THEME ARTICLE: Memristor-Based Computing

Not in Name Alone: A Memristive Memory Processing Unit for Real In-Memory Processing

Ameer Haj-Ali, Rotem Ben-Hur, Nimrod Wald, Ronny Ronen, and Shahar Kvatinsky
Technion – Israel Institute of Technology

Data movement between processing and memory is the root cause of the limited performance and energy efficiency in modern von Neumann systems. To overcome the data-movement bottleneck, we present the memristive Memory Processing Unit (mMPU)—a real processing-in-memory system in which the computation is done directly in the memory cells, thus eliminating the necessity for data transfer. Furthermore, with its enormous inner parallelism, this system is ideal for data-intensive applications that are based on single instruction, multiple data (SIMD)—providing high throughput and energy-efficiency.

Modern computers are typically based on the von Neumann architecture, in which the memory is separated from the processing space and programs are executed by moving data between the processing and memory units. This incessant data movement is the lead cause of the performance bottleneck known as the memory wall, which has increased in severity over the years as CPU speed improvements have surpassed those of memory speed and bandwidth. Furthermore, with the demise of Dennard scaling, energy-efficiency is becoming a major concern in modern computers; for example, moving data to an off-chip DRAM consumes four orders of magnitude more energy than the computation itself.[1]

One approach to addressing the challenges arising from data movement is to move the computation closer to the memory. Both DRAM and emerging non-volatile memory technologies can provide ample intrinsic parallelism, which goes unutilized today due to pin-limited integrated circuit interfaces. Processing in memory (PIM) can leverage this intrinsic parallelism by avoiding the need for high-latency and high-energy chip-to-chip data transfers, thus yielding massively parallel, high-performance, energy-efficient processing systems.

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

Widely investigated in the 1990s, the PIM concept is hardly new. For example, the authors of IRAM[2] attempted to increase the bandwidth between CPU and memory by designing them on the same die. However, despite this and other attempts, inadequate technology prevented the widespread adoption of PIM. Additional obstacles included the effort required to design custom PIM architecture and memory interfaces and the need for new programming abstractions. Although recent advances in technology—for example, the emergence of 3D die stacking—have led to renewed interest in PIM, the recently proposed machines still suffer from the same fundamental problem: the need to transfer data between the processing and the memory spaces. So, while new PIM machines such as the Micron's Automata processor rely on cutting down the distance between the processing and memory units, they do not attempt to eliminate the data transfer itself. Nor can DRAM cells provide the ultimate solution to the data-transfer problem, as they are incapable of performing logic; systems with DRAM as a memory require a separate resource to perform computation.

The von Neumann architecture limitations can only be fully overcome if logical functions are computed directly using the memory cells, precluding the need to move data or instantiate additional CMOS blocks for processing. We present a new PIM approach that does just that. The memristive Memory Processing Unit (mMPU) relies on emerging memristive technologies such as resistive random access memory (RRAM[3]) to give computational capabilities directly to the memory cells. By directly tackling the data-movement problem, this approach differs fundamentally from all previous PIM techniques. The evolution of this scheme is illustrated in Figure 1. Furthermore, with its massive innate parallelism, the mMPU yields considerably higher performance and energy-efficiency than the current state-of-the-art approaches.

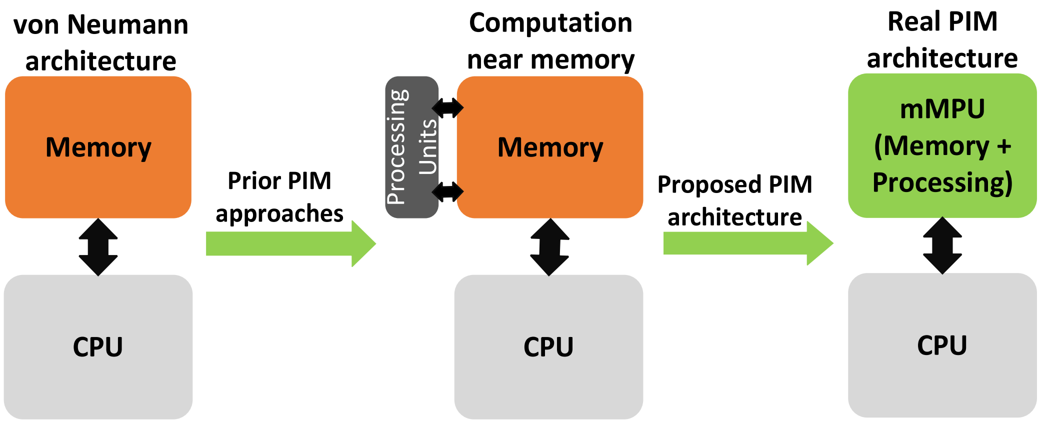

Figure 1. Architectural evolution from (left) von Neumann machines with separate computation and storage to (middle) near-data processing, and finally to (right) the proposed architecture in which logical operations are performed within the mMPU by the same cells that store the data—thus eliminating the von Neumann bottleneck.

MEMRISTOR-AIDED LOGIC (MAGIC)

We have recently proposed a stateful logic family called Memristor-Aided loGIC (MAGIC[4]). In MAGIC, only a single voltage is used to perform a NOR logic operation, and there are separate input and output memristors. Figure 2 shows a schematic of a MAGIC NOR gate operation performed over column vectors within a memristive memory. Since NOR is functionally complete, a MAGIC NOR operation is sufficient to execute any Boolean operation. For example, a recently fabricated memory crossbar demonstrated the feasibility of implementing an adder using MAGIC.[5] MAGIC enables true in-memory processing since data need not be read or sensed during the computation; the data is processed using only memory cells chosen by the memory controller, thus eliminating the data transfer. We propose that MAGIC NOR be employed as the basis for all data processing within the mMPU by dividing the desired function into a sequence of MAGIC NOR operations. These basic NOR operations will be executed one after the other using the memory cells as computational elements. Another virtue of MAGIC is its ability to perform logic operations in parallel on sets of data. Due to the structure of the crossbar array, applying the operating voltage V_G on any two selected columns and grounding a third column will result in NOR operations being performed on all selected rows (isolation of rows is possible if desired). This allows massive parallelism within the mMPU, which is independent of the data size. Due to the symmetry of memristive crossbar arrays, performing NOR operations on row vectors is feasible in a similar manner.

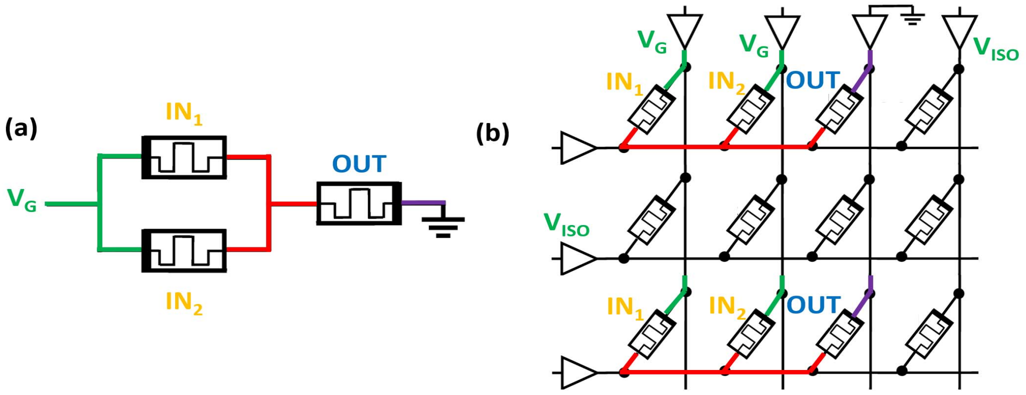

Figure 2. Schematic of (a) a MAGIC NOR gate and (b) a MAGIC NOR gate within a memristive memory array. IN_1 and IN_2 are the input memristors, and OUT is the output memristor. The inputs are the initial resistances (states) of the input memristors, and the output is the resistance (state) of the output memristor at the end of the computation. To perform the NOR operation, a single voltage V_G is applied to the bitline (BL) of the inputs, ground is applied to the BL of the output memristor, and V_{ISO} (the isolation voltage) is applied to unselected BLs and wordlines (WLs).

MEMRISTIVE MEMORY PROCESSING UNIT (MMPU)

The mMPU consists of a memristive memory and a CMOS controller. The structure is identical to a standard memristive memory; the only circuit modifications exist in the controller and the peripheral circuits (the row decoder and the voltage drivers). Hence, the advantages of a memristive crossbar array, such as density and nonvolatility, are maintained. Furthermore, because the mMPU can function as a standard memory, it is completely backward-compatible with von Neumann architecture and can operate either as a hybrid memory processing unit or as a standard memory. The structure of the proposed mMPU architecture is illustrated in Figure 3.

To support MAGIC NOR gates in the mMPU, only three modifications to the conventional RRAM architecture are required: (1) The on-chip memory controller must be made aware of MAGIC functionality to activate multiple MATs simultaneously and support the mapping of an application to MAGIC NOR gates, (2) the voltage drivers must include the voltages required to perform MAGIC, and (3) the row decoder of each bank should allow the simultaneous activation of multiple subarrays. For small datasets, the first two modifications might suffice, although only a single subarray (and its MATs) inside each bank could be activated in this case. These modifications allow us to perform MAGIC in multiple banks, subarrays, and MATs simultaneously in a SIMD scheme. Further modifications are also required to support PIM operations. For example, a new data mapping is necessary to maintain persistence and coherence, and the memory access protocol must be modified to handle the PIM operations.

Since parallel operations benefit most from the mMPU, they will mostly consist of simple SIMD operations, where a single operation requires only a few MAGIC cycles. One example is image convolution, which consists of many parallel additions and multiplications. To support in-memory applications, optimized algorithms can be manually or automatically developed. Manual optimization is ideal for relatively complex tasks or for delivering the last drop in performance.

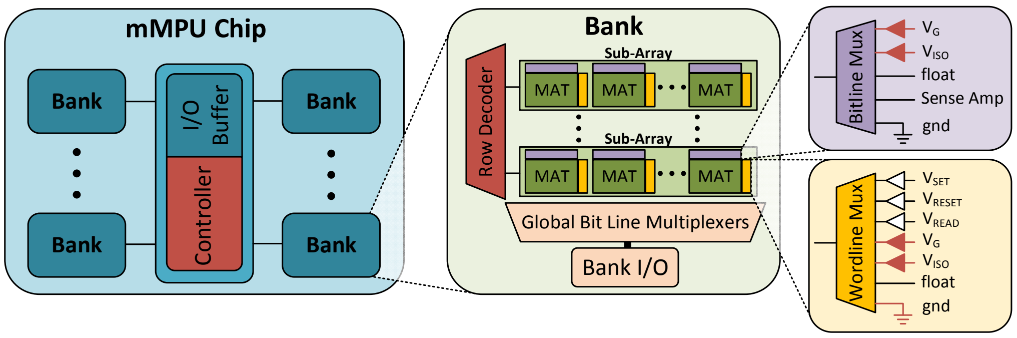

Figure 3. The mMPU chip architecture. The only modifications to conventional RRAM chip architecture (dark red) are in the controller, row decoders, and voltage drivers. A conventional RRAM chip typically has multiple banks. Banks in the same chip share the I/O. Each bank has several subarrays, each containing MATs. Each MAT has its private multiplexers (muxes) and sense amplifiers. Several adjacent BLs share one sense amplifier by a mux due to its large size. The sense amplifiers are also shared between two adjacent MATs by a mux.

For example, we have developed algorithms[6] that efficiently execute fixed-point (FiP) multiplication using MAGIC NOR gates and allow many such multiplications to be performed inside each MAT simultaneously, enabling the support of image-processing tasks.[7] For automatically generated algorithms, we use a framework[8] that allows optimal implementation of arbitrary logical functions within the memory by defining an optimal sequence of MAGIC NOR operations. Automatic mapping is great for simpler tasks. We believe that such automatic tools will ultimately serve as the basis for the mMPU operations, with manual mapping left for specific tasks requiring complex optimizations under different constraints, such as the number and size of available MATs, dataset size, and the complexity of the operation.

To make the rest of the system as oblivious as possible to the mMPU and avoid the burden of optimization tasks, a programmer will only have to determine the desired operation and the locations and sizes of the inputs and outputs (at least in the general case). We are exploring two approaches for activation of our PIM scheme. The first requires extending the ISA with additional instructions supported by the mMPU. In the second, we plan to use a library, which will include functions that support the different tasks the mMPU can perform (similar to CUDA in Nvidia GPUs). The parts of the program that could benefit from execution within the memory will be written using these functions, which initiate mMPU operations encapsulated as a write command to a reserved memory address; thus, they could be sent to the memory using the standard memory interface. Other portions will be executed in conventional von Neumann style (in the CPU with load and store operations for memory access). Figure 4 shows an example of a possible execution scheme of such functions.

THE CHALLENGES OF THE MMPU

There are several known challenges to overcome when building a system like the mMPU. Processing operands in the mMPU is constrained by the operands' physical addresses having to share the same WLs/BLs, since they serve as circuit connections among the inputs and outputs. If two operands that need to be processed are present in different WLs/BLs, they first must be aligned, meaning copied to addresses that share WLs/BLs with the other operands in the same MAT. Ideally, operands involved in MAGIC NOR operations should be initially mapped to the same WLs/BLs. This can be done using mMPU driver hints indicating that the data should be mapped to the same WLs/BLs. To the same end, if such mapping was not achieved, we also propose three data-transfer techniques for organizing the data after it was already stored, depending on the physical addresses of the input and output operands.[9] If the operands need to be transferred within the same MAT, one operand can be aligned to the other using MAGIC NOT operations. If the operands need to be transferred between different MATs in the same bank, a single operand can be read from one MAT using the sense amplifiers to the bank I/O and written to the other MAT. Finally, if the operands need to be transferred from one bank to another within the same chip, we read one operand to the bank's I/O through the sense amplifiers, transfer it to the bank I/O of the destination bank through the internal bus of the chip, and write it to the desired memory location juxtaposed to the other operand. Data organization using these techniques could increase the execution time by up to 1.5×. MAGIC also requires modifying the memory controller to support MAGIC-based operations and modifying the peripheral circuits to support additional voltage levels, consequently increasing the area of the memory.

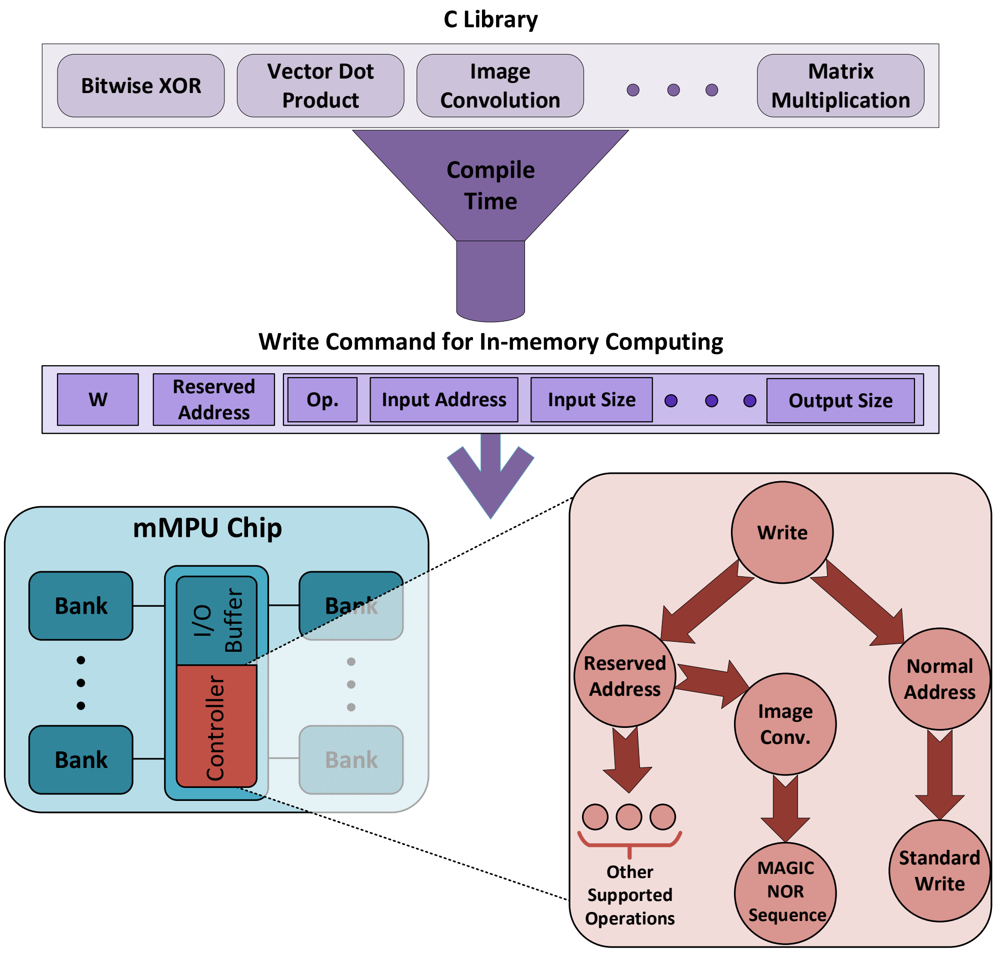

Figure 4. Example of functions supported in the mMPU. The function calls are converted by the compiler to an instruction for in-memory computing. The instruction contains a write operation to a reserved address mapped to a dedicated register within the mMPU controller. The payload data contains the relevant information for execution, such as the required operation, operands and result location, and size. The mMPU controller identifies such operations and performs the equivalent sequence of MAGIC NOR operations that results in the desired function.

Power and endurance limitations are crucial challenges. MAGIC incurs the overhead of writing multiple output memristors, which might result in high power consumption and memristor wear-out that decreases the memory lifetime. We analyzed the impact of power and endurance limitations in previous works.[7,9] Alleviating their impact is possible but might limit the performance or require technological improvements. For example, we achieved a worst-case (when all the MATs are simultaneously performing MAGIC gates in all the rows) lifetime of 0.8 years by using wear leveling.[7]

THE VIRTUES OF THE MMPU

Since the early 1990s, researchers have sought to scale the memory wall by bridging the gap between where data is stored and where it is processed. Although the term PIM is used to refer to processing the data closer to where it resides, true in-memory processing is possible only if the computation is carried out directly by the memory cells themselves. Hence, we argue that all previous PIM approaches are in fact near-memory approaches. Reducing the data-movement overhead is possible by processing data close to where it resides (such as in the periphery connected to the MAT rather than in off-chip computational units). This, however, will have no bearing on the fundamental cause of the problem: the very necessity for data movement between the memory and processing units.

By using the memory cells for computation, the mMPU overcomes the von Neumann bottleneck, eliminating the problem at its source. As such, the mMPU performs in-memory processing in practice and not in name alone. Furthermore, the added logic for supporting MAGIC in the mMPU has relatively low area overhead. By contrast, the conventional PIM approaches require considerable area overhead for the computational blocks added to the memory. MAGIC's natural parallelism can be put to full advantage within the mMPU, making it the ideal candidate for parallel data-intensive applications. Ultimately, any desired operation could be translated to a sequence of MAGIC NOR gates, which gives broad flexibility to the mMPU.

EXPLORING THE POTENTIAL OF THE MMPU

The massive parallelism and the elimination of data movement in the mMPU could be leveraged in many applications—for example, in digital image processing, which requires the same instruction on multiple data in parallel. Image manipulation requires data-intensive computations, often in real time, and the necessity for data movement only intensifies as image resolution becomes higher. Therefore, image-processing applications suffer from high energy consumption and a long processing time. They would thus benefit naturally from the mMPU, since many pixels are processed in place and in parallel, and the parallelism improves as the image dimensions increase.

As an initial step to demonstrate the potential of the mMPU, we compare its performance and energy-efficiency on three bitwise operations and two synthetic image-processing tasks (the Hadamard product and image convolution) with the recently published, state-of-the-art Pinatubo.[10] Pinatubo extended the capabilities of RRAM periphery to support bitwise XOR, AND, OR, and NOT operations. The inputs are read to the CMOS periphery, computed, and written back to the memory array. Thus, the computation in Pinatubo is very similar to that in the mMPU, except that Pinatubo performs the bitwise operations in the periphery while the mMPU performs them in the memory cells themselves. 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 element-wise multiplications between the images and 32×32 matrices used for pattern recognition and lossy compression algorithms such as JPEG.

We extended the MAGIC NOR-based FiP multiplication algorithm[6] to implement image convolution (and the Hadamard product)[7] in the mMPU by aligning the filters (or matrices) in the same WLs/BLs with the images and performing multiply-accumulate (or multiply) operations. We use similar algorithms in Pinatubo by replacing the MAGIC NOR gates with the optimal combination of XOR, AND, and OR gates; the multiplication is done using the sum of the partial products algorithm. We generate the partial products using AND gates, and each 1-bit full adder is implemented using two XOR gates, two AND gates, and a single OR gate. All the algorithms are verified using an in-house functional simulator that performs the logic gates cycle by cycle until the final result is obtained. The simulator starts the execution from a state in which the data is already stored properly in the mMPU and Pinatubo, since this overhead is equal in the two designs.

Each MAT is 512×512. To model the memristor, we use the VTEAM[11] model in which the device parameters fit the HfOx-based bipolar memristor.[12] The parameters for MAGIC NOR gates and read and write operations are extracted from SPICE simulations. The ratio of read, write, and MAGIC latencies is, respectively, 1:2.5:3.25, and the read latency is 8.9 ns.[10] For an apples-to-apples comparison, we exclude the overhead of the CMOS logic in the Pinatubo periphery. Note that the latency and energy of Pinatubo are higher in practice (when the CMOS logic overheads are included); our results are conservative in that they give Pinatubo an advantage. The RRAM capacity required to fit and compute the entire CIFAR-10 dataset is slightly lower than 1 Gbyte.

Figure 5 shows the speedup and normalized energy-efficiency of the mMPU on bitwise operations and image processing as compared to Pinatubo. Note that the latency and energy for performing MAGIC NOR in the mMPU is higher than that of the read and write operations in Pinatubo. However, the mMPU improves the performance by 65× and energy-efficiency by 4.6× in bitwise operations, on average, over Pinatubo. This improvement is primarily attributed to the higher parallelism enabled in the mMPU, where 512 MAGIC NOR operations can be performed simultaneously in each MAT. By contrast, in Pinatubo, since the bulky sense amplifiers are shared for every 32 BLs in every two adjacent MATs, the bandwidth for moving the data to and from the periphery is limited, consequently limiting overall system performance and energy-efficiency. Bitwise NOR operations improve the most, since they are the supported operation in the mMPU; supporting them in Pinatubo requires an OR followed by a NOT operation.

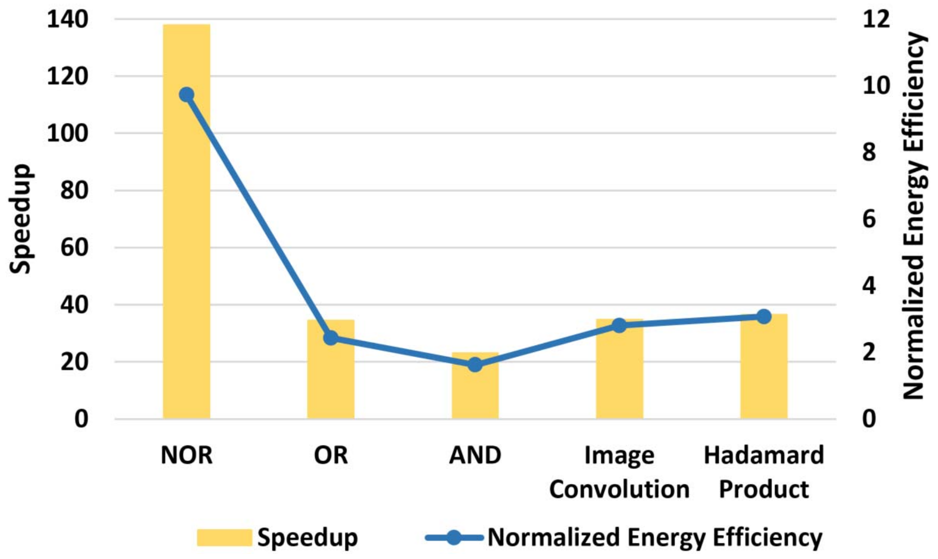

Figure 5. Speedup (left) and normalized energy-efficiency (right) of mMPU as compared to Pinatubo.

The improvements in the image-processing tasks are substantially lower than in the bitwise NOR operations because the mMPU can only perform NOR operations. In contrast to Pinatubo, which can perform the addition and multiplication operations more efficiently using fewer gates, the mMPU must serialize a sequence of NOR operations to perform the other logic functions. Performing image-processing tasks using only NOR gates is apparently not the most efficient method. However, the massive intrinsic parallelism and absence of data movement in the mMPU results in significantly higher performance and energy-efficiency for these tasks than state-of-the-art Pinatubo.

Both the mMPU and Pinatubo require activating multiple WLs and BLs. The mMPU, on the other hand, requires supporting additional voltage levels in the voltage drivers—support that is not necessary in Pinatubo. Therefore, the area overhead of the mMPU might be higher than that of Pinatubo. This area overhead could be reduced if necessary, but doing so might require limiting the space in the memory that supports PIM operations or using 1S1R arrays.

This comparison offers only a first glimpse into the potential of the mMPU. Both Pinatubo and the mMPU rely on the same technology and have similar performance overheads in the control, wires, peripheral circuits, and so on. Therefore, we could focus only on the differences between these architectures in our evaluation. Recently, we have shown that the mMPU can improve the performance in these tasks by 200× over APIM, which uses a combination of MAGIC gates and partial product generators in the periphery to speed up the multiplication.[7] However, a deeper exploration of performance and energy is still required, in addition to a comparison with fundamentally different architectures such as CPUs, GPUs, and FPGAs. Considering these overheads in such a whole system comparison will eventually be necessary.

CONCLUSION

In this article, we present the mMPU, a real PIM system that gives computational capabilities directly to the memory cells and thus differs fundamentally from all previous PIM techniques. With its elimination of data movement, massive innate parallelism, high performance, and energy-efficiency, the mMPU tackles the von Neumann bottleneck at its source. We look forward to unleashing the full potential of the mMPU by exploring additional applications, programming models, data mapping, and design of a new memory controller to support PIM.

With its elimination of data movement, massive innate parallelism, high performance, and energy-efficiency, the mMPU tackles the von Neumann bottleneck at its source.

ACKNOWLEDGMENTS

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), by the Viterbi Fellowship at the Technion Computer Engineering Center, by the EU ICT COST Action IC1401, and by the Israel Science Foundation grant no. 1514/17.

REFERENCES

1. A. Pedram et al., "Dark Memory and Accelerator-Rich System Optimization in the Dark Silicon Era," *IEEE Design & Test*, 2017.
2. D. Patterson et al., "Intelligent RAM (IRAM): the industrial setting, applications, and architectures," *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, 1997.
3. C. Xu et al., "Overcoming the challenges of crossbar resistive memory architectures," *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.
4. S. Kvatinsky et al., "MAGIC—Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2014.
5. H. Bae et al., "Functional Circuitry on Commercial Fabric via Textile-Compatible Nanoscale Film Coating Process for Fibertronics," *Nano Letters*, 2017.
6. A. Haj-Ali et al., "Efficient Algorithms for In-memory Fixed Point Multiplication Using MAGIC," *IEEE International Symposium on Circuits and Systems*, 2018.
7. A. Haj-Ali et al., "IMAGING: In-Memory AlGorithms for Image processiNG," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2018.
8. R. Ben-Hur et al., "SIMPLE MAGIC: Synthesis and In-memory MaPping of Logic Execution for Memristor-Aided loGIC," *International Conference on Computer-Aided Design*, 2017.

9. N. Talati et al., "Practical Challenges in Delivering the Promises of Real Processing-in-Memory Machines," *Design, Automation Test in Europe Conference Exhibition*, 2018.
10. S. Li et al., "Pinatubo: A Processing-in-memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories," *53nd ACM/EDAC/IEEE Design Automation Conference*, 2016.
11. S. Kvatinsky et al., "VTEAM: A General Model for Voltage-Controlled Memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2015.
12. H.Y. Lee et al., "Evidence and Solution of Over-RESET Problem for HfOX Based Resistive Memory with Sub-ns Switching Speed and High Endurance," *Electron Devices Meeting (IEDM), IEEE International*, 2010.

ABOUT THE AUTHORS

Ameer Haj-Ali is a graduate student at the Andrew and Erna Viterbi Faculty of Electrical Engineering at the Technion – Israel Institute of Technology. He has a bachelor's degree in computer engineering from the same university. Previously, he was a chip designer with Mellanox Technologies. His research focuses on novel computer architectures with emerging memory technologies and on design of energy-efficient and parallel architectures. Haj-Ali is an IEEE student member. Contact him at ameerh@berkeley.edu.

Rotem Ben-Hur is a graduate student working toward a PhD degree at the Andrew and Erna Viterbi Faculty of Electrical Engineering at the Technion – Israel Institute of Technology. She has a bachelor's degree in electrical engineering from the same university. Previously, she was an FPGA designer at Elbit Systems. Her research focuses on novel architectures for logic with emerging memory technologies. Contact her at rotembenhur@campus.technion.ac.il.

Nimrod Wald is a graduate student at the Technion – Israel Institute of Technology, working on novel circuits and architectures for logic with memristors. He has a bachelor's degree in electrical engineering and physics from the same university. Previously, he was a hardware design student at Qualcomm and worked in hardware architecture performance analysis. Contact him at nimrodw@campus.technion.ac.il.

Ronny Ronen is a senior researcher at the Andrew and Erna Viterbi Faculty of Electrical Engineering at the Technion – Israel Institute of Technology. He has a master's degree in computer science from the same university. Previously, he worked at Intel in various technical and managerial positions, including leading the Intel Collaborative Research Institute for Computational Intelligence and directing microarchitecture research at the Intel Development Center in Haifa. Ronen holds over 70 patents and has published over 20 papers. He is an IEEE Fellow. Contact him at ronny.ronen@technion.ac.il.

Shahar Kvatinsky is an assistant professor at the Andrew and Erna Viterbi Faculty of Electrical Engineering at the Technion – Israel Institute of Technology. He has a PhD in electrical engineering from the same university. Previously, he studied at the Hebrew University of Jerusalem and worked at Intel and Stanford University. Kvatinsky is an editor of *Microelectronics Journal* and has received numerous best-paper awards. His research focuses on circuits and architectures with emerging memory technologies and design of energy-efficient architectures. He is a senior IEEE member. Contact him at shahar@ee.technion.ac.il.