MatPIM: Accelerating Matrix Operations with Memristive Stateful Logic

Orian Leitersdorf, Ronny Ronen, and Shahar Kvatinsky
Viterbi Faculty of Electrical and Computer Engineering, Technion – Israel Institute of Technology, Haifa, Israel
orianl@campus.technion.ac.il, ronny.ronen@technion.ac.il, shahar@ee.technion.ac.il

***Abstract*—The emerging memristive Memory Processing Unit (mMPU) overcomes the memory wall through memristive devices that unite storage and logic for real processing-in-memory (PIM) systems. At the core of the mMPU is stateful logic, which is accelerated with memristive partitions to enable logic with massive inherent parallelism within crossbar arrays. This paper vastly accelerates the fundamental operations of matrix-vector multiplication and convolution in the mMPU, with either full-precision or binary elements. These proposed algorithms establish an efficient foundation for large-scale mMPU applications such as neural-networks, image processing, and numerical methods. We overcome the inherent asymmetry limitation in the previous in-memory full-precision matrix-vector multiplication solutions by utilizing techniques from block matrix multiplication and reduction. We present the first fast in-memory binary matrix-vector multiplication algorithm by utilizing memristive partitions with a tree-based popcount reduction (39× faster than previous work). For convolution, we present a novel in-memory input-parallel concept which we utilize for a full-precision algorithm that overcomes the asymmetry limitation in convolution, while also improving latency (2× faster than previous work), and the first fast binary algorithm (12× faster than previous work).**

***Index Terms*—Memristor, processing-in-memory, parallel algorithms, matrix multiplication, convolution.**

I. Introduction

Matrix operations construct the foundation for large-scale applications such as neural-networks, image processing, and numerical methods. Maximizing the efficiency of these operations has been thoroughly studied, such as Strassen's algorithm [1]. We vastly accelerate matrix operations in the emerging memristive Memory Processing Unit (mMPU) [2].

The mMPU is rapidly emerging as a technology that may overcome the *memory wall* [3] through memristive [4] crossbar arrays that enable real processing-in-memory (PIM) [5]–[7]. At its core is the memristor [4], a two-terminal resistive device whose resistance may be modified with an applied voltage. This enables binary storage through the resistance value (low resistance for logical one, high resistance for logical zero), with memristive crossbar arrays essentially storing binary matrices. The experimentally-demonstrated [8]–[10] digital *stateful logic* [6] technique observes that applying voltages on bitlines/wordlines of memristive crossbars induces parallel logic within the crossbar, performed using the same memristors responsible for storage [11]–[13]. For example, Figure 1(a) demonstrates that applying voltages on bitlines induces a logic gate (e.g., NOR) in each row of the crossbar; essentially, a bit-wise operation on two columns is computed and stored in a third column within a single cycle. Single-row algorithms [7], [14]–[21] utilize such parallelism for high-throughput vectored operation (e.g., vector addition) with latency independent of vector dimension: the arithmetic function (e.g., addition) is performed *serially* within a single row (e.g., a single NOR at a time), yet repeated along all rows simultaneously. Memristive partitions [7], [13]–[16], [22], [23] dynamically divide the crossbar using transistors to enable multiple concurrent column operations, see Figure 1(b), thereby enabling *parallel* single-row algorithms that perform multiple concurrent gates within each row, across all rows [14].

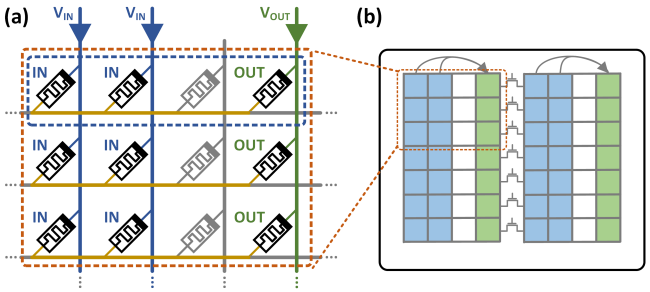

Fig. 1. (a) Memristive stateful logic with parallelism across rows: bitwise logic on two columns into a third column in a single cycle. (b) Memristive partitions enable multiple concurrent column operations.

We significantly advance in-memory matrix-vector multiplication [14], [19] and 2D convolution [18], [19], operations at the core of many applications [24]. Matrices are stored within a single crossbar array and the goal is to compute and store the outputs in the crossbar through stateful operations. As we operate within a crossbar array using stateful logic, data-transfer is significantly reduced and high throughput follows from the concurrent operation of multiple crossbars [25]. We consider both full-precision (e.g., for traditional neural-networks) and binary-precision (e.g., for binary neural networks [26], [27]) algorithms. Note that while memristive crossbar arrays can also be utilized in an analog fashion for fast matrix-vector multiplication [28], we focus on approaches based on digital stateful logic. For the full-precision algorithms, this is justified due to the high precision (e.g., 32-bit) that is required in many applications, yet cannot be matched by an analog domain. For the binary algorithms, we avoid the throughput-limited sense amplifiers by processing data *within* the crossbar; applications based on the proposed algorithms may process with high throughput by concatenating matrix operations within the same crossbar, thereby avoiding throughput-limited sense amplifiers.

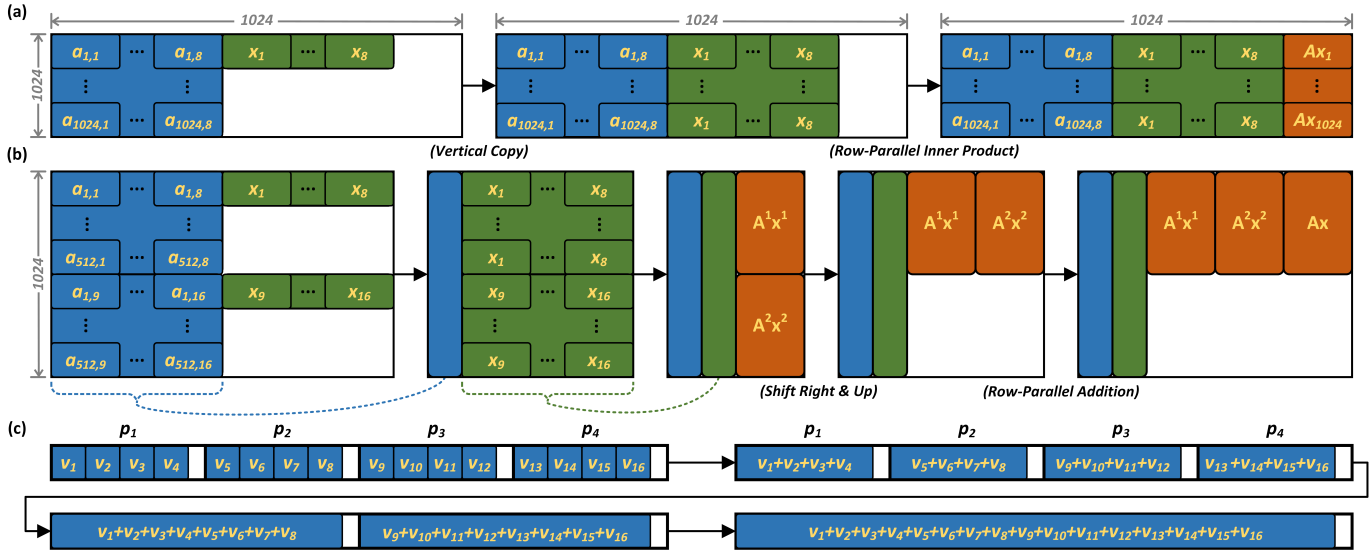

Fig. 2. (a) The previous approach to full-precision in-memory matrix-vector multiplication, supporting only matrices of drastically non-symmetric dimension, e.g. 1024×8. (b) The proposed approach to in-memory matrix-vector multiplication that splits the computation using block matrix multiplication, supporting matrices of nearly any dimension, e.g. 64×128. (c) The proposed single-row approach to binary popcount, accelerated with memristive partitions.

Previously proposed matrix-vector[1] multiplication concepts [14], [19] utilize parallelism across rows/columns in a straightforward manner. Yet, for full-precision matrix multiplication, a critical asymmetry enables only matrix multiplication of highly-peculiar dimensions: 1024×8 (1024 rows but only 8 columns in the matrix) for 32-bit integers and a crossbar sized 1024×1024 [14]. We propose a technique based on block matrix multiplication and reduction that overcomes this asymmetry, e.g., enabling dimension 64×128. We also present the first fast binary matrix-multiplication technique using a tree-based reduction with partitions, which is $39\times$ faster than the special case of 1-bit in previous approaches [14], [19].

Previous approaches for in-memory convolution either require non-trivial periphery [19] or suffer from the same asymmetry above [18], [19]. We propose a novel in-memory approach to convolution, enabling fast efficient operation for both full and binary precision. The proposed full-precision algorithm both overcomes the asymmetry limitation and even improves latency by $2\times$ over previous work [18], due to more efficient shifting. The binary algorithm improves latency by $12\times$ over the case of 1-bit integers in previous work [18].

II. In-Memory Matrix-Vector Multiplication

This section presents fast in-memory matrix-vector multiplication, outperforming the previous state-of-the-art [14], [19].

A. Full-Precision: Balanced Matrix-Vector Multiplication

We address the asymmetry in in-memory matrix multiplication, enabling fast full-precision matrix multiplication with flexible dimensions. Let matrix $\mathbf{A}$ be of dimensions $m \times n$ and vector $\mathbf{x}$ of dimension n be stored in the memory, the goal is to compute and store $\mathbf{Ax}$ within the crossbar, where each element in $\mathbf{A}, \mathbf{x}$, and $\mathbf{Ax}$ is an N-bit number.

[1]Matrix-matrix multiplication can be derived from repeated matrix-vector.

The basic previous concept is shown in Figure 2(a), storing each N-bit element horizontally. Vector $\mathbf{x}$ is duplicated to rows with stateful operations across rows, and then each row performs an inner-product (multiply and accumulate) between a row of $\mathbf{A}$ and $\mathbf{x}$. The inner product is performed in parallel across rows, using column operations [14], [19]. The overall latency is $O(m+nN\log N)$, due to (1) duplicating $\mathbf{x}$ in $O(m)$ cycles, and (2) multiplying two N-bit numbers within a single row in $O(N\log N)$ cycles, which is repeated n times [14]. While this elegantly utilizes parallelism, scalability is limited: as elements are stored horizontally ($1 \times N$ memristors), a 1024×1024 crossbar can support $m = 1024$ but only $n \leq 8$, for $N = 32$. This is problematic, as matrix-vector multiplication of dimension 1024×8 is highly uncommon.

Our proposed concept overcomes this asymmetry by splitting the matrix-vector multiplication into blocks. The concept follows from block matrix multiplication: consider splitting $\mathbf{A}$ to left and right halves, and $\mathbf{x}$ to top and bottom halves,

$$\mathbf{A} = \begin{pmatrix}\mathbf{A}^1 & \mathbf{A}^2\end{pmatrix}, \mathbf{x} = \begin{pmatrix}\mathbf{x}^1 \\ \mathbf{x}^2\end{pmatrix} \implies \mathbf{Ax} = \mathbf{A}^1\mathbf{x}^1 + \mathbf{A}^2\mathbf{x}^2. \quad (1)$$

We generalize (1) to α blocks. Our algorithm begins by computing each pair $\mathbf{A}^i\mathbf{x}^i$ across its own m rows (taking αm rows in total). Thus, the computation of all α pairs is performed in parallel. We continue by summing the vectors $(\mathbf{A}^1\mathbf{x}^1, \mathbf{A}^2\mathbf{x}^2, ..., \mathbf{A}^\alpha\mathbf{x}^\alpha)$ through a logarithmic technique inspired by reduction [25]. We start with α vectors stored vertically, shift half of them to the right and upwards, and add the vectors in parallel, reducing the task from α vectors to $\alpha/2$. Then, we continue recursively. Figure 2(b) demonstrates the case of $\alpha = 2$. Overall, we attain a latency of $O(\alpha m+(n/\alpha)N\log N+N\log\alpha+\alpha m)$, due to (1) the initial copying, (2) the parallel inner products, and (3) the reduction.

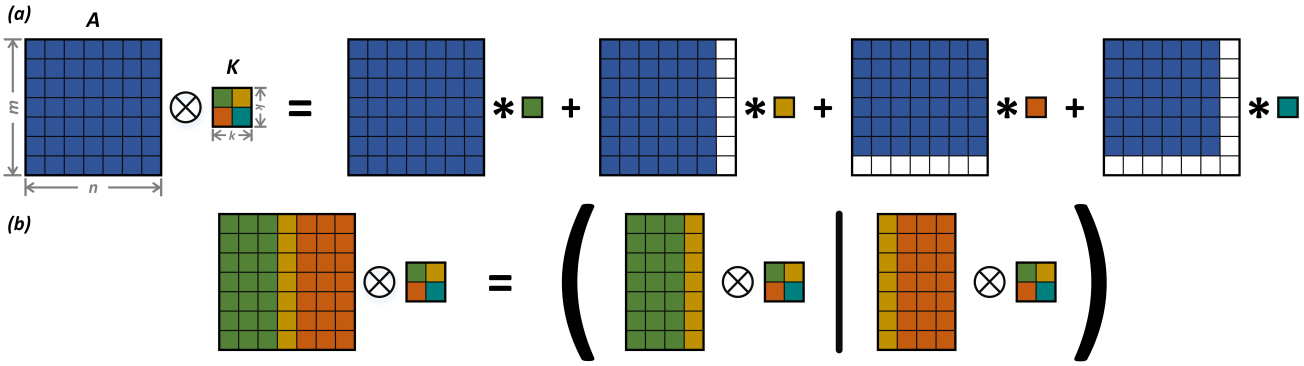

Fig. 3. (a) Input-parallel convolution concept which computes the sum of a shifted versions of $\mathbf{A}$ multiplied by single elements from $\mathbf{K}$. (b) Convolution property which enables splitting the input $\mathbf{A}$ into overlapping blocks $\mathbf{A}^1, ..., \mathbf{A}^\alpha$ and computing $\mathbf{A} \otimes \mathbf{K}$ by concatenating the results of $\mathbf{A}^1 \otimes \mathbf{K}, ..., \mathbf{A}^\alpha \otimes \mathbf{K}$.

B. Binary: Fast Matrix-Vector Multiplication

Binary matrix-vector multiplication is where the elements in $\mathbf{A}$ and $\mathbf{x}$ are binary (e.g., -1 or 1 [26], [27]), and the output is quantized (e.g., majority). The naive application of the previous approach [14], [19] performs the same algorithm for full-precision multiplication even for $N = 1$. The inner product in each row is performed by multiplying the 1-bit numbers (e.g., XNOR [26], [27]), adding each product to a counter, and then comparing the sum to $n/2$ (majority). In Section IV, we show this is highly inefficient.

As this summation is essentially a popcount operation, we propose two improvements that vastly improve performance. First, popcount can be implemented more efficiently with a tree rather than a counter, due to the representation size increasing throughout the summation. Second, using partitions, we accelerate the single-row popcount with a tree-based reduction. Assuming p partitions and a pop-count of n bits, we start with each partition storing n/p bits. All partitions start with a serial popcount operation on their n/p bits, in parallel across all partitions. We continue with a reduction tree amongst the partitions. For example, the first recursive step would connect every pair of partitions (set transistor between them to conducting) and add in parallel across $p/2$ partitions. Figure 2(c) illustrates the case of $n = 16$ and $p = 4$.

III. In-Memory Convolution

In-memory convolution is more complex than matrix-vector multiplication due to the required shifting [18], [19]. Thus, convolution in FloatPIM [19] requires barrel-shifters for each crossbar array, incurring undesirable area and control overhead [7]. Conversely, IMAGING [18] does not utilize barrel shifters, yet requires a complex output-parallel technique. We present a simple input-parallel technique that improves both of the previous methods, with both full-precision and binary algorithms. Consider the 2D convolution $\mathbf{A} \otimes \mathbf{K}$ of input $\mathbf{A}$ (dimension $m \times n$) and kernel $\mathbf{K}$ (dimension $k \times k$).

A. In-Memory Input-Parallel Convolution

IMAGING considered an output-parallel approach: for each element in the output matrix, they compute all of the products for that element in parallel; yet, this led to complex expensive movements between multiplications. Conversely, we propose an input-parallel approach: for each element a in the input $\mathbf{A}$, we consider the elements in $\mathbf{A} \otimes \mathbf{K}$ that a contributes to. We find that a is multiplied only once by each element in $\mathbf{K}$ for each of the neighbors of that element in $\mathbf{A} \otimes \mathbf{K}$. Thus, our approach constructs $\mathbf{A} \otimes \mathbf{K}$ from the sum of shifted versions of $\mathbf{A}$ multiplied by single elements from $\mathbf{K}$ (input-parallel), as shown in Figure 3(a). The in-memory implementation of this concept is described in Algorithm 1: we initialize a zero output matrix, and for each element x in $\mathbf{K}$, we add $\mathbf{A} * x$ to the current sum and then shift $\mathbf{A}$ to match the next element. Specifically, the multiplication is performed by duplicating x across all rows and then multiplying columns from $\mathbf{A}$ with x in parallel. The horizontal shifts are simulated as part of the access (similar to IMAGING), and the vertical shifts are performed with stateful gates alone (no barrel shifter) along rows. The barrel shifters from FloatPIM are not required since the shift is performed in parallel across the entire row (due to the input-parallel approach), and thus the latency of naive shift based on stateful logic is amortized. This shift parallelism is also the benefit of the proposed approach over IMAGING.

Algorithm 1 In-Memory Input-Parallel Convolution

Input: Input $\mathbf{A}$ (dim. $m \times n$), Kernel $\mathbf{K}$ (dim. $k \times k$)
Output: $\mathbf{A} \otimes \mathbf{K}$, with no padding (w.l.o.g.)

```
Out ← 0_{(m−⌈k/2⌉)×(n−⌈k/2⌉)}
for vert = 0 to k − 1 do
    for hori = 0 to k − 1 do
        for col = 0 to n − ⌈k/2⌉ − 1 do
            // Row-parallel addition/multiplication of columns
            Out[:, col] += A[:, col + hori] * K[vert][hori]
        end for
    end for
    Shift A vertically once (upwards).
end for
return Out
```

B. Balanced Full-Precision Convolution

Similar to matrix multiplication, the proposed concept in Section III-A for in-memory matrix convolution suffers from asymmetry in full-precision form, thus in this section a balanced implementation that overcomes that limitation is

TABLE I
MATRIX-VECTOR MULTIPLICATION LATENCY [CYCLES]

$\mathbf{A}$	$\mathbf{x}$	N	Baseline [14], [19]	Proposed
1024×8	8×1	32	4657	4657
512×16	16×1	32	Not Supported	5367
256×32	32×1	32	Not Supported	5822
128×64	64×1	32	Not Supported	6151
1024×384	384×1	1	14770	383

proposed. Figure 3(b) shows a simple property of convolutions that enables splitting $\mathbf{A}$ into multiple overlapping blocks, $\mathbf{A}^1, ..., \mathbf{A}^\alpha$, and computing the result $\mathbf{A} \otimes \mathbf{K}$ based on the concatenation of $\mathbf{A}^1 \otimes \mathbf{K}, ..., \mathbf{A}^\alpha \otimes \mathbf{K}$ ($\alpha = 2$ in the illustration). Therefore, while $\mathbf{A}$ may not fit in a crossbar when each element in $\mathbf{A}$ is stored horizontally ($1 \times N$) as the row size is the bottleneck, a block $\mathbf{A}^i$ can fit (as it contains less columns). Thus, we vertically concatenate the blocks $\mathbf{A}^1, ..., \mathbf{A}^\alpha$ within a single crossbar array, in a manner similar to the balanced matrix-vector multiplication. Note that the convolution occurs in parallel across all blocks as they are stacked vertically.

C. Fast Binary Convolution

We propose fast binary convolution based on the concept from Section III-A and the technique for splitting convolution introduced in Section III-B. Essentially, we compute each $\mathbf{A}_i \otimes \mathbf{K}$ in its own memristive partition. This case is simpler than the binary matrix-vector multiplication as each inner product fits within a single partition and thus no inter-partition communication is necessary. Furthermore, we replace the full-precision multiply and accumulate used in the full-precision convolution with the single-row single-partition popcount operation introduced in Section II-B.

IV. EVALUATION

We evaluate the proposed algorithms for full-precision and binary matrix-vector multiplication and convolution, demonstrating significant improvement as compared to the previous in-memory stateful-logic works in terms of flexibility with dimensions (full-precision) and latency (full-precision and binary). Results are verified via a custom cycle-accurate simulator[2] that models the logic operations in the array, with the proposed algorithms executing a sequence of stateful operations and the simulation environment verifying correctness. We choose a crossbar that supports the FELIX [13] suite of logic gates, and modify the results from previous works [18], [19] to assume the state-of-the-art arithmetic for addition and multiplication [14], providing a fair comparison of the algorithmic concepts rather than the specific stateful-logic technique. We consider a 1024×1024 crossbar array with 32 partitions within rows and columns.

A. Matrix Multiplication

Table I summarizes the results for in-memory matrix multiplication, for both the full-precision ($N = 32$)[3] and binary ($N = 1$) algorithms. The proposed full-precision algorithm supports far greater dimension flexibility than the previous works [14], [19]; for example, the previous works only supported 1024×8, while we support $512 \times 16, 256 \times 32, 128 \times 64$. For the binary algorithms, we present a substantial improvement of $39\times$ due to the combination of two techniques: optimized popcount and partition-based reduction tree.

[2]Available at https://github.com/oleitersdorf/MatPIM.

[3]$N = 32$ is chosen as an common example, we also support different N.

TABLE II
2D CONVOLUTION LATENCY [CYCLES]

$\mathbf{A}$	$\mathbf{K}$	N	Baseline [18]	Proposed
1024×4	3×3	32	28760	15352
1024×8	3×3	32	Not Supported	39897
512×16	3×3	32	Not Supported	49092
256×32	3×3	32	Not Supported	49592
128×64	3×3	32	Not Supported	49824
1024×8	5×5	32	Not Supported	81305
512×16	5×5	32	Not Supported	127728
256×32	5×5	32	Not Supported	128220
128×64	5×5	32	Not Supported	128436
1024×256	3×3	1	45312	3805

B. Matrix Convolution

Table II summarizes the results for in-memory 2D convolution, for both the full-precision ($N = 32$) and binary ($N = 1$) algorithms. The proposed full-precision algorithm both possess greater dimension flexibility than the previous work [18] and improves latency by $2\times$. In the binary case, we present a drastic improvement of $12\times$ due to the combination of the in-memory input-parallel concept and the distributed division of the convolution amongst partitions.

V. CONCLUSION

This paper vastly accelerates matrix-vector multiplication and convolution for the memristive Memory Processing Unit (mMPU). For matrix-vector multiplication, we extend the previous concept with block matrix multiplication to overcome an asymmetry challenge in full-precision algorithms, and with partitions to provide fast binary algorithms. For convolution, we propose an in-memory input-parallel approach, reducing the requirement for shift operations and not requiring an external barrel shifter; this provides significant improvement for full-precision algorithms and enables fast binary convolution. While alternative techniques such as analog memristive computing are widely explored for neural networks, the mMPU is also an attractive architecture for data-intensive applications due to the high-throughput *within* crossbar arrays and low area overhead; the expansion of the proposed algorithms to applications such as neural networks will be investigated in future work. Overall, we provide an efficient foundation for the operations at the core of many applications, thereby advancing the mMPU towards a new era of in-memory computing.

ACKNOWLEDGMENT

This work was supported in part by the European Research Council through the European Union's Horizon 2020 Research and Innovation Programe under Grant 757259, and in part by the Israel Science Foundation under Grant 1514/17.

References

[1] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[2] S. Kvatinsky, "Real processing-in-memory with memristive memory processing unit (mMPU)," in *IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160-052X, 2019, pp. 142–148.

[3] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, no. 2, pp. 39–50, 2017.

[4] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[5] R. Balasubramonian and B. Grot, "Near-data processing," *IEEE Micro*, vol. 36, no. 1, pp. 4–5, 2016.

[6] J. Reuben, R. Ben-Hur, N. Wald, N. Talati, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky, "Memristive logic: A framework for evaluation and comparison," in *27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[7] M. S. Q. Truong, E. Chen, D. Su, L. Shen, A. Glass, L. R. Carley, J. A. Bain, and S. Ghose, "RACER: Bit-pipelined processing using resistive memory," in *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, p. 100–116.

[8] N. Xu, T. Park, K. J. Yoon, and C. S. Hwang, "In-memory stateful logic computing using memristors: Gate, calculation, and application," *physica status solidi (RRL) – Rapid Research Letters*, vol. 15, no. 9, p. 2100208, 2021.

[9] B. Hoffer, V. Rana, S. Menzel, R. Waser, and S. Kvatinsky, "Experimental demonstration of memristor-aided logic (MAGIC) using valence change memory (VCM)," *IEEE Transactions on Electron Devices*, vol. 67, no. 8, pp. 3115–3122, 2020.

[10] Z. Sun, E. Ambrosi, A. Bricalli, and D. Ielmini, "Logic computing with stateful neural networks of resistive switches," *Advanced Materials*, vol. 30, no. 38, p. 1802554, 2018.

[11] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[12] 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, 2014.

[13] S. Gupta, M. Imani, and T. Rosing, "FELIX: Fast and energy-efficient logic in memory," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–7.

[14] O. Leitersdorf, R. Ronen, and S. Kvatinsky, "MultPIM: Fast stateful multiplication for processing-in-memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[15] O. Leitersdorf, B. Perach, R. Ronen, and S. Kvatinsky, "Efficient error-correcting-code mechanism for high-throughput memristive processing-in-memory," in *58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 199–204.

[16] O. Leitersdorf, R. Ronen, and S. Kvatinsky, "Making memristive processing-in-memory reliable," in *28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–6.

[17] A. Eliahu, R. Ben-Hur, R. Ronen, and S. Kvatinsky, "abstractPIM: Bridging the gap between processing-in-memory technology and instruction set architecture," in *IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, 2020, pp. 28–33.

[18] 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, 2018.

[19] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-memory acceleration of deep neural network training with high precision," in *ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 802–815.

[20] A. Haj-Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky, "Efficient algorithms for in-memory fixed point multiplication using MAGIC," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.

[21] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, N. Peled, 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*, vol. 39, no. 10, pp. 2434–2447, 2020.

[22] M. R. Alam, M. H. Najafi, and N. TaheriNejad, "Sorting in memristive memory," *ACM J. Emerg. Technol. Comput. Syst.*, 2022.

[23] Z. Lu, M. T. Arafin, and G. Qu, "RIME: A scalable and energy-efficient processing-in-memory architecture for floating-point operations," in *26th Asia and South Pacific Design Automation Conference*, 2021.

[24] J. Solomon, *Numerical algorithms: methods for computer vision, machine learning, and graphics*. CRC press, 2015.

[25] R. Ronen, A. Eliahu, O. Leitersdorf, N. Peled, K. Korgaonkar, A. Chattopadhyay, B. Perach, and S. Kvatinsky, "The bitlet model: A parameterized analytical model to compare PIM and CPU systems," *ACM J. Emerg. Technol. Comput. Syst.*, 2021.

[26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[27] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[28] Y. Xi, B. Gao, J. Tang, A. Chen, M.-F. Chang, X. S. Hu, J. V. D. Spiegel, H. Qian, and H. Wu, "In-memory learning with analog resistive switching memory: A review and perspective," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 14–42, 2021.