Algorithmic Considerations in Memristive Memory Processing Units (MPU)

Rotem Ben Hur, Nishil Talati, and Shahar Kvatinsky, *Member, IEEE*
Andrew & Erna Viterbi Faculty of Electrical Engineering
Technion – Israel Institute of Technology
Haifa 3200003, ISRAEL

***Abstract*—Memristive technologies are attractive candidates to replace conventional memory technologies, and can also be used to perform logic and arithmetic operations using a technique called 'stateful logic.' Combining data storage and computation in the memory array enables a novel non-von Neumann architecture, where both the operations are performed within a Memory Processing Unit (MPU). The use of an MPU alleviates the primary restriction on performance and energy in von Neumann machine, which is the data transfer between CPU and memory. To optimize the speed, energy, and area efficiency of the MPU, different algorithms need to be developed. This paper discusses the considerations in setting the sequence of computing operations in an MPU and presents examples of two operations that can benefit from processing within memristive memory.**

Keywords— Memristive systems, memristor, logic design, MAGIC, crossbar memory, memory controller, CPU, MPU.

I. Introduction

Currently, almost all general purpose computing systems use von Neumann architecture or an ameliorated version of it, which separates the processing unit(s) from the memory. Due to this separation, there is always a bus activity to and from processor and memory, which causes a massive overhead of power consumption and performance. This is called *von Neumann bottleneck*, and thus, researchers are trying to come up with substitute for this architecture.

Emerging nonvolatile resistive memory technologies, such as RRAM, PCM, STT-MRAM *etc*. (namely, memristors) are considered as attractive candidates to replace conventional memory technologies (*i.e.*, DRAM and Flash) by offering enhanced speed, lower power consumption, better scalability, and higher endurance [1]. In addition to standard storage capabilities, memristors can perform logic operations within the memory array using a technique called *stateful logic*, where the memristors are the primary building blocks of the logic gate and the resistance represents the logical state. Different voltage patterns across the bitlines and wordlines of the memory array lead to several stateful logic families [2-5]. In this paper, we use the *Memristor-Aided LoGIC (MAGIC)* family [5].

Adding computing capabilities to memristive memories enables the development of novel non-von Neumann architectures, where data storage and processing are combined, which we refer to as the *Memory Processing Unit* (MPU). This MPU maintains the structure of a standard memory and thus, it is compatible with conventional von Neumann architectures. Additionally, MPU enhances data processing and reduces energy for designated applications. Generally, the use of MPU allows alleviating the von Neumann bottleneck.

To maximize the benefits of the MPU, new algorithms for executing logic operations within memory are required to be defined and developed. These algorithms should be optimized by exploiting the parallelism offered by memristive memories. This paper describes the basic considerations of such in-memory processing algorithm design.

II. Processing Within Memristive Memories

MAGIC NOT and NOR are compatible for logic execution within memory due to the connection pattern among input and output memristors. In MAGIC NOT (NOR) execution, input memristor(s) is (are) excited with execution voltage V_0 and an output memristor is connected to ground [5]. This section discusses the concept of processing within memory, relevant design considerations and extension of MAGIC to various logic and arithmetic operations, with examples of full-adder and processing within an image. To exploit the symmetric feature of the memristive memory crossbar, a set of memory control and sense amplifiers can be replicated in the conventional memory circuit to access each memory cell from all directions (as opposed to only one direction in conventional memory). This is called the *transpose memory* [6], which adds the additional functionality to the memory crossbar in terms of logic execution.

A. Processing within an MPU - Concept and Considerations

To execute different logical and arithmetical operations, a definite sequence of NOT and NOR operations is performed within the MPU. Since performance, energy and area trade off in an MPU computation, different algorithmic approaches should be considered. For example, writing to many memristors simultaneously or duplicating data to different memristors prior to execution may dramatically improve the speed of computation. This performance optimization, however, increases the energy and occupied area. Alternatively, data can be moved or copied to any location within memory using two consecutive NOT operations (*i.e., A* = NOT(NOT(*A*))). A move/copy operation removes the wastage of memory space and energy in duplication, but lowers speed of execution. Furthermore, the choice of correct electrical parameters, such as the applied execution voltage V_0, influences the latency and energy of each MAGIC gate [6].

ISBN 978-3-8007-4252-3

Since every memristor is both a memory cell and also a part of the logic gate, storing and processing can be done at any desired location within a memory sub-array. Thus, each operation may be performed dynamically in different areas of the memory, controlled by a designated controller. However, to assure that the stored data is not erased while computing, expensive management of the memory by the operating system is essential. To alleviate the complexity of this control, processing areas either can be pre-defined, where the rest of the memory serves solely as storage, or can be defined dynamically in terms of location and size by the controller or the operating system. Furthermore, controlling different areas within the memory can also be used for wear leveling to increase the lifetime of the memory.

Execution of complex functions can be performed either by combining many lower levels of abstraction, which includes basic NOT/NOR operations or by using pre-developed higher abstraction level building blocks of simple operations (*e.g.*, AND/OR gates). The former may be optimized better but requires more investment in terms of algorithm design and a complicated memory controller.

Paralleling basic logical operations can be done at the lowest abstraction level of logic execution (logic gates). This data-level parallelism (DLP) is enabled, given a particular data distribution pattern within the memory. For example, the input memristors of MAGIC gates should be situated in the same bitlines within the same subarray (or the same wordlines for a transpose operation [6]); and output memristors should be situated in the same bitline (or same wordline for transpose memory [6]), as shown in Figure 1. Thus, when V_0 and ground voltages are asserted, multiple MAGIC gates can be implemented simultaneously. Hence, the effective execution time to compute logic over several such basic gates would be equal to a single gate delay, which can attain significant performance enhancement. This principle is further illustrated in the next subsection.

B. Vector and Matrix Operations – Toy Examples

To demonstrate the benefits from the parallelism offered by the MPU, we present addition operation of vectors. The one-bit full adder is implemented in terms of NOT and NOR operations. Note that both the sum and carry can be computed simultaneously under this representation [6]. Different sequences of operations can be performed to optimize either speed or area with approximately the same energy. The area-optimized approach computes the output of full adder using four redundant functional memristors (different from inputs and outputs) and 15 execution cycles [6]. The latency-optimized approach requires six functional memristors and 13 execution cycles to compute this output [6]. This bit-wise implementation of an adder can be extended to *N*-bit addition. Now, consider two vectors, each of length *M,* having *N*-bit integers as each argument. Thus, each *N*-bit addition can be executed in parallel assuming that the input vectors corresponding to each entry are present in different wordlines, in the same way as parallelizing the execution of MAGIC NOR gates. Hence, the effective time to compute vector addition boils down to the delay of adding a single entry of two vectors, which is independent of *M*.

One illustrative application of a matrix operation is image processing, where the same operation is performed on millions

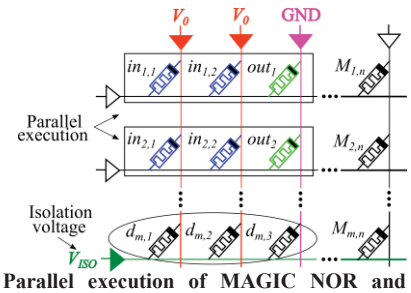

Figure 1: Parallel execution of MAGIC NOR and isolation of wordline(s) within MPU. Isolation is carried out by applying V_{ISO} [6].

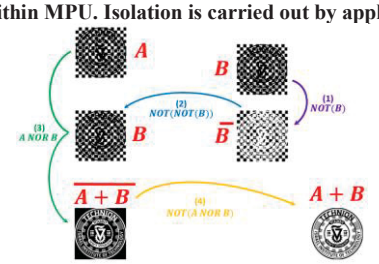

Figure 2: OR operation with an image of MxN pixels stored within a memristive memory using MAGIC: *A OR B = NOT(A NOR B)*
(1) NOT(B): M cycles and MxN operations. **(2)** ***NOT(NOT(B))=B***: N cycles and MxN operations. **(3)** ***A NOR B***: M cycles and MxN operations. **(4)** ***NOT(A NOR B)***: N cycles and MxN operations.
Takes a total of: 2M+2N cycles, 4MxN operations.

of pixels. Since an image is stored as a matrix in the memory, pixels from the same wordlines or bitlines can be processed simultaneously in the MPU. The number of cycles required to perform logic functions on an *M* by *N* image depends linearly on *MAX*(*N*, *M*), as illustrated in Figure 2. Since image processing is a data intensive application, the processing within memory, which diminishes the necessary data transfer, has the potential to dramatically reduce latency and power consumption, even for complex operations.

III. CONCLUSIONS AND FUTURE WORK

MPU is a recently developed architecture, where the memory element can function both as a data storage unit and a parallel processing element. To benefit from this concept, algorithmic framework has to be constructed. This paper discusses the algorithmic considerations and tradeoffs between performance, area and energy in such designs and describes data distribution considerations. Several such efforts have been made in this line of work, whereas further evaluations have to be made to determine which benefits the most between proposed approach and CMOS based computing.

ACKNOWLEDGMENT

This work is supported by Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI) and Viterbi Fellowship to the Technion Computer Engineering Center.

REFERENCES

[1] S. Kvatinsky *et al.*, "The Desired Memristor for Circuit Designers," *IEEE CAS Magazine*, Vol. 13, No. 2, pp. 17-22, Second Quarter 2013.

[2] S. Kvatinsky *et al.*, "Memristor-based IMPLY Logic Design Flow," *Proc. of IEEE ICCD*, pp.142-147, October 2011.

[3] J. Borghetti *et al.*, "Memristive Switches Enable Stateful Logic Operations via Material Implication," *Nature*, Vol. 464, pp. 873–876, Apr. 2010.

[4] E. Lehtonen *et al.*, "Recursive Algorithms in Memristive Logic Arrays," *IEEE JETCAS*, Vol. 5, No. 2, pp. 279-292, June 2015.

[5] S. Kvatinsky *et al.*, "MAGIC – Memristor Aided LoGIC," *IEEE TCAS-II*, Vol. 61, No. 11, pp. 895-899, November 2014.

[6] N. Talati *et al.*, "Logic Design within Memristive Memories Using Memristor-Aided loGIC (MAGIC)," *IEEE TNANO*, Vol. 15, No. 6, pp. 1-16, July 2016.