Memories - Materials, Devices, Circuits and Systems 4 (2023) 100022

Contents lists available at ScienceDirect

Memories - Materials, Devices, Circuits and Systems

journal homepage: www.elsevier.com/locate/memori

A survey on processing-in-memory techniques: Advances and challenges☆

Kazi Asifuzzaman *, Narasinga Rao Miniskar, Aaron R. Young, Frank Liu, Jeffrey S. Vetter

Oak Ridge National Laboratory, USA

ARTICLE INFO

Keywords:
Processing-in-memory
Near memory computing
Novel and emerging memory technologies

ABSTRACT

Processing-in-memory (PIM) techniques have gained much attention from computer architecture researchers, and significant research effort has been invested in exploring and developing such techniques. Increasing the research activity dedicated to improving PIM techniques will hopefully help deliver PIM's promise to solve or significantly reduce memory access bottleneck problems for memory-intensive applications. We also believe it is imperative to track the advances made in PIM research to identify open challenges and enable the research community to make informed decisions and adjust future research directions. In this survey, we analyze recent studies that explored PIM techniques, summarize the advances made, compare recent PIM architectures, and identify target application domains and suitable memory technologies. We also discuss proposals that address unresolved issues of PIM designs (e.g., address translation/mapping of operands, workload analysis to identify application segments that can be accelerated with PIM, OS/runtime support, and coherency issues that must be resolved to incorporate PIM). We believe this work can serve as a useful reference for researchers exploring PIM techniques.

1. Introduction

Conventional Von-Neumann architectures have been the dominant model for computing systems across almost all domains. Such architectures have a separate memory device to provide data to computing units for processing and an I/O device to display the results. This model has been proven efficient, and much of today's modern computer architectures are entrenched to it.

However, with emerging applications, data movement to and from memory has surfaced as an expensive operation in terms of time and energy [1]. This bottleneck occurs when applications require a volume of data to be moved from memory to computing units at a rate that cannot be sustained to provide optimum performance and energy efficiency, even with the deployment of highly efficient cache memories. In simple terms, when an application is exceedingly memory bound (e.g., due to last-level cache misses), the time and energy it takes to fetch and move data from the memory to the computing unit is highly inefficient: transferring data from DRAM to the processing unit through the cache hierarchy takes about two orders of magnitude more energy than performing a floating point operation on the processing unit once the data has arrived [2]. Several factors exacerbate this issue: (1) conventional memory systems (i.e., DRAM) usually reside on a DIMM connected to the processing units through a narrow bus, and these connections are limited by a finite number of pins; (2) DRAM modules operate at a significantly lower frequency than the processing units; and (3) each application has very different memory requirements and access patterns. Therefore, not all applications benefit from the standard configurations.

To tackle these problems, several approaches have been adopted, including introducing cache hierarchy and out-of-order processing to mask the penalty incurred by memory accesses. New memory technologies and novel configurations are also being investigated. However, these approaches are not keeping up with the exponential growth of memory required by modern applications, and this has challenged the conventional structure of computing systems.

Processing-in-memory (PIM) is one of the approaches that deviates from the Von-Neumann architecture to bring computation into or near the memory instead of transferring large amounts of data to and from the computing unit [3]. The idea of PIM is not particularly new. However, in recent years, a tremendous surge of research activity in this field has resulted from the ever-increasing memory requirements

This research was supported in part by the DOE Advanced Scientific Computing Research Program Sawtooth Project and the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle LLC for DOE.
* Corresponding author.
E-mail address: asifuzzamank@ornl.gov (K. Asifuzzaman).

https://doi.org/10.1016/j.memori.2022.100022
Received 2 September 2022; Received in revised form 21 November 2022; Accepted 15 December 2022

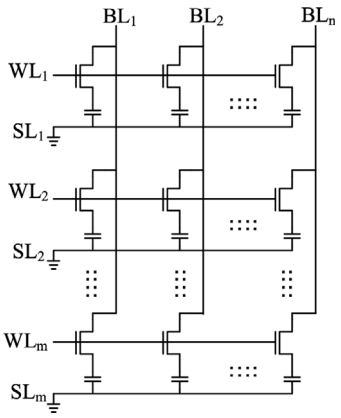

Fig. 1. DRAM array [5].

of modern applications, which have driven researchers to consider a paradigm shift to tackle the problem. Although PIM appears as a generic term, it has several unique aspects and features. PIM can be used at the cell level of the data array (i.e., simple operations), at the sense amplifier/row-buffer level (i.e., more complex operations), or by using simple cores near memory banks that can execute a subset of CPU instructions (e.g., for processing a larger/memory-bound part of the application). Additional design complexity occurs when these different PIM approaches are realized by using various memory technologies (e.g., DRAM, high-bandwidth memory [HBM], hybrid memory cube [HMC], spin-transfer-torque magnetic RAM [STT-MRAM], resistive RAM [ReRAM], phase change memory [PCM]). Among alternative memory technologies, non-volatile SRAM designs have also enabled in-memory computation [4]. Additionally, PIM units may need to be designed for a specific application field for optimal performance. For example, specific compute patterns of deep neural network (DNN) applications may achieve maximum efficiency by performing computations using the unique data-array structure of ReRAM.

Here, we analyze all the aforementioned aspects of recent studies that evaluate PIM to understand how far the technique has advanced and how it is evolving. We also describe different evaluation methodologies used by researchers to model PIM designs as well as the challenges and opportunities that lie ahead in this endeavor.

2. Memory technologies

Because this article discusses PIM as it applies to a range of memory technologies, we first briefly describe the design and organization of the main memory technologies that frequently appear in research and commercial designs.

2.1. DRAM

DRAM, or dynamic random access memory, is the most widely used and implemented commodity memory technology for modern computing systems. DRAM has been developed over decades of persistent research and is the most affordable main memory technology.

DRAM memory systems comprise three main components: memory controller, memory bus, and DRAM devices organized in dual-in-line memory modules or DIMMs. Most modern CPUs have integrated memory controllers and memory channels connected to DRAM DIMMs for transmitting data, commands, and addresses. DRAMs usually have separate dedicated slots on the motherboard so that they can be easily replaced or serviced.

DRAM stores data in *cells*, which are constituted by one transistor and one capacitor; the fully charged capacitor represents the logical value 1, and the discharged capacitor holds the logical value 0. These

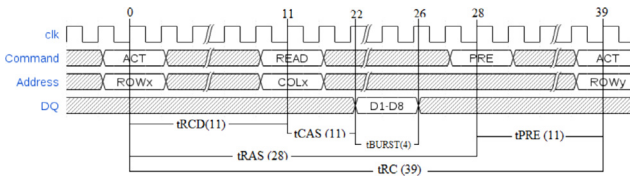

Fig. 2. DRAM READ timing.

DRAM cells are connected through word-lines and bit-lines that form an array structure (Fig. 1). When a row of this array is activated, it brings the data of that array to the row buffer, and selected data can then be transmitted to the CPU through the bus.

Because DRAM is the de-facto standard for main memory systems, one must understand its commands and timing to appreciate the modifications and changes required for the emerging memory technologies and protocols. To access data from DRAM, the specified *row address* must be activated by issuing an `ACT` command, which brings the data to the row buffer. Then, to carry a read operation, the memory controller must issue a `READ` command to provide the column address on the row that it needs to read. The data selected by these row/column addresses are then placed on the data bus to be transmitted. DRAM read is self-destructive, which means that once data is loaded into the row buffer, it clears the data in the row; therefore, the data must be written back to the row when the row is closed. A `PRECHARGE` operation is needed to carry out this procedure. Fig. 2 shows the timing constraints (not to scale) that DRAM must maintain between the commands that it issues.

DRAM memory technology is facing several challenges to remain the dominant memory technology. Extreme scaling requires DRAM capacitors to shrink so that more of them can fit within the same space constraints, thereby making them increasingly vulnerable to errors. Also, by having a *planar* layout, DRAM is constrained by the limited number of physical pins and is unable to sustain the bandwidth requirements of modern applications.

2.2. HBM

The 3D-stacked DRAMs were developed to accommodate the increasing bandwidth demands of modern applications. HBM is a variant of 3D-stacked DRAM, which stacks multiple DRAM dies on a base logic layer [7]. Each DRAM die usually has two independent channels, each of which can be divided into two pseudo-channels (HBM2). Multiple HBM stacks can be connected to the processing unit while being on the same silicon interposer, and this facilitates a 1024-bit connection to the processing unit. Although HBM is essentially built on DRAM devices, its unique organization and wider data connection enable it to achieve much higher bandwidth and capacity when multiple stacks are used (Fig. 3).

2.3. HMC

HMC is another DRAM derivative that adopts 3D-stacking of DRAM dies [8]. Unlike HBM, each DRAM die in HMC is distributed in partitions that are vertically connected with other partitions of adjacent dies, thereby forming a *vault*, which is controlled by a memory controller that resides in the logic base layer (Fig. 4). Vault controllers are connected to other HMCs or host devices through a packet-based communication protocol, which is implemented with a high-speed serialization/de-serialization circuit.

2.4. STT-MRAM

STT-MRAM is an emerging non-volatile memory technology based on the magneto-resistance caused by spin-polarized current [10]. The

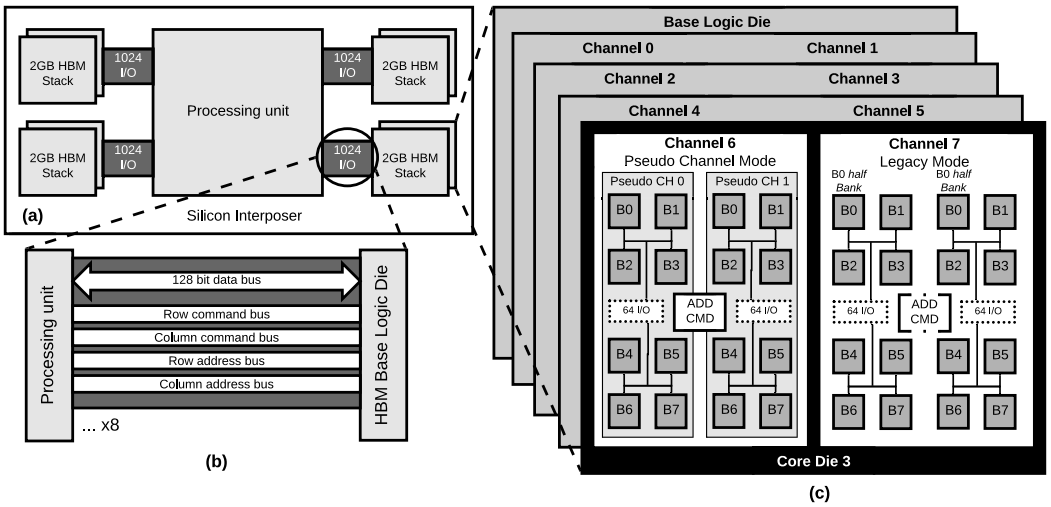

Fig. 3. (a) General organization of HBM stacks with a processing unit. (b) Per-channel data/bus connections with processing unit via interconnect circuitry (e.g., memory controller). (c) Internal structure of a 4-die HBM stack with arrangements of banks for pseudo channel mode (Channel 6) and legacy mode (Channel 7) [6].

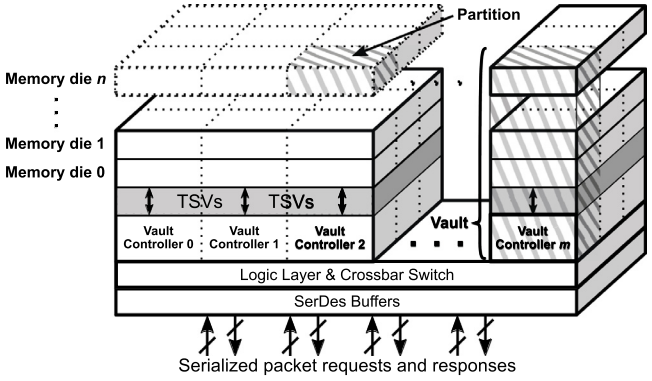

Fig. 4. HMC or hybrid memory cube [9].

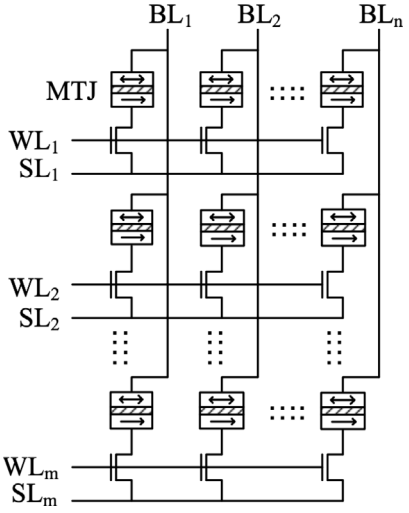

Fig. 5. A cell array of STT-MRAM or spin-transfer-torque magnetic RAM [5].

storage and programmability of STT-MRAM revolve around a magnetic tunneling junction (MTJ), which is composed of a thin tunneling dielectric layer sandwiched between two ferromagnetic layers. One of these layers has a fixed magnetization, and the other layer can be changed by passing a large amount of current through it. When both layers have the same polarity, then the MTJ exerts low resistance, which represents logical 0. On the other hand, when the layers have different polarities, the MJT represents a high-resistance state and a logical 1. As shown in Fig. 5, STT-MRAM's cell array structure can be very similar to that of DRAM [5].

2.5. Reram

ReRAM is another non-volatile memory technology that could be adopted as an alternative. A ReRAM cell is a two-terminal device with a metal–insulator–metal structure. These cells can establish a low-resistance state or a high-resistance state by creating or dissolving a conductive filament that works as the metal oxide insulator. Three main operations can be performed with ReRAM cells: SET, RESET, and READ. The SET operation invokes the low-resistance state, and the RESET operation transforms the cell from the low-resistance state to a high-resistance state. A READ operation detects the current resistance state of the cell by applying a small sensing voltage that does not change the cell's resistance [11].

2.6. PCM

PCM is another emerging non-volatile memory that has garnered considerable attention as an alternative memory technology. A PCM cell has two electrodes separated by a phase-change substance (e.g., chalcogenide material such as Ge2Sb2Te5). These substances can have two distinct electrical resistance properties (i.e., crystalline or amorphous), which indicates it could be an ideal candidate for a memory technology. Like ReRAM, PCM cells also have three main operations: SET (to 1), RESET (to 0), and READ [12]. Although READ and RESET operations on PCM are relatively fast, the SET operation is very slow, thereby making PCM an unlikely candidate as the main memory technology for a high-performance computing system. Additionally, the low endurance of PCM cells means it may not be a practical choice for enterprise systems.

3. In-depth analysis of PIM literature

There has been a tremendous surge of investigations of different variants of PIM. In this section, we present an analysis of recent studies in this field. We organize our evaluation of PIM studies into

four categories: design type, application field, memory technology, and evaluation methodology.

3.1. PIM by design type

In this section, we discuss the three main types of PIM designs, which are classified by where the processing takes place in memory.

3.1.1. Processing at the Data Array (DA)

Some studies propose adopting compute capabilities in the data-array level [13–21]. Generally, these studies explore and investigate the possibilities of leveraging the cell-level structures to introduce some computing capabilities.

Sun et al. [13] propose a coprocessor based on STT-MRAM for convolutional neural network (CNN) acceleration. The authors also report implementing an advanced technology node of 22 nm on CMOS, SRAM, and STT-MRAM. The proposed CNN processing block simultaneously performs 3×3 convolution on a 2D image at $P \times P$ pixel locations by using input from the input buffer and filter coefficients from colocated on-chip MRAM memory.

Imani et al. [14] exploit the analog characteristics of the conventional crossbar memory to enable essential operations in memory. Unlike previous efforts that compute bit-wise operations on the sense amplifier of each memory block, this work supports bit-wise operations *internally* in memory without reading the values out of the block. The authors report adopting row-parallel computation, achieving 1000 parallel addition/multiplication on memory with 1000 rows.

Leitersdorf et al. [15] propose to speed up in-memory multiplication by using novel, partition-based computation techniques for broadcasting/shifting data among partitions along with replacing the Wallace tree with a carry-save-add-shift (CSAS) multiplier and introducing a novel full-adder design. This study also exploits a unique ReRAM (memristive crossbar) feature: voltage-controlled variable resistance to support logic gates (e.g., NOT, NOR, OR, NAND). Peng et al. [16] propose a novel mapping method and dataflow to maximize the reuse of weight and input data on an 8-bit ReRAM-based PIM architecture. The concept is simulated with NeuroSim simulator [22], and the results suggest that the proposed techniques save 90% latency and 68% energy in interconnect and buffers on the ResNet-34 benchmark.

Long et al. [17] present another ReRAM-based PIM architecture specifically for recurrent neural network (RNN) acceleration (Fig. 6). The authors propose an in-memory processing unit with three main subarrays: crossbar subarrays for matrix–vector multiplication, special function units for nonlinear functions, and multiplier subarrays for element-wise operations. Reportedly, the proposed techniques result in a 79× improvement on average over the GPU baseline.

Lu et al. [18] explore reconfigurable design methodologies for compute in memory (CIM)–based accelerator to support CNNs running on prefabricated chips. Authors evaluate the concept with a modified DNN + NeuroSim framework running a system-level performance benchmark.

Kazemi et al. [20] introduce multibit in-memory hyperdimensional computing (HDC) inference that supports multibit operations that use ferroelectric field effect transistor (FeFET) crossbar arrays for multiply-and-add and FeFET multibit content addressable memories for associative search.

3.1.2. Processing at the row buffer

Few studies propose low-level techniques to enable processing capability in or around the row buffer or sense amplifiers.

Roy et al. [23] propose a novel multiplication scheme inside DRAM at the subarray level with negligible changes to the DRAM subarrays. Because the multiplication is performed by addition and AND operations, the study proposes a fast, lightweight, bit-wise, and in-subarray AND operation in DRAM to reduce the overall multiplication cost. They incorporate a novel PIM-DRAM bank architecture depicted in

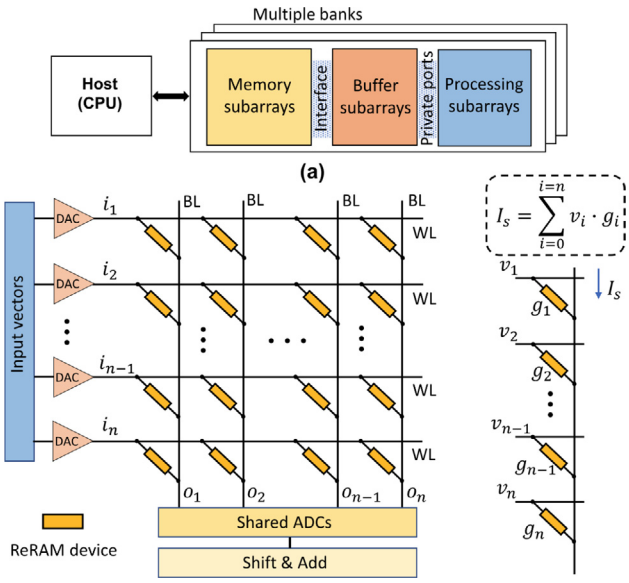

Fig. 6. PIM architecture and ReRAM crossbar for matrix–vector multiplication proposed by Long et al. [17].

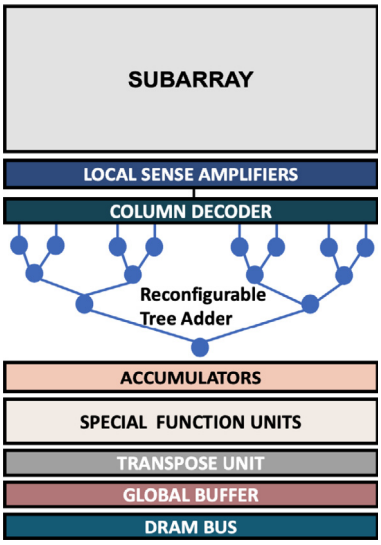

Fig. 7. PIM-DRAM bank architecture proposed by Roy et al. [23].

Fig. 7, including adder trees and nonlinear activation function units in each DRAM bank for efficient machine learning (ML) acceleration. The proposed architecture supports the ReLU, batch-normalization, and pooling operations needed for a wide range of ML models. The proposed PIM operations include a bit-wise AND operation that dedicates two extra rows in the DRAM subarray, and in-DRAM multiplication is broken down into AND and ADD operations that use nine compute rows. In the proposed data mapping, every DNN layer is allocated to a DRAM bank. In the mapping algorithm, the outermost loop runs across all layers in the neural network.

Long et al. [24] present a FeFET-based PIM architecture to accelerate the inference of DNNs. This work proposes a digital in-memory vector–matrix multiplication engine design that uses the FeFET crossbar to enable bit-parallel computation and eliminate analog-to-digital conversion in prior mixed-signal PIM designs. A dedicated hierarchical network-on-chip is developed for input broadcasting and on-the-fly partial results processing, thereby reducing the data transmission volume and latency. Simulations of a 28 nm CMOS technology show

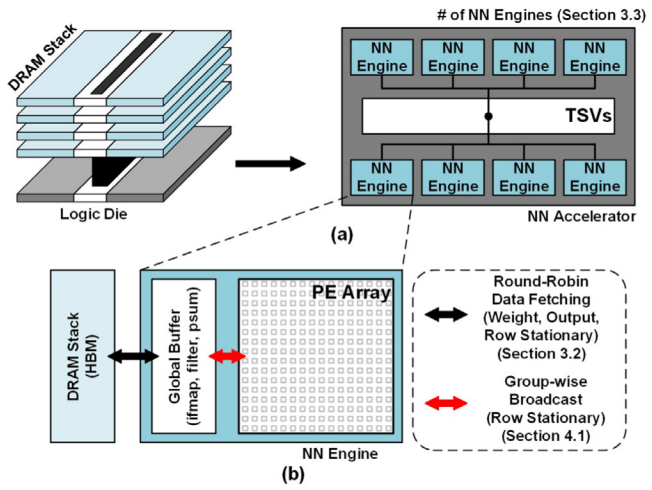

Fig. 8. Overall architecture proposed by Park et al. [32]. (a) Neural network accelerator on the HBM logic die. (b) Neural network engine with the data-fetching scheme proposed in the study.

115× higher computing efficiency (gigaflops/W) over a desktop GPU (NVIDIA GTX 1080 Ti) and 6.3× higher computing efficiency over a ReRAM-based design.

Lee et al. [25] present an HBM2-based experimental design that includes 16-cycle MAC (multiply-and-accumulate) units and 8-cycle reducers for matrix–vector multiplication. The study reports having achieved 406% and 35.2% performance improvement for all bank and per-bank scheduling, respectively. Several other studies propose innovative PIM techniques to accelerate neural networks [26–28].

3.1.3. Processing in a unit near memory banks

The majority of the studies that we analyze propose a near-memory compute unit that implements a simple core to execute a subset of CPU instructions to reduce memory traffic to and from the processing unit [1,2,29–48].

Olgun et al. [29] argue that some processing-using-memory (PuM) mechanisms require special memory allocation and alignment schemes that are not provided by the existing memory allocation primitives. Also, in-DRAM copy operations require efficient handling of memory coherence, and this makes it difficult to analyze PuM techniques on proprietary computing systems or simulators. To tackle the issue, the study develops a field-programmable gate array (FPGA)-based prototype to demonstrate end-to-end integration and evaluation of PuM mechanisms using real DRAM chips. To that end, the proposal incorporates two hardware components: a PuM operations controller that works between the application and the memory controller as a memory-mapped module to provide instruction set architecture (ISA)-transparent control of PuM techniques, and the custom memory controller that provides refresh, scheduling, and timing used for DDRx sequences that trigger PuM operations. Software components include the PuM operations library (pumolib), which exposes PuM operations to the application developer, and a custom supervisor software that provides necessary OS primitives (i.e., virtual memory management, memory allocation, and alignment).

Park et al. [32] discuss the high-performance, near-memory neural network accelerator architecture that uses the logic die in 3D HBM-like memory (Fig. 8). Because most of the previously reported 3D memory-based, near-memory neural network accelerator designs used HMC memory, the first focus was to identify the key differences between HBM and HMC in terms of near-memory neural network accelerator design. The study introduces the round-robin data fetching and group-wise broadcast schemes to exploit the centralized through-silicon-via (TSV) channels. The article shows that an efficient scheme designed to fetch data from the DRAM dies to the neural network accelerator on the logic die in HBM is different from an efficient data-fetching scheme for neural network accelerators in HMC. To be more exact, the bottom buffer die of current generation HBM is implemented in a DRAM process. Because a logic process is needed to implement a high-performance neural network accelerator, it is assumed that the bottom die of HBM is implemented in the logic process in this study. Authors propose (1) for the data movement from DRAM stacks to neural network engines on the logic die in HBM, the round-robin fashion of data fetching is more efficient than the conventional distributed data fetching, and (2) with conventional architectures using a multicast scheme, increasing the bit width leads to a huge routing overhead of connecting wide I/Os to all processing elements. On the contrary, by using a group-wise broadcast scheme with predetermined and grouped interconnects, the proposed architecture can remove the routing overhead.

Kwon et al. [33] demonstrate that as the models and the datasets used to train deep learning models scale, the system architects are faced with new challenges, one of which is the memory capacity bottleneck, at which the limited physical memory inside the accelerator device constrains the algorithm that can be studied. The study proposes a memory-centric deep learning system that can transparently expand the memory capacity available to the accelerators while also providing fast interdevice communication for parallel training. This proposal aggregates a pool of memory modules locally within the device-side interconnect, and these modules are decoupled from the host interface and function as a vehicle for transparent memory capacity expansion. Compared with conventional systems, this technique achieves an average speedup of 2.8× on eight deep learning applications and increases the system-wide memory capacity to tens of TBs.

Lee et al. [2] propose an innovative PIM architecture that can seamlessly work with unmodified commercial processors as a clean replacement for standard DRAM. To demonstrate its feasibility and efficiency at the system level, the authors proposed a PIM architecture based on a commercial HBM2 DRAM die design fabricated with a 20 nm DRAM technology, integrated the fabricated PIM-HBM with an unmodified commercial processor, and developed the necessary software stack. The PIM architecture consists of (1) a PIM-HBM DRAM die; (2) a bank coupled with a PIM execution unit comprising a single instruction multiple data (SIMD) floating-point unit (FPU), command register file (CRF), general register file (GRF), and a scalar register file (SRF); and (c) the data path of the PIM execution unit (Fig. 9). In PIM mode, PIM execution units across all the banks concurrently respond to a standard DRAM column command (e.g., `READ` or `WRITE`) from the host processor, executing one wide-SIMD operation commanded by a PIM instruction with deterministic latency in a lock-step manner. A PIM execution unit consists of three components: (1) a 16-wide SIMD FPU, (2) register files, and (3) a controller (Fig. 10). The PIM execution unit is divided into up to five pipeline stages to satisfy the DRAM internal timing for reading/writing data. The first stage fetches and decodes a PIM instruction. The second stage loads 256-bit data from the `EVEN BANK` or the `ODD BANK` to either a GRF or an input of the SIMD FPU. The third stage is `MULT`, and the fourth stage is `ADD`. That is, the MAC goes through both the third and fourth stages, whereas `MULT` skips the fourth stage, and `ADD` skips the third stage. The last stage writes the result to a GRF.

Ghose et al. [35] examine three key domains for the practical construction and widespread adoption of PIM architectures. First, they describe their work on systematically identifying opportunities for PIM in real applications and quantify potential gains for popular emerging applications (e.g., ML, data analytics, genome analysis). Second, they aim to solve several key issues in programming these applications for PIM architectures. Third, they describe challenges that remain for the widespread adoption of PIM. A function is a PIM target candidate in a consumer device if it meets the following conditions: (1) it consumes the most energy out of all functions in the workload, as

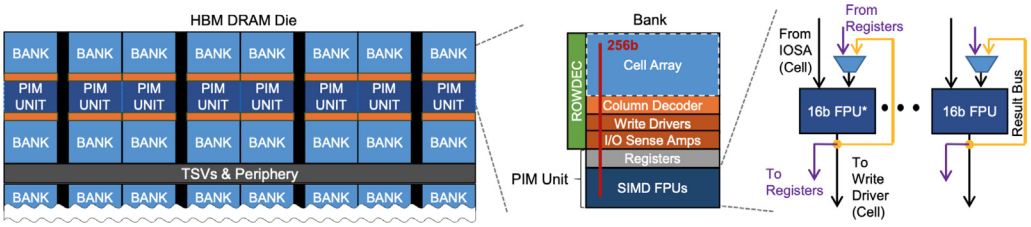

Fig. 9. HBM DRAM die organization with PIM unit and its data path, as proposed by Lee et al. [2].

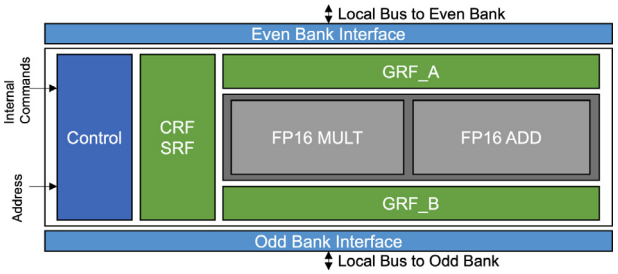

Fig. 10. Proposed microarchitecture of PIM execution unit with control, register, and single instruction multiple data (SIMD) units [2].

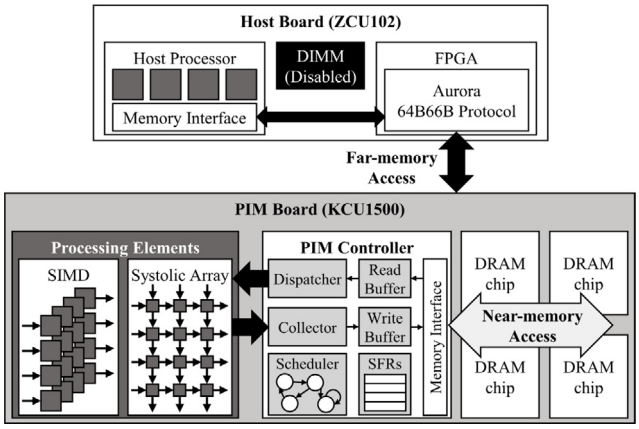

Fig. 11. PIMCaffe system architecture proposed by Jeon et al. [34].

energy reduction is a primary objective in consumer workloads; (2) its data movement consumes a significant fraction (i.e., >20%) of the total workload energy to maximize the potential energy benefits of offloading to PIM; (3) it is memory-intensive (i.e., its last-level cache misses per kilo instruction [MPKI] is >10), as the energy savings of PIM is higher when more data movement is eliminated; and (4) data movement is the single largest component of the function's energy consumption. The study identifies four key issues that affect the programmability of PIM architectures: (1) the different granularities of an offloaded PIM kernel, (2) how to handle data sharing between PIM kernels and CPU threads, (3) how to efficiently provide PIM kernels with access to essential virtual memory address translation mechanisms, and (4) how to automate the identification and offloading of PIM targets (i.e., portions of an application that are suitable for PIM).

Several papers propose near-memory processing techniques for accelerating neural networks [32,36,37], and Huang et al. [44] aim to achieve energy-efficient graph processing with heterogeneous PIM hardware/software co-design.

3.2. PIM by application field

Recent PIM studies focus on attempts to use PIM to accelerate application segments in ML, AI, and neural network domains [13,16–18,23,24,26–28,31–34,36,37]. Because applications from these domains frequently use matrix–vector multiplication operations, the coefficients can be naturally mapped to the word lines and bit lines and use the data array to compute the dot product in each cell and then eventually accumulate them along the source lines [16,17]. In another approach, some studies use the logic die of 3D-stacked memory to place processing elements directly beneath the DRAM dies with a shorter distance to the coefficient data for faster computation in each layer [32,37]. Imani et al. [14] reportedly achieved graph processing and query processing capabilities along with ML acceleration with PIM. Huang et al. [44] propose a heterogeneous PIM design that incorporates memristors and CMOS-based technologies to accommodate the heterogeneity requirements of graph applications.

3.3. PIM by memory technology

There are some interesting observations when using particular memory technologies in certain PIM designs. Most of the studies that propose processing in a DA use emerging non-volatile memory (e.g., ReRAM) because the data-array structure of this memory is particularly suitable for such implementations [13–15,21,25,41,44]. On the other hand, work that builds on the idea of having a processing unit near memory mainly opts for 3D-stacked memories (e.g., HBM, HMC) to leverage the additional logic layer [2,32,42,45,46,48,49]. Few studies propose PIM optimizations on commodity DRAM technology [23,28–30,34,38,39], and three studies focus their designs on UPMEM, which is a commercially developed DRAM-based DIMM with in built-in compute capability [1,40,50].

3.4. PIM by evaluation methodology

Researchers are adopting various methodologies to design and test PIM techniques. We categorize these methodologies in three segments: analytical models, simulation models, and hardware implementations.

3.4.1. Analytical model

A theoretical model was proposed for the design and analysis of PIM-based parallel algorithms [51]. The proposed model combines the CPU side of parallel cores with fast access to a shared memory and a PIM side that consists of local memory and a processor core. It also proposes standard parallel complexity metrics for both shared memory and distributed memory computing. The proposed model is evaluated for a skip-list algorithm with seven operations: `GET(key)`, `Update(key, value)`, `Delete(key)`, `Predecessor(key)`, `Successor(key)`, `Upsert(key, value)`, and `RangeOperation(lkey, rkey, function)`.

3.4.2. Simulation model

Simulators played a vital role in evaluating and boosting the research for PIM-based systems because no real commercially available PIM hardware is available to the researchers.

Ghose et al. [30] use a GEM5 full-system simulator with DRAMSim2 to evaluate the efficient in-memory accelerator for pointer chasing,

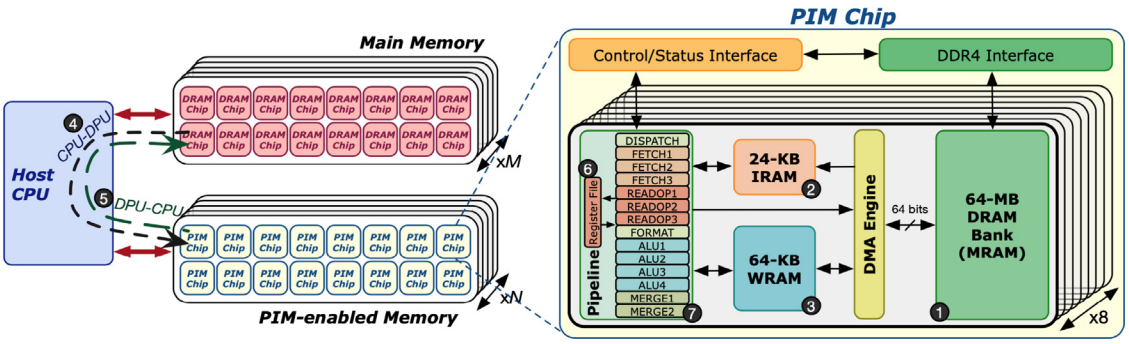

Fig. 12. Architecture and organization of an UPMEM-based PIM system [1].

which can handle address translation entirely within DRAM. DRAMSim2 provides accurate memory modeling and DRAM energy analysis.

Roy et al. [23] propose a DRAM-based PIM multiplication primitive to accelerate matrix–vector operations in ML workloads and evaluated DNNs by using these operations on HSPICE circuit simulations. However, it is not clear how the analog circuit simulator is used for these experiments.

Zhou et al. [19] propose a full-stack simulation infrastructure to explore the design space of digital PIM. This infrastructure incorporates a software library, a configurable compiler layer, and a fast and accurate PIM-enabled architecture model. Authors claim that the proposed simulator provides 10.3× faster simulation with 6.3% deviation compared with a validated simulator.

Xu et al. [52] present PIMSIM (processing-in-memory simulator), which is equipped with a partitioner at the front end capable of recognizing and distributing PIM instructions. It also provides dynamic feedback support to determine if a PIM instruction should be executed in memory. The simulator supports three simulation modes: fast simulation, instrument-driven simulation, and full system simulation.

Lu et al. [22] develop and present the NeuroSim simulator for CIM hardware simulation. NeuroSim can estimate statistics related to the area, latency, dynamic energy, and leakage power consumption of the hardware performance. The simulator is validated against actual silicon data from a 40 nm 16 kb CIM macro that uses TSMC's 40 nm RRAM process, whereas the peripheral circuit modules (e.g., decoders, switch matrix, MUX [multiplexer], adders) are validated with SPICE simulations.

Xie el al. [53] propose MPU-Sim for general-purpose near-bank processing architectures to model several MPU cores inside a processor connected through on-chip network links. Each MPU core includes several near-bank processing units on DRAM dies. MPU-Sim provides support for the SIMT (single instruction, multiple threads) programming model to exploit massive bank-level parallelism and address the challenge of control logic and communication overheads.

MultiPim [54] and DAMOV [55] are simulation infrastructures built on ZSim [56] (a system simulator) and Ramulator [57] (a memory simulator). Both simulators support offloading kernels to PIM units. MultiPim employs a multistack interconnect, crossbar switches, PIM core coherence, and virtual memory, whereas DAMOV performs an extensive workload characterization to identify data-movement bottlenecks across a range of applications and functions.

3.4.3. Hardware implementations

Jaio et al. [31] propose a chiplet-based PIM design strategy evaluated with a Tiny-Yolo DNN running on an FPGA. However, the FPGA's details are not provided. The proposed layer-wise method partitions a monolithic accelerator's workload to a multichiplet pipeline.

PimCaffe [34] develops a PIM-emulating FPGA platform with SIMD and systolic array computing engines that can perform vector and matrix multiplication on the PIM device. A high-level block diagram of the proposed design in shown in Fig. 11.

Samsung [2] proposes an industrial PIM prototype fabricated with a 20 nm DRAM process. This PIM execution unit consists of sixteen 16-bit SIMD lanes, each with an floating-point (FP) adder, FP multiplier, 32-entry command register, 16-entry general register, and 16-entry scalar registers.

UPMEM [1] is another PIM-based technology in commercial production. Fig. 12 shows an UPMEM based PIM system with a host CPU, standard DRAM main memory, and PIM-enabled main memory. An UPMEM module is based on a standard DDR4 DIMM with several PIM chips. Each PIM chip consists of 8 DRAM processing units (DPUs), and each DPU has access to a 64 MB DRAM bank, 24 KB of instruction memory, and 64 KB of scratchpad memory. The 64 MB DRAM banks are accessible by the host CPU for copying input data from main memory and retrieving results.

Table 1 presents a summary of our survey and provides a high-level view of the application domain, publication year, PIM category, memory technology used, evaluation methodology, and discussions of several other issues relevant to PIM implementation.[1]

4. Challanges

Through the literature review, we identify the areas that should be further developed to ensure PIM can be adopted as a standalone and efficient solution. Most of the studies surveyed focus on only a part of the PIM implementation, thereby leaving crucial questions unanswered.

No matter how intriguing PIM sounds, it comes with a range of challenges across the implementation framework. These challenges must be addressed before PIM can be adopted as a universal alternative or complement to traditional memory technologies. PIM is expected to accelerate certain memory-bound tasks, but much of the computation will still be executed on the CPU. Therefore, overhead costs will still apply for classifying tasks for the PIM execution unit or for the main processing unit. Also, the execution units in/near memory are supposed to be a simple core that supports limited instructions, and they are unable to execute programs that use a variety of instructions. To efficiently allocate application portions to be processed in memory, runtime and OS support is needed. Furthermore, significant changes in memory controller design would be warranted. Therefore, we classify the main challenges of adopting PIM as workload analysis and classification, address translation and mapping, OS and runtime support, compiler and programming models, and coherency and consistency. These issues are explored in more detail below.

[1] DA = data array, RB = row buffer, NM = near memory, DR3 = DDR3, DR4 = DDR4, MRM = MRAM, UPM = UPMEM, ReR = ReRAM, HBM = HBM, 3D = 3D-stacked memory, PCM = PCM, FFT = FeFET, HMC = HMC, GE5 = Gem5, SPC = SPICE, CAC = CACTI, INH = in-house, CF3 = Caffe3, DS2/3 = DramSim2/3, RML = Ramulator, NRO = NeuroSim, PIN = Pin, GGU = GPGPUSim, SYN = Synopsys, RSV = RISC V, x86 = x86, NVD = NVIDIA, CDA = CUDA, CNN = convolutional neural network, DNN = deep neural network, NN = neural network, DL = deep learning, BData = big data, AQUAB = Aquabolt-XL, BNN = binarized neural network, OFFLD = offloading technique, SIM = simulator study, and HDC = hyperdimensional computing.

Table 1
A summary of the survey to provide a high-level view of the application domain (if specified), publication year, PIM category, memory technology used, evaluation methodology, and discussions of several other issues relevant to PIM implementation. Green highlights indicate detailed discussions, and yellow highlights indicate limited discussions.

Study	Appl. area	Author	Year	P. IN/Near MEM	Memory type	Mapping	Simulation	FPGA/HW	Bench/app	Compiler/PM	Runtime/OS	Scheduling	Addr.Trans	Coher/consi
[29]		Olgun et al.	2021	NM	DR3									
[30]		Ghose et al.	2018	NM	DR3		GE5							
[13]	CNN	Baohua et al.	2018	DA	MRM									
[23]	DNN	Anand R et al.	2021	RB	DR3		SPC							
[1]		GomezLuna et al.	2021	NM	UPM									
[14]	BData	Rosing et al.	2019	DA	ReR									
[31]	DL	Jiao et al.	2021	NM										
[32]	CNN	KIM et al.	2021	NM	HBM		CAC							
[24]	DNN	Long et al.	2019	RB										
[33]	DL	Kwon et al.	2018	NM			INH							
[2]	AQUAB	S. Lee et al.	2021	NM	HBM									
[15]	PIM	Kvatinsky et al.	2021	DA	ReR									
[26]	NN	Gupta et al.	2018	RB				SYN						
[16]	CNN	Peng et al.	2020	DA										
[34]	ML	Won et al.	2021	NM	DR3									
[35]		Ghose et al.	2019	NM										
[27]	CNN	Roohi et al.	2019	RB										
[36]	NN	Liu et al.	2018	NM				SYN						
[17]	RNN	Long et al.	2018	DA										
[37]	CNN	Wang et al.	2018	NM			CF3							
[38]	REC	Ke et al.	2020	NM	DR3									
[39]	GCN	Zhou et al.	2021	NM	DR4		DS3							
[40]		Nider et al.	2021	NM	UPM									
[49]		Xie et al.	2020	RB	3D		RML							
[41]		Nader et al.	2021	NM	PCM		GE5							
[18]	CNN	Lu et al.	2021	DA	FFT		NRO							
[42]		Ahn et al.	2015	NM	HMC		PIN							
[28]	BNN	Lin et al.	2021	RB	DRM		GE5	RSV						
[43]		Boroumand et al.	2019	NM	HMC		GE5	x86						
[44]	Graph	Huang et al.	2020	NM	ReR		DS3							
[45]	OFFLD	Pattnaik et al.	2016	NM	3D		GGU	NVD						
[46]		Zhang et al.	2020	NM	HBM			RSV						
[48]		Drumond et al.	2017	NM	HMC		FLX							
[25]		LEE et al.	2019	RB	HBM		DR2							
[19]	SIM	Zhou et al.	2021	DA	ReR									
[20]	HDC	Kazemi et al.	2021	DA	FFT		INH							
[52]	SIM	XU et al.	2019	NM	ALL									
[21]		Jung et al.	2022	DA	MRM									
[22]		Lu et al.	2021	DA	ReR									
[50]		Ginnoula et al.	2022	NM	UPM									
[54]	SIM	Yu et al.	2021	NM	HMC									
[55]	SIM	Oliveira et al.	2021	NM	HMC									

4.1. Workload analysis and classification

To better understand the potential use and expected acceleration of PIM execution, we must understand and characterize the candidate applications and functions. Generally, application segments that suffer from the poor locality and have higher last-level cache misses are good candidates for PIM. Oliveira et al. [55] present a detailed methodology for workload characterization in which the authors classify candidate functions in six categories based on various combinations of temporal locality, arithmetic intensity, last-to-first-level cache miss ratio (LFMR), and MPKI to determine if they are good candidates to be executed on a processing unit near memory. The study determines that such an approach can be beneficial for five categories except for the one in which functions demonstrate high temporal locality, low LFMR, and high arithmetic intensity.

4.2. Address translation and mapping

Deploying PIM techniques requires revising the mapping and address translation process of conventional memory systems. Olgun et al. [29] argue that data mapping and allocation requirements must be augmented to satisfy PIM techniques. The authors identify that in their PIM design, source and destination operands of `COPY` should reside in the same DRAM subarray, referring to it as a *mapping problem* and proposing a custom supervisor software to handle it. Lee et al. [2] avoid the overhead of virtual–physical address translation by reserving memory space for PIM operations during the booting process. They also set this reserved memory space in an uncacheable region, so when the host processor sends a DRAM command for memory access to the PIM memory space, the PIM device driver allocates physically contiguous memory blocks.

4.3. OS and runtime support

PIM incorporation requires OS and runtime support to ensure correct and efficient functionality of the modified memory controller required for PIM [46]. Ahn et al. [42] propose a simplified hardware structure that oversees the locality of data accessed by PIM-enabled instructions at runtime to determine if the instruction should be executed on the host processor instead of the PIM unit. Wang et al. [37] observe

that, for the deterministic behavior of convolutional connections, a runtime based approach can determine the mapping of parallelism onto the corresponding hardware. Gómez-Luna et al. [1] use the UPMEM runtime library to handle library calls to move instructions among different memories (e.g., MRAM, IRAM) within PIM units. TUPIM [58] can extract PIM-friendly instructions automatically and offload them to the memory at runtime. TUPIM selects PIM-friendly instructions that (1) reduce intermediate data movements between on-chip caches and main memories, (2) are not well served by the caches, and (3) are repeatedly executed. The study reports that the proposed enhancements achieve 2.2× speedup on average with a 15.7% energy reduction compared with CPU-only execution.

4.4. Compiler and programming models

Olgun et al. [29] reportedly modify the RISC-V GNU compiler tool chain to expose a special instruction, CFLUSH, to C/C++ applications. CFLUSH is used to flush physically addressed dirty cache blocks to maintain coherence. Ghose et al. [30] propose offloading portions of the code to PIM cores by using two macros: #PIM_begin and #PIM_end. The compiler converts these macros to the instruction added to the ISA to trigger and end PIM execution.

4.5. Coherency and consistency

Olgun et al. [29] identify coherence management as a challenge for PIM. Conventional systems deploy caches to keep copies of data in the main memory for fast access to frequently used data. The STORE operation updates the cache data, but the main memory data is not immediately updated, and this becomes challenging for PIM. As mentioned, the authors implement a new custom RISCV instruction called CFLUSH to flush physically addressed dirty cache blocks, thereby solving the memory coherence problem in a minimally invasive manner. Ghose et al. [30] propose the LazyPIM mechanism to maintain cache coherence between PIM processing logic and CPU cores without having to send coherence requests for every memory access. Instead, PIM processing logic speculatively acquires coherence permissions and later sends batched coherence look-ups to the processing unit to ascertain if its speculative permission acquisition violated the memory ordering defined by the programming model. Ahn et al. [42] invoke back-invalidation and back-write-back for the requested cache blocks to the last-level cache before sending the PIM operation to memory. This technique prevents having a stale copy of the data in on-chip caches or in main memory before or after a PIM operation. CuckooPIM [59] proposes a new, variable-grained coherence scheme that dynamically monitors system behavior and strategically grants data ownership. It does not induce notable combinational logic complexity to a PIM system.

5. Previous surveys of PIM

Gagandeep et al. [60] provide a detailed review of near-memory computing architectures with a focus on processing at the data's location, which can mitigate the data movement problem for data-intensive operations. Surveying the literature up to 2018, the authors analyze an extensive body of work on the limited aspects of the memory level at which this paradigm is applied and the granularity of applications for near-memory computing units. The paper limits its analysis to near-memory computing and excludes CIM analysis. Authors classify the state-of-the-art of near-memory computing literature into five broad categories: memory, processing, evaluation technique, interoperability, and target application domain. They also consider different memory properties: hierarchy, memory type, and integration. In the processing category, the authors consider different types of host compute units (e.g., CPU, GPU, CGRA, FPGA, accelerator), implementation of near-memory compute logic (e.g., programmable, fixed function, reconfigurable), and granularity (e.g., instruction level, kernel level, application level). In the evaluation category, the authors separate the literature into analytic, simulation, and prototype/hardware. In interoperability, the authors consider programmable interface support, cache coherence mechanism, and virtual memory support. Authors also survey literature on design space exploration for near-memory systems to understand and evaluate their space. Literature on both microarchitecture dependent [61] and independent [62–64] workload characterization based design space exploration approaches are also considered. The literature also reveals evaluation methods used in state-of-the-art papers, such as analytical modeling-based approaches [65,66] and simulation-based modeling approaches [67–70]. The case study compares a CPU-centric multicore system that uses DDR3 as the main memory (traditional) against a data-centric approach that uses HMC-like 3D-stacked memory instead of DDR3. The authors conclude that applications with low locality should leverage the near-memory approach, whereas high locality applications can benefit more from the traditional approach.

Li et al. [71] summarize progress in the development of in-memory processing for three mainstream eNVM technologies: STT-MRAM, PCM, and ReRAM. The authors classify the studies by memory type, location in the memory hierarchy, design level (i.e., device, circuit, or system), function type (i.e., logic, arithmetic, associative, vector, or matrix–vector multiplication), and application group.

Yu et al. [72] also describe studies that explore CIM with emerging non-volatile memories. The authors particularly focus on prototype chips that monolithically integrate eNVMs with CMOS periphery for deep learning. The study describes an RRAM CIM macro [73] and explains how low on-state resistance of most eNVM technologies may affect analog read out accuracy. Also, the analog MUX at the column end must be significantly sized up to avoid voltage drop. Additional technical challenges come with the write voltage requirements, which are higher for eNVMs such as RRAM and PCM.

Mittal et al. [74] focus their survey on spintronic architectures for PIM to target neural network processing. The authors identify the type of spintronic technologies used, the organization of the proposed PIM accelerator designs, and PIM operations supported by these devices. The study concludes that neither conventional memories and compute-centric architectures nor spintronic memories can meet the grand challenges of AI.

Our study complements previous survey papers with the latest advancements and developments of memory technologies and techniques by uniquely classifying the studies by PIM category, application fields, memory technology, and evaluation models and discussing unresolved challenges on the path forward.

6. Conclusions

In this survey, we summarize the recent advances in PIM. To that end, we introduce technologies that PIM techniques generally build on and analyze research articles, technical papers, and industrial product details to understand how PIM techniques have advanced as of this writing. We describe the prevalent PIM architectures proposed by the community, the application domains that benefit from PIM operations, and the evaluation methodologies used to conduct PIM experiments. We also discuss strategies to mitigate key challenges for implementing PIM techniques, including address translation and mapping of operands, workload analysis to identify application segments that can be accelerated with PIM, OS and runtime support needed to incorporate PIM, and coherency issues for PIM. We believe this survey will serve as a useful reference for future studies that explore PIM techniques.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] J. Gómez-Luna, I.E. Hajj, I. Fernandez, C. Giannoula, G.F. Oliveira, O. Mutlu, Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture, 2021, arXiv:2105.03814.

[2] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, N.S. Kim, Hardware architecture and software stack for PIM based on commercial DRAM technology : Industrial product, in: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture, ISCA, 2021, pp. 43–56, http://dx.doi.org/10.1109/ISCA52012.2021.00013.

[3] X. Zou, S. Xu, X. Chen, L. Yan, Y. Han, Breaking the von Neumann bottleneck: architecture-level processing-in-memory technology, Sci. China Inf. Sci. 64 (6) (2021) 160404, http://dx.doi.org/10.1007/s11432-020-3227-1.

[4] S.R. Sundara Raman, S.S.T. Nibhanupudi, J.P. Kulkarni, Enabling in-memory computations in non-volatile SRAM designs, IEEE J. Emerg. Sel. Top. Circuits Syst. 12 (2) (2022) 557–568, http://dx.doi.org/10.1109/JETCAS.2022.3174148.

[5] K. Asifuzzaman, R.S. Verdejo, P. Radojković, Enabling a reliable STT-MRAM main memory simulation, in: Proceedings of the International Symposium on Memory Systems, in: MEMSYS '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 283–292, http://dx.doi.org/10.1145/3132402.3132416.

[6] K. Asifuzzaman, M. Abuelala, M. Hassan, F.J. Cazorla, Demystifying the characteristics of high bandwidth memory for real-time systems, in: 2021 IEEE/ACM International Conference On Computer Aided Design, ICCAD, 2021, pp. 1–9, http://dx.doi.org/10.1109/ICCAD51958.2021.9643473.

[7] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, K. Kim, HBM (High Bandwidth Memory) DRAM technology and architecture, in: 2017 IEEE International Memory Workshop, IMW, 2017, pp. 1–4, http://dx.doi.org/10.1109/IMW.2017.7939084.

[8] R. Hadidi, B. Asgari, B.A. Mudassar, S. Mukhopadhyay, S. Yalamanchili, H. Kim, Demystifying the characteristics of 3D-stacked memories: A case study for Hybrid Memory Cube, in: 2017 IEEE International Symposium on Workload Characterization, IISWC, 2017, pp. 66–75, http://dx.doi.org/10.1109/IISWC.2017.8167757.

[9] M. Radulovic, D. Zivanovic, D. Ruiz, B.R. de Supinski, S.A. McKee, P. Radojković, E. Ayguadé, Another trip to the wall: How much will stacked DRAM benefit HPC?, in: Proceedings of the 2015 International Symposium on Memory Systems, in: MEMSYS '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 31–36, http://dx.doi.org/10.1145/2818950.2818955.

[10] Y. Xie, Modeling, architecture, and applications for emerging memory technologies, IEEE Des. Test Comput. 28 (1) (2011) 44–51, http://dx.doi.org/10.1109/MDT.2011.20.

[11] H.-S.P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F.T. Chen, M.-J. Tsai, Metal–oxide RRAM, Proc. IEEE 100 (6) (2012) 1951–1970, http://dx.doi.org/10.1109/JPROC.2012.2190369.

[12] I.G. Thakkar, S. Pasricha, DyPhase: A dynamic phase change memory architecture with symmetric write latency and restorable endurance, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (9) (2018) 1760–1773, http://dx.doi.org/10.1109/TCAD.2017.2762921.

[13] B. Sun, D. Liu, L. Yu, J. Li, H. Liu, W. Zhang, T. Torng, MRAM co-designed processing-in-memory CNN accelerator for mobile and IoT applications, 2018, arXiv:1811.12179.

[14] M. Imani, S. Gupta, Y. Kim, M. Zhou, T. Rosing, DigitalPIM: Digital-based processing in-memory for big data acceleration, in: Proceedings of the 2019 on Great Lakes Symposium on VLSI, GLSVLSI '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 429–434, http://dx.doi.org/10.1145/3299874.3319483.

[15] O. Leitersdorf, R. Ronen, S. Kvatinsky, MultPIM: Fast stateful multiplication for processing-in-memory, 2021, arXiv:2108.13378.

[16] X. Peng, R. Liu, S. Yu, Optimizing weight mapping and data flow for convolutional neural networks on RRAM based processing-in-memory architecture, in: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1–5, http://dx.doi.org/10.1109/ISCAS.2019.8702715.

[17] Y. Long, T. Na, S. Mukhopadhyay, ReRAM-based processing-in-memory architecture for recurrent neural network acceleration, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 26 (12) (2018) 2781–2794, http://dx.doi.org/10.1109/TVLSI.2018.2819190.

[18] A. Lu, X. Peng, Y. Luo, S. Huang, S. Yu, A runtime reconfigurable design of compute-in-memory–based hardware accelerator for deep learning inference, ACM Trans. Des. Autom. Electron. Syst. 26 (6) (2021-06) http://dx.doi.org/10.1145/3460436.

[19] M. Zhou, M. Imani, Y. Kim, S. Gupta, T. Rosing, DP-sim: A full-stack simulation infrastructure for digital processing in-memory architectures, in: ASPDAC '21: 26th Asia and South Pacific Design Automation Conference, Tokyo, Japan, January 18-21, 2021, ACM, 2021, pp. 639–644, http://dx.doi.org/10.1145/3394885.3431525.

[20] A. Kazemi, M.M. Sharifi, Z. Zou, M.T. Niemier, X.S. Hu, M. Imani, MIMHD: Accurate and efficient hyperdimensional inference using multi-bit in-memory computing, in: IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2021, Boston, MA, USA, July 26-28, 2021, IEEE, 2021, pp. 1–6, http://dx.doi.org/10.1109/ISLPED52811.2021.9502498.

[21] S. Jung, H. Lee, S. Myung, H. Kim, S.K. Yoon, S.-W. Kwon, Y. Ju, M. Kim, W. Yi, S. Han, B. Kwon, B. Seo, K. Lee, G.-H. Koh, K. Lee, Y. Song, C. Choi, D. Ham, S.J. Kim, A crossbar array of magnetoresistive memory devices for in-memory computing, Nature 601 (7892) (2022) 211–216, http://dx.doi.org/10.1038/s41586-021-04196-6.

[22] A. Lu, X. Peng, W. Li, H. Jiang, S. Yu, NeuroSim simulator for compute-in-memory hardware accelerator: Validation and benchmark, Front. Artif. Intell. 4 (2021) 659060, http://dx.doi.org/10.3389/frai.2021.659060.

[23] S. Roy, M. Ali, A. Raghunathan, PIM-DRAM: Accelerating machine learning workloads using processing in commodity DRAM, 2021, arXiv:2105.03736.

[24] Y. Long, D. Kim, E. Lee, P. Saha, B.A. Mudassar, X. She, A.I. Khan, S. Mukhopadhyay, A ferroelectric FET-based processing-in-memory architecture for DNN acceleration, IEEE J. Explor. Solid-State Comput. Devices Circuits 5 (2) (2019) 113–122, http://dx.doi.org/10.1109/JXCDC.2019.2923745.

[25] W.J. Lee, C.H. Kim, Y. Paik, J. Park, I. Park, S.W. Kim, Design of processing-"inside"-memory optimized for DRAM behaviors, IEEE Access 7 (2019) 82633–82648, http://dx.doi.org/10.1109/ACCESS.2019.2924240.

[26] S. Gupta, M. Imani, H. Kaur, T.S. Rosing, NNPIM: A processing in-memory architecture for neural network acceleration, IEEE Trans. Comput. 68 (9) (2019) 1325–1337, http://dx.doi.org/10.1109/TC.2019.2903055.

[27] A. Roohi, S. Angizi, D. Fan, R.F. DeMara, Processing-in-memory acceleration of convolutional neural networks for energy-efficiency, and power-intermittency resilience, 2019, arXiv:1904.07864.

[28] C.-C. Lin, C.-L. Lee, J.-K. Lee, H. Wang, M.-Y. Hung, Accelerate binarized neural networks with processing-in-memory enabled by RISC-V custom instructions, in: 50th International Conference on Parallel Processing Workshop, Association for Computing Machinery, 2021, URL https://doi.org/10.1145/3458744.3473351.

[29] A. Olgun, J.G. Luna, K. Kanellopoulos, B. Salami, H. Hassan, O. guz Ergin, O. Mutlu, PiDRAM: A holistic end-to-end FPGA-based framework for processing-in-DRAM, 2021, arXiv:2111.00082.

[30] S. Ghose, K. Hsieh, A. Boroumand, R. Ausavarungnirun, O. Mutlu, Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions, 2018, arXiv:1802.00320.

[31] B. Jiao, H. Zhu, J. Zhang, S. Wang, X. Kang, L. Zhang, M. Wang, C. Chen, Computing utilization enhancement for chiplet-based homogeneous processing-in-memory deep learning processors, in: Proceedings of the 2021 on Great Lakes Symposium on VLSI, Association for Computing Machinery, 2021, pp. 241–246, URL https://doi.org/10.1145/3453688.3461499.

[32] N. Park, S. Ryu, J. Kung, J.-J. Kim, High-throughput near-memory processing on CNNs with 3D HBM-like memory, ACM Trans. Des. Autom. Electron. Syst. 26 (6) (2021) http://dx.doi.org/10.1145/3460971.

[33] Y. Kwon, M. Rhu, Beyond the memory wall: A case for memory-centric HPC system for deep learning, 2019, arXiv:1902.06468.

[34] W. Jeon, J. Lee, D. Kang, H. Kal, W.W. Ro, PIMCaffe: Functional evaluation of a machine learning framework for in-memory neural processing unit, IEEE Access 9 (2021) 96629–96640, http://dx.doi.org/10.1109/ACCESS.2021.3094043.

[35] S. Ghose, A. Boroumand, J.S. Kim, J. Gómez-Luna, O. Mutlu, Processing-in-memory: A workload-driven perspective, IBM J. Res. Dev. 63 (6) (2019) 3:1–3:19, http://dx.doi.org/10.1147/JRD.2019.2934048.

[36] J. Liu, H. Zhao, M.A. Ogleari, D. Li, J. Zhao, Processing-in-memory for energy-efficient neural network training: A heterogeneous approach, in: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2018, pp. 655–668, http://dx.doi.org/10.1109/MICRO.2018.00059.

[37] Y. Wang, W. Chen, J. Yang, T. Li, Towards memory-efficient allocation of CNNs on processing-in-memory architecture, IEEE Trans. Parallel Distrib. Syst. 29 (6) (2018) 1428–1441, http://dx.doi.org/10.1109/TPDS.2018.2791440.

[38] L. Ke, U. Gupta, B.Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H.S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C.-J. Wu, M. Hempstead, X. Zhang, RecNMP: Accelerating personalized recommendation with near-memory processing, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), 2020, pp. 790–803, http://dx.doi.org/10.1109/ISCA45697.2020.00070.

[39] Z. Zhou, C. Li, X. Wei, G. Sun, GCNear: A hybrid architecture for efficient GCN training with near-memory processing, 2021, arXiv:2111.00680.

[40] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, A. Fedorova, A case study of processing-in-memory in off-the-shelf systems, in: 2021 USENIX Annual Technical Conference (USENIX ATC 21), USENIX Association, 2021, pp. 117–130, URL https://www.usenix.org/conference/atc21/presentation/nider.

[41] M.S. Hosseini, M. Ebrahimi, P. Yaghini, N. Bagherzadeh, Near volatile and non-volatile memory processing in 3D systems, IEEE Trans. Emerg. Top. Comput. (2021) 1, http://dx.doi.org/10.1109/TETC.2021.3115495.

[42] J. Ahn, S. Yoo, O. Mutlu, K. Choi, PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture, in: 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture, ISCA, 2015, pp. 336–348, http://dx.doi.org/10.1145/2749469.2750385.

[43] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, R. Ausavarungnirun, K. Hsieh, N. Hajinazar, K.T. Malladi, H. Zheng, O. Mutlu, CoNDA: Efficient cache coherence support for near-data accelerators, in: Proceedings of the 46th International Symposium on Computer Architecture, ISCA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 629–642, http://dx.doi.org/10.1145/3307650.3322266.

[44] Y. Huang, L. Zheng, P. Yao, J. Zhao, X. Liao, H. Jin, J. Xue, A heterogeneous PIM hardware-software co-design for energy-efficient graph processing, in: 2020 IEEE International Parallel and Distributed Processing Symposium, IPDPS, 2020, pp. 684–695, http://dx.doi.org/10.1109/IPDPS47924.2020.00076.

[45] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A.K. Mishra, M.T. Kandemir, O. Mutlu, C.R. Das, Scheduling techniques for GPU architectures with processing-in-memory capabilities, in: 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), 2016, pp. 31–44, http://dx.doi.org/10.1145/2967938.2967940.

[46] J. Zhang, Y. Zha, N. Beckwith, B. Liu, J. Li, MEG: A RISCV-based system emulation infrastructure for near-data processing using FPGAs and high-bandwidth memory, ACM Trans. Reconfigurable Technol. Syst. 13 (4) (2020) http://dx.doi.org/10.1145/3409114.

[47] L. Chang, C. Li, Z. Zhang, J. Xiao, Q. Liu, Z. Zhu, W. Li, Z. Zhu, S. Yang, J. Zhou, Energy-efficient computing-in-memory architecture for AI processor: device, circuit, architecture perspective, Sci. China Inf. Sci. 64 (6) (2021) 160403, http://dx.doi.org/10.1007/s11432-021-3234-0.

[48] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, D. Pnevmatikatos, Algorithm/architecture co-design for near-memory processing, SIGOPS Oper. Syst. Rev. 52 (1) (2018) 109–122, http://dx.doi.org/10.1145/3273982.3273992.

[49] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, Y. Xie, iPIM: Programmable in-memory image processing accelerator using near-bank architecture, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, ISCA, 2020, pp. 804–817, http://dx.doi.org/10.1109/ISCA45697.2020.00071.

[50] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, O. Mutlu, SparseP: Towards efficient sparse matrix vector multiplication on real processing-in-memory systems, 2022, http://dx.doi.org/10.48550/ARXIV.2201.05072, URL https://arxiv.org/abs/2201.05072.

[51] H. Kang, P.B. Gibbons, G.E. Blelloch, L. Dhulipala, Y. Gu, C. McGuffey, The processing-in-memory model, in: Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 295–306, http://dx.doi.org/10.1145/3409964.3461816.

[52] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, X. Li, PIMSim: A flexible and detailed processing-in-memory simulator, IEEE Comput. Archit. Lett. 18 (1) (2019) 6–9, http://dx.doi.org/10.1109/LCA.2018.2885752.

[53] X. Xie, P. Gu, J. Huang, Y. Ding, Y. Xie, MPU-Sim: A simulator for in-DRAM near-bank processing architectures, IEEE Comput. Archit. Lett. 21 (1) (2022) 1–4, http://dx.doi.org/10.1109/LCA.2021.3135557.

[54] C. Yu, S. Liu, S. Khan, MultiPIM: A detailed and configurable multi-stack processing-in-memory simulator, IEEE Comput. Archit. Lett. 20 (1) (2021) 54–57, http://dx.doi.org/10.1109/LCA.2021.3061905.

[55] G.F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, O. Mutlu, DAMOV: A new methodology and benchmark suite for evaluating data movement bottlenecks, 2021, http://dx.doi.org/10.48550/ARXIV.2105.03725, URL https://arxiv.org/abs/2105.03725.

[56] D. Sanchez, C. Kozyrakis, Zsim: Fast and accurate microarchitectural simulation of thousand-core systems, SIGARCH Comput. Archit. News 41 (3) (2013) 475–486, http://dx.doi.org/10.1145/2508148.2485963.

[57] Y. Kim, W. Yang, O. Mutlu, Ramulator: A fast and extensible DRAM simulator, IEEE Comput. Archit. Lett. 15 (1) (2016) 45–49, http://dx.doi.org/10.1109/LCA.2015.2414456.

[58] S. Xu, X. Chen, X. Qian, Y. Han, TUPIM: A transparent and universal processing-in-memory architecture for unmodified binaries, in: Proceedings of the 2020 on Great Lakes Symposium on VLSI, in: GLSVLSI '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 199–204, http://dx.doi.org/10.1145/3386263.3406896.

[59] S. Xu, X. Chen, Y. Wang, Y. Han, X. Li, CuckooPIM: An efficient and less-blocking coherence mechanism for processing-in-memory systems, in: Proceedings of the 24th Asia and South Pacific Design Automation Conference, in: ASPDAC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 140–145, http://dx.doi.org/10.1145/3287624.3287646.

[60] G. Singh, L. Chelini, S. Corda, A.J. Awan, S. Stuijk, R. Jordans, H. Corporaal, A.-J. Boonstra, Near-memory computing: Past, present, and future, Microprocess. Microsyst. 71 (2019).

[61] A.J. Awan, V. Vlassov, M. Brorsson, E. Ayguade, Node architecture implications for in-memory data analytics on scale-in clusters, in: Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, in: BDCAT '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 237–246, http://dx.doi.org/10.1145/3006299.3006319.

[62] K. Hoste, L. Eeckhout, Microarchitecture-independent workload characterization, IEEE Micro 27 (3) (2007) 63–72, http://dx.doi.org/10.1109/MM.2007.56.

[63] M. Wei, M. Snir, J. Torrellas, R.B. Tremaine, A near-memory processor for vector, streaming and bit manipulation workloads, in: In the Second Watson Conference on Interaction Between Architecture, Circuits, and Compilers, 2005.

[64] A. Anghel, L.M. Vasilescu, R. Jongerius, G. Dittmann, G. Mariani, An instrumentation approach for hardware-agnostic software characterization, in: Proceedings of the 12th ACM International Conference on Computing Frontiers, in: CF '15, Association for Computing Machinery, New York, NY, USA, 2015, http://dx.doi.org/10.1145/2742854.2742859.

[65] R. Jongerius, A. Anghel, G. Dittmann, G. Mariani, E. Vermij, H. Corporaal, Analytic multi-core processor model for fast design-space exploration, IEEE Trans. Comput. 67 (6) (2018) 755–770, http://dx.doi.org/10.1109/TC.2017.2780239.

[66] L. Xu, D.P. Zhang, N. Jayasena, Scaling Deep Learning on Multiple In-Memory Processors, 2015.

[67] H. Choe, S. Lee, S. Park, S.J. Kim, E. Chung, S. Yoon, Near-data processing for machine learning, 2016, CoRR abs/1610.02273. arXiv:1610.02273. URL http://arxiv.org/abs/1610.02273.

[68] Y.-Y. Jo, S.-W. Kim, M. Chung, H. Oh, Data mining in intelligent SSD: Simulation-based evaluation, in: 2016 International Conference on Big Data and Smart Computing, BigComp, 2016, pp. 123–128, http://dx.doi.org/10.1109/BIGCOMP.2016.7425810.

[69] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, et al., The GEM5 simulator, ACM SIGARCH Comput. Archit. News 39 (2) (2011) 1–7.

[70] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, D. Ortega, COTSon: Infrastructure for full system simulation, SIGOPS Oper. Syst. Rev. 43 (1) (2009) 52–61, http://dx.doi.org/10.1145/1496909.1496921.

[71] B. Li, B. Yan, H. Li, An overview of in-memory processing with emerging non-volatile memory for data-intensive applications, in: H. Homayoun, B. Taskin, T. Mohsenin, W. Zhao (Eds.), Proceedings of the 2019 on Great Lakes Symposium on VLSI, GLSVLSI 2019, Tysons Corner, VA, USA, May 9-11, 2019, ACM, 2019, pp. 381–386, http://dx.doi.org/10.1145/3299874.3319452.

[72] S. Yu, X. Sun, X. Peng, S. Huang, Compute-in-memory with emerging nonvolatile-memories: Challenges and prospects, in: 2020 IEEE Custom Integrated Circuits Conference, CICC 2020, Boston, MA, USA, March 22-25, 2020, IEEE, 2020, pp. 1–4, http://dx.doi.org/10.1109/CICC48029.2020.9075887.

[73] C. Yu, S. Liu, S. Khan, MultiPIM: A detailed and configurable multi-stack processing-in-memory simulator, IEEE Comput. Archit. Lett. 20 (1) (2021) 54–57, http://dx.doi.org/10.1109/LCA.2021.3061905.

[74] S. Umesh, S. Mittal, A survey of spintronic architectures for processing-in-memory and neural networks, J. Syst. Archit. 97 (2019) 349–372, http://dx.doi.org/10.1016/j.sysarc.2018.11.005.