; *Redundancy*; • **Computing methodologies** → Simulation support systems;

Additional Key Words and Phrases: Computation in-memory, memristor, architecture, simulator, ISA

ACM Reference format:
Mahdi Zahedi, Muah Abu Lebdeh, Christopher Bengel, Dirk Wouters, Stephan Menzel, Manuel Le Gallo, Abu Sebastian, Stephan Wong, and Said Hamdioui. 2022. MNEMOSENE: Tile Architecture and Simulator for Memristor-based Computation-in-memory. *J. Emerg. Technol. Comput. Syst.* 18, 3, Article 44 (January 2022), 24 pages.
https://doi.org/10.1145/3485824

1 INTRODUCTION

Modern-day applications such as machine learning, image processing, and encryption are widely employed on real-time embedded systems as well as back-end data centers. Nowadays, since the amount of data has to be processed for these types of applications increased considerably (e.g., BERT has around 110 million parameters [15]), performance and power consumption of the systems that fulfill these processes has become crucial and draw increasingly more attention. Currently, traditional von-Neumann architectures have been stretched to attain adequate performance at reasonable energy levels, but are clearly showing limitations for further improvements. The main limitation is the conceptual separation of the processing unit and its memory, which makes the data movement between memory and processing unit the main performance and energy bottleneck [25, 36]. For instance, communication between off-chip DRAM and the processor takes up to 200 cycles [62] and it consumes power more than 2 orders of magnitude than a floating-point operation [1, 14, 17]. Consequently, new architectures need to be sought to achieve the performance levels required for these new applications by means of mitigating this costly communication.

To address this challenge, early researches [12, 47] suggested integrating more on-chip memory next to the processing unit, which imposes significant fabrication costs to the system. One of the promising solutions to tackle this challenge is performing computation within or near the memory rather than moving the data to the processing unit. Some valuable works focus on near-memory computing, especially after 3D stacking technology was introduced, in which computation was performed in the DRAM chip by adding extra logic next to the memory structure [2, 3, 10, 19, 34, 45]. However, recently, **computation in-memory (CIM)**, as a concept, has gained a huge interest from the research community in combining memory storage and actual computing in the same memory array. This is achieved by exploiting special characteristics of emerging non-volatile memories called memristors such as **resistive RAM (ReRAM)** [26], **phase change memory (PCM)** [40], and **spin-transfer torque magnetic RAM (STT-RAM)** [30]. No matter which technology is used for fabrication, memristor technology has great scalability, high density, near-zero standby power, and non-volatility [26, 63, 74]. Accordingly, memristor technology with the aforementioned characteristics opens up new horizons toward new ways of computing and computer architectures.

Until now, the main focus of researchers was to enhance the characteristics of memristor devices such as latency and endurance [8, 29, 32, 51, 74]. Researchers have already proposed different innovative circuit designs based on memristor devices to exploit their capabilities of co-locating computation and storage together [7, 31, 37, 42–44, 46, 61, 76]. Moreover, within a single memory array as well as at inter-array level huge parallelism can be flexibly achieved as each memory tile becomes a powerful computation core. It was demonstrated that due to these two main features

of memristor-based designs, significant energy and performance improvement can be gained [28]. It is widely accepted that the dot-product (and, in turn, the matrix-matrix multiplication) operation is the most suited for the memristor-based designs. Consequently, convolutional and deep neural network are the potential applications that have been widely studied by the researchers to exploit memristive crossbar structures [9, 57, 64, 66, 67, 73]. However, researchers have also proposed other types of operations, e.g., Boolean operations [56, 60, 72] or arithmetic operations like additions [58]. More information on available researches regarding device characteristics and potential applications for in-memory computing can be found in References [49, 53].

Having said this, there is no work in the research community that allows for easy comparison of these supported operations at the application-kernel level between different technologies (e.g., ReRAM, PCM, and STT-RAM) nor is it possible to emulate complex operations, e.g., matrix-matrix multiply, when the underlying technology does not allow for direct implementation. Furthermore, the interactions between the analog memory array and its supporting digital periphery is largely overlooked.

In this article, we generalized the existing CIM-tile architectures (a CIM-tile comprises the analog array itself and all necessary analog and digital periphery) to introduced a generic execution model that allows operations in the digital periphery and the analog array to operate in parallel through pipelining. In this model, a new **instruction set architecture (ISA)** is introduced with the twin objectives of orchestrating digital and analog components of memristor tiles and bridging the gap between high-level programming languages and the CIM architecture. By rescheduling the proposed fine-grained instructions, maximum flexibility on executing a program targeting the CIM-tile can be attained. Our compiler written in C++ performs instruction lowering from high-level kernels, which are supposed to execute on the CIM tile down to in-memory instructions at compile time. The compiler is aware of the architecture configuration, the technology constraints, and the datatype size requested by the application. Subsequently, based on this information, the required in-memory instructions are generated in a proper sequence.

The execution model is aimed to work with different memristor technologies as well as different configurations/circuits for peripheral devices. To accomplish design space exploration targeting performance and energy, we designed a modular simulator written in SystemC in which the designer can track all the data and control signals. Timing and power numbers of the modules within the CIM architecture are obtained from low-level models. These numbers configure specific parameters of the simulator.

In conclusion, the contributions of our work are summarized below:

- we generalize existing CIM tile architectures and execution models for in-memory computing, which enables an application to leverage a variety of in-memory operations. Additionally, we proposed a pipelining approach for an architecture that has digital and analog modules, which can have widely varying (unbalanced) latencies. Finally, the architecture can be easily extended with additional peripheral modules.
- we define a new and generic in-memory ISA for the aforementioned design to obtain maximum flexibility while dealing with different constraints, configurations, and requirements. Furthermore, we developed a compiler that is able to translate higher-level operations (e.g., defined in higher-level programming languages) into a sequence of instructions in our ISA. The ISA can also be extended when additional peripheral modules are added.
- we develop a fully parameterized simulator that is capable of executing the newly introduced instructions and simulate our CIM tile architecture. Design parameters (e.g., height/width of memory array, type and number of peripheral modules like the **Analog-to-Digital Converter (ADC)**) as well as technology-dependent parameters (e.g., ReRAM/PCM/STT-RAM,

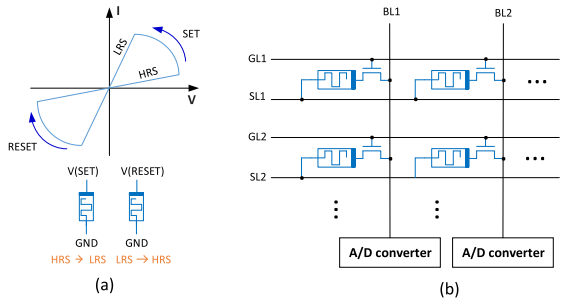

Fig. 1. (a) Memristor behavior. (b) Memristor 1T1R crossbar structure.

latencies, number of bits per cell, power utilization per module, clock frequency of the digital periphery) are supported.
- we perform an initial set of design-space explorations based on well-accepted design parameters and current-day memristor technologies and present the results.

2 BACKGROUND AND RELATED WORK

This section first provides a preliminary background on memristor devices and the different ways that they employed to perform computation. Subsequently, the existing architectures exploiting in-memory computing using memristor devices as well as available simulators will be discussed.

2.1 Memristor Devices

Memristor devices are categorized as non-volatile memory in which data can be represented as resistance levels. Figure 1(a) depicts the current-voltage behavior of a bipolar memristor device. For a unipolar device, SET and RESET would occur at the same voltage polarity. The resistance states are altered by the application of suitable voltage or current pulses during a write or programming operation. Usually the term SET is used to denote the transition from a **low resistive state (LRS)** to a **high resistive state (HRS)** while the term RESET means the transition from a HRS to a LRS. Since ReRAM and STT-MRAM are both bipolar devices, one voltage polarity SETs the device while the other RESETs it. The polarity depends on the definition of the reference potential. PCM is a unipolar device, which means that SET and RESET occur for the same voltage polarity but at different amplitudes [69]. By utilizing the LRS/HRS resistance states we can store the logic levels "1" and "0," respectively [22, 29]. In the case that the device can be programmed to multiple resistance levels, obviously it can hold more than one bit of data [35, 74]. Reading resistance state of the device requires a small voltage applied to the device, which prevents data disturbance, and then the current through or the voltage across the device can be sensed.

2.2 Memristor-based Computation Circuits

Several papers proposed circuit designs that employ memristive devices to be used as either computation or storage unit. In addition to their appealing characteristics in terms of area and energy, some of these designs have demonstrated the ability to enable both computing and storing data at the same location; i.e., data is not shuttled back and forth between the memory and the processor [27]. Most of these novel CIM circuit designs can be classified based on the location of the results generated by the memory core [39], which consists of a *memory array* and *peripheral circuits*. Figure 2 illustrates this classification. According to the classification, if the result of computation is produced outside the memory core, then it is referred to as ***Computation-Outside-Memory (COM)***. In this case, the computation can take place either in extra logic circuit inside the

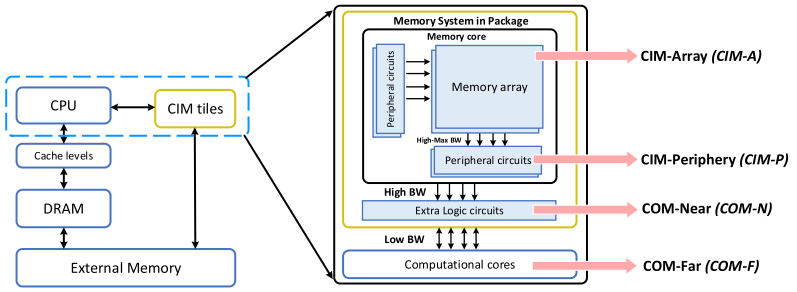

Fig. 2. Potential high-level computer architecture using CIM tiles.

memory **System-in-Packages (SiP)** or like a traditional architecture in computational cores (CPU or GPU), which are referred to ***COM-Near (COM-N)*** or ***COM-Far (COM-F)***, respectively. The two main classes of CIM circuit designs using **memristive devices** where the results produced inside the memory core are described in the following:

CIM-Array (CIM-A): The result of the computation is produced within *the array* and represented by resistances [7, 37, 42, 55, 59, 61]. CIM-A circuit designs have generally the following advantages in common:

- They achieve the maximum bandwidth of "transferring" data between the computation and storage units, as both computation and storage happen physically within the same memory array.
- They perform computation independently from the sense amplifiers. This can provide high parallelism.
- They can potentially enable logic cascading of universal functions.

However, they share some limitations as well such as:

- Frequent write operations can lead to endurance and energy issues.
- Performance overhead due to the high latency of device programming and logic cascading to achieve complex functions.
- Significant design efforts both for the memory array (e.g., the memory array must support wider bit-lines to handle the excess of currents when multiple values are written simultaneously) and its controller (e.g., the controller requires complex state machines to apply a long sequence of different control voltages to the memory array).

CIM-Periphery (CIM-P): The result of the computation is produced within *the periphery* and represented by voltage levels [33, 54, 72]. CIM-P circuit designs generally have the following advantages in common:

- They do not exacerbate the endurance of the memory array as the memory states do not change during and after the computation.
- They require relatively less redesign efforts in the memory array; i.e., the currents during computation are lower or close to the write current.
- They allow high to maximum bandwidth depending on the complexity and area of the periphery.

However, they share some limitations such as:

- They may have to share sense amplifiers or analog-to-digital converters per multiple bit-lines, which can degrade the performance.

- They cannot cascade functions without read/write operations.

The pros/cons listed for the two classes point us to CIM-P as a prime candidate to be exploited in our CIM-tile organization. In this decision, we considered parameters like endurance, performance, and the complexity of the controller administrates the memory array. Among different types of operations, vector logical operations, as well as vector-matrix multiplication, have a great potential to be executed on the crossbar structure. Figure 1(b) depicts the structure of 1T1R memristor crossbar for the CIM-P circuit class. For example, to perform logical operations using this structure, first the desired rows have to be activated and the read voltage has to be dropped across each cell. Subsequently, according to the scouting logic [72], just by changing the reference input of the **sense amplifiers (SAs)**, different Boolean operations (AND, OR, XOR) can be performed. For vector-matrix multiplication, the matrix has to be stored inside the crossbar and then the vector encoded to analog voltages is applied to source lines of the corresponding rows. In the end, the analog result is received for all the columns in a single shot.

2.3 In-memory Architectures and Simulators

Some other studies are proposing in-memory accelerator designs using memristor devices that mostly focused on accelerating **Neural Networks (NN)**. PRIME [13] designed a new architecture based on ReRAM devices for NN applications in which some portion of crossbars can be configured for computation and the rest for storage. By reducing the communication between memory and processing units, their estimation demonstrates significant improvement in energy and performance. Similar to PRIME, ISAAC [54] proposed a full-fledged accelerator with memristor crossbars for CNNs, which exploits pipelining in the level of application. Recently, new in-memory instructions were proposed [23]. In this promising work, in-memory instructions are complete operations that have to be performed on crossbars based on the **Single Instruction, Multiple Data (SIMD)** execution model to enjoy massive data parallelism. The basic operations supported in the crossbar are multiplication, addition, and subtraction. However, to perform logical operations, it was just assumed that **Look Up Tables (LUTs)** can be placed beside the crossbars. Similar to ISAAC, PUMA [4] proposed an accelerator tailored for vector-matrix-multiplication. The designed architecture has three piplined stages (fetch, decode, and execute) and the new in-memory instructions defined there are mainly responsible for communicating data between memory units or performing scalar operations in digital peripheries. Finally, ReVAMP [6] proposed a general-purpose **Very Long Instruction Word (VLIW)** architecture platform based on ReRAM devices. In this work, two high-level instructions are defined to indicate *read* and *compute* operations and control the architecture based on them. Although some abstract architectures are proposed in the literature, there is still a lack of studies on the detailed execution model of in-memory architectures. In this work, we propose new in-memory instructions to be able to orchestrate the analog and digital circuits inside the tile. The proposed instructions are defined to be as generic as possible, which enables the programmers to build other operations on top of them.

Many factors have to be considered when an in-memory memristor-based architecture has to be designed, since they can have a non-negligible effect on the performance, energy, area, or even accuracy of the system. Therefore, the necessity of optimization and design space exploration at reasonable simulation time derive the researchers to develop high-level simulators for this new type of architecture. There are a few simulators among which NVSim [16] and NVMain [50] are the first memory-oriented simulators designed for non-volatile memories. These tools, which are inspired by CACTI [68], can estimate the access time, energy, and area of non-volatile memories. Similar to NVSim, MNSIM [71] proposed a behavioral simulation platform for neuromorphic accelerators that estimates design parameters using analytical equations. Recently, another system-level simulator [41] was proposed, in which by using probability functions to capture the

accuracy of ReRAM cells, the behavior of an application in terms of accuracy can be evaluated. Contrary to existing works that estimate execution times, our execution model allows us to build a tile-level cycle-accurate simulator by actually executing in-memory instructions—after translating high-level kernels to our ISA. Furthermore, our simulator allows the user to track all the control signals and the content of crossbar/registers and produces more accurate performance and energy consumption results. Finally, due to the modular programming of the simulator, the user can easily investigate different memristor technologies, circuit designs, and more advanced crossbar modeling (e.g., considering read/write variability).

3 CIM-TILE ARCHITECTURE

A CIM tile can be either employed as a standalone accelerator or integrated to the conventional computer architecture. Accelerators are designed to lessen the execution time of certain functionalities, which are frequently used by the intended applications. With respect to their degree of flexibility, they can be designed and implemented for dedicated functionalities or they can be flexible enough to support wider range of tasks [5, 11, 65]. However, in the context of in-memory computing, the accelerators are storage units enabled to perform certain functionalities. This is aligned with the main goal of this context, which is data movement reduction. Figure 2 depicts one potential way in which a CIM tile can be seen as an off-/on-chip accelerator from the CPU. In this architecture, CIM-tiles are not integrated to the memory hierarchy of the system and allocated into different address space. In this section, we focus on the architecture of our CIM tile and how it is organized using an in-memory instructions set architecture. Afterwards, we present our compiler, which is capable of translating higher-level programs, e.g., matrix-matrix multiplication, into a sequence of the newly introduced ISA. Finally, we propose the execution model that allows pipelining at the tile-level between the analog and digital peripheries.

3.1 Overview

As mentioned earlier, we focus on the CIM-P 1T1R structure in which the (computational) results of the (memory) array operations are captured in the (digital) periphery. Figure 3 depicts the architecture of the CIM tile, which includes the required components and signals that can control digital or analog data. The operations that can be executed on the crossbar are divided into two categories: (1) write and (2) read and computational operations. The computational operations include *addition, multiplication*, and *logical operations*. In the following, we will describe our tile architecture and its main modules considering these two categories (the discussion of the control signals are left to the subsequent section):

(1) **Write operation.** To write data to the memristor crossbar, we have to specify in which row and column the data has to be written. Therefore, three registers are employed to capture this information. The data itself has to be written to the ***Write Data (WD)*** register whose length depends on the width of the crossbar as well as the number of levels supported by the memristor cells. Considering endurance issues and potential energy savings, it is not always necessary to write data to all the array columns. For this purpose, the ***Write Data Select (WDS)*** register is used to select which columns should be activated. This is especially relevant when considering implementing a write-verify operation. Finally, the ***Row Select (RS)*** register is employed to activate the row in which data has to be written. A more detailed description is presented in the following.

Voltages that have to be applied to the crossbar depend on the crossbar technology and they are usually different than the voltage used for digital part of the system. Therefore, we need a device to convert the information from digital to analog domain called **Digital Input Modular (DIM)**. Selecting a row requires that two different voltage levels have to be provided for both source and gate line of the target row as depicted in Figure 1(b), which means two DIMs (*Source/Gate DIM*) are

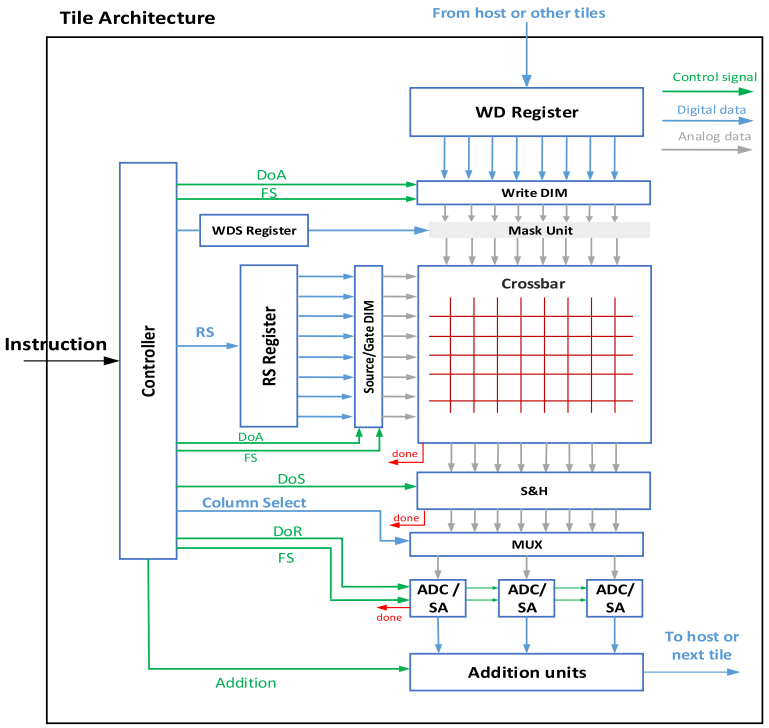

Fig. 3. The overall tile architecture.

required to drive both of them. Therefore, considering one DIM for the crossbar columns (*Write DIM*), we need three DIMs in this architecture in total. Based on the operation and the data stored in the RS and WD registers, DIMs can apply proper voltage levels to the crossbar. In addition, the data in the WDS register is used by the *Mask unit* to prevent extra switches for the cells that the data has not to be written into them.

(2) **Read and computational operations.** In this category, the operations generate an output and it has to be read by the periphery circuits in the architecture. The generated output can be the outcome of either a normal memory read or computational operation. In contrast to the write operation, there is no need to fill the WD and WDS registers. The RS register again is used for row activation. However, among computation operations, **matrix-matrix multiplication (MMM)** is a little different than others in the sense that the RS not only has to indicate the active rows but also can be considered as the data for one of the matrices. When the operation performed inside the crossbar, the generated analog output has to be captured by the ***Sample & Hold* (S&H)** unit. This allows for a clear separation of the execution within the array and the read-out circuitry, which can be used for pipelining of the system (explained in Section 3.4).

After the S&H module has captured the result from the array, the ADCs (or the sense amplifiers) can be used to convert the analog results into the digital domain. Since ADCs consume much energy and area, usually it is not possible to allocate one ADC per column. Therefore, we need analog multiplexers to share several columns with one ADC. Besides, certain high-level operations, e.g., the integer MMM, require additional processing steps and these are performed in the *Addition units* that are specific to the integer MMM. Our scheme for this unit is proposed in Reference [75] where the design utilizes minimum size adder to impose as less latency/power as possible to the system. The design considers technology/circuit/application-driven restrictions such as the

Table 1. List of Instructions

Instruction	Semantic	Operand
Row select	RS	data to fill RS register
Write data	WD	data to fill WD register
Write data select	WDS	data to fill WDS register
Function select	FS	data to configure the drivers/ADCs
Do array	DoA	—
Do sample	DoS	—
Columns select	CS	data for the select of MUX
Do read	DoR	—

maximum number of active crossbar rows, number of ADCs, and datatype size. When other (high-level) operations are needed in the future, this unit can be altered or substituted with others.

3.2 Instruction-set Architecture

As discussed in Section 3.1, a complex sequence of steps need to be performed in the CIM tile that can be different depending on the (higher-level) CIM tile operation, e.g., read/write, dot-matrix multiplication, Boolean operations, and integer matrix-matrix multiplication. Similar to the concept of microcode, we introduce an instruction-set architecture for our CIM tile that would allow for different schedules for different CIM tile operations. The "Controller" in Figure 3 is responsible for translating these instructions to the actual control signals (highlighted in green). The list of instructions is presented in Table 1. In the following, we discuss each instruction:

- ***Row Select (RS):*** the RS instruction is responsible for setting up the RS register (see Figure 3) that is subsequently used to correctly control the source and gate drivers to provide the right voltage levels for the crossbar. At this moment, the number of bits to set the RS register is as large as the height of the crossbar (means the input precision is 1 bit). As the size of crossbar increased, this is impractical for hardware implementations and left as a future optimization. However, for simulation purposes, the impact is negligible and actually allows for investigations to utilization patterns of the RS register.
- ***Write Data (WD):*** the WD instruction is responsible for setting up the WD register that is used to write data into the crossbar. Similar to the RS instruction and with the same reasoning, the size of the instruction is as large as the size of the WD register. We envision that this instruction to be replaced when another mechanism is chosen to load data into the crossbar that is more hardware friendly. For simulation purposes, it is now the only way to load data into the crossbar.
- ***Write Data Select (WDS):*** the WDS instruction sets up the WDS register to control which bits of the WD register need to be written. This allows for a flexible manner to write data into the crossbar in the light of potential endurance issues associated with current-day memristor technologies. It is especially useful when a write-verify operation needs to be performed when writing data into the crossbar. Correctly, written bits can be masked out, which helps to improve the endurance of the crossbar.
- ***Function Select (FS):*** the FS instruction is needed for several reasons. Functionally, the DIMs need to set up differently when writing data into the crossbar or when reading out data or performing compute in the crossbar. Furthermore, it is envisioned that future periphery needs additional control signals. These are now conceptually captured in the FS instruction. For example, the control signals needed to control the "Addition units" are more than one, i.e., more than one instruction is needed to control these units. The details are intentionally left out.

- ***Do Array (DoA):*** the DoA instruction is used to actually initiate the DIMs after they have been set up using the RS, WD, and FS instructions. This instruction will have a variable latency depending on the operation that is performed in the crossbar. These delays are specified as parameters in the simulator. More importantly, this instruction allows for a clear (conceptual) separation between the setup stage and the execute stage (see Figure 3) that would allow for pipelining (discussed in Section 3.4) within the CIM tile.
- ***Do Sample (DoS):*** the DoS instruction is used to signal the S&H module to start copying the result from crossbar into its own internal storage. It must be noted that this module still operates in the analog domain. The introduction of the the DoS instruction allows for a conceptual separation of the execute stage and read stage. After the values are copied into the S&H module, the crossbar can basically be issued the next DoA instruction.
- ***Column Select (CS):*** the CS instruction is used to set up the CS register that controls the multiplexers (in the MUX module) in the read stage. In case that each column of the crossbar can be associated with its own ADC/SA, there is no need to have the CS instruction. In all other cases, the CS register flexibly controls which column (in the S&H module) is connected to a ADC/SA. The length of the CS instruction is related to the number of columns of the crossbar. For the same reasons as with the RS and WDS (and WD) instructions, it is purposely defined as it is now and implemented as such in the simulator to allow for further investigation in the future. It is expected to be optimized or replaced when a hardware implementation is considered.
- ***Do Read (DoR):*** the DoR instruction initializes the modules "ADC/SA" to start converting the output of the S&H module (via the MUX) into a digital representation. It is expected that multiple iterations of the CS and DoR instructions need to be issued in order completely read out all the columns of the crossbar. However, depending on the complexity of the module, the latency can vary and thus that a "done"-signal is needed (see Figure 3) to signal the end of the readout before the next DoR instruction can be issued.

It is important to note that the crossbar, the S&H modules, and the ADC modules (related to the DoA, DoS, DoR instructions, respectively) need to be able to signal to the controller that their operation is finished. It is only after this signal, that subsequently instructions can be issued by the controller. If this turns out to be impossible in a real hardware implementation, then we envision the need to setup counters that are initialized to the specific operations to achieve the same functionality. Both approaches are already supported in our simulator.

3.3 Compiler

The compiler translates high-level operations intended for the CIM tile into a sequence of instructions to be executed within the CIM tile. The high-level operations (e.g., MMM) are provided by the front-end compiler, which is responsible for searching for the operations within the application program that can be performed using the memristor crossbar (see Figure 4). The front-end compiler receives the application in *Tensor Comprehensions* representation. This is then converted to a polyhedral representation to identify computational patterns suitable for acceleration by employing *Loop Tactics*. The front-end compiler is written by our partners in MNEMOSENE project and more information can be found in Reference [18]. Based on the requirements or constraints that come from either the tile architecture or technology side, our back-end compiler translates high-level operations to in-memory instructions. As depicted in Figure 4, this information is written to the configuration file and passed to the compiler. For example, there might be a constraint on the number of rows that can be activated at once. This constraint can come either from the precision of ADCs or even technology capability. Therefore, if an operation wants to activate more rows, the compiler splits it into several steps and takes care of other changes that might be needed. The

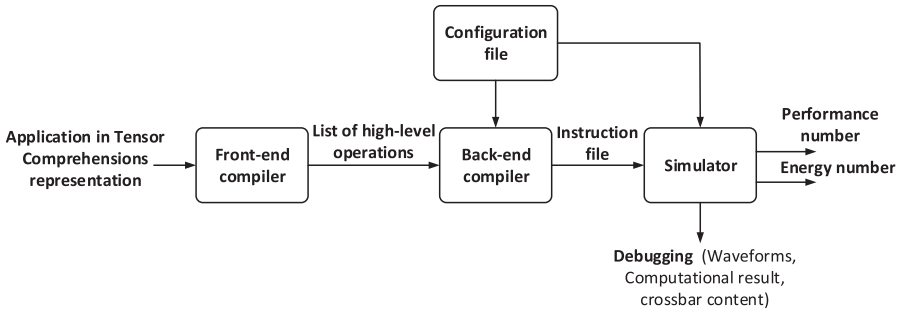

Fig. 4. The overall system flow from tensor comprehensions down to fine-grained in-memory instructions.

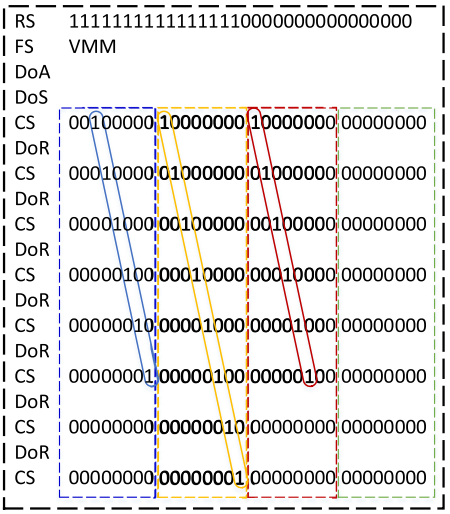

Fig. 5. An example of instructions sequence to read out the bit-lines with special pattern while activating certain rows.

flexibility brought by our in-memory instructions helps to overcome constraints, requirements, and sparse patterns. Figure 5 illustrates an example for VMM operation where four ADCs have to read columns in a special pattern (each 8 bit-lines share one ADC). This example demonstrates how the instructions deal with these kinds of patterns. It is important to note that the sequence of instructions generated by the compiler changes whenever the tile configuration changes. Therefore, by putting this complexity into the compiler, we try to keep the tile controller as simple as possible.

3.4 Pipelining

The operations in the digital periphery and the analog array can be divided into the following stages: (indicated by different colors in Figure 6)

(1) Set up stage (digital): all the control registers (and write data register) are initialized
(2) Execution stage (analog): perform the actual operation in the analog array
(3) Read out stage (analog): convert the analog results into digital values
(4) Addition stage (digital): perform the necessary operations for the integer matrix-matrix multiplication

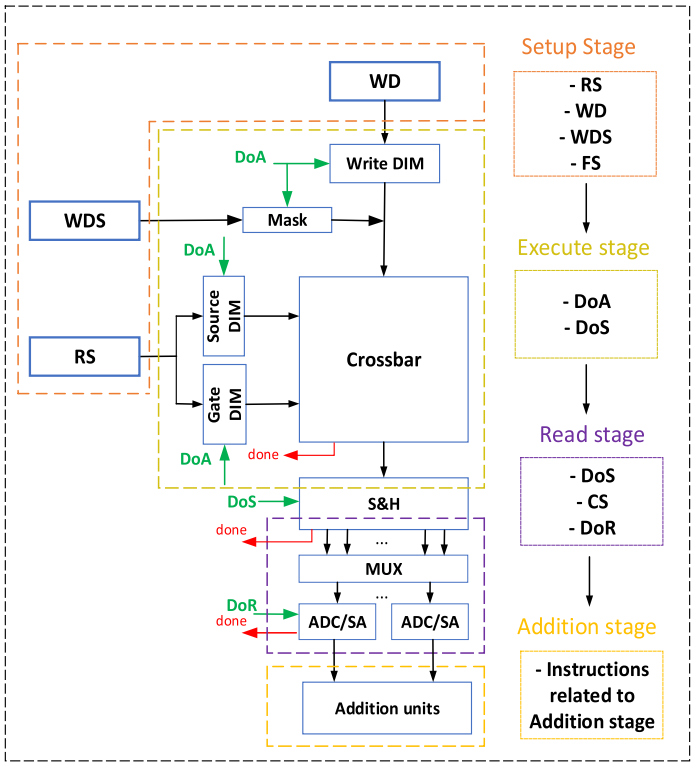

Fig. 6. Pipelining of the crossbar tile.

These stages sequentially follow each other while performing higher level operations translated to a sequence of instructions in our ISA. It should be clear that the pipelining described here is different from the traditional instruction pipelining. In the latter, the latency of each stage should be matched with each other to have a balanced pipeline. In the CIM tile, the latency of the operation performed in the analog array is expected to be much longer than the latency of a single clock cycle in the digital periphery. Therefore, it is important that the right signaling is performed between the stages to enable pipelining. The introduced execution model to pipeline the operations within the CIM tile, will allow for trade-off investigations between different NVM technologies and the (speed of the) digital periphery. Considering the aforementioned stages, the designer should evaluate the latency of each stage (which depends on the configuration of the tile, memristor technology, etc.) to realize the contribution of each stage into the total latency of the tile and merge some of them, in case there is a stage that has far less latency than others. This analog/digital behavior of the CIM-tile restricts the choices and their effectiveness regarding the pipelining stages. It is worth mentioning that, the two analog stages (*Execution* and *Read out* stages) cannot be split into more stages. This is the same for the *Set up Stage* as well, since the registers initialized here cannot be changed before activating the crossbar to ensure the correct functionality of the system.

In contrast to traditional processors where the execution of instructions is split into different stages to enable pipelining, our tile architecture associates different instructions to each tile stage. In the first stage, registers should be filled with new data and the drivers have to be configured. In the second stage, to activate the crossbar using *DoA* instruction, the operation latency for the previous activation must elapse. The latency of the crossbar, which depends on its technology as

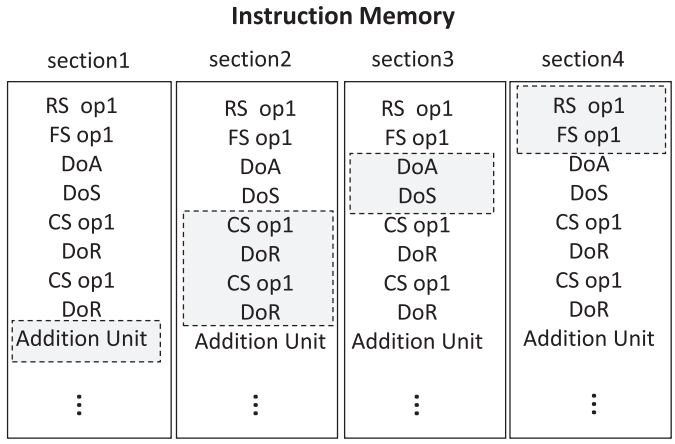

Fig. 7. An example of instruction execution on the pipelined CIM tile.

well as the operation supposed to be executed, can be captured by either a counter or done signal generated by the circuit itself. Based on the operation, which can be *write* and *read/computational*, done signals issued by the crossbar and S&H unit should be used to synchronize the first two stages, respectively. In the third stage, the latency of the Read stage depends not only on the latency of ADCs, but on the number of columns that have to be read as well. Accordingly, groups of columns are read by ADCs sequentially and this stage would be available for the next S&H activation after the last columns already translated to the digital domain. Figure 7 depicts the instruction memory where it is divided into four sections each working on one stage of the tile in parallel.

4 CIM SIMULATOR

The proposed CIM architecture is generalized making it capable of targeting different technologies with different configurations of the peripheral circuit. The simulator, written in SystemC, models the architecture presented in Section 3.1 and generates performance and energy numbers by executing applications. The simulator takes as input the program generated by the compiler (presented in Section 3.3), which is currently stored as simple (human-readable) text. Besides the program, to simplify design space exploration, the configuration of architecture has to be sent to the simulator via a configuration file in which the user is able to specify many parameters. The simulator produces as output the following: (1) energy and performance numbers, (2) content of the crossbar (over time), (3) waveforms of all control signals, and (4) the computational results. All outputs are written into text files to be used for further evaluation. Figure 8 illustrates the control and data flow of the simulator considering just 2-stage pipelining. To synchronize the stages, the controller can insert a stall to hold the execution of each stage. By decoding an instruction, the data embedded in it along with the control signals are passed to the corresponding component related to that specific instruction in the data path. Then, the status will be returned to the controller to indicate the execution of the instruction is finished.

The simulator has been written in a modular way, which helps us to easily modify or replace the components shown in the tile architecture with new designs/circuits. Moreover, new attributes can be easily added to the components model and their impacts are captured in the kernel level (e.g., read/write variability for the crossbar). In this fully parameterized simulator, each components has its own characteristics like energy, latency, and precision written into the configuration file. Table 2 shows all the parameters that can be set in the file to be used for the early-stage design space

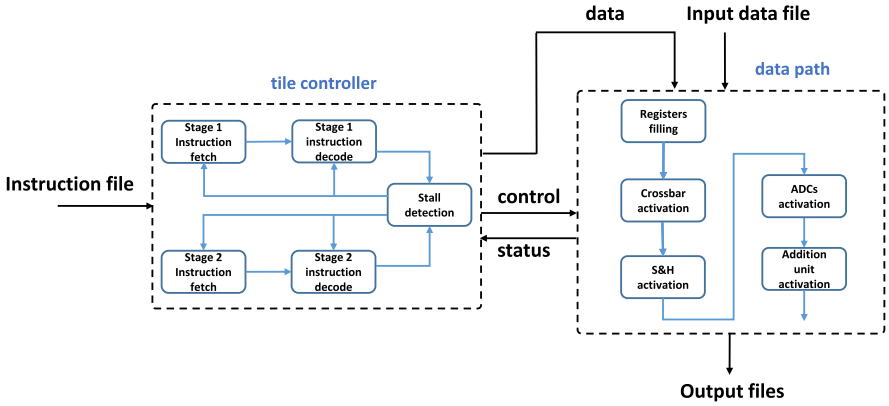

Fig. 8. Simplified flowchart of the simulator comprising control flow and data path.

Table 2. List of Parameters Used in the Configuration File

crossbar	drivers/analog peripheries	digital peripheries
- number of rows/columns - cell levels	- number of ADCs - precision of ADCs	- clock frequency - datatype size
- cell resistances - cell read/write voltages	- ADC power - read/write drivers power	- energy per adder in addition unit
- write latency - read latency	- ADC latency per conversion - SH latency	- RS/WD/WDS/CS filling cycle - instruction decoding cycle - latency per adder

exploration. Furthermore, the first of its kind feature of our simulator is the ability to calculate energy number according to the data provided by the application. Existing simulators estimate average energy number regardless of data. Our simulator takes into account the data stored in the array to estimate the energy consumption in the crossbar and its drivers. This is achieved by taking into account the data stored in the crossbar cell resistance level, the number of activated rows, and the equivalent resistance of the crossbar.

The power consumption of the crossbar and read drivers regarding read/compute operations is given in Equation (1), where R_{rc} is the resistance level of the memristor cell located in row "r" and column "c," $P_{DIM_{read}}$ is the read drivers power, and $V(read)$ is the read voltages. In general, R_{rc} and $V(read)$ are members of two sets contain possible resistance and voltage levels, respectively. Furthermore, $activation_r$ is a binary value that indicates whether row "r" is activated and contributes to the power of the crossbar or not. Considering read and compute operations, the summation is performed for the selected rows and all the columns. In addition, for simplicity, the resistance of access transistors, as well as bit-lines, are ignored. The power consumption of write operations is shown in Equation (2), where $P_{DIM_{write}}$ is the write drivers power. $V(write)$ and $I(write)$ are the write voltage and programming current, respectively. In general, $V(write)$ is a member of a set including different write voltage levels. Finally, $activation_c$ determines whether the column "c" is activated and would contribute to the crossbar energy or not. The summation is performed over the selected columns in just one activated row. The energy consumption of read/computational as well write operations are shown in Equations (3) and (4), respectively, in which $T_{Xbar(write)}$ as well as $T_{Xbar(read)}$ are the latency of the crossbar for write and read/computational operations.

The latency of the crossbar for read/computational operations also depends on the peripheral circuits used to capture or read the analog values generated by the crossbar (S&H). Using S&H unit to capture the result, its capacitance is charged with different gradient according to the equivalent resistance of the crossbar. Therefore, the result should be captured at the right time when there is a maximum voltage difference on the capacitance of S&H unit for different crossbar equivalent resistances, which helps to be distinguished by the ADC easily (implies that the crossbar latency also depends on ADC capability). It is worth mentioning that the equations are data-dependent and provide the worst-case energy numbers for the crossbar and its drivers:

$$\begin{aligned} P_{(read,compute)} &= \sum_{r=1}^{\#rows} \left[\left(\sum_{c=1}^{\#columns} \frac{V^2(read)}{R_{rc}} \right) + P_{DIM_{read}} \right] * activation_r \\ R_{rc} &\in \{L1, L2, \ldots, Ln\} \quad activation_r \in \{0, 1\} \\ V(read) &\in \{V(r1), V(r2), \ldots, V(rn)\}, \end{aligned} \tag{1}$$

$$\begin{aligned} P_{(write)_r} &= \sum_{c=1}^{\#columns} (V(write) * I(write) * activation_c + P_{DIM_{write}}) \\ V(write) &\in \{V(w1), V(w2), \ldots, V(wn)\}, \end{aligned} \tag{2}$$

$$E_{(read,compute)} = P_{(read,compute)} * T_{Xbar(read,compute)}, \tag{3}$$

$$E_{(write)} = P_{(write)} * T_{Xbar(write)}. \tag{4}$$

Due to the execution model and flexibility of our instructions, the simulator is able to add the energy of ADCs, which are active during the program execution to the total energy of the tile. Besides the hardware implementation of our digital controller, which can provide an accurate number for this unit, a more advanced model that incorporates more attributes of the crossbar are our main focus for future work. Considering that limited bits can be stored on a single cell, the data for the crossbar programming has to be distributed over multiple cells, depending on datatype size. In addition, in the case of MMM, due to the limitation on the number of levels which can be supported by the drivers and ADCs, the data for the multiplier should be sent to the crossbar's rows in several steps, depending on datatype size. Therefore, extra processing in "*Addition unit*" (see Figure 3) required to fulfill analog computations for calculation over integer numbers. An efficient structure tailored for the CIM-tile was proposed in Reference [75]. Thanks to our in-memory ISA and the flexibility provided by rescheduling them using our back-end compiler, the architecture and subsequently the simulator can perform computation with different integer datatype sizes at the same time.

5 EVALUATION AND DISCUSSION

The defined CIM tile architecture, ISA, compiler, and simulator allows for various design-space explorations. These explorations can give insight into the future CIM-tile fabrication and the influence of different parameters on the performance and energy of the tile considering different technologies. In this section, we will present several of these explorations that are currently possible with our tools.

5.1 Simulation Setup

Energy and performance model. The values used in our experiments regarding the (technology) parameters are summarized in Table 3. The needed values related to the digital periphery

Table 3. Value of Parameters Used for the Experiments

Component	Parameters	Spec		
Memristive devices		ReRAM	PCM	STT-MRAM
	cell levels	2	2	2
	LRS	5k	20k	5k
	HRS	1M	10M	10K
	read voltage	0.2V	0.2V	0.9V
	write voltage	2V	1V	1.5V
	write current	100 uA	300 uA	200 uA
	read time	10 ns	10 ns	10 ns
	write time	100 ns	100 ns	60 ns
Crossbar	structure	1T1R		
	num. columns	256		
	num. rows	256		
S&H	number	256		
	hold time	9.2 ms		
	latching energy	0.25 pJ		
	latency	0.6 ns		
DIM		read DIM		write DIM
	number	256		256
	power	1 mW		1 mW
ADC	power	2.6 mW		
	precision	8 bits		
	latency	1.2 GSps		

were obtained by using Cadence Genus targeting the standard cell 90 nm UMC library. The values related to the three targeted technologies (ReRAM, PCM, and STT-MRAM) were taken from References [20, 21, 24, 38, 48, 70]. For all the experiments, we assume the size of the crossbar is 256 by 256 and its input precision is one bit. In addition, the cycles required to fill the tile registers are computed based on the crossbar size and we assumed the databuses to be 32 bits wide.

The power and latency values for the ADCs were taken from Reference [54]. Using Equation (5), where "b" is the ADC resolution, we can derive the energy and latency of the ADC in performing a single conversion for different resolutions. These values are utilized in our simulations. Finally, the power model for the read and write drivers (DIMs) are obtained from Reference [52]:

$$power_{ADC} = \frac{energy_{ADC}}{latency_{ADC}} = \frac{[(64 * 34fJ)(2^{b-8})]}{([(1.2GS/sec) * (2^{8-b})])^{-1}}. \tag{5}$$

Benchmark. As a benchmark, the linear-algebra kernel "GEMM" from the Polybench/C benchmark suite was chosen. In this kernel, first, the multiplicands are written into the crossbar (write operation) and then the actual multiplication (compute operation) is performed. The datatype size for both multiplier and multiplicand are considered 8 bits meaning that for each number 8 cells have to be employed (assuming each cell stores one bit). This benchmark was chosen as it intensively utilizes the memory array as well as its digital peripheries given that we want to perform DSE targeting different technologies for the memory array.

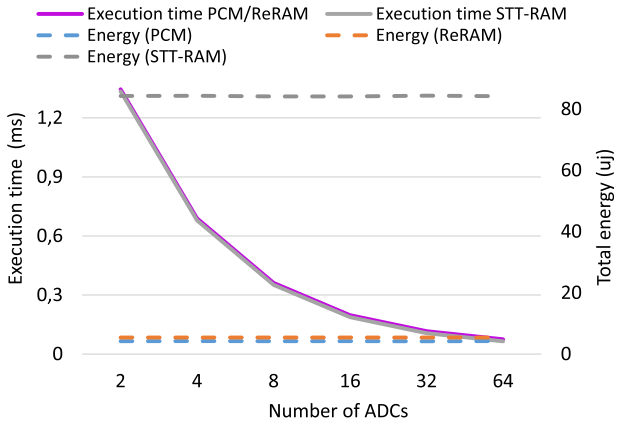

Fig. 9. The impact of number of ADCs on execution time of GEMM benchmark.

5.2 Simulation Results

In this section, we will present several design-space explorations that are currently possible using our simulator. The insights obtained from these analyses will lead designers to take better decisions for the actual implementation.

Performance versus number of ADCs. In Figure 9, we plotted the (normalized) execution time of running the GEMM benchmark targeting three different technologies for the crossbar array, namely, PCM, ReRAM, and STT-MRAM. The simulations were performed assuming a 1 GHz clock frequency for the digital periphery and an 8-bit ADC resolution.

We can clearly observe that the number of ADCs greatly impacts the execution time. By adding more ADCs, the total execution time can be reduced as the cycles needed to read out the data from the crossbar array can be reduced. Although STT-MRAM has faster write time, due to the less number of write operations to program the crossbar compared to the computational operations, the improvement on the execution time is negligible. An interesting observation is that the performance does not improve much when moving from 32 to 64 ADCs. This can be explained by the fact that at some point, the latency of the read stage is no longer dominant and further reducing the readout time has little impact on the total execution time. Finally, regardless of the number of ADCs, the energy consumption is almost constant (small fluctuation due to the data randomness), since the number of conversions is always fixed.

Performance versus frequency of digital periphery. The latency of the operations in the (analog) crossbar array is a constant number. Therefore, it is interesting to determine how fast the digital periphery should be clocked to "match" this latency to make the pipeline more balanced. In the following investigation, we have fixed the number of ADCs to 16 and ran the GEMM benchmark at different frequencies.

Figures 10 clearly shows that performance improvements can be gained by raising the frequency of the digital periphery. However, increasing the clock frequency beyond 1 GHz does not result in much better execution times as the analog circuits (relatively) are becoming the bottleneck. In addition, the performance improvement due to the pipelining will be reduced, since the stages are more unbalanced. This DSE allows a designer to make the different stages of the tile more balanced. A positive side-effect is that pipelining more balanced stages will usually lead to better performance improvements over a non-pipelined design.

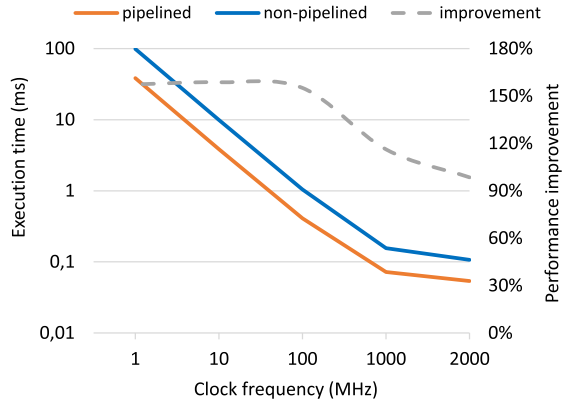

Fig. 10. Performance improvement due to the unbalanced pipelining of tile used ReRAM/PCM device for GEMM benchmark.

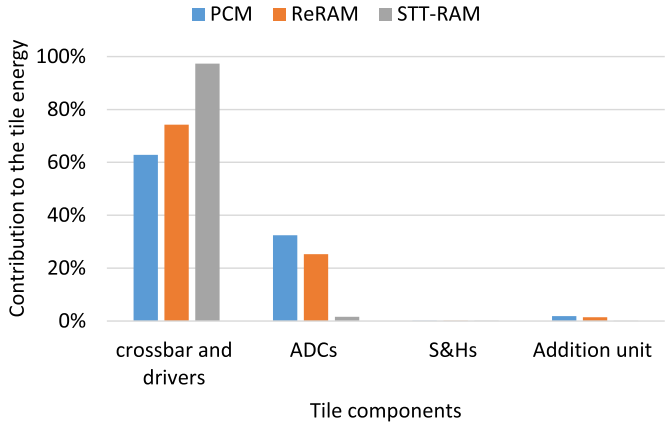

Fig. 11. Contribution of different components to the energy consumption for GEMM benchmark.

Energy contribution per module. Figure 11 depicts the relative energy spent in the different modules when running the GEMM benchmark for 16 ADCs and using an 8-bit datatype. We can clearly observe that the largest energy consumer is still the crossbar and its drivers. Although the power consumption of ADC is high, due to the low latency of this component (around 1 ns [54]), the energy consumption of ADC does not dominate. In the PCM case, the relative energy consumption of the ADCs and crossbar are close to each other (compared to other technologies). The reason for this is that the power consumption of PCM is relatively lower compared to the ReRAM technology due to the higher cell resistance (see Table 3). This in turn increases the relative energy consumption of the ADCs.

It is worth mentioning that the energy consumption of the crossbar depends on the input and programming data. As more devices are programmed to LRS and more rows are activated, clearly more energy is consumed during the computation. Since the simulator can actually execute kernels, the energy number obtained for the crossbar is data-dependent. Figure 12 depicts the energy consumption of the crossbar with different levels of sparsity. This figure illustrates the sparsity of logic value 1 for matrix-matrix multiplication. For this experiment, it is considered that the

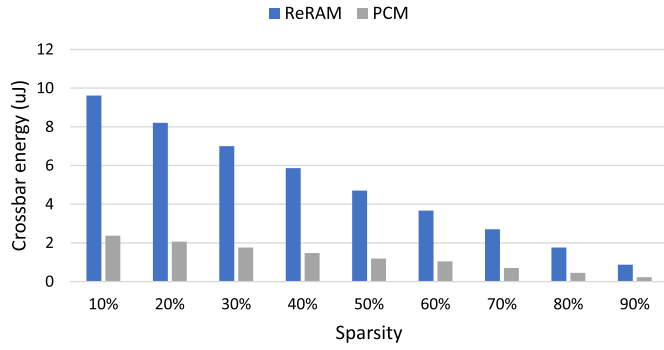

Fig. 12. Energy consumption of the crossbar with respect to the percentage of sparsity for both input and programming data.

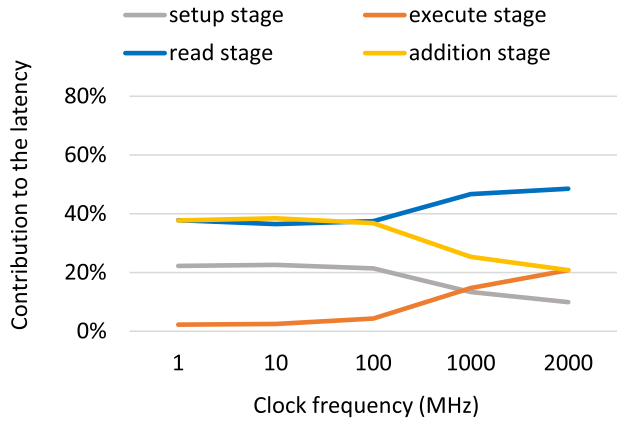

Fig. 13. Contribution of each pipeline stage on the latency of the tile considering different clock frequencies.

multiplier (input) and the multiplicand (programmed) matrix have the same sparsity. In addition, logic value 1 (0) as an input implies the corresponding row is activated (deactivated) and as programming data indicates the memristive device is programmed to LRS (HRS). The simulation is performed for ReRAM and PCM devices and an interesting observation is that although the ratio of HRS to LRS is lower in ReRAM devices, since PCM devices have higher LRS compared to ReRAM, the sparsity leads to less variation in the energy consumption of the crossbar made with PCM.

Relative latency contribution per stage. Figure 13 depicts the relative time that the GEMM application spends in each of the 4 stages plotted against the frequency of the digital periphery. Since several columns share an ADC, reading all the columns should be performed in multi-steps. In addition, after each ADC activation, digital processing is required in *"addition unit."* Therefore, we can clearly observe that with a low frequency, the read and addition stages are completely dominant in the total latency. Using an efficient structure and minimum-sized adder, the latency of the addition unit is no longer than the read stage. With low clock frequency, the latency of the analog circuits is hidden in one clock period. However, As the clock frequency is increased more, the latency of the analog components starts to rise (relatively). Consequently, we can observe that the latency of the read stage, which is composed of analog (latency of ADC) and digital (decoding latency) latency, as well as the execute stage impose more latency. This information can be used to determine the number of pipeline stages for the actual implementation.

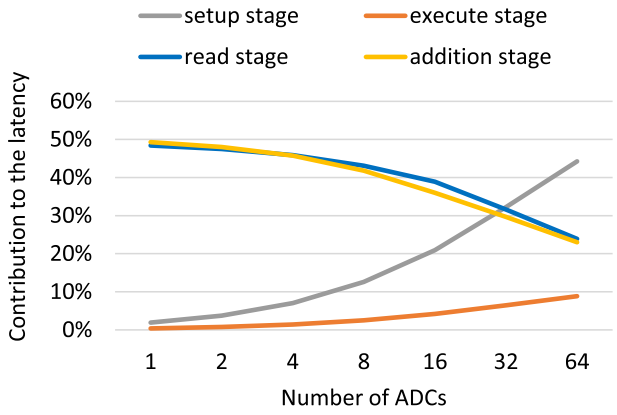

Fig. 14. Effect of number of ADCs on the latency of pipeline stages in 100 MHz clock frequency.

Figure 14 depicts the relative time that the GEMM application spends in each of the four stages plotted against the number of utilized ADCs. It should be clear that increasing number of ADCs, the number cycles spend in the read out stage is greatly reduced. Consequently, we can observe that the relative contribution of the setup stage to the total latency grows accordingly. The contribution of the other stages to the total latency is almost negligible. With the advent of advanced ADC design to have more ADCs per tile, this figure brings an insight into its implications and helps for future design decisions.

6 CONCLUSIONS

In this article, we proposed a general architecture and execution model that captures how a memristor-based compute-in-memory tile would function when it produces its results in the periphery (instead of in the crossbar array). This architecture is defined to accommodate current and future NVM technologies used to implement the crossbar array. Example technologies are PCM, ReRAM, and STT-RAM. Our abstraction allowed for the definition of an ISA that describes/captures the functionalities within the CIM tile and allows for sequencing of these instructions to perform higher-level operations achievable with such a CIM tile, e.g., integer matrix-matrix multiplication. In addition, a compiler was developed that can translate such high-level operations into a sequence of instructions that can be executed by the CIM tile. Subsequently, we developed a parameterized cycle-accurate simulator for our CIM architecture that would allow for design space exploration. The parameters are either technology-dependent parameters or design parameters of our architecture. Finally, we ported the GEMM benchmark and demonstrated in this article how various design space explorations for this benchmark can be performed targeting different technologies and other design parameters. In future work, we will extend the simulator to comprise more CIM tiles. This will help us to explore the communication between tiles, implement complex applications, and evaluate the scalability of our simulator.

REFERENCES

[1] Micron.com. 2017. Calculating Memory Power for DDR4 SDRAM. Retrieved from https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf.

[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. ACM, New York, NY, 105–117. https://doi.org/10.1145/2749469.2750386

[3] Berkin Akin, Franz Franchetti, and James C. Hoe. 2015. Data reorganization in memory using 3D-stacked DRAM. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. ACM, New York, NY, 131–143. https://doi.org/10.1145/2749469.2750397

[4] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W. Hwu, John Paul Strachan, Kaushik Roy, et al. 2019. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. 715–731.

[5] Lars Bauer, Muhammad Shafique, Simon Kramer, and Jörg Henkel. 2007. RISPP: Rotating instruction set processing platform. In *Proceedings of the 44th Annual Design Automation Conference*. 791–796.

[6] Debjyoti Bhattacharjee, Rajeswari Devadoss, and Anupam Chattopadhyay. 2017. ReVAMP: ReRAM based VLIW architecture for in-memory computing. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*. IEEE, 782–787.

[7] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. Stewart, and R. Stanley Williams. 2010. Memristive switches enable stateful logic operations via material implication. *Nature* 464, 7290 (2010), 873.

[8] G. W. Burr, M. J. Brightsky, A. Sebastian, H. Cheng, J. Wu, S. Kim, N. E. Sosa, N. Papandreou, H. Lung, H. Pozidis, E. Eleftheriou, and C. H. Lam. 2016. Recent progress in phase-change memory technology. *IEEE J. Emerg. Select. Topics Circ. Syst.* 6, 2 (June 2016), 146–162. https://doi.org/10.1109/JETCAS.2016.2547718

[9] Fuxi Cai, Justin M. Correll, Seung Hwan Lee, Yong Lim, Vishishtha Bothra, Zhengya Zhang, Michael P. Flynn, and Wei D. Lu. 2019. A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. *Nature Electron.* 2, 7 (July 2019), 290–299. https://doi.org/10.1038/s41928-019-0270-x

[10] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu. 2016. Low-Cost Inter-Linked Subarrays (LISA): Enabling fast inter-subarray data movement in DRAM. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. 568–580. https://doi.org/10.1109/HPCA.2016.7446095

[11] Xiaolin Chen, Andreas Minwegen, Yahia Hassan, David Kammler, Shuai Li, Torsten Kempf, Anupam Chattopadhyay, and Gerd Ascheid. 2012. FLEXDET: Flexible, efficient multi-mode MIMO detection using reconfigurable ASIP. In *Proceedings of the IEEE 20th International Symposium on Field-programmable Custom Computing Machines*. 69–76. https://doi.org/10.1109/FCCM.2012.22

[12] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*. IEEE Computer Society, Washington, DC, 609–622. https://doi.org/10.1109/MICRO.2014.58

[13] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.

[14] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. 2011. Memory power management via dynamic Voltage/Frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*. ACM, New York, NY, 31–40. https://doi.org/10.1145/1998582.1998590

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. Retrieved from https://arXiv:1810.04805.

[16] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. 2012. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. Comput.-aided Design Integr. Circ. Syst.* 31, 7 (2012), 994–1007.

[17] J. Doweck, W. Kao, A. K. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz. 2017. Inside 6th-Generation intel core: New microarchitecture code-named skylake. *IEEE Micro* 37, 2 (Mar. 2017), 52–62. https://doi.org/10.1109/MM.2017.38

[18] Andi Drebes, Lorenzo Chelini, Oleksandr Zinenko, Albert Cohen, Henk Corporaal, Tobias Grosser, Kanishkan Vadivel, and Nicolas Vasilache. 2020. TC-CIM: Empowering tensor comprehensions for computing-in-memory. In *Proceedings of the 10th International Workshop on Polyhedral Compilation Techniques (IMPACT'20)*.

[19] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 283–295. https://doi.org/10.1109/HPCA.2015.7056040

[20] K. Fleck, N. Aslam, S. Hoffmann-Eifert, V. Longo, F. Roozeboom, W. M. M. Kessels, U. Böttger, R. Waser, and S. Menzel. 2016. The influence of non-stoichiometry on the switching kinetics of strontium-titanate ReRAM devices. *J. Appl. Phys.* 120, 24 (2016), 244502.

[21] K. Fleck, U. Böttger, Rainer Waser, N. Aslam, S. Hoffmann-Eifert, and Stephan Menzel. 2016. Energy dissipation during pulsed switching of strontium-titanate based resistive switching memory devices. In *Proceedings of the 46th European Solid-state Device Research Conference (ESSDERC'16)*. IEEE, 160–163.

[22] Scott W. Fong, Christopher M. Neumann, and H.-S. Philip Wong. 2017. Phase-change memory–Towards a storage-class memory. *IEEE Trans. Electron Devices* 64, 11 (2017), 4374–4385.

[23] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. 2018. In-memory data parallel processor. In *ACM SIGPLAN Notices*, Vol. 53. ACM, 1–14.

[24] W. J. Gallagher, Eric Chien, Tien-Wei Chiang, Jian-Cheng Huang, Meng-Chun Shih, C. Y. Wang, Chih-Hui Weng, Sean Chen, Christine Bair, George Lee, et al. 2019. 22 nm STT-MRAM for reflow and automotive uses with high yield, reliability, and magnetic immunity and with performance and shielding options. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'19)*. IEEE, 2–7.

[25] Amir Gholami. 2018. AI and memory wall. Retrieved from https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8.

[26] B. Govoreanu, G. S. Kar, Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. J. Wouters, J. A. Kittl, and M. Jurczak. 2011. 10*10nm2 Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation. In *Proceedings of the International Electron Devices Meeting*. 31.6.1–31.6.4. https://doi.org/10.1109/IEDM.2011.6131652

[27] Said Hamdioui, Mottaqiallah Taouil, Hoang Anh Du Nguyen, Adib Haron, Lei Xie, and Koen Bertels. 2015. Memristor: The enabler of computation-in-memory architecture for big-data. In *Proceedings of the International Conference on Memristive Systems (MEMRISYS'15)*. IEEE, 1–3.

[28] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike, and Jan van Lunteren. 2015. Memristor based computation-in-memory architecture for data-intensive applications. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*. EDA Consortium, San Jose, CA, 1718–1725.

[29] Alexander Hardtdegen, Camilla La Torre, Felix Cüppers, Stephan Menzel, Rainer Waser, and Susanne Hoffmann-Eifert. 2018. Improved switching stability and the effect of an internal series resistor in hfo 2/TiO x bilayer ReRAM cells. *IEEE Trans. Electron Devices* 65, 8 (2018), 3229–3236.

[30] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. 2005. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'05)*. 459–462. https://doi.org/10.1109/IEDM.2005.1609379

[31] M. Imani, Y. Kim, and T. Rosing. 2017. MPIM: Multi-purpose in-memory processing using configurable resistive memory. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. 757–763. https://doi.org/10.1109/ASPDAC.2017.7858415

[32] Hao Jiang, Can Li, Peng Lin, Shuang Pi, Jianhua Yang, and Qiangfei Xia. 2019. Scalable 3D ta:sio x memristive devices. *Adv. Electronic Mater.* (Feb. 2019), 1800958. https://doi.org/10.1002/aelm.201800958

[33] Geethan Karunaratne, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abbas Rahimi, and Abu Sebastian. 2020. In-memory hyperdimensional computing. *Nature Electron.* 3, 6 (2020), 327–337.

[34] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay. 2016. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA'16)*. 380–392. https://doi.org/10.1109/ISCA.2016.41

[35] Yusung Kim, Xuanyao Fong, Kon-Woo Kwon, Mei-Chin Chen, and Kaushik Roy. 2014. Multilevel spin-orbit torque MRAMs. *IEEE Trans. Electron Devices* 62, 2 (2014), 561–568.

[36] Kunal Korgaonkar, Ronny Ronen, Anupam Chattopadhyay, and Shahar Kvatinsky. 2019. The bitlet model: Defining a litmus test for the bitwise processing-in-memory paradigm. Retrieved from https://arXiv:1910.10234.

[37] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. 2014. MAGIC–Memristor-Aided logic. *IEEE Trans. Circ. Syst. II: Express Briefs* 61, 11 (Nov. 2014), 895–899. https://doi.org/10.1109/TCSII.2014.2357292

[38] Manuel Le Gallo, Abu Sebastian, Giovanni Cherubini, Heiner Giefers, and Evangelos Eleftheriou. 2018. Compressed sensing with approximate message passing using in-memory computing. *IEEE Trans. Electron Devices* 65, 10 (2018), 4304–4312.

[39] Muath Abu Lebdeh, Uljana Reinsalu, Hoang Anh Du Nguyen, Stephan Wong, and Said Hamdioui. 2019. Memristive device based circuits for computation-in-memory architectures. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'19)*. IEEE, 1–5.

[40] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2010. Phase change memory architecture and the quest for scalability. *Commun. ACM* 53, 7 (July 2010), 99–106. https://doi.org/10.1145/1785414.1785441

[41] Matthew Kay Fei Lee, Yingnan Cui, Thannirmalai Somu, Tao Luo, Jun Zhou, Wai Teng Tang, Weng-Fai Wong, and Rick Siow Mong Goh. 2019. A system-level simulator for RRAM-based neuromorphic computing chips. *ACM Trans. Architect. Code Optimiz.* 15, 4 (2019), 64.

[42] Lei Xie, Hoang Anh Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels. 2015. Fast boolean logic mapped on memristor crossbar. In *Proceedings of the 33rd IEEE International Conference on Computer Design (ICCD'15)*. 335–342. https://doi.org/10.1109/ICCD.2015.7357122

[43] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the 53rd Annual Design Automation Conference (DAC'16)*. ACM, New York, NY, Article 173, 6 pages. https://doi.org/10.1145/2897937.2898064

[44] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, B. Liu, Y. Chen, H. Li, Qing Wu, and Hao Jiang. 2015. A spiking neuromorphic design with resistive crossbar. In *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC'15)*. 1–6. https://doi.org/10.1145/2744769.2744783

[45] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao. 2018. Processing-in-memory for energy-efficient neural network training: A heterogeneous approach. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. 655–668. https://doi.org/10.1109/MICRO.2018.00059

[46] Hiwa Mahmoudi, Thomas Windbacher, Viktor Sverdlov, and Siegfried Selberherr. 2013. Implication logic gates using spin-transfer-torque-operated magnetic tunnel junctions for intrinsic logic-in-memory. *Solid-state Electron.* 84 (2013), 191–197. https://doi.org/10.1016/j.sse.2013.02.017

[47] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha. 2011. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'11)*. 1–4. https://doi.org/10.1109/CICC.2011.6055294

[48] S. R. Nandakumar, Manuel Le Gallo, Christophe Piveteau, Vinay Joshi, Giovanni Mariani, Irem Boybat, Geethan Karunaratne, Riduan Khaddam-Aljameh, Urs Egger, Anastasios Petropoulos, et al. 2020. Mixed-precision deep learning based on computational memory. Retrieved from https://arXiv:2001.11773.

[49] Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui, and Francky Catthoor. 2020. A classification of memory-centric computing. *ACM J. Emerg. Technol. Comput. Syst.* 16, 2 (2020), 1–26.

[50] Matt Poremba and Yuan Xie. 2012. Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*. IEEE, 392–397.

[51] Saeed Rashidi, Majid Jalili, and Hamid Sarbazi-Azad. 2019. A survey on PCM lifetime enhancement schemes. *ACM Comput. Surv.* 52, 4, Article 76 (Aug. 2019), 38 pages. https://doi.org/10.1145/3332257

[52] Mehdi Saberi, Reza Lotfi, Khalil Mafinezhad, and Wouter A. Serdijn. 2011. Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs. *IEEE Trans. Circ. Syst. I: Regular Papers* 58, 8 (2011), 1736–1748.

[53] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. 2020. Memory devices and applications for in-memory computing. *Nature Nanotechnol.* (2020), 1–16.

[54] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with *in situ* analog arithmetic in crossbars. *ACM SIGARCH Comput. Architect. News* 44, 3 (2016), 14–26.

[55] Anne Siemon, Thomas Breuer, Nabeel Aslam, Sebastian Ferch, Wonjoo Kim, Jan van den Hurk, Vikas Rana, Susanne Hoffmann-Eifert, Rainer Waser, Stephan Menzel, et al. 2015. Realization of boolean logic functionality using redox-based memristive devices. *Adv. Function. Mater.* 25, 40 (2015), 6414–6423.

[56] A. Siemon, R. Drabinski, M. J. Schultis, X. Hu, E. Linn, A. Heittmann, R. Waser, D. Querlioz, S. Menzel, and J. S. Friedman. 2019. Stateful three-input logic with memristive switches. *Sci. Rep.* 9, 1 (2019), 1–13.

[57] A. Siemon, S. Ferch, A. Heittmann, R. Waser, D. J. Wouters, and S. Menzel. 2019. Analyses of a 1-layer neuromorphic network using memristive devices with non-continuous resistance levels. *APL Mater.* 7, 9 (2019), 091110.

[58] Anne Siemon, Stephan Menzel, Debjyoti Bhattacharjee, Rainer Waser, Anupam Chattopadhyay, and Eike Linn. 2019. Sklansky tree adder realization in 1S1R resistive switching memory architecture. *Eur. Phys. J. Special Topics* 228, 10 (2019), 2269–2285.

[59] Anne Siemon, Stephan Menzel, Anupam Chattopadhyay, Rainer Waser, and Eike Linn. 2015. In-memory adder functionality in 1S1R arrays. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'15)*. IEEE, 1338–1341.

[60] Anne Siemon, Dirk Wouters, Said Hamdioui, and Stephan Menzel. 2019. Memristive device modeling and circuit design exploration for computation-in-memory. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'19)*. IEEE, 1–5.

[61] G. Snider. 2005. Computing with hysteretic resistor crossbars. *Appl. Phys. A* 80, 6 (2005), 1165–1172.

[62] Sriseshan Srikanth, Lavanya Subramanian, Sreenivas Subramoney, Thomas M. Conte, and Hong Wang. 2018. Tackling memory access latency through DRAM row management. In *Proceedings of the International Symposium on Memory Systems (MEMSYS'18)*. ACM, New York, NY, 137–147. https://doi.org/10.1145/3240302.3240314

[63] Sung Hyun Jo, T. Kumar, S. Narayanan, W. D. Lu, and H. Nazarian. 2014. 3D-stackable crossbar resistive memory based on Field Assisted Superlinear Threshold (FAST) selector. In *Proceedings of the IEEE International Electron Devices Meeting*. 6.7.1–6.7.4. https://doi.org/10.1109/IEDM.2014.7046999

[64] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang. 2017. Binary convolutional neural network on RRAM. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. 782–787. https://doi.org/10.1109/ASPDAC.2017.7858419

[65] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E.M. Panainte. 2004. The MOLEN polymorphic processor. *IEEE Trans. Comput.* 53, 11 (2004), 1363–1375. https://doi.org/10.1109/TC.2004.104

[66] Zhongrui Wang, Saumil Joshi, Sergey E. Savel'ev, Wenhao Song, Rivu Midya, Yunning Li, Mingyi Rao, Peng Yan, Shiva Asapu, Ye Zhuo, Hao Jiang, Peng Lin, Can Li, Jung Ho Yoon, Navnidhi K. Upadhyay, Jiaming Zhang, Miao Hu, John Paul Strachan, Mark Barnell, Qing Liang Wu, Huaqiang Wu, R. Stanley Williams, Qiangfei Xia, and J. Joshua Yang. 2018. Fully memristive neural networks for pattern classification with unsupervised learning. *Nature Electron.* 1 (2018), 137–145.

[67] Zhongrui Wang, Can Li, Peng Lin, Mingyi Rao, Yongyang Nie, Wenhao Song, Qinru Qiu, Yunning Li, Peng Yan, John William Strachan, Ning Ge, Nathan McDonald, Qing wu, Miao Hu, Huaqiang Wu, Stan Williams, Qiangfei Xia, and Jianhua Joshua Yang. 2019. *In situ* training of feed-forward and recurrent convolutional memristor networks. *Nature Mach. Intell.* 1 (Sept. 2019), 434–442. https://doi.org/10.1038/s42256-019-0089-1

[68] Steven J. E. Wilton and Norman P. Jouppi. 1996. CACTI: An enhanced cache access and cycle time model. *IEEE J. Solid-state Circ.* 31, 5 (1996), 677–688.

[69] Dirk J. Wouters, Rainer Waser, and Matthias Wuttig. 2015. Phase-change and redox-based resistive switching memories. *Proc. IEEE* 103, 8 (2015), 1274–1288.

[70] Lizhou Wu, Siddharth Rao, Guilherme Cardoso Medeiros, Mottaqiallah Taouil, Erik Jan Marinissen, Farrukh Yasin, Sebastien Couet, Said Hamdioui, and Gouri Sankar Kar. 2019. Pinhole defect characterization and fault modeling for STT-MRAM testing. In *Proceedings of the IEEE European Test Symposium (ETS'19)*. IEEE, 1–6.

[71] Lixue Xia, Boxun Li, Tianqi Tang, Peng Gu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, and Huazhong Yang. 2017. MNSIM: Simulation platform for memristor-based neuromorphic computing system. *IEEE Trans. Comput.-aided Design Integr. Circ. Syst.* 37, 5 (2017), 1009–1022.

[72] Lei Xie, Hoang Anh Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, and Said Hamdioui. 2017. Scouting logic: A novel memristor-based logic design for resistive computing. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'17)*. IEEE, 176–181.

[73] Peng Yao, Huaqiang Wu, Bin Gao, Sukru Burc Eryilmaz, Xueyao Huang, Wenqiang Zhang, Qingtian Zhang, Ning Deng, Lu ping Shi, H.-S. Philip Wong, and He Qian. 2017. Face classification using electronic synapses. In *Nature Communications*. Nature Research, London.

[74] S. Yu and P. Chen. 2016. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-state Circ. Mag.* 8, 2 (Spring 2016), 43–56. https://doi.org/10.1109/MSSC.2016.2546199

[75] Mahdi Zahedi, Mahta Mayahinia, Muath Abu Lebdeh, Stephan Wong, and Said Hamdioui. 2020. Efficient organization of digital periphery to support integer datatype for memristor-based CIM. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'20)*. IEEE, 216–221.

[76] Mahdi Zahedi, Remon van Duijnen, Stephan Wong, and Said Hamdioui. 2021. tile architecture and hardware implementation for computation-in-memory. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'21)*. IEEE, 108–113.

Received January 2021; revised July 2021; accepted September 2021