1. Motivation & Problem

The increasing prevalence and growing size of data in modern applications has led to high costs for computation in traditional processor-centric computing systems. Moving large volumes of data between memory devices (e.g., DRAM) and computing elements (e.g., CPUs, GPUs) across bandwidth-limited memory channels can consume more than 60% of the total energy in modern systems [1, 2]. To mitigate these costs, the processing-in-memory (PIM) [1, 3–9] paradigm moves 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 [4]: (1) processing-near-memory (PnM) [2, 10–77], where PIM logic is added to the same die as memory or to the logic layer of 3D-stacked memory [78–80]; and (2) processing-using-memory (PuM) [44, 81–103], which uses the operational principles of memory cells to perform computation (for example, by exploiting DRAM’s analog operation to perform bulk bitwise AND, OR, and NOT logic operations [83, 84, 87, 96, 97, 99]).

Many works from academia [2, 10–12, 15–23, 25, 31, 35–39, 48, 81–83, 85, 86, 90, 99, 104–112] and industry [34, 41–43, 50–54] have shown the benefits of PnM and PuM for a wide range of workloads from different domains. However, fully adopting PIM in commercial systems is still very challenging due to the lack of tools and system support for PIM architectures across the computer architecture stack [4], which includes: (i) workload characterization methodologies and benchmark suites targeting PIM architectures; (ii) frameworks that can facilitate the implementation of complex operations and algorithms using the underlying PIM primitives (e.g., simple PIM arithmetic operations [19], bulk bitwise Boolean in-DRAM operations [83, 84, 92]); (iii) compiler support and compiler optimizations targeting PIM architectures; (iv) operating system support for PIM-aware virtual memory, memory management, data allocation and mapping; and (v) efficient data coherence and consistency mechanisms.

Our goal in this work is to provide tools and system support for PnM and PuM architectures, aiming to ease the adoption of PIM in current and future systems. With this goal in mind, we address two limitations of prior works related to (i) identifying and characterizing workloads suitable for PnM offloading and (ii) enabling complex operations in PuM architectures. First, we develop a methodology, called DAMOV, that identifies sources of data movement bottlenecks in applications and associates such bottlenecks with PIM suitability. Second, we propose an end-to-end framework, called SIMDRAM, that enables the implementation of complex in-DRAM operations transparently to the programmer.

2. DAMOV: Identifying and Characterizing Data Movement Bottlenecks

DAMOV introduces the first rigorous methodology to characterize memory-related data movement bottlenecks in modern workloads and the first benchmark suite for data movement related studies. We develop a new methodology to correlate application characteristics with the primary sources of data movement bottlenecks and to determine the potential benefits of three example data movement mitigation mechanisms: (1) a deep cache hierarchy, (2) a hardware prefetcher, and (3) a general-purpose PnM architecture.

Our methodology has three steps. In Step 1 (Fig. 1), we use a hardware profiling tool [113] to identify memory-bound functions across applications. In Step 2 (2), we use an architecture-independent profiling tool [114, 115] to collect metrics that provide insights about the memory access behavior of each function. In Step 3 (3), we collect architecture-dependent metrics and analyze the performance and energy of each function on our three data movement mitigation mechanisms. By combining the three steps, we systematically classify the leading causes of data movement bottlenecks in an application or function into different bottleneck classes.

![Figure 1: Overview of our three-step workload characterization methodology.](image-url)
4. Applications with high temporal locality and low LLC MPKI are bottlenecked by L3 cache contention at high core counts. In such cases, the PnP system provides a cost-effective way to alleviate cache contention over increasing the L3 cache capacity.

5. Applications with high temporal locality, low LLC MPKI, and low arithmetic instruction (AI) are bottlenecked by the L1 cache capacity. The three candidate data movement mitigation mechanisms achieve similar performance and energy consumption for these applications.

6. Applications with high temporal locality, low LLC MPKI, and high AI are compute-bound. These applications benefit from a deep cache hierarchy and hardware prefetchers, but the PnP system degrades their performance.

We publicly release our 144 representative data movement bottlenecked functions from 74 applications as the first open-source benchmark suite for data movement, called DAMOV benchmark suite, along with the complete source code for our new characterization methodology and simulator [116]. For more information on our extensive data movement bottleneck characterization and on our DAMOV benchmark suite, along with our detailed contributions (including four use cases of our benchmark suite), please refer to our full paper [5, 117].

3. SIMDRAM: Enabling Complex Operations using DRAM

A common approach for PuM 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 [83, 84, 87, 96, 97, 99], an in-DRAM PuM accelerator, proposes exploiting DRAM’s analog operation to perform bulk bitwise majority-of-three (MAJ) computation, which can be manipulated to perform AND, OR, and NOT logic operations. Inspired by Ambit, many prior works have explored DRAM and emerging non-volatile memory (NVM) [118–154] designs that are capable of performing in-memory bitwise operations [89, 92, 100, 106, 155–157]. 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. Our goal 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.

To this end, we propose SIMDRAM, the first end-to-end framework for processing-using-DRAM. At its core, we build the SIMDRAM framework around a DRAM substrate that enables two previously-proposed techniques: (1) vertical data layout in DRAM to support bit-shift operations, and (2) majority-based logic. SIMDRAM consists of three key steps, illustrated in Figure 2, 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. We briefly describe these steps.
greater than, maximum, minimum); (3) arithmetic operations (e.g., addition, subtraction, multiplication, division); (4) prediction (e.g., if-then-else); and (5) other complex operations such as bitcount and ReLU [158].

We compare the benefits of SIMDRAM to different state-of-the-art computing platforms (CPU, GPU, and Ambit [83, 84, 87, 96, 97, 99]). We comprehensively evaluate SIMDRA M’s reliability, area overhead, throughput, and energy efficiency. We leverage the SIMDRA M framework to accelerate seven application kernels from machine learning, databases, and image processing (VGG-13 [159], VGG-16 [159], LeNET [160], kNN [161], TPC-H [162], BitWeaving [163], brightness [164]). Using a single DRAM bank, SIMDRA M provides (1) $2.0 \times$ the throughput and $2.6 \times$ the energy efficiency of Ambit [83], averaged across the 16 implemented operations; and (2) $2.5 \times$ the performance of Ambit, averaged across the seven application kernels. Compared to a CPU and a high-end GPU, SIMDRA M using 16 DRAM banks provides (1) $257 \times$ and $31 \times$ the energy efficiency, and $88 \times$ and $5.8 \times$ the throughput of the CPU and GPU, respectively, averaged across the 16 operations; and (2) $21 \times$ and $2.1 \times$ the performance of the CPU and GPU, respectively, averaged across the seven application kernels. SIMDRA M incurs no additional area overhead on top of Ambit, and a total area overhead of only 0.2% in a high-end CPU. We also evaluate the reliability of SIMDRA M 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.

For more information on our SIMDRA M framework and our extensive evaluation results (including a comparison to an alternative framework for processing-using-cache architectures), please refer to our full paper [165, 166].

4. Discussion

Few prior works tackle the challenge of providing end-to-end support for PIM. We describe these works and their limitations for in-DRAM computing.

Workload Characterization and Benchmark Suites for PIM. We highlight two prior works, [167] and PrIM [41–43] that also focus on characterizing workloads and providing benchmark suites for PIM architectures. In [167], the authors propose the first work that characterizes workloads for PIM. They analyze the benefits a PIM architecture similar to [71], where vector processing compute units are integrated into the DDRx memory modules, provides for five applications. Even though [167] has a similar goal to DAMOV, it understandably does not provide insights into modern data-intensive applications and PIM architectures as it dates from 2001. The authors of [41–43] propose PrIM, a benchmark suite of 16 workloads from different application domains (e.g., dense/sparse linear algebra, databases, data analytics, graph processing, neural networks, bioinformatics, image processing) tailored to fit the characteristics of a real PIM architecture (i.e., the UPMEM-based PIM system [34]). PrIM is open-source and publicly available at [168]. Unlike these prior works, DAMOV is applicable to and can be used to study other PIM architectures than processing-in-/near DRAM, including processing-in-/near cache [68, 93–95, 169–171], processing-in-/near storage [40, 172–181], and processing-in-/near emerging NVMs [81, 82, 90, 91, 100, 182, 183]. This is possible since DAMOV’s methodology and benchmarks are mainly concerned with broadly characterizing data movement bottlenecks in an application, independent of the underlying PIM architecture.

Frameworks for PIM. DualityCache [95] is an end-to-end framework for in-cache computing, which executes a fixed set of operations in a single-instruction multiple-thread (SIMT) manner. Employing DualityCache in DRAM is not straightforward due to the fundamental differences between in-cache computing and in-DRAM computing (e.g., the destructive behavior of DRAM operations and cost-sensitivity of DRAM chips). Two prior works, Hyper-A [184] and IMP [185], propose frameworks for in-emerging-NVM computing. Since Hyper-A and IMP target in-emerging-NVM substrates that utilize different computing paradigms (e.g., associative processing [186, 187]) or rely on particular structures of the NVM array (such as analog-to-digital/digital-to-analog converters) to perform computation, they are not applicable to an in-DRAM substrate that performs bulk bitwise operations. Olgun et al. propose the PiDRAM [188] framework, a flexible end-to-end and open-source FPGA-based framework that enables system integration studies and evaluation of in-DRAM computing techniques (e.g., in-DRAM copy and initialization [86, 98] and in-DRAM true random generation [108, 189, 190]) using real unmodified DRAM chips. PiDRAM is publicly available at [191] and can be used to prototype our SIMDRA M framework in a real system.

5. Conclusion & Future Work

This paper summarizes two of our recent efforts toward providing holistic system-level support for processing-in-memory (PIM) systems. We provide (i) a methodology to identify and characterize sources of data movement bottlenecks in a workload that can enable the programmer to assess whether a processing-near-memory (PnM) architecture can mitigate the identified data movement bottlenecks; (ii) the first benchmark suite (i.e., DAMOV) tailored for analyzing data movement bottlenecks and effects of near-data processing; and (iii) an end-to-end framework (i.e., SIMDRA M) that enables efficient and programmer-transparent computation of a wide range of arbitrary and complex operations by employing processing-using-memory (PuM) in DRAM. We believe that DAMOV can enable (1) simple and practical identification of PIM-suitable workloads and functions, (2) a research substrate (with our benchmark suite and simulator) for PIM-related architecture and system studies. SIMDRA M can facilitate the broader adoption of PuM architectures by more workloads and programmers. We hope that our work inspires future research on system-level solutions and tools that can aid the research, development, and implementation of PIM architectures.

Acknowledgments

We thank the SAFARI Research Group members for valuable feedback and the stimulating intellectual environment they provide. We acknowledge the generous gifts provided by our industrial partners, including ASML, Facebook, Google, Huawei, Intel, Microsoft, and VMware. We acknowledge
support from the Semiconductor Research Corporation and the ETH Future Computing Laboratory.

This invited extended abstract is a summary version of our two prior works DAMOV [5, 117] (published at IEEE Access 2021) and SIMDRAM [165, 166] (published at ASPLOS 2021). Presentations that describe DAMOV can be found at [192] (short talk video), [193] (long talk video), and [194] (tutorial on the DAMOV framework and benchmarks). A presentation that describes SIMDRAM can be found at [195].

References

[1] O. Multh et al., “Processing Data Where It Makes Sense: Enabling In-Memory Computation,” MIPRO, 2019.
[2] A. Boroumand et al., “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” in ASPLOS, 2018.
[3] S. Ghose et al., “Processing-in-Memory: A Workload-Driven Perspective,” IBM JRD, 2019.
[4] O. Mutlu et al., “A Modern Primer on Processing in Memory,” Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann, 2020.
[5] G. F. Oliveira et al., “DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks,” IEEE Access, 2021.
[6] S. Ghose et al., “The Processing-in-Memory Paradigm: Mechanisms to Enable Adoption,” in Beyond-CMOS Technologies for Next Generation Computer Design, 2019.
[7] O. Mutlu et al., “Enabling Practical Processing in and Near Memory for Data-Intensive Computing,” in DAC, 2019.
[8] O. Mutlu and L. Subramanian, “Research Problems and Opportunities in Memory Systems,” SUPERFRI, 2014.
[9] O. Mutlu, “Memory Scaling: A Systems Architecture Perspective,” in IJMS, 2013.
[10] J. Ahn et al., “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing,” in ISCA, 2015.
[11] L. Nai et al., “GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks,” in HPCA, 2017.
[12] A. Boroumand et al., “LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory,” IEEE CAL, 2017.
[13] D. Zhang et al., “TOP-PIM: Throughput-Oriented Programmable Processing in Memory,” in HPDC, 2014.
[14] M. Gao and C. Kozyrakis, “HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing,” in HPCA, 2016.
[15] J. S. Kim et al., “GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping using Processing-In-Memory Technologies,” in APBC, 2018.
[16] M. Drumond et al., “The Mondrian Data Engine,” in ISCA, 2017.
[17] P. C. Santos et al., “Operand Size Reconfiguration for Big Data Processing in Memory,” in DATE, 2017.
[18] G. F. Oliveira et al., “NIM: An HMC-Based Machine for Neuron Computation,” in ARC, 2017.
[19] J. Ahn et al., “PIM-Enabled Instruments: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture,” in ISCA, 2015.
[20] M. Gao et al., “TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory,” in ASPLOS, 2017.
[21] D. Kim et al., “Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory,” in ISCA, 2016.
[22] P. Gu et al., “Leveraging 3D Technologies for Hardware Security: Opportunities and Challenges,” in GLSVLSI, 2016.
[23] A. Boroumand et al. and CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators,” in ISCA, 2019.
[24] K. Hsieh et al., “Transparent Offloading and Mapping (TOM) Enabling Programmer-Transparent Near-Data Processing in GPU Systems,” in ISCA, 2016.
[25] D. S. Cali et al., “GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis,” in MICRO, 2020.
[26] S. H. Pagsley et al., “NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads,” in ISPASS, 2014.
[27] A. Farmahini-Farahani et al., “NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules,” in HPCA, 2015.
[28] G. H. Loh et al., “A Processing in Memory Taxonomy and a Case for Studying Fixed-Function PIM,” in WoNDF, 2013.
[29] A. Patnaik et al., “Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities,” in PACT, 2016.
[30] B. Akin et al., “Data Reorganization in Memory Using 3D-Stacked DRAM,” in ISCA, 2016.
[31] K. Hsieh et al., “Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation,” in ICCD, 2016.
[32] O. O. Babarinsa and S. Ideos, “JAFAR: Near-Data Processing for Databases,” in SIGMOD, 2015.
[33] J. H. Lee et al., “BBSync: Processing Near Memory for Machine Learning Workloads with Bounded Stalleness Consistency Models,” in PACT, 2015.
[34] F. Devaux, “The True Processing in Memory Accelerator,” in HCS, 2019.
[35] A. Boroumand et al., “Mitigating Edge Machine Learning Inference Bottlenecks: An Empirical Study on Accelerating Google Edge Models,” arXiv:2103.00768 [cs.AR], 2021.
[36] A. Boroumand et al., “Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks,” in PACT, 2021.
[37] A. Boroumand et al., “Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design,” in ICDE, 2022.
[38] A. Boroumand et al., “Polynesia: Enabling Effective Hybrid Transactional/Analytical Database Systems in Advanced Hardware/Software Co-Design,” arXiv:2103.00798 [cs.AR], 2021.
[39] A. Boroumand, “Practical Mechanisms for Reducing Processor-Memory Data Movement in Modern Workloads,” Ph.D. dissertation, Carnegie Mellon University, 2022.
[40] N. M. Ghiasi et al., “GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis,” in ASPLOS, 2022.
[41] J. Gómez-Luna et al., “Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware,” in CUP, 2021.
[42] J. Gómez-Luna et al., “Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture,” arXiv:2105.03814 [cs.AR], 2021.
[43] J. Gómez-Luna et al., “Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System,” IEEE Access, 2022.
[44] M. Besta et al., “SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems,” in MICRO, 2021.
[45] C. Giannoula et al., “SyncCron: Efficient Synchronization Support for Near-Data Processing Architectures,” in HPCA, 2021.
[46] I. Fernandez et al., “NATSAT: A Near-Data Processing Accelerator for Time Series Analysis,” in ICCD, 2020.
[47] G. Singh et al., “NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling,” in FPL, 2020.
[48] S. H. S. Rezai et al., “NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories,” CAL, 2020.
[49] G. Singh et al., “NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning,” in DAC, 2019.
[50] S. Lee et al., “A 1ynm 1.25V 8Gb, 16Gb/s/Pin GDDR6-Based Accelerator-in-Memory Supporting ITFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications,” in ISSCC, 2022.
[51] Y.-C. Kwon et al., “25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2 TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications,” in ISSCC, 2021.
[52] S. Lee et al., “Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product,” in ISCA, 2021.
[53] J. Ke et al., “Near-Memory Proof of Concept: Accelerating Personalized Recommendation with AxDIMM,” IEEE Micro, 2021.
[54] D. Niu et al., “18QPSWx64Mbit/mm2 3D Logic-to-DRAM Hybrid Bonding with Process-Near-Memory Engine for Recommendation System,” in ISCC, 2022.
[55] C. Giannoula et al., “SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Systems,” in SIGMETRICS, 2022.
[56] H. Shin et al., “McDRAM: Low Latency and Energy-Efficient Matrix Computations in DRAM,” IEEE TCADICS, 2018.
[57] S. Cho et al., “McDRAM v2: In-Dynamic Random Access Memory Systolic Array Accelerator to Address the Large Model Problem in Deep Neural Networks on the Edge,” IEEE Access, 2020.
[58] Q. Zhu et al., “Accelerating Sparse Matrix-Matrix Multiplication with 3D-Stacked Logic-in-Memory Hardware,” in HPEC, 2013.
5
