Real Processing-in-Memory with Memristive Memory Processing Unit (mMPU)

(Invited Paper)

Shahar Kvatinsky
Viterbi Faculty of Electrical Engineering
Technion – Israel Institute of Technology
Haifa, Israel 3200003
shahar@ee.technion.ac.il

Abstract— **Memristive technologies are attractive candidates to replace conventional memory technologies and can also be used to perform logic and arithmetic operations. In this paper, we show how memristors are used to combine data storage and computation in the memory, thus enabling a novel non-von Neumann architecture called the 'memristive memory processing unit' (mMPU). The mMPU relies on a memristive logic technique called 'memristor aided logic' (MAGIC) that requires no modification to the memory array structure. By greatly reducing the data transfer between the CPU and the memory, the mMPU alleviates the primary restriction on performance and energy efficiency in modern computing systems. This paper describes basic principles and design considerations of MAGIC and the mMPU and presents a case study of digital image processing to demonstrate the benefits.**

Keywords— memristor, memristive device, RRAM, mMPU, MAGIC

I. INTRODUCTION

General purpose computing systems are typically designed in von Neumann architecture, or an ameliorated version of it, which separates the memory and processing space. In these systems, programs are executed by moving data between the processing unit and memory using specific operations (load/store). While this programming model is simple, the performance of the system is limited by the memory access time, which is substantially higher than the computing time itself. This performance bottleneck has become even more severe over the years because CPU speed has improved much more than memory speed and bandwidth [1]. Moreover, many modern workloads have high and unstructured data volumes with limited locality, reducing the effectiveness of data caching.

With the demise of Dennard scaling [2], energy efficiency in computer systems has also become a daunting concern, and most modern computers are power limited [3]. A significant contributor to the power consumption of the system is the high energy cost of data movement, and especially of memory accesses [4]. For example, performing an add operation on 16-bit numbers in 45nm CMOS technology requires 0.18 pJ, while moving the same data on-chip requires about 11 pJ per mm (60X). Sending the same data to an off-chip DRAM consumes 640 pJ, 3600X more energy than the computation itself [5].

This separation of the processing and memory space – and thus the required transfer of data between them – constitute two main bottlenecks in current computing systems: speed (*'memory wall'*) and energy efficiency (*'power wall'*). A promising approach to overcome these challenges is to push the computation closer to the memory. Both DRAM and emerging non-volatile memory have ample intrinsic parallelism, which goes unutilized today because of the pin-limited integrated circuit interface. *Processing-In-Memory* (PIM) can tap this intrinsic parallelism, avoiding the need for high-latency and high-energy chip-to-chip transfers, thus yielding massively parallel, high-performance, energy-efficient processing [6].

Early research into PIM dates back to the '90s, but four major challenges prevented its widespread adoption [7]. The first challenge was inadequate implementation technology. Although attempts were made to integrate the memory and CPU on the same die, the incompatibility of DRAM and CPU fabrication technologies made it difficult to incorporate these approaches in practical computing systems. The second was the lack of a processor architecture that could use the high bandwidth enabled by proximity to memory. Early PIM research required custom architectures, involving huge design and development efforts. The third challenge was to develop interfaces that allowed both the PIM computing units and the external processing units to access memory. Early efforts required the design and adoption of custom memory interfaces. The fourth challenge was the programming models. The early approaches had to develop the programming abstractions from the bottom up.

In the modern age, advancements in the technologies and methodology of building computer systems have made it easier to address these challenges. For example, the first challenge – adequate implementation technology – has been addressed by the emergence of 3D die stacking, which enables heterogeneous integration of logic and memory, and by emerging memory technologies, which enable 3D fabrication of memory arrays on top of CMOS substrates [8]. Evolution of various other processing platforms, such as GPGPUs and custom accelerators, have solved the second problem by efficiently

This research is partially supported by the European Research Council under the European Union's Horizon 2020 Research and Innovation Programme (grant agreement no. 757259) and by the Israel Science Foundation grant no. 1514/17.

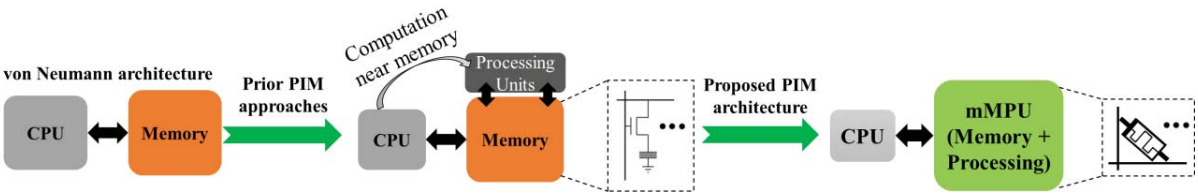

Figure 1. Architectural evolution of eliminating the von Neumann bottleneck by moving processing into the memory. Moving from von Neumann machines with separate computation and storage to near data processing, and finally to the proposed architecture that eliminates a significant amount of data transfer using the same cells that store the data to perform logical operations within the mMPU.

utilizing the high memory bandwidth within the thermal constraints of the memory modules [9]. Recent die-stacked memory interface standards (such as High Bandwidth Memory [10]) and off-chip memory interfaces that expose load-store semantics (such as Hybrid Memory Cube [11]) meet nearly all the memory interface requirements of PIM, thus overcoming the third challenge. Recent frameworks such as Heterogeneous System Architecture and the associated software tools for accelerators have addressed the fourth challenge. However, a whole new set of problems has arisen in the wake of these advancements. For example, workload heterogeneity can be difficult to maintain when different algorithms result in different memory layouts and access patterns, and the computations they execute have varying degrees of parallelism and complexity. Viable methods for fabricating embedded non-volatile memory (eNVM) are another critical issue.

Current state-of-the-art efforts in PIM using standard technology include computation in the periphery of SRAM [12] and DRAM [13, 14] and required modifications in the peripheral circuits to allow computation in the sensing circuitry, as well as in specific memory cells. While these efforts more tightly integrate processing and storage, data still moves between the memory cells and the periphery. Furthermore, DRAM and especially SRAM are relatively small memories and therefore cannot provide a platform for computing massive amounts of data (tens or hundreds of GBs).

Emerging *memristive technologies*, such as Resistive Random Access Memory (RRAM or ReRAM) [15-17], might serve to enable high capacity non-volatile memory that can also compute. Memristive accelerators have been proposed for enhancing artificial neural networks by using the analog computation capabilities of memristors [18-21]. Other memristive accelerators have been based on performing content addressable searches [21] and associative computing [22-23].

In this paper, we present a fundamentally different approach that tackles the data movement problem directly by computing logical functions in the memory cells themselves, without any need to instantiate additional CMOS blocks for processing. We propose to solve the von Neumann bottleneck by giving computational capabilities directly to the memristive memory cells, as illustrated in Figure 1.

The memristive memory cells can both store data and be used to construct a logic gate using the same circuit and different control signals. Several memristive logic techniques have been proposed in the literature. The focus of this paper is *Memristor Aided Logic* (MAGIC) [24]. One advantage of MAGIC is its ability to efficiently perform logic operations in parallel on sets of data. We exploit the processing capabilities that MAGIC adds to memristive memories to design a novel non-von Neumann architecture that significantly reduces data transfer. The basic element in this architecture is the *memristive Memory Processing Unit* (mMPU) [25], which has both storage and processing capabilities. The mMPU consists of a memristive memory and CMOS control, exactly the same structure as a standard memristive memory, where the primary circuit modifications are made in the controller. Hence, the advantages of a memristive crossbar array, such as density and nonvolatility, are maintained. Furthermore, the mMPU is completely compatible with von Neumann architecture, and can operate either as a hybrid memory-processing unit or as a standard memory. The structure of a conventional von Neumann architecture and that of the proposed architecture with an mMPU as the memory are illustrated in Figure 2.

The proposed architecture dramatically reduces data movement and has enormous parallelism. Nevertheless, its design requires that we take a fresh look at computer architecture and reconsider various aspects such as the memory controller, programming model, memory hierarchy and so forth. In this paper, we provide background on memristive logic (Section II) and specifically on MAGIC (Section III). Then, we present the mMPU architecture (Section IV) and discuss different aspects of the controller design. Finally, we illustrate the benefits of the mMPU using an image processing application example and show that processing images within the mMPU can provide two orders of magnitude speedup versus state-of-the-art accelerators (Section VII).

II. Memristive Logic Approaches

Emerging memristive technologies store data as the resistance of the memory cell, a two-terminal passive circuit element with varying resistance called a *memristor*. High and low resistances are considered, respectively, as logic '0' and '1'. Since the resistance value is usually continuous, it is possible to consider the data as multi-level (*i.e.*, store more than a single bit per cell). The resistance of the memristor can be changed by applying a voltage across it, where the exact mechanism differs in each memristive technology (*e.g.*, movement of oxygen vacancies in Valence Change Memory). Memristors are usually fabricated using dielectric material between two metal electrodes. Hence, they can be used in a crossbar array as a crosspoint of two metal wires, enabling an extremely dense memory [15], [26-27]. The memory cell can consist solely of a memristor, or it can include a selector to eliminate undesired

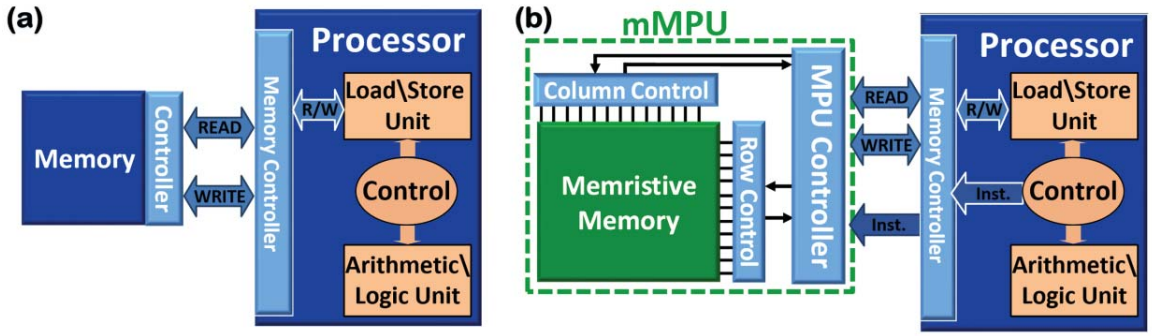

Figure 2. Structure of a (a) von-Neumann architecture and (b) the proposed architecture with an mMPU as the memory. In addition to standard memory operations, the mMPU can get a computing instruction to be performed in-memory.

sneak path currents [28-33]. The symmetry of the crossbar structure enables a *transpose memory* structure [34] with added flexibility by applying voltages from both horizontal and vertical directions.

Due to their high switching speed, low operating power, scalability, and high endurance [35], memristors are considered as attractive candidates to replace conventional memory and storage technologies (*e.g.*, DRAM and Flash). Memristive technologies have also been explored for additional applications such as analog and radiofrequency circuits [36-42], neuromorphic circuits [43-48], and logic circuits, which are the focus of this paper. Different methods for using memristors to perform logical operations have been proposed. Several techniques have been proposed to perform processing near the memory, similar to CMOS-based PIM, where memristors are used only as memory cells, exploiting their density and tight integration with CMOS periphery circuits [18-24], [49-52]. In some other logic families, memristors are integrated with CMOS logic structures as configurable switches or as logic gates [53-57]. In these logic families, the logical values are represented by voltage levels, and therefore these techniques cannot be used to perform computation within the memory cells unless the data is read explicitly and transformed from resistance to voltage.

Several logic families use the structure of a memristive memory to perform logical operations to overcome the memory wall [58-63], and we have defined a classification for memristive logic in that context [64]. This classification has three categories:

- *Statefulness* – A memristive logic family is said to be stateful if the Boolean variable is represented only as the state of the memristor (*i.e.*, resistance) and computation is performed by manipulating this state. In other words, inputs are represented as resistance, and the output(s) after computation is (are) also stored as the resistance of the memristor. Statefulness is a fundamental classification because it has far–reaching effects on the compatibility of the logic family with other units, such as CMOS-based circuits and memristive memory cells. If the circuits are incompatible, state conversion (from resistance to voltage or vice versa) of consecutive logic operations will be required, influencing performance, power, and area. Consequently, statefulness is a desired characteristic for computation within memristive memory since computation is performed using the same logic state variables as represented in the memory cells. On the other hand, non-stateful families benefit from better integration with CMOS.
- *Proximity of computation* – We redefine PIM and near-memory computing with respect to the proximity of the data to the memory array during computation and call this the proximity of computation. We define the memory array as a regular array of memory cells to store data, replicated in two dimensions, the wordline (WL) and the bitline (BL), and not including its auxiliary circuit. We redefine processing-in-memory as 'in-memory computing' and define it as the computing model in which the data resides within the memory array during the entire computation. We redefine 'near-memory computing' as the computing model that requires data movement to the auxiliary circuit (*e.g.*, for state conversion) during the computation, even if some (or most) of the computation is carried out by the memory cells. Hence, the memory array is the point of reference in our definition. In out-of-memory computing, computation may even be performed in another die (*e.g.*, a logic die beneath a DRAM die as in the hybrid memory cube) or in another chip (as in conventional von Neumann machines).
- *Flexibility* - A memristive logic family is said to be flexible if a variety of operations can be executed using the same computing elements. To achieve flexibility, a logic family must provide a basic operation (or a set thereof) that is functionally complete, and allow different control signal sequences to result in different outcomes. Some logic families are similar to ASIC, where the functionality of each computing element is determined prior to the fabrication process. Hence, they can perform a fixed function (or a set of functions). Other logic techniques can be executed using the same computing units at different execution times, and therefore the functionality can be dynamically chosen during runtime. Flexible families require a controller that conducts the execution of the desired program using the adjustable computing elements and synchronizes the sequence of basic logic operations supported by the family. Some sort of compiler or logic synthesis tool is necessary to generate

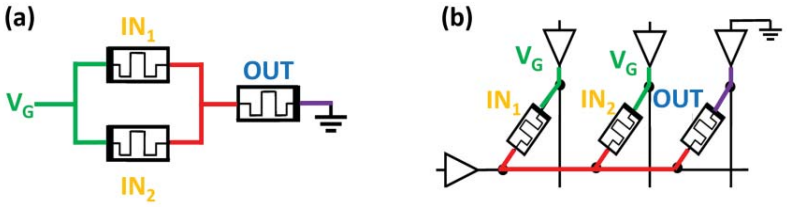

Figure 3. Schematic of (a) MAGIC NOR gate and (b) MAGIC NOR gate within a memristive memory array. IN_1 and IN_2 are the input memristors and OUT is the output memristor. A single voltage V_G is applied to perform the NOR operation [24].

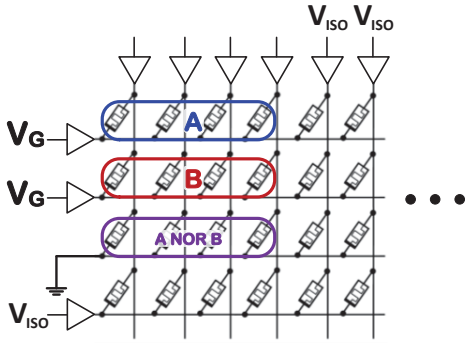

Figure 4. A MAGIC NOR operation between two row vectors A and B is performed within the memristive memory array by applying V_G to the wordlines of the input memristors, ground to the wordline of the output memristor, and V_{ISO} to isolate unselected bitlines and wordlines. The operation takes a single clock cycle regardless of the vector size of A and B [34].

an efficient sequence of basic logic operations to realize a desired function. Using an inadequate synthesis tool can lead to an inefficient logic implementation in terms of performance and/or power, while the proper use of it can result in a cost-effective design. Non-flexible families can become programmable in a similar manner to a general purpose CPU, where the designed fixed-functions are sufficient to perform any required task and construct a desired datapath. Programmable non-flexible families can compute any desired operation but cannot be used in or near memory since they cannot be made compatible with the memory array.

III. Memristor Aided Logic (MAGIC)

Recently, we have proposed *Memristor-Aided loGIC* (MAGIC) [24], a stateful, in-memory, flexible logic family. In MAGIC, only a single voltage V_G is used to perform a NOR logic operation and there are separate input and output memristors, as shown in Figure 3. Additionally, MAGIC gates do not require additional devices to perform the operation (unlike some families that require an additional resistor for each wordline). Since NOR is a complete logic function, a MAGIC NOR operation is sufficient to execute any Boolean operation. Hence, MAGIC NOR can be the basis for performing all desired processing within memory by dividing the desired function into a sequence of MAGIC NOR operations. These basic NOR operations are executed one after the other using the memory cells as computation elements. MAGIC can also be used to perform logic operations in parallel on sets of data. The crossbar array is structured such that applying the operating voltage V_G on any two selected rows and grounding a third row will will result in NOR operations being performed on all columns that were not isolated by applying an isolation voltage V_{ISO}. The schematic of a MAGIC gate operation, performed over row vectors within a memristive memory, is shown in Figure 4. Note that due to the symmetry of memristive crossbar arrays (*i.e.*, transpose memory), performing NOR operations on column vectors is similarly feasible.

IV. Memristive Memory Processing Unit Architecture

The mMPU [25], [65] is a standard RRAM memory with a few modifications that enable the support of MAGIC-based PIM instructions. In other words, the mMPU functions as a standard memory that supports memory operations (*i.e.*, read and write) with additional PIM capabilities, and thus it is backward compatible with the von Neumann computing scheme. The mMPU architecture is shown in Figure 2b.

To support PIM instructions, the memory controller [66], the memory protocol [67], and the peripheral circuits (*i.e.*, voltage drivers and row/column decoders) must be modified to support MAGIC instructions [68-69]. The mapping of data is also modified to maintain persistency and coherence. Note, however, that the memory crossbar array structure itself is not modified.

V. mMPU Controller Design

The mMPU CMOS controller is a finite state machine that supports standard and PIM memory instructions by generating the necessary control signals. The controller receives the commands from the CPU and performs the decoded instruction. The PIM instructions are translated to a pre-synthesized and optimized sequence of MAGIC NOR gates. To execute different applications in-memory using MAGIC, algorithms that translate these applications to an optimized sequence of

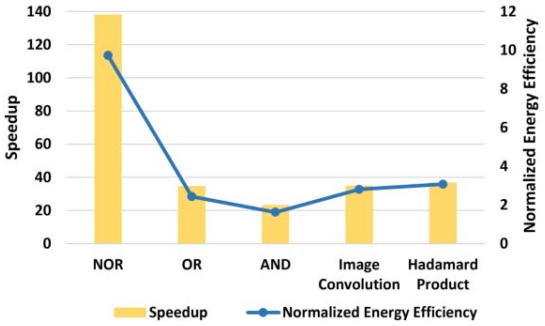

Figure 5. Speedup (left) and normalized energy efficiency (right) of mMPU as compared to Pinatubo [74].

MAGIC NOR/NOT gates must be developed [64]. These can be generated manually or automatically.

We have proposed optimized, manual algorithms for performing Fixed-Point (FiP) addition [34] and multiplication [70] using MAGIC. For automatically generated algorithms, we proposed SIMPLE [71]: an automatic synthesis tool that receives any Boolean function and automatically generates the equivalent, optimal sequence of MAGIC NOR operations. The operation is converted to a NOR CMOS based netlist which is mapped to a sequence of MAGIC NOR gates by solving an optimization problem. Such a tool will serve as the basis for the mMPU controller design, with the manual mapping left for specific tasks.

Due to the parallel nature of MAGIC, the tasks that benefit most from execution in the mMPU consist of simple SIMD operations. Each operation will be performed in a single row. For example, to add two vectors of 512 elements, each in a MAT of size 512X512, each row in the MAT will store two elements, one from each vector, and all the elements from each vector will share the same columns so that all the elements will be added simultaneously. This execution scheme substantially improves the throughput (number of executions per cycle). We further extended the synthesis tool to SIMPLER [72]. SIMPLER maximizes the throughput by performing the computation on a single row and concurrently executes numerous MAGIC NOR operations on multiple rows.

VI. Image Processing Case Study

We have demonstrated the benefits from an mMPU for image processing [65], [73], where numerous pixels are processed simultaneously and the same instruction is executed in parallel on multiple data. Image manipulation therefore requires data-intensive computations, often in real time, and the necessity for data movement only intensifies as image resolution becomes higher.

We evaluated different bit-wise operations and image processing kernels such as image convolution and Hadamard product. To execute the image processing kernels, we extended the manual algorithm for fixed-point multiplication [IMAGING]. For the image-processing tasks, we use the CIFAR-10 image classification benchmark dataset, a test set of 10,000 images, where instances are 32×32 color (RGB) images representing airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. For image convolution, we run a layer of 3×3 filters used for sharpening and edge detection on the dataset. The filters are slid over the images, and their values are multiplied by the corresponding pixel values. For the Hadamard product, we perform elementwise multiplications between the images and 32×32 matrices used for pattern recognition and lossy compression algorithms such as JPEG. The full evaluation methodology is detailed in [65].

Figure 5 compares the mMPU to Pinatubo [74], a state-of-the-art memristive accelerator. Our results demonstrate 20X to 120X speedup for bit-wise operations, and more than 30X speedup for Hadamard product and image convolution with a more modest 2X to 9X improvement in energy efficiency.

VII. Conclusions

The memristive memory processing unit (mMPU) is a true processing-in-memory approach, where the same cells are used both for data storage and computation. Its massive parallel processing capabilities, together with the elimination of data movement, make the mMPU a great contender to replace the standard memory unit, while maintaining backward compatibility with von Neumann machines.

To make the mMPU feasible, further research is required. For instance, the mMPU/CPU interface must be specified by defining an instruction set architecture (ISA) and an associated programming model. Additionally, memristive logic families such as MAGIC must be demonstrated in large scale memory arrays. Finally, the superiority of the mMPU over the GPU, CPU and other types of processing units must be demonstrated for additional applications.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Morgan Kauffman, 2017.

[2] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, "Design of Ion-Implanted MOSFET's With Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, Vol. 9, No. 5, pp. 256–268, Oct. 1974.

[3] S. W. Keckler, "GPU Computing and the Road to Extreme-Scale Parallel Systems," *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, p. 1, Nov. 2011.

[4] M. Horowitz, "Computing's Energy Problem (and what we can do about it)," *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 10–14, Feb. 2014.

[5] A. Pedram, S. Richardson, S. Galal, S. Kvatinsky, and M. Horowitz, "Dark Memory and Accelerator-Rich System Optimization in the Dark Silicon Era," *IEEE Design and Test*, Vol, 34, No. 2, pp. 39-50, April 2017.

[6] R. Balasubramonian and B. Grot, "Near-Data Processing," *IEEE Micro*, Vol. 36, No. 1, pp. 4–5, Jan. 2016.

[7] N. Jayasena, "Overcoming Challenges to Near-Data Processing," *IEEE Micro*, Vol. 36, No. 1, pp. 8–9, Feb. 2016.

[8] HSA Foundation, "Harmonizing the Industry around Heterogeneous Computing." [Online]. Available: http://www.hsafoundation.com/.

[9] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal Feasibility of Die-Stacked Processing in Memory," *Proceedings of the 2nd Workshop Near-Data Processing*, Dec. 2014.

[10] JEDEC Solid State Technology Association, "High Bandwidth Memory (HBM) DRAM," JESD235A, 2015. [Online]. Available: http://www.jedec.org/standards-documents/results/jesd235.

[11] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 1.0." 2013.

[12] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 383-396, June 2018.

[13] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 273-287, October 2017.

[14] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: a DRAM-based Reconfigurable In-Situ Accelerator, *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 288-301, October 2017.

[15] H. S. P. Wong, H. Y. Lee, S. Yu, Y. S. Chen, Y. Wu, P. S. Chen, B. Lee, F. T. Chen, and M. J. Tsai, "Metal-oxide RRAM," *Proceedings of the IEEE*, Vol. 100, No. 6, pp. 1951–1970, Jun. 2012.

[16] W. Woods, M. M. A. Taha, S. J. Dat Tran, J. Burger, and C. Teuscher, "Memristor Panic - A Survey of Different Device Models in Crossbar Architectures," *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 106–111, Jul. 2015.

[17] J. Lee, M. Jo, D. Seong, J. Shin, and H. Hwang, "Materials and Process Aspect of Cross-Point RRAM," *Microelectronic Engineering*, Vol. 88, No. 7, pp. 1113–1118, Jul. 2011.

[18] M. N. Bojnordi and E. Ipek, "Memristive Boltzmann Machine: A Hardware Accelerator for Combinatorial Optimization and Deep Learning," *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–13, Mar. 2016.

[19] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory," *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 27-39, Jun. 2016.

[20] A. Shafiee *et al*., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pp. 14-26, June 2016.

[21] Q. Guo, X. Guo, Y. Bai, and E. İpek, "A Resistive TCAM Accelerator for Data-Intensive Computing Categories and Subject Descriptors," *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 339–350, 2011.

[22] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: Associative Computing with STT-MRAM," *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pp. 189-200, June 2013.

[23] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive Associative Processor," *IEEE Computer Architecture Letters*, Vol. 14, No. 2, pp. 148–151, Jul. 2015.

[24] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC - Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 61, No. 11, pp. 895–899, Nov. 2014.

[25] R. Ben Hur and S. Kvatinsky, "Memory Processing Unit for In-Memory Processing," *Proceedings of the IEEE International Symposium on Nanoscale Architectures*, pp. 171-172, July 2016.

[26] Z. Jiang, P. Huang, L. Zhao, S. Kvatinsky, S. Yu, X. Liu, J. Kang, Y. Nishi, and H.-S. P. Wong, "Analysis and Predication on Resistive Random Access Memory (RRAM) 1S1R Array," *Proceedings of the International Memory Workshop*, pp. 1–4, May 2015.

[27] R. Waser, R. Dittmann, C. Staikov, and K. Szot, "Redox-Based Resistive Switching Memories Nanoionic Mechanisms, Prospects, and Challenges," *Advanced Materials*, Vol. 21, No. 25–26, pp. 2632–2663, July 2009.

[28] S.-S. Sheu, P.-C. Chiang, W.-P. Lin, H.-Y. Lee, P.-S. Chen, Y.-S. Chen, T.-Y. Wu, F. T. Chen, K.-L. Su, M.-J. Kao, K.-H. Cheng, and M.-J. Tsai, "A 5ns Fast Write Multi-Level Non-Volatile 1 K Bits RRAM Memory with Advance Write Scheme," *IEEE Symposium on VLSI Circuits*, pp. 82–83, June 2009.

[29] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-Based Memory: The Sneak Paths Problem and Solutions," *Microelectronics Journal*, Vol. 44, No. 2, pp. 176–183, 2013.

[30] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-Path Constraints in Memristor Crossbar Arrays," *Proceedings of the IEEE International Symposium on Information Theory*, pp. 156–160, JulY 2013.

[31] A. Doz, I. Goldstein, and S. Kvatinsky, "Analysis of the Row Grounding Method in a Memristor-Based Crossbar Array," *International Journal of Circuit Theory and Applications*, Vol. 46, No. 1, pp. 122-137, January 2018.

[32] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Information-Theoretic Sneak Path Mitigation in Memristor Crossbar Arrays," *IEEE Transactions on Information Theory*, Vol. 62, No. 9, pp. 4801-4814, September 2016.

[33] F. Pan, C. Chen, Z. Wang, Y. Yang, J. Yang, and F. Zeng, "Nonvolatile Resistive Switching Memories-Characteristics, Mechanisms and Challenges," *Progress in Natural Science: Materials International*, Vol. 20, pp. 1–15, Nov. 2010.

[34] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC)," *IEEE Transactions on Nanotechnology*, Vol. 15, No. 4, pp. 635–650, Jul. 2016.

[35] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "The Desired Memristor for Circuit Designers," *IEEE Circuits and Systems*, Vol. 13, No. 2, pp. 17–22, Jun. 2013.

[36] Y. V. Pershin and M. Di Ventra, "Practical Approach to Programmable Analog Circuits with Memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 57, No. 8, pp. 1857–1864, Aug. 2010.

[37] D. Mahalanabis, V. Bharadwaj, H. J. Barnaby, S. Vrudhula, and M. N. Kozicki, "A Nonvolatile Sense Amplifier Flip-Flop Using Programmable Metallization Cells," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol. 5, No. 2, pp. 205–213, Jun. 2015.

[38] M. Itoh and L. O. Chua, "Memristor Oscillators," *International Journal of Bifurcation and Chaos*, Vol. 18, No. 11, pp. 3183–3206, Nov. 2008.

[39] L. Danial, N. Wainstein, S. Kraus, and S. Kvatinsky, "Breaking Through the Speed-Power-Accuracy Tradeoff in ADCs using a Memristive Neuromorphic Architecture," *IEEE Transactions on Emerging Topics in Computational Intelligence*, Vol. 2, No.5, pp. 396-409, October 2018.

[40] N. Wainstein and S. Kvatinsky, "A Lumped RF Model for Nanoscale Memristive Devices and Non-Volatile Single-Pole Double-Throw Switches," *IEEE Transactions on Nanotechnology*, Vol. 17, No. 5, pp. 873-883, September 2018.

[41] N. Wainstein and S. Kvatinsky, "TIME – Tunable Inductors using MEmristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 65, No. 5, pp. 1505-1515, May 2018.

[42] L. Danial, N. Wainstein, S. Kraus, and S. Kvatinsky, "DIDACTIC: A Deeply Intelligent Digital-to-Analog Converter with a Trainable Integrated Circuit using Memristors," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol. 8, No. 1, pp. 146-158, March 2018.

[43] A. Afifi, A. Ayatollahi, and F. Raissi, "Implementation of Biologically Plausible Spiking Neural Network Models on the Memristor Crossbar-Based CMOS/Nano Circuits," *Proceedings of the European Conference on Circuit Theory and Design*, pp. 563–566, Aug. 2009.

[44] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 26, No. 10, pp. 2408–2421, Oct. 2015.

[45] S. P. Adhikari, C. Yang, H. Kim, and L. O. Chua, "Memristor Bridge Synapse-Based Neural Network and its Learning," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 23, No. 9, pp. 1426–1435, Sep. 2012.

[46] X. Liu, Z. Zeng, S. Member, and S. Wen, "Implementation of Memristive Neural Network With Full-function Pavlov Associative Memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 63, No. 9, pp. 1454–1463, Sep. 2016.

[47] T. Greenberg-Toledo, R. Mazor, A. Haj Ali, and S. Kvatinsky, "Supporting the Momentum Algorithm Using a Memristor-Based Synapse," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 66, No. 4, pp. 1571-1583, April 2019.

[48] E. Rosenthal, S. Greshnikov, D. Soudry, and S. Kvatinsky, “A Fully Analog Memristor-Based Multilayer Neural Network with Online Backpropagation Training,” *Proceeding of the IEEE International Symposium on Circuits and Systems*, pp. 1394-1397, May 2016.

[49] A. Morad, L. Yavits, S. Kvatinsky, and R. Ginosar, “Resistive GP-SIMD Processing-In-Memory,” *ACM Transactions on Architecture and Code Optimization*, Vol. 12, No. 4, pp. 1–22, Jan. 2016.

[50] K. Eshraghian, K. R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, “Memristor MOS Content Addressable Memory (MCAM): Hybrid Architecture for Future High Performance Search Engines,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 19, No. 8, pp. 1407–1407, Aug. 2010.

[51] A. Rahimi, A. Ghofrani, K. Cheng, L. Benini, and R. K. Gupta, “Approximate Associative Memristive Memory for Energy-Efficient GPUs,” *Proceedings of the Design, Automation, and Test in Europe Conference & Exhibition (DATE)*, pp. 1497–1502, Mar. 2015.

[52] M. Imani and T. Rosing, "CAP: Configurable Resistive Associative Processor for Near-Data Computing," *Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED)*, pp. 346-352, March 2017.

[53] D. B. Strukov and K. K. Likharev, “CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits with Two-Terminal Nanodevices,” *Nanotechnology*, Vol. 16, No. 6, pp. 888–900, Apr. 2005.

[54] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, “MRL — Memristor Ratioed Logic,” *Proceedings of the International Workshop on Cellular Nanoscale Networks and their Applications*, pp. 1–6, Aug. 2012.

[55] A. K. Maan, D. S. Kumar, S. Sugathan, and A. P. James, “Memristive Threshold Logic Circuit Design of Fast Moving Object Detection,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 23, No. 10, pp. 2337–2341, Oct. 2014.

[56] P. E. Gaillardon, M. H. Ben-Jamaa, G. Betti Beneventi, F. Clermidy, and L. Pemiola, “Emerging Memory Technologies for Reconfigurable Routing in FPGA Architecture,” *Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 62–65, Dec. 2010.

[57] W. Wei, T. T. Jing, and B. Butcher, “FPGA Based on Integration of Memristors and CMOS Devices,” *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1963–1966, May 2010.

[58] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, “Beyond von Neumann—Logic Operations in Passive Crossbar Arrays Alongside Memory Operations,” *Nanotechnology*, Vol. 23, No. 30, p. 305205, Jun. 2012.

[59] P. E. Gaillardon, L. Amaru, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, “The Programmable Logic-in-Memory (PLiM) Computer,” *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 427–432, Mar. 2016.

[60] A. Siemon, S. Menzel, R. Waser, and E. Linn, “A Complementary Resistive Switch-Based Crossbar Array Adder,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol. 5, No. 1, pp. 64–74, Mar. 2015.

[61] Y. Levy, J. Bruck, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaakobi, and S. Kvatinsky, “Logic Operations in Memory Using a Memristive Akers Array,” *Microelectronics Journal*, Vol. 45, No. 11, pp. 1429–1437, Nov. 2014.

[62] L. Xie, H. A. Du Nguyen, M. Taouil, and K. Bertels Said Hamdioui, “Fast Boolean Logic Mapped on Memristor Crossbar,” *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pp. 335–342, Oct. 2015.

[63] G. Snider, “Computing with Hysteretic Resistor Crossbars,” *Applied Physics A: Materials Science and Processing*, Vol. 80, No. 6, pp. 1165–1172, Mar. 2005.

[64] J. Reuben, R. Ben Hur, N. Wald, N. Talati, A. Haj Ali, P.-E. Gaillardon, and S. Kvatinsky, "Memristive Logic: A Framework for Evaluation and Comparison," *Proceeding of the IEEE International Symposium on Power and Timing Modeling, Optimization and Simulation*, pp. 1-8, September 2017.

[65] A. Haj Ali, R. Ben Hur, N. Wald, R. Ronen, and S. Kvatinsky, "Not in Name Alone: A Memristive Memory Processing Unit for Real In-Memory Processing," *IEEE Micro*, Vol. 38, No. 5, pp. 13-21, September/October 2018.

[66] R. Ben-Hur and S. Kvatinsky, "Memristive Memory Processing Unit (MPU) Controller for In-Memory Processing", *Proceedings of the IEEE International Conference on Science of Electrical Engineering*, pp. 1-5, November 2016.

[67] N. Talati, H. Ha, B. Perach, R. Ronen, and S. Kvatinsky, "CONCEPT: A Column Oriented Memory Controller for Efficient Memory and PIM Operations in RRAM," *IEEE Micro,* Vol/ 39, No. 1, pp. 33-43, January/February 2019.

[68] N. Wald and S. Kvatinsky, "Influence of Parameter Variations and Environment for Real Processing-In-Memory using Memristor Aided Logic (MAGIC)," *Microelectronics Journal*, Vol. 86, pp. 22-33, April 2019.

[69] N. Talati, A. Haj Ali, R. Ben Hur, N. Wald, R. Ronen, P.-E. Gaillardon, and S. Kvatinsky, "Practical Challenges in Delivering the Promises of Real Processing-in-Memory Machines," *Proceedings of the Design Automation and Test in Europe*, pp. 1628-1633, March 2018.

[70] A. Haj Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky, "Efficient Algorithms for In-Memory Fixed Point Multiplication Using MAGIC," *Proceeding of the IEEE International Symposium on Circuits and Systems*, pp. 1-5, May 2018.

[71] R. Ben Hur, N. Wald, N. Talati, and S. Kvatinsky, "SIMPLE MAGIC: Synthesis and Mapping of Boolean Functions for Memristor Aided Logic (MAGIC)," *Proceeding of the IEEE International Conference on Computer Aided Design*, pp. 225-232, November 2017.

[72] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, and S. Kvatinsky, "SIMPLER MAGIC: Synthesis and Mapping of In-Memory Logic Executed in a Single Row to Improve Throughput," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (to appear).

[73] A. Haj Ali, R. Ben-Hur, N. Wald, R. Ronen, and S. Kvatinsky, "IMAGING - In-Memory AlGorithms for Image ProcessiNG," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 65, No. 12, pp. 4258-4271, December 2018.

[74] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie,. "Pinatubo: a Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," *Proceedings of the Annual Design Automation Conference (DAC)*, Article 173, June 2016.