CiMC: A Computing-in-Memory Controller for Memristive Crossbar Array

Kai Bu*, Zhiwei Li and Wei Wang

College of Electronic Science and Technology, University of Defence Technology, 109 Deya Road, Changsha, Hunan, China

*Corresponding author Email: booquai@nudt.edu.cn

Abstract. The significant challenges for the exascale system are the relatively limited computational resources and data movement between processors and memories. Memristor has the potential to build a universal digital nonvolatile computing system to break the energy-efficiency bottleneck caused by the separation of storage and processing in the traditional computing framework. Computing-in-Memory(CiM) based on memristor is a potential approach to addressing the data transfer bottleneck between processor and memory in a computing system. CiM could reduce or even remove the data movement between processors and memory. The unique properties of Memristor allow both logic and memory function within an element, opening up the possibility of directly computing on the data stored in memory. We propose a Compute-in-Memory Controller, a design for computing and storing in a memristive crossbar array. Both logic operation and arithmetic functions are implemented with the controller. We also propose a two-layer instruction set to perform CiM functions, which addresses the question of how a CiM memory could be integrated into a general-purpose computing system.

1. Introduction

Under the background of the artificial intelligence era, the traditional computing system based on CMOS technology and von-Neumann architecture is facing more and more strict performance and energy consumption challenges in the application of data-centric computing[1,2]. The development of electronic information technology has entered the post-Moore era. The further reduction on the feature size of semiconductor devices is facing dual constraints from fabricating technics and physical mechanisms. And the performance improvement is also more and more difficult for CPU processors designed following Von-Neumann computing architecture. The "storage wall" is becoming a severe problem caused by the access bottleneck between processor and memory. The CPU processor could not meet the requirements of performance and energy consumption for data-centric computing in the intelligent era[xx].

Memristor has the potential feature to build a universal digital nonvolatile computing system to break the energy-efficiency bottleneck caused by the separation of storage and processing in the traditional computing framework. High-performance devices, efficient logic gates, and reconfigurable architecture are the basis for improving nonvolatile computing systems. The new computing-in-memory architecture based on memristor technology is a meaningful way to break through the performance bottleneck of the traditional von-Neumann computing system[16]. As a new passive nanoscale information device, memristor has brought new possibilities to the design of computing...
architecture with its characteristics of both logic and storage[4,5]. A memristor crossbar array can easily perform matrix multiplication, which is a very expensive operation for CMOS-based neural networks. In a memristor crossbar array, there is a memristor at each crosspoint of a row and column. Voltage pulse could be transfer to one end of each memristor through its row and an summation of output current could be generated along a column[15].

The Computing-in-Memory (CiM) technology has built a new type of computing architecture[5-8], which breaks the separation between computing and storage, and is expected to solve the problem of Memory-Wall. Profit from its unique physical mechanism of resistance, Memristor has become a valuable research content of integrated logic calculation based on the ability of both computation and storage. Recently, many efforts have been paid on the implementation of memristive logic gates for computing in-memory[4,5]. Many circuits and architectures based on memristors have been proposed to implement primitive Boolean logic functions[7,8,9], while some works have focused more on particular computing function circuits such as multiply-accumulate, full-adder and multiplier[10,11]. At present, the research is mainly focused on how to build the function unit, such as 16 kinds of Boolean logic and adder, which is only a primary element to design a processor. However, this fundamental element only can be utilized in a narrow range of applications[12-14].

To solve practical computing problems in real applications, we need an instruction set to supporting the program and running a task. Therefore, a general-purpose processor is preferred to handle the diverse applications. We use memory to build a general-purpose processor to do both data computing and storage. This paper proposes CiMC processor architecture, as shown in Figure 1. CiMC employs stateful logic operations where the same device serves as both logic and memory simultaneously. The main contributions of this paper are:

• A general-purpose microarchitecture utilizing a memristor crossbar array to both stores and operate data in-situ.
• A two-layer instruction set architecture to support compute-in-memory operations that can realize a range of the store, boolean logic, and arithmetic without data moving.
• Optimized state-full logic operations and address mapping to reduce their delay.

Figure 1. Computing-in-Memory Controller(CiMC) architecture

2. State-full logic based on memristor crossbar array
The resistances of the memristors can be switched between two digital states, low resistance state corresponding to logic-high ‘1’ and high resistance state corresponding to logic-low ‘0’, by applying two different voltage pulses on the two ends of a memristor. Crossbar is usually employed as memory architecture based on memristor for high integration density. Further, a memristor with crossbar architecture, showing the outstanding characteristics of reconfigurability and potential for parallel computing, is suitable for in-memory computing.
Here, we use the memristor with the Cu/α-Si/α-C/Pt structure, which has excellent uniformity of HRS/LRS and set/reset voltage and the ability to prevent resistance shift. A stateful logic scheme has been presented to execute logic operations by altering trigger signal configuration and initialization of memristors, which makes devices connected in series or parallel and dropped on by different voltage amplitudes to achieve all kinds of logic operations.

![Diagram of NAND Operation](image)

Figure 2. NAND logic operations.

As showned in Figure 2a, NAND logic operations are implemented by the devices sharing the common Word Line. The resistance of devices A and B are logic inputs, and the resistance of R will be the output result decided by the states of the memristors A and memristor B. Voltage pulse $V_N$ is applied on memristor R to trigger the resistance switching between the input and output devices. Figure 2b shows the equivalent circuit diagram for NAND logic sharing a common Word Line with initialized resistance of (A,B,R) = (0,0,1), (1,0,1) and (1,1,1). Figure 2c shows resistance states of inputs before NAND, excitation pulses and response current of NAND logic gate, resistance states of outputs after NAND. Result states are read out with a weak voltage pulse.

Based on the Cu/α-Si/α-C/Pt device with high performance and principle of resistance interaction, NAND is the essential operation of all the Boolean logic, which is enough to synthesize all the logics by proper cascading. Figure 2a shows the unit structure and working principle of the logic gate designed without changing the crossbar array's architecture. A word line connects three memristors A, B, and R on top electrodes, and their bottom electrodes are linked to different bit lines. The logic realized can be expressed as $R = \sim AB$, which is written as NAND(A, B) for short in the following illustration. Memristor A and B hold the input information, and device R will store the operation result. The BitLines of Memristors A and B are connected to signal ground to trigger the resistance state switching. Meanwhile, the voltage pulse $V_N$ is applied on memristor R’s BitLine and the three devices’ common WordLine remains dangling, which is different from the former stateful logic gates. The equivalent circuit is just shown in Figure 2b. The magnitude of $V_N$ should be larger than RESET to switch the memristor to high resistance state, due to the voltage drop across memristor A and B. Also, to leave the resistance state of input devices unchanged which is important to multi-gate data transfer, the magnitude of the stimulating voltage pulse $V_N$ must meet the condition that $1/2 \lt V_N < V_{\text{RESET}} < 2/3V_N$. Based on the characteristics of Cu/α-Si/α-C/Pt memristor, a pulse with 3V and 100 ns is assigned as $V_N$ to ensure the logic operation performs steadily and correctly. Figure 2c shows resistance states of memristors before NAND, excitation pulses and response current of NAND logic
gate, and resistance states of memristors after NAND. In the NAND logic gate, output memristor R is initialized to low resistance state. Logic inputs, which are the resistance states of memristor A and B, are initialized to certain states by applying appropriate voltage pulse across the crossbar array. Then the resistance states are read out with a small pulse voltage $V_{READ}$ to verify the input information and the initialized results. To get a stable resistance state reflected by the read current signal, long time pulses are used. To realize the NAND logic operations, a trigger voltage pulse signal $V_N$ (3 V, 100 ns) is applied to produce the resistance state interaction among memristors. After that, a small voltage pulse $V_{READ}$ is applied on memristor A, B, and R to get each memristor’s final states after logic operating without destroying the resistance states of them.

3. Computing-in-Memory Controller architecture

3.1. Architecture

As shown in Figure 1, the Computing-in-Memory Controller (CiMC) architecture consists of an Instruction Memory, a control unit and several special function Registers. The CiMC is built on CMOS digital circuits, which sets excitation voltage on memristor crossbar through ADC, DA Cand analog selector devices in the analog zone.

The Instruction Memory is used to store programs of computing tasks. It uses an internal bus to transfer instructions to special function registers. The control unit manages the execution of the overall architecture. It fetches an instruction from the instruction arrays by sending a read instruction to it. The control unit consists of an instruction decoder, a finite state machine, a Micro-OP sequence RAM, two Micro-OP Excitation Decoders, an ADC controller, a DAC controller, and a Selector controller.

Micro-OP Excitation Decoders are divided into two types: Pulse-Type Decoder and Address Decoder. DecoderPulse-Type Decoder generates the excitation voltage type that should be set on each line of the memristor crossbar while the Address Decoder generates the address of the data that would be operated according to the present micro-operations stored in the Micro-operation sequence RAM. Micro-OP Excitation Decoder doesn’t drive the memristor crossbar directly with digital signal but by driving DAC controller, ADC controller and Selector controller. Micro-OP sequence RAM is used to store all the micro-operation sequences that corresponding to each CiM Instructions listed in Table 3.

3.2. Special Function Registers Table

Special Function Registers Table consists of eight Special Function Registers described in Table 1. The accessing mode is of the CiMC is an indirect mode. When we start a logic or arithmetic operations, it doesn’t need transfer data from one address to a unit like ALU (this unit doesn’t exist in the CiMC ), we just need to set both the base address and the index address of the data to be computed in registers. To execute an instruction, the control unit sends a read instruction to instruction memory to fetch the instruction. Its address is stored in the program counter register Ri.

| Address | Register name | Function                  |
|---------|---------------|---------------------------|
| 0x00    | Rn[31:0]      | Row address of the first operand |
| 0x01    | Rn2[31:0]     | Row address of the second operand |
| 0x02    | Rd[31:0]      | Row address of the result  |
| 0x03    | Rm[31:0]      | Immediate operand         |
| 0x04    | Ro[31:0]      | Output data               |
| 0x05    | Re[31:0]      | Control and status of CiMC |
| 0x06    | Rs[31:0]      | Reserved                  |
| 0x07    | Ri[31:0]      | Programm counter          |

4. Two-layer Instruction Set Architecture for CiM Controller

4.1. The first layer of CiMC Instruction

We present a two-layer instruction set architecture. The first layer is standard instruction used to be used in assemble programming by the assembler. It works in the same way as the instruction of traditional CPU. We design a basic version of the CiMC first layer instruction set, which contains tens
of instruction dived into three classes: Memory access class, Basic Class and Functional Class. Thus we could program based on the first layer instruction set to implement multiple processing functions such as data read, data write, boolean logic, complex logic, addition, subtraction and so on.

A CiMC instruction is in 32bit width, consisting of Byte OP, Rn1, Rn2 and Rd as shown in Table 2. Byte Rn1 and Rn2 contain the row addresses of the instruction inputs while Rd the output address.

| Table 2. Format of CiMC instruction |
|-------------------------------------|
| [31:24] | [23:16] | [15:8] | [7:0] |
| OP | Rn1 | Rn2 | Rd |
| Operation code | First operand address register | Second operand address register | Result operand address register |

The syntax and definition of all instructions are given in Table 3. CiMC use memory access instructions to transfer data between memristor crossbar and the outside environment. To transfer data within the crossbar, a copy instruction is used. The copy instruction can copy a row.

| Table 3. CiMC Instruction Set |
|-------------------------------|
| Instruction | Class | Opcode | Description |
| READ Rd | Memory Access | 0x01 | Read data in Row of Rd, and store the data in register Ro |
| WRITE Rd | Memory Access | 0x02 | Write data of register Rm to Row of Rd |
| SET Rd | Memory Access | 0x03 | Write zeros to Row of Rd |
| RESET Rd | Memory Access | 0x04 | Write ones to Row of Rd |
| COPY Rn1,Rn2 | Basic Class | 0x05 | Copy data in Row of Rn1 to Row of Rn2 |
| AND1 Rn1,Rn2,Rd | Basic Class | 0xF0 | Boolean AND logic of the first bit in Row Rn1 and Rn2 |
| NOR1 Rn1,Rn2,Rd | Basic Class | 0xF1 | Boolean NOR logic of the first bit in Row Rn1 and Rn2 |
| XOR1 Rn1,Rn2,Rd | Basic Class | 0xFF | Boolean XOR logic of the first bit in Row Rn1 and Rn2 |
| SLR1 Rn1 | Function Class | 0x11 | Shift data in Row of Rn1 to write |
| CMP Rn1,Rn2,Rd | Function Class | 0x21 | Copy data in Row of Rn1 and Row of Rn2 |
| ADD16 Rn1,Rn2,Rd | Function Class | 0x3F | Addition of 16bit data in Row of Rn1 and Row of Rn2 |
| MULT8 Rn1,Rn2,Rd | Function Class | 0x47 | Multiplication of 8bit data in Row of Rn1 and Row of Rn2 |

4.2. Micro Operation

The second layer instruction is named micro-operation, which is one excitation step on crossbar such as "COPY", "OR" and "NAND" logic operations, which are executed via resistance state interaction in one operation cycle. Micro operation is the basic operation to implement a standard instruction. Each instruction contains a serial of micro-operations, we call it Mico-OP Sequence. The operation time of a basic logic unit is equal to that of the memristor switching, which is the only speed limit of the computing gate. Arbitrary information - the resistance state stored in memristors can be involved in logic computing and the same devices can execute different logic operations with proper initialization and voltage excitation configuration during the runtime. Every one of the 16 Boolean logic can be achieved by logic units cascading within three operation steps. Complex computing operations, such as full adders, can be performed by synthesizing the primary and derived logic operations. The structure of units is also suitable for parallel computing, increasing the calculation efficiency. As illustrated in Table 4, a Micro Operation instruction consists of Micro Operation code and Crossbar address, which defines the Excitation Parameter and address for both row and column of the memristor crossbar. Five basic micro-operation codes could build all 16 boolean logics and any other functions. The type of the micro-operation determines the control voltages applied to the columns and rows as they are independent with the input data stored in each row.
### Table 4. Micro Operation instruction format, take NANDWL(1,2)-->4 for example

| WL  | VWL | WL_offset | VBL | BL_offset |
|-----|-----|-----------|-----|-----------|
| [63:48] | [47:32] | [31:16] | [15:0] |
| 0x0036 | 0x0000 | 0x0001 | 0x0004 |

| Row Excitation | Column Excitation | Row Offset | Column Offset |
|----------------|-------------------|------------|---------------|
| Parameter      | Parameter         | Address    | Address       |

| Micro Operation code | VWL | VBL |
|----------------------|-----|-----|
| ORWL                | [31:16] | [15:0] |
| NANDWL              | 0x0036 | 0x0000 |
| NA_ALWL             | 0x1031 | 0x0000 |
| NMAJWL              | 0x1041 | 0x0000 |
| COPYWL              | 0x0021 | 0x0000 |

| Row/Column Excitation Parameter |
|----------------------------------|
| Voltage Pulse Type | Voltage Pulse Bias | Base Row/Column number | Quantity of row/column |
| [15:12] | [11:8] | [7:4] | [3:0] |

With the basic micro-operations, not only all the Booleans but also some combination logic operations can be completed efficiently and robustly. Take full adder for example, which is a key and basic role in all computing systems. It takes 5n memristors and a 6n+4 Micro-OP Sequence to experimentally demonstrate the functions of n-bit full adder. The Mico-OP Sequence is illustrated in Table 5.

### Table 5. Micro-OP Sequence of n-bit addition

| STEP | LOGIC OPERATION | A0 | B0 | C0 | T0 | S0 | T1 | A1 | B1 | C1 | T2 | S1 | T4 | C2 | T5 |
|------|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1    | NMAt(A0,B0,C0)=T2 | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 1  | 0  | 0  | 1  | 1  | 1  |
| 2    | NAND(T2,T3)=T4  | A0 | B0 | C0 | 0  | 0  | C1 | A1 | B1 | 1  | 0  | 0  | C1 | 1  | 1  |
| 3    | NMAt(A1,B1,T0)=C1| A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 1  | 0  | 0  | C1 | 1  | 1  |
| 4    | NAND(C1,C2)=C2  | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 1  | 0  | 0  | C1 | 1  | 1  |
| 5    | RESET(C1)→C1    | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 0  | 0  | 0  | C1 | C2 | 1  |
| 6    | COPY(T5)→T4     | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 0  | 0  | 0  | C1 | C2 | 1  |
| 6n+1 | OR(A0,B0,T0)    | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 0  | 0  | C1 | 2  | 1  |
| 6n+2 | NA(A0,B0,T0)=T1 | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 0  | 0  | 0  | C1 | 2  | 1  |
| 6n+3 | OR(C1,T3)=S0    | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 0  | 0  | 0  | C1 | 2  | 1  |
| 6n+4 | NA(A0,C1,T0,S0)=S1 | A0 | B0 | C0 | 0  | 0  | C2 | A1 | B1 | 0  | 0  | 0  | C1 | 2  | 1  |

5. Conclusion

This paper proposes CiMC architecture including a controller architecture and a two-layer instruction set architecture. CiMC is the potential to significantly improve the execution time while using the same order of memory, as compared to von Neumann architectures. It is more important that CiMC provides a new angle to use memory to build a general-purpose processor for both data computing and storage. The two-layer CiMC instruction set supports read, write and copy, arithmetic and logical operations on memristor crossbar.
References

[1] Waldrop, M. M. The chips are down for Moore's law. Nature 530, 144-147 (2016).
[2] Wong, H.-S. P. & Salahuddin, S. J. n. Memory leads the way to better computing. 10, 191 (2015).
[3] Huang, P. et al. Reconfigurable Nonvolatile Logic Operations in Resistance Switching Crossbar Array for Large-Scale Circuits. Adv Mater 28, 9758-9764, DOI:10.1002/adma.201602418 (2016).
[4] Sun, Z., Ambrosi, E., Bricalli, A. & Ielmini, D. J. A. M. Logic Computing with Stateful Neural Networks of Resistive Switches. 1802554 (2018).
[5] Siemon, A. et al. Realization of Boolean Logic Functionality Using Redox-Based Memristive Devices. Advanced Functional Materials 25, 6414-6423, DOI:10.1002/adfm.201500865 (2015).
[6] Borghetti, J. et al. ‘Memristive’ switches enable ‘stateful’ logic operations via material implication. Nature 464, 873, DOI:10.1038/nature08940.
[7] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, “PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture,” in Proceedings of the 42nd Annual International Symposium on Computer Architecture. ACM, 2015, pp. 336-348.
[8] Gaillardon P E , Luca Amarú, Siemon A , et al. The Programmable Logic-in-Memory (PLiM) computer[C] 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2016.
[9] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-Memory for Energy-efficient Neural Network Training: A Heterogeneous Approach," in 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2018, pp. 655-668.
[10] M. Alian, S. W. Min, H. Asgharimoghaddam, A. Dhar, D. K. Wang, T. Roewer, A. McPadden, O. O’Halloran, D. Chen, J. Xiong, D. Kim, W. Hwu, and N. S. Kim, “Application-Transparent Near-Memory Processing Architecture with Memory Channel Network,” in 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Oct 2018, pp. 802-814.
[11] Zidan M., Jeong Y., Shin J.H., Du C., Zhang Z., Lu W. Field-programmable crossbar array (FPCA) for reconfigurable computing. IEEE Trans. Multi-Scale Comput. Syst. 2017.
[12] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in Memory With Spin-Transfer Torque Magnetic RAM," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 3, pp. 470-483, March 2018.
[13] Xumeng Zhang, Wei Wang, Qi Liu. An Artificial Neuron Based on a Threshold Switching Memristor[J]. IEEE Electron Device Letters, 2018, 39(2):308-311.
[14] S. Sun, J. Li, Z. Li, H. Liu, Q. Li, and H. Xu, “Low-consumption neuromorphic memristor architecture based on convolutional neural networks,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2018, pp. 1–6.
[15] S.-Y. Sun, Z. Li, J. Li, H. Liu, H. Liu, and Q. Li, “A memristor-based convolutional neural network with full parallelization architecture,” IEEICE Electron. Express, vol. 16, no. 3, 2019, Art. no. 20181034.
[16] He Zhang, Wang Kang, Lezhi Wang, Kang L. Wang, Weisheng Zhao. "Stateful Reconfigurable Logic via a Single-Voltage-Gated Spin Hall-Effect Driven Magnetic Tunnel Junction in a Spintronic Memory", IEEE Transactions on Electron Devices, 2017.