Lightning Talk: Memory-Centric Computing

Onur Mutlu

ETH Zürich

Modern computing systems are processor-centric. Data processing (i.e., computation) happens only in the processor (e.g., a CPU, GPU, FPGA, ASIC). As such, data needs to be moved from where it is generated/captured (e.g., sensors) and stored (e.g., storage and memory devices) to the processor before it can be processed. The processor-centric design paradigm greatly limits the performance & energy-efficiency, as well as scalability & sustainability, of modern computing systems. Many studies show that even the most powerful processors and accelerators waste a large fraction (e.g., >60%) of their time simply waiting for data and energy on moving data between storage/memory units to the processor. This is so even though most of the hardware real estate of such systems is dedicated to data storage and communication (e.g., many levels of caches, DRAM chips, storage systems, and interconnects).

Memory-centric computing aims to enable computation capability in and near all places where data is generated and stored. As such, it can greatly reduce the large negative performance and energy impact of data access and data movement, by fundamentally avoiding data movement and reducing data access latency & energy. Many recent studies show that memory-centric computing can greatly improve system performance and energy efficiency. Major industrial vendors and startup companies have also recently introduced memory chips that have sophisticated computation capabilities.

This talk describes promising ongoing research and development efforts in memory-centric computing. We classify such efforts into two major fundamental categories: 1) processing using memory, which exploits analog operational properties of memory structures to perform massively-parallel operations in memory, and 2) processing near memory, which integrates processing capability in memory controllers, the logic layer of 3D-stacked memory technologies, or memory chips to enable high-bandwidth and low-latency memory access to near-memory logic. We show both types of architectures (and their combination) can enable orders of magnitude improvements in performance and energy consumption of many important workloads, such as graph analytics, databases, machine learning, video processing, climate modeling, genome analysis. We discuss adoption challenges for the memory-centric computing paradigm and conclude with some research & development opportunities.

1. Memory-Centric Computing

Memory-centric computing (also called *processing in memory, PIM*) is a processing paradigm where data processing is performed near and in devices where data is generated (e.g., sensors) or stored (e.g., memory and storage devices) [1]. This paradigm enables computing to be more efficient by offering an alternative to modern systems, which overwhelmingly use the processor-centric paradigm where data processing is performed *only* in the processor (which can be a CPU, GPU, FPGA, ASIC in modern systems). Memory-centric computing has several advantages over processor-centric computing. First, it fundamentally reduces the data movement bottleneck [2], which plagues processor-centric systems that have to move data to the processor before processing it. Second, it enables low-latency and low-energy access to data by reducing the distance between processing units and data storage & sensing units. Third, it can exploit large amounts of parallelism present in modern memory, storage, and sensor arrays to perform massively parallel (bit-level) computation [3]. As such, memory-centric computing promises to improve both performance and energy-efficiency at the same time.

Memory-centric computing systems can be categorized into two types [1,4], based on the fundamental way in which computation is performed: 1) processing using memory (PuM), and 2) processing near memory (PnM). We briefly describe these next and give examples from recent works. These two approaches can be combined to obtain the best of both approaches.

1.1. Processing using Memory (PuM)

A memory device has analog operational properties that enable it to perform (varying types and amounts of) computation. PuM exploits these properties to perform computation *using* the memory device (including memory cells, bitlines, wordlines, sensing structures, and peripheral circuitry). As such, the PuM approach can enable computation *without* adding logic to perform computation into a memory device, which makes it fundamentally different from modern processor-centric systems as well as PnM systems that add such logic near or in memory devices. PuM approach can be made more powerful by designing the memory device to increase its capability to perform analog computation.

PuM approaches have been demonstrated in DRAM (e.g., [3, 5–10]), NVM (e.g., [11–13]), NAND flash (e.g., [14, 15]) and SRAM (e.g., [16,17]) devices. For example, recent works [5, 18, 19] show that data copy and initialization can be performed inside a DRAM chip by exploiting internal connectivity in the DRAM chip, even in existing real DRAM chips that do *not* explicitly support these operations. Latency of a 4KB data copy can be improved by more than 11X and energy by 77X compared to a state-of-the-art processor-centric solution. Recent works [6, 8, 9, 18] also show that bulk bitwise operations (Majority, AND, OR, NOT) and true random number generation [20, 21] can be performed in commodity DRAM chips with small modifications or by violating timing parameters. Frameworks and compilers have been introduced to implement any type of operation using such bulk bitwise computation capability, with little effort required from the programmer [9]. Real NAND flash memory chips can also perform bulk bitwise operations (AND, OR NOT, XOR) using inherent operational properties of NAND flash cells and strings as well as peripheral circuitry [14, 15]. Some emerging memory technologies are capable of performing matrix-vector multiplication operations in the analog domain due to their crossbar array structure [12, 13], and various test chips have been designed to demonstrate this as proof-of-concept prototypes.

1.2. Processing near Memory (PnM)

PnM adds processing logic (similar to modern processors and accelerators) close to or inside a memory device such that the distance between the processing logic and memory device is much smaller than in processor-centric systems. Such logic can be added to memory controllers, the logic layer of 3D-stacked memories, around peripheral circuitry in a memory chip, near memory subarrays in a memory chip, etc. The closer the logic is to the data storage parts of memory, the lower the amount of data movement. As such, PnM is not fundamentally different from modern systems where processing logic and memory structures are distinct, yet PnM greatly reduces the distance between them and in more aggressive implementations places logic and memory together in a tightly-integrated manner.

Many recent works (e.g., [22–25]) have shown the benefits of the PnM approach, by especially focusing on how various different types of applications can be accelerated using such an approach with varying levels of modifications to applications. For example, rewriting the entire application and changing the programming model to execute graph analytics near memory can greatly improve both performance and efficiency, by more than an order of magnitude [22,23]. Less intrusive PnM approaches offload specific functions or instructions to near-memory logic [2,26–30], with lower but still large performance and energy benefits.

1.3. Real PIM Systems

Recently, several real DRAM-based PnM systems were introduced as commercial systems or promising prototypes. The UPMEM company, for example, introduced a system where DRAM chips contain a general-purpose multithreaded processor next to each DRAM bank [31]. Several studies of the UPMEM system (e.g., [32–36]) demonstrate the benefits and tradeoffs of this first commercial memory-centric system on various workloads and present benchmark suites and libraries for it. These studies show large performance and energy benefits when the workload is carefully designed to fit the constraints present in the PnM system, which is limited in terms of the computation power within

the near-memory processors and the communication capability present between such processors and the host CPU. These studies also indicate how future general-purpose PnM systems can be improved to be much more powerful and effective.

Several major vendors developed specialized PnM systems targeted toward machine learning applications and recommendation systems. For example, Samsung introduced FIMDRAM [37], which is intended to accelerate floating-point based matrix operations (with native support for FP multiply and accumulates) in a DRAM chip. FIMDRAM incorporates processing units next to DRAM banks. To accelerate similar applications, SK-Hynix introduced the AiM-DRAM system [38], which also incorporates near-bank computation units. Two other PnM systems were introduced by Alibaba [39] and Samsung/Meta [40] to accelerate recommendation systems. The former modifies a DRAM chip to perform specialized computation tailored towards recommendation inference. The latter includes a processing buffer chip in a DRAM module that performs similarly specialized computation on data coming from many DRAM chips surrounding it.

Systems where computation can be offloaded to FPGAs that are equipped with high-bandwidth memory (HBM) also exist [41]. These systems can provide significant performance and energy benefits on various applications (e.g., [41, 42]), including weather modeling and genome analysis.

1.4. Adoption Challenges

Even though real PIM systems exist, memory-centric computing is far from being adopted in a widespread manner. To reach that level and thus realize the full potential and benefits of memory-centric computing, a number of challenges likely need to be solved. Many of these adoption challenges are common to PnM and PuM systems. We briefly cover some of these challenges, as they constitute important areas to investigate both in research and development of memory-centric computing systems.

First, it is important to accurately and comprehensively demonstrate which workloads and algorithms can benefit from memory-centric computing and by how much. This can enable a larger momentum for adopting PIM systems. It is especially critical to maximize benefits on important workloads. Second, widespread adoption of PIM requires such systems to be easy to program [43], which in turn requires support for seamless programming and compilation. Third, system and security support is needed to enable high efficiency and ease of use/programming. This wide topic includes support for data coherence between PIM and other computation units (e.g., CPUs) [44, 45], synchronization [46], virtual memory [29], multiprogramming and sharing of PIM computation units, isolation between processes executing on PIM units, and communication interfaces to access PIM units. Fourth, it is important to design runtimes and compilation systems to decide what code should be executed in PIM units [47], how data should be mapped to facilitate PIM execution, and how access control and data sharing should be managed. Fifth, there is continual need for infrastructures and benchmarks (e.g., [19, 33–35, 47–49]) that help both hardware designers and software designers to accurately assess benefits, tradeoffs, and feasibility of different types of memory-centric computing systems. Finally, it is important to lower cost and demonstrate TCO benefits.

PuM systems have specific additional challenges due to their analog nature of computation. These include how to tolerate circuit variation and noise, how to ensure reliable operation, and how to enable computation on large memory arrays for scalable performance. In addition, some PUM systems implemented using memories that have endurance problems exacerbate lifetime and endurance problems. Due to such challenges, we believe PuM systems are harder to adopt in the short term even though their benefits can be fundamentally higher than PnM systems.

1.5. Future Opportunities and Outlook

Memory-centric computing can enable balanced and efficient system designs where computation and memory access are fundamentally balanced and the processor-memory dichotomy is eliminated. These systems can provide greatly higher performance and efficiency than existing processor-centric systems. They can also enable potentially new applications and computing platforms. However, as with any new paradigm, memory-centric computing systems pose significant adoption challenges. We be-

lieve the processor-centric mindset that is ingrained in essentially every decision made in modern computing systems is likely the largest adoption challenge memory-centric systems face. We conclude that the future of memory-centric computing is very bright, but there is a lot more exciting research and development to do.

References

[1] O. Mutlu *et al.*, "A Modern Primer on Processing in Memory," *Emerging Computing: From Devices to Systems*, 2021, https://arxiv.org/pdf/2012.03112.pdf.
[2] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[3] V. Seshadri and O. Mutlu, "In-DRAM Bulk Bitwise Execution Engine," *arXiv*, 2020.
[4] O. Mutlu *et al.*, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," *MicPro*, 2019.
[5] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[6] V. Seshadri *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.
[7] V. Seshadri *et al.*, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," *arXiv*, 2016.
[8] V. Seshadri *et al.*, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[9] N. Hajinazar *et al.*, "SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM," in *ASPLOS*, 2021.
[10] J. D. Ferreira *et al.*, "pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables," in *MICRO*, 2022.
[11] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016.
[12] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[13] P. Chi *et al.*, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation In ReRAM-Based Main Memory," in *ISCA*, 2016.
[14] J. Park *et al.*, "Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory," in *MICRO*, 2022.
[15] C. Gao *et al.*, "ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory based SSDs," in *MICRO*, 2021.
[16] M. Kang *et al.*, "An Energy-Efficient VLSI Architecture for Pattern Recognition via Deep Embedding of Computation in SRAM," in *ICASSP*, 2014.
[17] S. Aga *et al.*, "Compute Caches," in *HPCA*, 2017.
[18] F. Gao *et al.*, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in *MICRO*, 2019.
[19] A. Olgun *et al.*, "PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM"," *TACO*, 2023.
[20] J. Kim *et al.*, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in *HPCA*, 2019.
[21] A. Olgun *et al.*, "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAMs," in *ISCA*, 2021.
[22] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[23] M. Besta *et al.*, "SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems," in *MICRO*, 2021.
[24] I. Fernandez *et al.*, "NATSA: A Near-Data Processing Accelerator for Time Series Analysis," in *ICCD*, 2020.
[25] N. M. Ghiasi *et al.*, "GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis," in *ASPLOS*, 2022.
[26] K. Hsieh *et al.*, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[27] A. Boroumand *et al.*, "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks," in *PACT*, 2021.
[28] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[29] K. Hsieh *et al.*, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.
[30] A. Boroumand *et al.*, "Polynesia: Enabling Effective Hybrid Transactional Analytical Databases with Specialized Hardware Software Co-Design," in *ICDE*, 2022.
[31] F. Devaux, "The True Processing In Memory Accelerator," in *Hot Chips*, 2019.
[32] J. Gómez-Luna *et al.*, "Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-In-Memory Hardware," in *IGSC*, 2021.
[33] J. Gómez-Luna *et al.*, "Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System," *IEEE Access*, 2022.
[34] C. Giannoula *et al.*, "Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-in-Memory Architectures," in *SIGMETRICS*, 2022.
[35] J. Gomez-Luna *et al.*, "Evaluating Machine Learning Workloads on Memory-Centric Computing Systems," in *ISPASS*, 2023.
[36] S. Diab *et al.*, "A Framework for High-throughput Sequence Alignment using Real Processing-in-Memory Systems"," *Bioinformatics*, 2023.
[37] 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.
[38] S. Lee *et al.*, "A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications," in *ISSCC*, 2022.
[39] D. Niu *et al.*, "184QPS/W 64Mb/mm2 3D Logic-to-DRAM Hybrid Bonding with Process-Near-Memory Engine for Recommendation System," in *ISSCC*, 2022.
[40] L. Ke *et al.*, "Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM," *IEEE Micro*, 2021.
[41] G. Singh *et al.*, "FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications," *IEEE Micro*, 2021.
[42] G. Singh *et al.*, "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling," in *FPL*, 2020.
[43] S. Ghose *et al.*, "Processing-in-Memory: A Workload-Driven Perspective," *IBM JRD*, 2019.
[44] A. Boroumand *et al.*, "CoNDA: Efficient Cache Coherence Support for near-Data Accelerators," in *ISCA*, 2019.
[45] A. Boroumand *et al.*, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2016.
[46] C. Giannoula *et al.*, "SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures," in *HPCA*, 2021.
[47] G. F. Oliveira *et al.*, "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks," *IEEE Access*, 2021.
[48] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.
[49] G. Singh *et al.*, "NAPEL: Near-memory Computing Application Performance Prediction via Ensemble Learning," in *DAC*, 2019.