# SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM
## *Extended Abstract*

*Nastaran Hajinazar◇★     *Geraldo F. Oliveira◇     Sven Gregorio◇     João Dinis Ferreira◇     Nika Mansouri Ghiasi◇
Minesh Patel◇     Mohammed Alser◇     Saugata Ghose☉     Juan Gómez-Luna◇     Onur Mutlu◇

◇*ETH Zürich*     ★*Simon Fraser University*     ☉*University of Illinois at Urbana–Champaign*

## 1. Motivation & Limitations of State-of-the-Art

The increasing prevalence and growing size of data in modern applications has led to high costs for computation in traditional computer architectures. Moving large volumes of data between memory devices (e.g., DRAM) and the CPU across bandwidth-limited memory channels can consume more than 60% of the total energy in modern systems [10, 48]. To mitigate these costs, researchers have proposed a new computing paradigm, known as *processing-in-memory* (PIM). The key idea of PIM is to move computation closer to where the data resides, reducing (and in some cases eliminating) the need to move data between memory and the processor.

There are two main approaches to PIM [27, 49]: (1) processing-near-memory, where PIM logic is added to the same die as memory or to the logic layer of 3D-stacked memory [2–4, 8, 10–13, 17, 18, 20, 25, 26, 30–35, 39–41, 46, 51–55, 69, 70]; and (2) processing-using-memory, which makes use of the operational principles of the memory cells themselves to perform computation by enabling interactions between cells [1, 15, 16, 19, 22, 24, 43, 57–61, 64, 65, 68]. Since processing-using-memory operates directly in the memory cells, it benefits from the large internal bandwidth and parallelism available inside the memory arrays, which are significantly higher than those for processing-near-memory solutions.

A common approach for processing-using-memory architectures is to make use of bulk bitwise computation. Many widely-used data-intensive applications (e.g., databases, neural networks, graph analytics) heavily rely on a broad set of simple (e.g., AND, OR, XOR) and complex (e.g., equality check, multiplication, addition) bitwise operations. Ambit [56, 58], an in-DRAM processing-using-memory accelerator, was the first work to propose exploiting DRAM's analog operation to perform bulk bitwise AND, OR, and NOT logic operations. Inspired by Ambit, many prior works have explored DRAM (as well as NVM) designs that are capable of performing in-memory bitwise operations [5–7, 24, 44, 68]. However, a major shortcoming prevents these proposals from becoming widely applicable: they support only basic operations (e.g., Boolean operations, addition) and fall short on flexibly supporting new and more complex operations. Some prior works propose processing-using-DRAM designs that support more complex operations [16, 43]. However, such designs (1) require significant changes to the DRAM subarray, and (2) support only a limited and specific set of operations, lacking the flexibility to support new operations and cater to the wide variety of applications that can potentially benefit from in-memory computation. **Our goal** in this paper is to design a framework that aids the adoption of processing-using-DRAM by efficiently implementing complex operations and providing the flexibility to support new desired operations.

## 2. The Proposal

We propose SIMDRAM, an end-to-end processing-using-DRAM framework that provides the programming interface, the ISA and the hardware support for (1) efficiently computing *complex* operations, and (2) providing the ability to implement *arbitrary* operations as required, all in an in-DRAM massively-parallel SIMD substrate. At its core, we build the SIMDRAM framework around a DRAM substrate that enables two previously-proposed techniques: (1) vertical data layout in DRAM, and (2) majority-based logic for computation.

**Vertical Data Layout.** Supporting bit-shift operations is essential for implementing complex computations, such as addition or multiplication. Prior works show that employing a vertical layout [5, 9, 19, 22, 24, 62] for the data in DRAM, such that all bits of an operand are placed in a single DRAM column (i.e., in a single bitline), eliminates the need for adding extra logic in DRAM to implement shifting [16, 43]. Accordingly, SIMDRAM supports efficient bit-shift operations by storing operands in a vertical fashion in DRAM. This provides SIMDRAM with two key benefits. First, a bit-shift operation can be performed by simply copying a DRAM row into another row (using RowClone [57], LISA [14] or FIGARO [67]). For example, SIMDRAM can perform a left-shift-by-one operation by copying the data in DRAM row $j$ to DRAM row $j + 1$. (Note that while SIMDRAM supports bit shifting, we can optimize many applications to avoid the need for explicit shift operations, by simply changing the row indices of the SIMDRAM commands that read the shifted data). Second, SIMDRAM enables massive parallelism, wherein each DRAM column operates as a SIMD lane by placing the source and destination operands of an operation on top of each other in the same DRAM column.

**Majority-Based Computation.** Prior works use majority operations to implement basic logical operations [24, 43, 56, 58] (e.g., AND, OR) or addition [5, 6, 16, 23, 24, 43]. These basic operations are then used as basic building blocks to implement the target in-DRAM computation. SIMDRAM extends the use of the majority operation by directly using the logically complete set of majority (MAJ) and NOT operations to implement in-DRAM computation. Doing so enables SIMDRAM to achieve higher performance, throughput, and reduced energy consumption compared to using basic logical operations as building blocks for in-DRAM computation. We find that a computation typically requires fewer DRAM commands using

---

MAJ and NOT than using basic logical operations such as AND, OR, and NOT.

## 3. SIMDRAM Framework

SIMDRAM is the first end-to-end framework for processing-using-DRAM. SIMDRAM consists of three key steps to enable a desired operation in DRAM: (1) building an efficient MAJ/NOT-based representation of the desired operation, (2) mapping the operation input and output operands to DRAM rows and to the required DRAM commands that produce the desired operation, and (3) executing the operation. These three steps ensure efficient computation of a wide range of arbitrary and complex operation in DRAM. The first two steps give users the flexibility to efficiently implement and compute any desired operation in DRAM. The third step controls the execution flow of the in-DRAM computation, transparently from the user. We briefly describe these steps.

The goal of the first step is to use logic optimization to minimize the number of DRAM row activations, and therefore the compute latency required to perform a specific operation. Accordingly, for a desired computation, the first step is to derive its *optimized* MAJ/NOT-based implementation from its AND/OR/NOT-based implementation.

The second step translates the MAJ/NOT-based implementation into DRAM row activations. This step includes (1) mapping the operands to the designated rows in DRAM, and (2) defining the sequence of DRAM row activations that are required to perform the computation. SIMDRAM chooses the operand-to-row mapping and the sequence of DRAM row activations to minimize the number of DRAM row activations required for a specific operation.

The third step is to program the memory controller to issue the sequence of DRAM row activations to the appropriate rows in DRAM to perform the computation of the operation from start to end. To this end, SIMDRAM uses a *control unit* in the memory controller that transparently executes the sequence of DRAM row activations for each specific operation.

## 4. System Integration

To incorporate SIMDRAM into a real system, we address three integration challenges as part of our work: (1) managing memory with both vertical and horizontal layouts in a system, (2) exposing SIMDRAM functionality to programmers and compilers, and (3) dealing with potential RowHammer-based security exploits [21, 36, 37, 47, 50]. As part of the support for system integration, we introduce two components.

First, SIMDRAM adds a *transposition unit* in the memory controller that transforms the data layout from the conventional horizontal layout to vertical layout (and vice versa), as required, thereby allowing both layouts to coexist. Using the transposition unit, SIMDRAM provides the ability to store only the data that is required for in-DRAM computation in the vertical layout. As a result, SIMDRAM maintains the horizontal layout for the rest of the data and allows the CPU to read/write its operands from/to DRAM in a horizontal layout and at full bandwidth.

Second, SIMDRAM extends the ISA to enable the user/compiler to communicate with the SIMDRAM control unit. These extensions include instructions for (1) transposing data and (2) indicating specific operations to be issued by the control unit during in-DRAM execution.

## 5. Key Results and Contributions

The end-to-end support enables SIMDRAM as a holistic approach that facilitates the adoption of processing-using-DRAM. The SIMDRAM framework efficiently supports a wide range of operations of different types. In this work, we demonstrate the functionality of the SIMDRAM framework using an example set of operations including (1) *N*-input logic operations (e.g., AND/OR/XOR of more than 2 input bits); (2) relational operations (e.g., equality/inequality check, greater than, maximum, minimum); (3) arithmetic operations (e.g., addition, subtraction, multiplication, division); (4) predication (e.g., if-then-else); and (5) other complex operations such as bitcount and ReLU [29]. The SIMDRAM framework is not limited to these operations, and can enable processing-using-DRAM for other existing and future operations.

We compare the benefits of SIMDRAM to different state-of-the-art computing platforms (CPU, GPU, and the Ambit [58] in-DRAM computing mechanism). We comprehensively evaluate SIMDRAM's reliability, area overhead, throughput, and energy efficiency. Our evaluation shows that SIMDRAM provides up to $5.1\times$ higher throughput and $2.5\times$ higher energy efficiency compared to Ambit [58] for 16 different operations, while incurring less than 1% DRAM area overhead.

We leverage the SIMDRAM framework to accelerate seven application kernels from machine learning, databases, and image processing (VGG-13 [63], VGG-16 [63], LeNET [38], kNN [42], TPC-H [66], BitWeaving [45], Brightness [28]). SIMDRAM provides up to $2.5\times$ speedup for the kernels compared to Ambit [58]. Compared to a CPU and a high-end GPU, SIMDRAM is $257\times$ and $31\times$ more energy efficient, while providing $93\times$ and $6\times$ higher throughput, respectively. We also evaluate the reliability of SIMDRAM under different degrees of manufacturing process variation, and observe that it guarantees correct operation as the DRAM process technology node scales down to smaller sizes.

We make the following key contributions:
- To our knowledge, this is the first work to propose a framework to enable efficient computation of a flexible and wide range of operations in a massively parallel SIMD substrate built via processing-using-DRAM.
- SIMDRAM provides a three-step framework to develop efficient and reliable MAJ/NOT-based implementations of a wide range of operations. We design this framework, and add hardware and ISA support, to (1) address key system integration challenges and (2) allow programmers to employ new SIMDRAM operations without hardware changes.
- We provide a detailed reference implementation of SIMDRAM, including required changes to the user applications, ISA, and hardware.
- We evaluate the reliability of SIMDRAM under different degrees of process variation and observe that it guarantees correct operation as the DRAM technology scales to smaller node sizes.

# References

[1] S. Aga *et al.*, "Compute Caches," in *HPCA*, 2017.
[2] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[3] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[4] B. Akin *et al.*, "Data Reorganization in Memory Using 3D-Stacked DRAM," in *ISCA*, 2016.
[5] M. F. Ali *et al.*, "In-Memory Low-Cost Bit-Serial Addition Using Commodity DRAM Technology," in *TCAS-I*, 2019.
[6] S. Angizi and D. Fan, "GraphiDe: A Graph Processing Accelerator Leveraging in-DRAM-Computing," in *GLSVLSI*, 2019.
[7] S. Angizi *et al.*, "IMCE: Energy-Efficient Bitwise In-Memory Convolution Engine for Deep Neural Network," in *ASP-DAC*, 2018.
[8] O. O. Babarinsa and S. Idreos, "JAFAR: Near-Data Processing for Databases," in *SIGMOD*, 2015.
[9] K. E. Batcher, "Bit-Serial Parallel Processing Systems," in *IEEE Trans. Comput.*, 1982.
[10] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[11] A. Boroumand *et al.*, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *IEEE CAL*, 2017.
[12] A. Boroumand *et al.*, "CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators," in *ISCA*, 2019.
[13] D. S. Cali *et al.*, "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis," in *MICRO*, 2020.
[14] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[15] P. Chi *et al.*, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *ISCA*, 2016.
[16] Q. Deng *et al.*, "DrAcc: a DRAM based Accelerator for Accurate CNN Inference," in *DAC*, 2018.
[17] F. Devaux, "The True Processing in Memory Accelerator," in *HCS*, 2019.
[18] M. Drumond *et al.*, "The Mondrian Data Engine," in *ISCA*, 2017.
[19] C. Eckert *et al.*, "Neural Cache: Bit-serial in-Cache Acceleration of Deep Neural Networks," in *ISCA*, 2018.
[20] A. Farmahini-Farahani *et al.*, "NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules," in *HPCA*, 2015.
[21] P. Frigo *et al.*, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *IEEE S&P*, 2020.
[22] D. Fujiki *et al.*, "Duality Cache for Data Parallel Acceleration," in *ISCA*, 2019.
[23] P.-E. Gaillardon *et al.*, "The Programmable Logic-in-Memory (PLiM) Computer," in *DATE*, 2016.
[24] F. Gao *et al.*, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in *MICRO*, 2019.
[25] M. Gao and C. Kozyrakis, "HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing," in *HPCA*, 2016.
[26] M. Gao *et al.*, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017.
[27] S. Ghose *et al.*, "Processing-in-Memory: A Workload-Driven Perspective," *IBM JRD*, 2019.
[28] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. Addison-Wesley, 2002.
[29] I. Goodfellow *et al.*, *Deep Learning*. MIT Press, 2016.
[30] P. Gu *et al.*, "Leveraging 3D Technologies for Hardware Security: Opportunities and Challenges," in *GLSVLSI*, 2016.
[31] K. Hsieh *et al.*, "Transparent Offloading and Mapping (TOM) Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[32] K. Hsieh *et al.*, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.
[33] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification Rev. 2.0," http://www.hybridmemorycube.org/.
[34] D. Kim *et al.*, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *ISCA*, 2016.
[35] J. S. Kim *et al.*, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping using Processing-in-Memory Technologies," in *APBC*, 2018.
[36] J. S. Kim *et al.*, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA*, 2020.
[37] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[38] Y. LeCun *et al.*, "LeNet-5, Convolutional Neural Networks," http://yann.lecun.com/exdb/lenet, 2015.
[39] D. U. Lee *et al.*, "25.2 A 1.2V 8Gb 8-Channel 128GB/s High-Bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test Methods Using 29nm Process and TSV," in *ISSCC*, 2014.
[40] D. Lee *et al.*, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," *ACM TACO*, 2016.
[41] J. H. Lee *et al.*, "BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models," in *PACT*, 2015.
[42] Y. Lee, "Handwritten Digit Recognition Using k-Nearest-Neighbor, Radial-Basis Function, and Backpropagation Neural Networks," *Neural Computation*, 1991.
[43] S. Li *et al.*, "DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.
[44] S. Li *et al.*, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," in *DAC*, 2016.
[45] Y. Li and J. M. Patel, "BitWeaving: Fast Scans for Main Memory Data Processing," in *SIGMOD*, 2013.
[46] G. H. Loh *et al.*, "A Processing in Memory Taxonomy and a Case for Studying Fixed-Function PIM," in *WoNDP*, 2013.
[47] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.
[48] O. Mutlu *et al.*, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," *MICPRO*, 2019.
[49] O. Mutlu *et al.*, "A Modern Primer on Processing in Memory," *Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann*, 2021.
[50] O. Mutlu and J. S. Kim, "RowHammer: A Retrospective," in *TCAD*, 2019.
[51] L. Nai *et al.*, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," in *HPCA*, 2017.
[52] G. F. Oliveira *et al.*, "NIM: An HMC-Based Machine for Neuron Computation," in *ARC*, 2017.
[53] A. Pattnaik *et al.*, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.
[54] S. H. Pugsley *et al.*, "NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads," in *ISPASS*, 2014.
[55] P. C. Santos *et al.*, "Operand Size Reconfiguration for Big Data Processing in Memory," in *DATE*, 2017.
[56] V. Seshadri *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," in *IEEE CAL*, 2015.
[57] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient in-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[58] V. Seshadri *et al.*, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[59] V. Seshadri and O. Mutlu, "The Processing Using Memory Paradigm: In-DRAM Bulk Copy, Initialization, Bitwise AND and OR," arXiv:1610.09603 [cs.AR], 2016.
[60] V. Seshadri and O. Mutlu, "In-DRAM Bulk Bitwise Execution Engine," arXiv:1905.09822 [cs.AR], 2019.
[61] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[62] W. Shooman, "Parallel Computing with Vertical Data," in *EJCC*, 1960.
[63] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556 [cs.CV], 2014.
[64] L. Song *et al.*, "PipeLayer: A Pipelined ReRAM-based Accelerator for Deep Learning," in *HPCA*, 2017.
[65] L. Song *et al.*, "GraphR: Accelerating Graph Processing Using ReRAM," in *HPCA*, 2018.
[66] Transaction Processing Performance Council, "TPC-H," http://www.tpc.org/tpch/.
[67] Y. Wang *et al.*, "FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching," in *MICRO*, 2020.
[68] X. Xin *et al.*, "ELP2IM: Efficient and Low Power Bitwise Operation Processing in DRAM," in *HPCA*, 2020.
[69] D. Zhang *et al.*, "TOP-PIM: Throughput-Oriented Programmable Processing in Memory," in *HPDC*, 2014.
[70] Q. Zhu *et al.*, "Accelerating Sparse Matrix-Matrix Multiplication with 3D-Stacked Logic-in-Memory Hardware," in *HPEC*, 2013.