iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** V **Month of publication:** May 2024

DOI: https://doi.org/10.22214/ijraset.2024.62684

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue V May 2024- Available at www.ijraset.com

Processing-In-Memory Techniques: Survey, Advances, and Challenges

Maj Vipul Pathak
Faculty of Communication Engineering, Military College of Telecommunication Engineering, Street, Ambedkar Nagar, 453441, Madhya Pradesh, India

Abstract: ***Researchers studying computer architecture are paying close attention to processing-in-memory (PIM) techniques and a lot of time and money has been spent exploring and developing them. It is hoped that increasing the amount of research done on PIM approaches will help fulfill PIM's promise to eliminate or greatly minimize memory access bottleneck issues for memory-intensive applications. In order to uncover unresolved issues, empower the research community to make wise judgments, and modify future research trajectories, we also think it is critical to keep track of PIM research advancements. In this review, we examine recent research that investigated PIM methodologies, highlight the innovations, contrast contemporary PIM designs, and pinpoint the target application domains and appropriate memory technologies. We also talk about ideas that address problems with PIM designs that have not yet been solved, such as the translation and mapping of operands, workload analysis to find application segments that can be sped up with PIM, OS/runtime support, and coherency problems that need to be fixed before PIM can be used. We think that this work can be a helpful resource for researchers looking into PIM methods.***
Keywords: Novel and emerging memory technologies, Processing-in-memory, Near memory computing

I. INTRODUCTION

In practically all disciplines, traditional Von-Neumann architectures have been the predominant model for computing systems. These architectural designs have a distinct memory device to supply data to the processor units, and an I/O device to display the results. Since it has been shown to be effective, this concept has been incorporated into many modern computer architectures. Data movement to and from memory, however, has become an expensive operation in terms of time and energy with emerging applications [1]. Even with the use of highly effective cache memo- ries, this bottleneck arises when applications need a volume of data to be transferred from memory to computational units at a rate that cannot be maintained to give optimum performance and energy efficiency. In plain English, when an application is severely memory-constrained (for example, as a result of last-level cache misses), the time and energy required to fetch and move data from the memory to the computing unit is extremely inefficient. For example, performing a floating point operation on the processing unit after the data has arrived from DRAM requires about two orders of magnitude more energy than moving the data through the cache hierarchy [2]. Several things make this problem worse:

1) DRAM modules run at a significant lower frequency than the processing units;
2) Each application has very different memory requirements and access patterns;
3) Conventional memory systems (i.e., DRAM) typically reside on a DIMM connected to the processing units through a narrow bus, and these connections are limited by a finite number of pins. As a result, not all apps profit from the default setups.

Several methods have been used to address these issues, such as adding a hierarchy to the cache and using out-of-order processing to hide the cost of memory accesses. Investigations are also being done into unique combinations and new memory tech- nologies. However, the exponential development in memory demands for contemporary applications is outpacing these approaches, which has put the traditional design of computing systems under pressure. Instead of sending and receiving huge volumes of data to and from the computing unit, one method that departs from the Von-Neumann architecture is called "processing-in-memory" (PIM) [3]. PIM is not a particularly novel concept. However, the ever-increasing memory requirements in recent years have led to a great boom in research interest in this subject of contemporary applications, which has prompted scholars to think about a paradigm shift to approach the issue. Although PIM appears to be a generic term, it has a number of distinctive characteristics.

PIM can be applied at the sense amplifier/row buffer level (i.e., more complex operations), at the cell level of the data array (i.e., simple operations), or by using simple cores near memory banks that can execute a limited set of CPU instructions (for example, when processing a more substantial or memory-bound portion of the application). When these multiple PIM techniques are implemented using different memory technologies (e.g., DRAM, high-bandwidth memory [HBM], hybrid mem- ory cube [HMC], spin-transfer-torque magnetic RAM [STT-MRAM], resistive RAM [ReRAM], and phase change memory [PCM]), additional design complexity results. Non-volatile SRAM designs have also made it possible to perform in-memory comput- ing among alternative memory technologies [4]. In order to perform at their best, PIM units can also need to be developed for a particular application field. For instance, specific computation patterns for deep neural network (DNN) applications may be optimised by employing ReRAM's distinctive data-array format. In order to compre- hend how far PIM has come and how it is changing, we analyse all the aforementioned components of current research that evaluate PIM in this article. We also discuss the many evaluation methodologies employed by researchers to model PIM designs, as well as the difficulties and chances that this project may face in the future.

II. MEMORY TECHNOLOGIES

We first provide a quick overview of the design and structure of the primary memory technologies that typically arise in research and commercial designs because this article covers PIM as it relates to a variety of memory technologies.

A. DRAM

The most extensively used and implemented commodity memory technology for con- temporary computer systems is DRAM, or dynamic random access memory. DRAM is the most cheap main memory technology and was developed over decades of steadfast development. Memory controller, memory bus, and DRAM devices arranged in dual- in-line memory modules, or DIMMs, make up the three basic components of DRAM memory systems. The majority of contemporary CPUs feature built-in memory con- trollers and memory channels connected to DRAM DIMMs for data, command, and address transmission. DRAMs often have their own specific slots on the motherboard, making it simple to repair or replace them.

Word-lines and bit-lines connecting these DRAM cells together form an array structure. (Fig. 1). When a row in this array is activated, the data from that row is sent to the row buffer, from which the CPU can receive selected data via the bus. DRAM is the de-facto standard for main memory systems, therefore understanding its commands and timing is necessary to fully grasp the modifications and adjustments needed for the newest memory technologies and protocols. Data from DRAM must be brought to the row buffer by providing an ACT command to activate the specified row address. The memory controller must then send a READ command with the col- umn address on the row it needs to read in order to execute a read operation. These row/column addresses choose the data, which is subsequently loaded onto the data bus for transmission. When data is put into the row buffer, the DRAM read is self- destructive, clearing the row's contents; as a result, the data must be written back to the row when the row is closed. To perform this process, a PRECHARGE surgery is required. Fig. 2 illustrates the timing restrictions that DRAM must uphold between commands it issues (not to scale). To continue to be the dominant memory technology, DRAM memory technology must overcome a number of obstacles. DRAM capacitors must get smaller during extreme scaling in order to put more of them into the same amount of space. This makes them more prone to errors. Additionally, due to its pla- nar architecture, DRAM is restricted by the few physical pins available and is unable to support the bandwidth demands of contemporary applications.

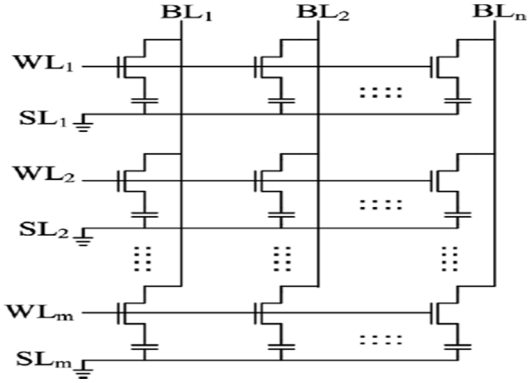


Fig. 1 DRAM Array.

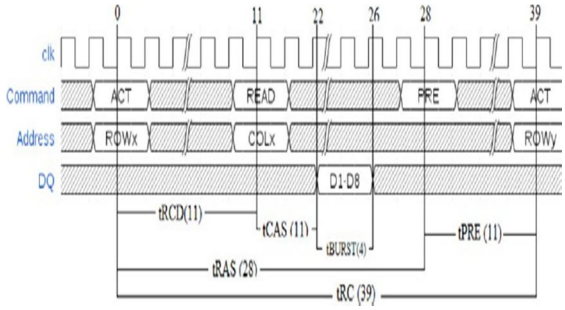


Fig. 2 DRAM Read Timing.

B. HBM

The 3D-stacked DRAMs were created to meet the growing bandwidth requirements of contemporary applications. Multiple DRAM dies are placed on a base logic layer in the HBM variation of 3D-stacked DRAM [5]. Typically, a DRAM die has two independent channels that can be split into two pseudo-channels (HBM2) on either side. A 1024- bit connection to the processor unit is made possible by the ability to attach several HBM stacks to it while they are on the same silicon interposer. Despite being primarily based on DRAM chips, HBM may reach substantially better bandwidth and capacity when several stacks are employed thanks to the way it is organised and the broader data connection it uses (Fig. 3).

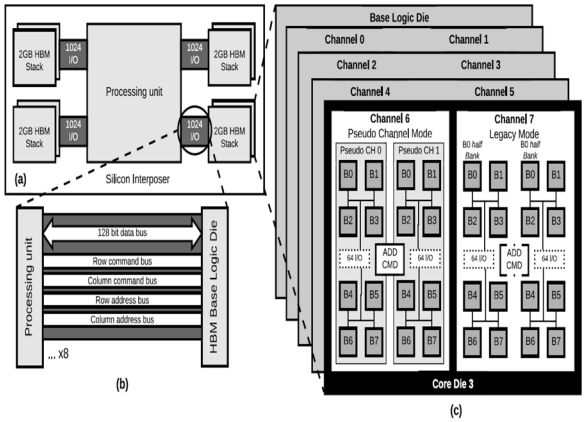


Fig. 3 (a) General organization of HBM stacks with a processing unit. (b) Per-channel data/bus connections with processing unit via interconnect circuitry (e.g., memory controller). (c) Internal structure of a 4-die HBM stack with arrangements of banks for pseudo channel mode (Channel 6) and legacy mode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

C. HMC

Another DRAM variant that uses 3D-stacking of DRAM dies is HMC.[6]. Each DRAM die in HMC is dispersed in partitions that are vertically coupled with other parti- tions of neighbouring dies, creating a vault that is managed by a memory controller that is housed in the logic base layer, in contrast to HBM (Fig. 4). A high-speed seri- alization/deserialization circuit is used to construct a packet-based communication protocol that links vault controllers to other HMCs or host devices.

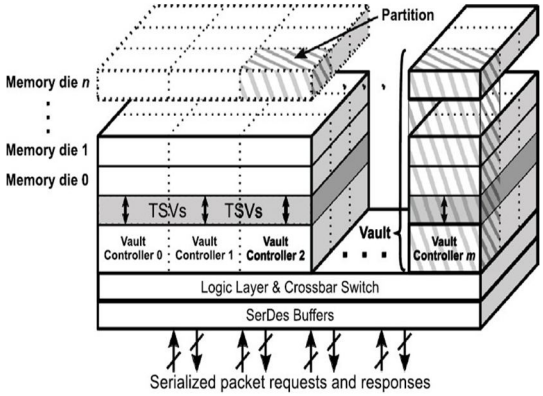


Fig. 4 HMC or hybrid memory cube

D. STT-MRAM

A new non-volatile memory device called STT-MRAM is based on the magneto- resistance that spin-polarized current causes [7]. A magnetic tunnelling junction (MTJ), which is made up of a thin tunnelling electronic layer sandwiched between two ferromagnetic layers, is the central component of STT-MRAM storage and pro- grammability. One of these layers has a constant magnetization, while the other can fluctuate depending on how much current is applied to it. The MTJ produces negligi- ble resistance when both layers are equally polarised, which is logical 0. On the other hand, the MJT indicates a high-resistance state and a logical 1 when the layers have different polarities. According to Fig. 5 the cell array structure of STT-MRAM and DRAM may be strikingly similar. [8].

E. ReRam

Another non-volatile memory technology that might be used instead is ReRAM. A metal-insulator-metal structure is included in a two-terminal device known as a ReRAM cell. By forming or dissolving a conductive filament that serves as the metal oxide insulator, these cells can achieve either a low- or high-resistance state. ReRAM cells provide the following three main operations: SET, RESET, and READ. The low- resistance state is activated by the SET operation, and the cell is changed to the high-resistance state by the RESET operation. A tiny sensing voltage that has no effect on the cell's resistance is applied during a READ operation to determine the current resistance state of the cell. [9].

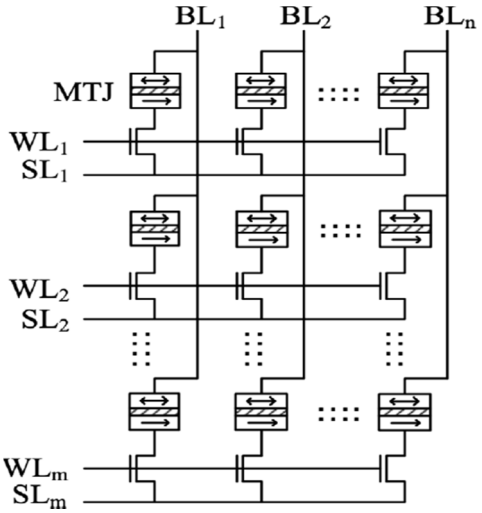


Fig. 5 A cell array of STT-MRAM or spin-transfer-torque magnetic RAM

F. PCM

Another new non-volatile memory that has drawn a lot of interest as an alternative memory technology is PCM. Two electrodes are separated from one another in a PCM cell by a phase-change material, such as Ge2Sb2Te5. These materials can have either crystalline or amorphous electrical resistance, which suggests that they might be a good fit for a memory technology. PCM cells have the same three primary operations as ReRAM: SET (to 1), RESET (to 0), and READ [10]. PCM is a poor choice as the primary memory technology for a high-performance computer system because, despite its comparatively quick READ and RE-SET operations and slow SET operations. Additionally, PCM cells may not be a viable option for corporate systems due to their poor endurance.

III. DETAILED EXAMINATION OF PIM LITERATURE

Investigations into various PIM variations have increased dramatically. We analyse recent works in this area in the section below. Design type, application field, memory technology, and assessment technique make up our four categories for categorising our evaluation of PIM research.

A. Design Type

We go over the three primary PIM design types in this part, which are categorised according to where the processing happens in memory.

1) Data Array Processing

According to several studies, data-array level computation capabilities should be implemented [11]–[12]. Generally speaking, these research look into and test the viability of using cell-level structures to add certain computational capabilities. A coprocessor based on STT-MRAM is suggested by Sun et al. [11] for convolutional neural network (CNN) acceleration. The implementation of a 22 nm advanced tech- nology node on CMOS, SRAM, and STT-MRAM is also reported by the authors. The proposed CNN processing block utilises input from the input buffer and filter coeffi- cients from a co-located on-chip MRAM memory to conduct 3 3 convolution on a 2D picture at P P pixel positions. Imani et al.'s [13] use of the conventional crossbar mem- ory's analogue properties enables crucial memory operations. This technique supports bit-wise operations internally in memory without reading the values out of the block, in contrast to earlier efforts that computed bit-wise operations on the sense ampli- fier of each memory block. The authors claim that by using row-parallel computation, 1000 concurrent additions and multiplications on 1000 rows of memory were accom- plished. By replacing the Wallace tree with a carry-save-add-shift (CSAS) multiplier, introducing a unique full-adder architecture, and applying novel partition-based com- putation techniques for broadcasting/shifting data around partitions, Leitersdorf et al. [14] propose to speed up in-memory multiplication. The voltage-controlled change- able resistance property of ReRAM (memristive crossbar) is also utilised in this work to implement logic gates (such as NOT, NOR, OR, and NAND). In order to maximise the reuse of weight and input data on an 8-bit ReRAM-based PIM architecture, Peng et al. [15] offer a unique mapping mechanism and dataflow. The concept is simulated with NeuroSim simulator [16] on the ResNet-34 benchmark, the findings show that the suggested solutions save 90Another ReRAM-based PIM architecture designed specifi- cally for recurrent neural network (RNN) acceleration is shown by Long et al. [17] (Fig. 6). The authors suggest an in-memory processing unit with three primary subarrays: multiplier subarrays for element-wise operations, crossbar subarrays for matrix-vector multiplication, and special function units for nonlinear functions. According to reports, the suggested methods improve on the GPU baseline by an average of 79. In order to accommodate CNNs running on prefabricated chips, Lu et al. [18] investigate reconfig- urable design techniques for compute in memory (CIM)-based accelerators. A modified DNN + NeuroSim framework running a system-level performance benchmark is used by the authors to assess the notion. By using FeFET multibit content addressable memories for associative search and ferroelectric field effect transistor (FeFET) cross- bar arrays for multiply-and-add, Kazemi et al. [19] introduce multibit in-memory hyperdimensional computing (HDC) inference.

2) Row Buffer Processing

There aren't many research that suggest low-level methods to increase processing capacity near or inside row buffers or sense amplifiers. With very minor adjustments to the DRAM subarrays, Roy et al. [20] offer a novel multiplication strategy inside DRAM at the subarray level. The study suggests a quick, light-weight, bit-wise, in-subarray

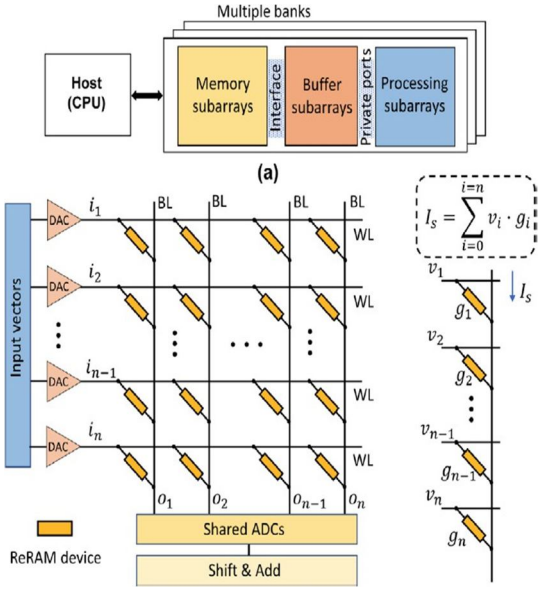


Fig. 6 PIM architecture and ReRAM crossbar for matrix–vector multiplication

AND operation in DRAM to decrease the overall cost of multiplication because addi- tion and AND operations are used to perform the multiplication. For effective machine learning (ML) acceleration, they implement a new PIM-DRAM bank architecture with adder trees and nonlinear activation function units in each DRAM bank, as shown in Fig. 7. ReLU, batch-normalization, and pooling operations—which are necessary for a variety of ML models—are supported by the suggested architecture. In-DRAM multi- plication is divided into AND and ADD operations that each use nine compute rows in the proposed PIM operations, and a bit-wise AND operation that uses two additional rows in the DRAM subarray is also included. Each DNN layer in the suggested data mapping is assigned to a DRAM bank. The outermost loop of the mapping algorithm traverses every layer of the neural network. A FeFET-based PIM architecture is pre- sented by Long et al. [21] to speed up the inference of DNNs. The FeFET cross-bar is used in this study to enable bit-parallel computation and do away with analog-to- digital conversion in previous mixed-signal PIM systems. This work also presents a digital in-memory vector-matrix multiplication engine design. For input broadcasting and on-the-fly partial results processing, a specialised hierarchical network-on-chip is created, lowering the volume and latency of data transfer.

Simulations of a 28 nm CMOS technology demonstrate a 115 times (gigaflops/W) increase in processing efficiency over a desktop GPU (NVIDIA GTX 1080 Ti) and a 6.3 times increase over a ReRAM-based solution. A matrix-vector multiplication experimental design with 16-cycle MAC (multiply-and-accumulate) units and 8-cycle reducers is presented by Lee et al. [22] on the basis of HBM2. According to the study, scheduling across all banks and inside each bank saw performance improvements of406% and 35.2%, respectively. Other studies [23]–[24] offer creative PIM methods to quicken neural networks.

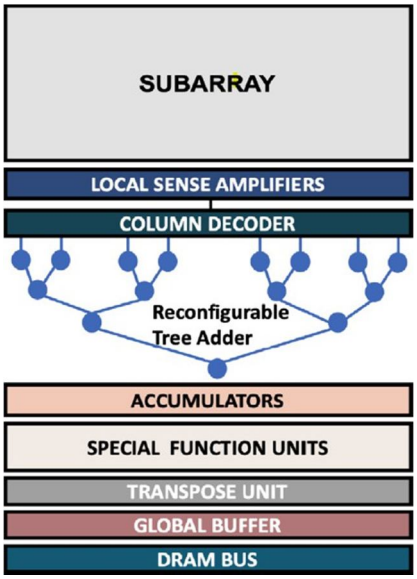


Fig. 7 PIM-DRAM bank architecture

3) Unit Near Memory bank Processing

In order to limit memory traffic to and from the processing unit, the bulk of the papers we review suggest a near-memory compute unit that implements a simple core to execute a select few CPU instructions [1],[2],[25]-[26]. According to Olgun et al. [25], some processing-using-memory (PuM) methods call for unique memory alloca- tion and alignment strategies that are not offered by the memory allocation primitives that are currently in use. Additionally, because in-DRAM copy operations need to handle memory coherence well, it is challenging to analyse PuM approaches on pri- vate computing systems or simulators. The study creates an FPGA-based prototype to illustrate end-to-end integration and evaluation of PuM methods utilising actual DRAM chips in order to address the issue. In order to achieve this, the proposal incor- porates two hardware elements: a custom memory controller that provides refresh, scheduling, and timing used for DDRx sequences that trigger PuM operations, and a PuM operations controller that functions as a memory-mapped module between the application and the memory controller. The PuM operations library (pumolib), which makes PuM operations available to application developers, and a special supervisor programme that provides the required OS primitives (such as virtual memory manage- ment, memory allocation, and alignment), are examples of software components.

The high-performance, near-memory neural network accelerator architecture that makes use of the logic die in 3D HBM-like memory is covered by Park et al. [27] (Fig. 8). The first goal was to pinpoint the main distinctions between HBM and HMC in terms of near-memory neural network accelerator design because the majority of the pre- viously reported 3D memory-based, near-memory neural network accelerator designs utilised HMC memory. The study presents group-wise broadcasting and round-robin data fetching systems to take use of the centralised through-silicon-via (TSV) channels. The article demonstrates how an efficient data-fetching technique for neural network accelerators in HMC differs from an efficient scheme designed to fetch data from the DRAM dies to the neural network accelerator on the logic die in HBM. To be more precise, the DRAM process is used to implement the bottom buffer die of the current generation HBM. It is assumed that the bottom die of HBM is implemented in the logic process in this study since a logic process is required to create a high-performance neural network accelerator. The authors suggest that (1) round-robin data fetching is more efficient than conventional distributed data fetching for moving data from DRAM stacks to neural network engines on the logic die in HBM, and (2) increasing bit width in conventional architectures using a multicast scheme results in significant routing overhead for connecting wide I/Os to all processing elements. On the other hand, the suggested design can do away with the routing overhead by employing a group-wise broadcast scheme with preset and grouped interconnects. Kwon et al. [28] show how system architects face new difficulties as deep learning models and datasets grow in size. One of these difficulties is the memory capacity bottleneck, where the amount of physical memory available inside the accelerator device limits the algorithms that can be researched. The paper suggests a memory-centric deep learning system that can quickly communicate between devices for parallel training and transparently increase the memory capacity made accessible to the accelerators. This idea assembles a pool of memory modules that are separated from the host interface and serve as a means of transparent memory capacity extension locally within the device-side interconnect. With this technology, eight deep learning applications average a 2.8x speedup over conventional computers, while the total system memory capacity is increased to tens of TBs. As a clean alternative to conventional DRAM, Lee et al.'s [2] revolutionary PIM architecture can operate with unaltered commercial processors. The authors pro- posed 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 unal- tered commercial processor, and created the necessary software stack to demonstrate its viability and effectiveness at the system level. The PIM architecture is made up of the following components: a PIM-HBM DRAM die; a bank connected to a PIM exe- cution unit that includes a single instruction multiple data (SIMD) floating-point unit (FPU), command register file (CRF), general register file (GRF), and scalar register file (SRF); and c) the PIM execution unit's data path (Fig. 9). (1) When operating in PIM mode, PIM execution units from every bank simultaneously reply to a nor- mal DRAM column command (such as READ or WRITE) from the host processor by carrying out one wide-SIMD operation ordered by a PIM instruction in a lock-step manner with deterministic latency. Three parts make up a PIM execution unit: a 16- wide SIMD FPU, register files, and a controller (Fig. 10). To accommodate the DRAM internal timing for reading/writing data, the PIM execution unit is separated into up to five pipeline phases. An instruction from a PIM is fetched and decoded in the first stage. In the second step, 256-bit data is loaded from either the EVEN BANK or the ODD BANK to a GRF or an input of the SIMD FPU. MULT is the third stage, and ADD is the fourth. MULT skips the fourth step, while ADD skips the third, however the MAC goes through both the third and fourth stages. The final step writes the outcome to a GRF.

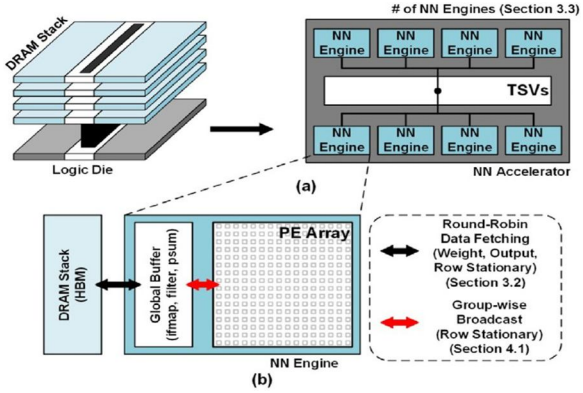


Fig. 8 Overall architecture proposed by [27]. (a) Neural network accelerator on the HBM logic die. (b) Neural network engine with the data-fetching scheme proposed in the study.

Three crucial areas are examined by Ghose et al. [29] for the effective creation and broad use of PIM structures. They begin by outlining their work on methodically finding PIM prospects in practical applications and quantifying possible benefits for well-liked developing applications (including ML, data analytics, and genomic analy- sis). Second, they seek to address a number of crucial problems associated with writing these applications for PIM structures. Third, they discuss the obstacles still standing in the way of the widespread use of PIM.

A function is a PIM target candidate in a consumer device if it meets the following criteria: (1) it consumes the most energy out of all functions in the workload, as energy reduction is a top priority in consumer workloads; (2) to maximise the potential energy benefits of offloading to PIM, its data movement consumes a significant portion (i.e.,¿20%) of the workload energy; (3) it is memory-intensive (i.e., its last-level cache misses per kilo. The paper finds four major problems that have an impact on how programmatable PIM architectures are: The different granularities of an offloaded PIM kernel, how to handle data sharing between PIM kernels and CPU threads, how to effectively give PIM kernels access to crucial virtual memory address translation mechanisms, and how to automatically identify and offload PIM targets (i.e., parts of an application that are suitable for PIM) are just a few of the topics covered in this paper. With the use of heterogeneous PIM hardware and software co-design, Huang et al. [30] hope to achieve energy-efficient graph processing, which is a goal of several articles that offer near-memory processing strategies for accelerating neural networks [27],[31],[32].

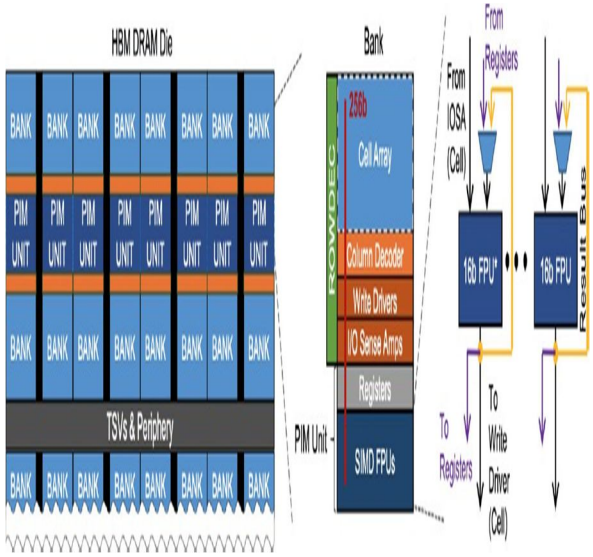


Fig. 9 HBM DRAM die organization with PIM unit and its data path

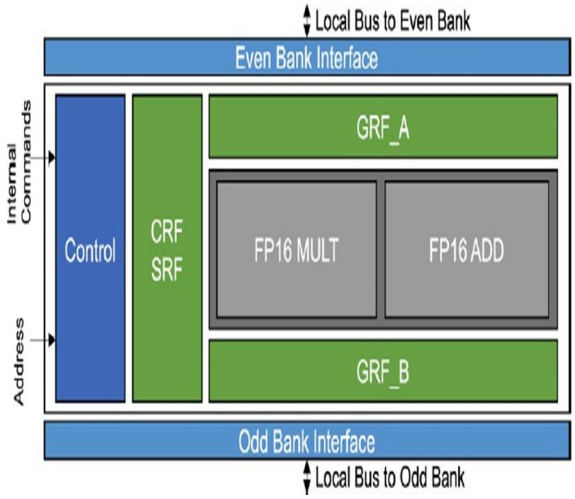


Fig. 10 Proposed microarchitecture of PIM execution unit with control, register, and single instruc- tion multiple data (SIMD) units [2]

B. Application Field

Recent PIM research focuses on efforts to leverage PIM to speed up application segments in the fields of machine learning, artificial intelligence, and neural networks [11],[15]–[18], [20],[21]–[24],[33]–[34],[31],[32]. The coefficients can be intuitively trans- lated to the word lines and bit lines and used to compute the dot product in each cell using the data array because applications from these areas typically use matrix-vector multiplication operations [15],[17]. After that, the coefficients can be accumulated along the source lines. Another method involves placing processing elements directly beneath the DRAM dies with a shorter distance to the coefficient data for faster computation in each layer [27],[32] using the logic die of 3D-stacked memory. Accord- ing to Imani et al. [13], ML acceleration with PIM enabled graph processing and query processing capabilities. To meet the heterogeneity needs of graph applications, Huang et al. [30] offer a heterogeneous PIM design that incorporates memristors and CMOS-based technologies.

C. Memory Technology

When using specific memory technologies with various PIM systems, several intrigu- ing insights can be made. Since the data-array format of this memory is particularly well-suited for such implementations, the majority of studies that advocate process- ing in a DA employ it (for example, ReRAM) [11]-[14],[12],[22],[35],[30]. Contrarily, research that expands on the notion of placing a processing unit next to memory pri- marily uses 3D-stacked memories (such as HBM and HMC) to benefit from the extra logic layer [2],[27],[36],[37],[38],[26],[39]. Few research [20],[24]-[40],[34],[41],[42] offer
PIM optimisations on commodity DRAM technology, while three studies [1],[43],[44] concentrate their designs on UPMEM, a commercially available DRAM-based DIMM with integrated computation capability.

D. Evaluation Methodology

Different strategies are being used by researchers to develop and test PIM techniques. We divide these techniques into three groups: hardware implementations, simulation models, and analytical models.

1) Analytical Model

For the purpose of designing and analysing parallel algorithms based on PIM, a theoretical model was put forth [45]. The suggested approach combines a PIM side made up of local memory and a processor core with a CPU side made up of par- allel cores with quick access to shared memory. Additionally, it suggests common parallel complexity metrics for both distributed memory computing and computing with shared memory. 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)—the proposed model is assessed.

2) Simulation Model

Since there is no real, commercially accessible PIM hardware available to the researchers, simulators were essential in evaluating and advancing the search for PIM-based systems.
Ghose et al.'s evaluation of the effective in-memory pointer chasing accelerator, which can handle address translation fully within DRAM, uses DRAMSim2 and a GEM5 full-system simulator. Accurate memory modelling and DRAM energy analysis are provided by DRAM- Sim2. In order to speed up matrix-vector operations in ML workloads, Roy et al. [20] propose a DRAM-based PIM multiplication primitive and tested DNNs employing these operations on HSPICE circuit simulations. The analogue circuit simulator's use in these studies is not entirely evident, though.
A full-stack simulation infrastructure is suggested by Zhou et al. [46] to explore the design space of digital PIM. This infrastructure includes a software library, a layer- based compiler that is flexible, and an accurate and quick architecture model that supports PIM. The proposed simulator, according to the authors, offers 10.3 times faster simulation with a 6.3Processing-in-memory simulator (PIMSIM) is presented by Xu et al. [47]. It has a partitioner at the front end that can identify and distribute PIM instructions. In order to decide if a PIM instruction should be executed in memory, it also offers dynamic feedback support. Fast simulation, instrument-driven simulation, and full system simulation are the three simulation modes that the simulator provides. The NeuroSim simulator is created and introduced by Lu et al. [16] for CIM hard- ware simulation. The size, latency, dynamic energy, and leaky power consumption of the hardware performance can all be estimated by NeuroSim.

The peripheral circuit modules (e.g., decoders, switch matrix, MUX [multiplexer], adders) are validated with SPICE simulations as opposed to actual silicon data from a 40 nm 16 kb CIM macro that uses TSMC's 40 nm RRAM process. For general-purpose near-bank processing architectures, Xie et al.'s MPU-Sim [48] proposal models several MPU cores connected by on-chip network links inside a processor. On DRAM dies, each MPU core con- tains a number of near-bank processing units. The SIMT (single instruction, multiple threads) programming approach is supported by MPU-Sim in order to take advantage of large bank-level parallelism and overcome the issue of control logic and commu- nication overheads. ZSim [49] (a system simulator) and Ramulator [50] (a memory simulator) serve as the foundation for the simulation infrastructures MultiPim [51] and DAMOV [52]. Both simulators allow for kernel offloading to PIM units. While DAMOV performs an extensive workload evaluation to pinpoint data-movement bot- tlenecks across a variety of applications and functions, Mul- tiPim uses a multistack connection, crossbar switches, PIM core coherence, and virtual memory.

3) Hardware Implementation

A chiplet-based PIM design technique is proposed by Jaio et al. [33] and is evaluated with a Tiny-Yolo DNN running on an FPGA. The specifics of the FPGA are not pro- vided, though. The workload of a monolithic accelerator is divided into a multichiplet pipeline using the suggested layer-wise strategy. PimCaffe [34] creates an FPGA plat- form that emulates the PIM and has SIMD and systolic array computing engines that can multiply vectors and matrices on the PIM device. Fig. 11 displays a high-level block diagram of the suggested concept. A 20 nm DRAM-fabricated industrial PIM prototype is offered by Samsung [2]. This PIM execution unit has sixteen 16-bit SIMD lanes, each of which has a 32-entry command register, a 16-entry general register, and sixteen 16-entry scalar registers.

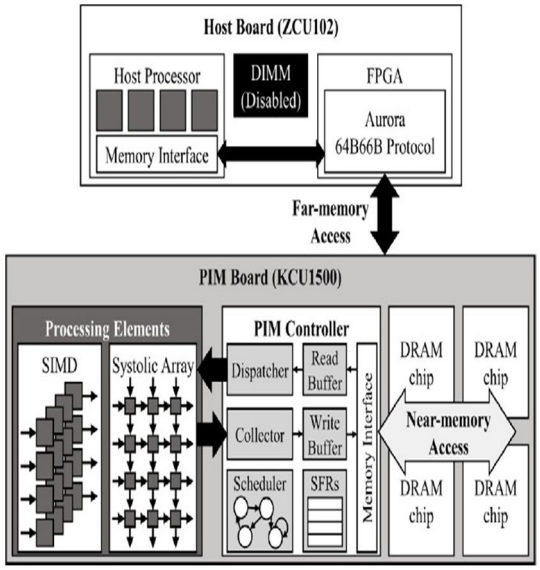


Fig. 11 PIMCaffe system architecture proposed [34]

Another PIM-based technology in commercial production is UPMEM [1]. An UPMEM-based PIM system with a host CPU, typical DRAM main memory, and main memory that supports PIM is shown in Fig. 12. On a conventional DDR4 DIMM with many PIM chips, a UPMEM module is based. A 64 MB DRAM bank, 24 KB of instruction memory, and 64 KB of scratchpad memory are all accessible to each of the 8 DRAM processing units (DPUs) that make up a PIM chip. The host CPU can transfer input data from main memory and retrieve results using the 64 MB DRAM banks.

IV. CHALLENGES

In order to ensure that PIM can be used as a stand-alone and effective solution, we identify the areas that need to be further developed through the literature review. The majority of the studies reviewed concentrate on just one aspect of the PIM deployment, leaving important questions unaddressed. No matter how exciting PIM may sound, there are a number of implementation-related difficulties. Before PIM may be used as

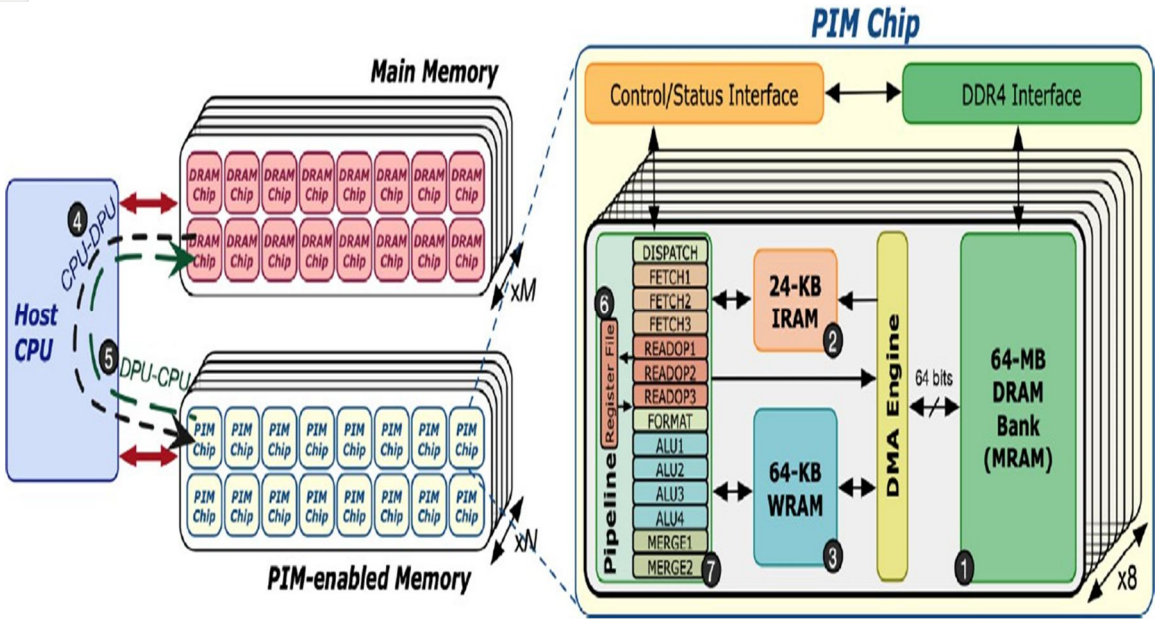


Fig. 12 Architecture and organization of an UPMEM-based PIM system [1]

a universal replacement or addition to conventional memory technology, these difficul- ties must be overcome. Although PIM is predicted to speed up some memory-bound operations, the CPU will still be used for the majority of the calculation. Therefore, overhead expenses will still be incurred when classifying activities for the main process- ing unit or the PIM execution unit. Additionally, the execution units in/near memory are incapable of running programmes that use a range of instructions because they are designed to have a simple core and support a small number of instructions. Runtime and OS support are required for the efficient allocation of programme sections to be processed in memory. It would also be necessary to make considerable changes to the design of the memory controller. Therefore, we categorise the key obstacles to adopt- ing PIM as workload classification and analysis, address translation, OS and runtime support, compiler support, programming model support, coherency and consistency, and address translation and mapping. Below, these topics are covered in more detail.

A. Workload Analysis and Classification

We must comprehend and categorise the probable candidate applications and functions in order to better understand the potential utilisation and anticipated acceleration of PIM execution. Generally speaking, application segments with poor locality and a higher last-level cache miss rate make suitable PIM candidates. In order to determine whether candidate functions are suitable to be executed on a processing unit close to memory, Oliveira et al. [52] present a thorough methodology for workload character- ization. The authors divide candidate functions into six categories based on different combinations of temporal locality, arithmetic intensity, last-to-first-level cache miss ratio (LFMR), and MPKI. The study finds that this strategy can be advantageous for five categories, with the exception of the one where functions exhibit high temporal localization, low LFMR, and high arithmetic intensity.

B. Address Translation and Mapping

The mapping and ad- hoc translation procedure of conventional memory systems must be revised in order to implement PIM approaches. Olgun et al. [25] contend that in order to support PIM approaches, data mapping and allocation criteria must be increased. Source and destination operands of COPY should be located in the same DRAM subarray, according to the authors' PIM design. They refer to this as a mapping problem and suggest a special supervisor software to handle it. By allocating memory for PIM operations at booting, Lee et al. [2] reduce the expense of virtual-physical address translation. Additionally, they placed this reserved memory space in an area that cannot be cached, causing the PIM device driver to allocate physically adjacent memory blocks when the host processor sends a DRAM signal for memory access to the PIM memory space.

C. OS and Runtime Support

For the redesigned memory controller necessary for PIM to perform correctly and efficiently, PIM inclusion needs OS and runtime support [38]. Ahn et al.'s [36] simplified hardware arrangement is designed to monitor the locality of data accessible by PIM- enabled instructions during runtime and decide whether the host processor or the PIM unit should be used to execute the instruction. According to Wang et al. [32], a runtime based technique can identify the mapping of parallelism onto the appropriate hardware for the predictable behaviour of convolutional connections. The UPMEM runtime library is used by Gómez-Luna et al. [1] to handle library calls to transfer instructions between various memory (such as MRAM and IRAM) inside PIM units. PIM-friendly instructions can be automatically extracted by TUPIM [53] and offloaded to memory at runtime. TUPIM chooses PIM-friendly instructions that, among other things, limit the amount of time that data must travel between on-chip caches and main memories, are poorly served by the caches, and are repeatedly carried out. According to the study, as compared to CPU-only execution, the suggested improvements provide an average speedup of 2.2 and a 15.7% energy reduction.

D. Compiler and Programming Models

According to Olgun et al. [25], the RISC-V GNU compiler tool chain has been modified such that C/C++ applications can access the special instruction CFLUSH. To keep coherence, dirty cache blocks that are physically referenced are flushed via CFLUSH. Ghose et al. [40] suggest utilising the two macros #PIM begin and #PIM end to offload sections of the code to PIM cores. These macros are transformed by the compiler into the instruction that is added to the ISA to start and stop PIM execution.

E. Coherency and consistency

Coherence management is cited by Olgun et al. [25] as a PIM challenge. Caches are used in conventional systems to store copies of data in main memory for quick access to frequently used data. It becomes difficult for PIM since the STORE operation updates the cache data but does not instantly update the data in main memory. The authors address the memory coherence issue in a minimally invasive way by implementing a new custom RISCV instruction called CFLUSH to flush physically referenced dirty cache blocks. In order to preserve cache coherence between PIM processing logic and CPU cores without having to issue coherence requests for each memory access, Ghose et al. [40] offer the LazyPIM technique. Instead, PIM processing logic speculatively acquires coherence permissions and then instructs the processing unit to perform batch coherence look-ups to check whether its speculative permission acquisition violates the programming model's defined memory ordering. Before sending the PIM operation to memory, Ahn et al. [36] invoke back- invalidation and back-write-back for the requested cache blocks to the last-level cache. By using this method, a stale copy of the data cannot exist in main memory or in on-chip caches before or after a PIM transaction. A novel, variable-grained coherence approach is proposed by CuckooPIM [54] that strategically assigns data ownership while dynamically monitoring system behaviour. It does not significantly increase the complexity of combinational logic in a PIM system.

V. PIM – PREVIOUS SURVEYS

With a focus on processing at the location of the data, Gagandeep et al.'s [55] extensive examination of near-memory computing architectures can help to reduce the problem of data transportation in data-intensive processes. The authors review a large amount of research on the granularity of applications for near-memory computing units and the restricted characteristics of the memory level at which this paradigm is imple- mented in the literature up to 2018. The paper eliminates CIM analysis and restricts its examination to near-memory computing. The literature on near-memory computing's state-of-the-art is categorised by authors into five primary groups: memory, process- ing, assessment technique, interoperability, and target application domain. They also take into account the hierarchy, kind, and integration of memory. The authors take into account various host compute units (such as CPU, GPU, CGRA, FPGA, and accelerator), implementation of near-memory compute logic (such as programmable, fixed function, and reconfig- urable), and granularity (such as instruction level, kernel level, and application level). The authors divide the literature into analytical, simula- tion, and prototype/hardware categories for the evaluation category. The authors take into account virtual memory support, cache coherence mechanisms, and programmable interface support when discussing interoperability. In order to comprehend and assess the design space for near-memory systems, authors also review relevant literature.

Research on workload characterization based design space exploration methods that are dependent on microarchitecture [56] and independent [57]–[58] is also taken into consideration. Additionally, the literature discloses the evaluation techniques utilised in cutting-edge publications, such as analytical modeling-based approaches [59],[60] and simulation-based modelling approaches [61]-[62]. The case study contrasts a CPU- centric multicore system with regular DDR3 main memory with a data-centric strategy that uses 3D-stacked memory that resembles HMC in place of traditional DDR3. The authors come to the conclusion that while high locality applications can gain more from the traditional approach, low locality applications should make use of the near-memory approach. Three popular eNVM technologies—STT-MRAM, PCM, and ReRAM—have made progress in the development of in-memory processing, according to Li et al. [63]. According to memory type, hierarchy position, design level (device, cir- cuit, or system), function type (logic, arithmetic, associative, vector, or matrix-vector multiplication), and application group, the authors categorise the investigations. Stud- ies that investigate CIM with developing non-volatile memory are also described by Yu et al. [64]. The authors concentrate especially on deep learning prototype chips that monolithically integrate eNVMs with CMOS peripheral. The article illustrates how low on-state resistance of most eNVM technologies may influence analogue read out accuracy and provides a description of an RRAM CIM macro [65]. In order to pre- vent voltage drop, the analogue MUX at the end of the column must also be greatly enlarged up. The write voltage needs, which are higher for eNVMs like RRAM and PCM, present additional technical problems. Spintronic designs for PIM are the main focus of Mittal et al.'s [66] survey, which aims to process neural networks. The organ- isation of the suggested PIM accelerator designs, the sort of spintronic technologies employed, and the PIM operations supported by these devices are all identified by the authors. The study comes to the conclusion that spintronic memories, conven- tional memories, and compute-centric architectures are all unable to handle the major AI concerns. By categorising the studies in a novel way according to PIM category, application fields, memory technology, and evaluation models and discussing unre- solved challenges on the path forward, our study complements previous survey papers with the most recent advancements and developments of memory technologies and techniques.

VI. CONCLUSION

We have compiled the most recent PIM advancements in this study. In order to com- prehend how PIM strategies have developed as of this writing, we explain technologies that PIM techniques typically build on and analyse research articles, technical papers, and industrial product information. We outline the most popular PIM architectures put forth by the community, the application areas that profit from PIM activities, and the assessment procedures utilised while running PIM experiments. We also go over ways to deal with issues like address translation and operand mapping, workload analysis to find application segments that can be accelerated with PIM, OS and run- time support required to implement PIM, and coherency problems for PIM. We think that this poll will be a helpful resource for future research on PIM methods.

VII. DECLARATION OF COMPETING INTEREST

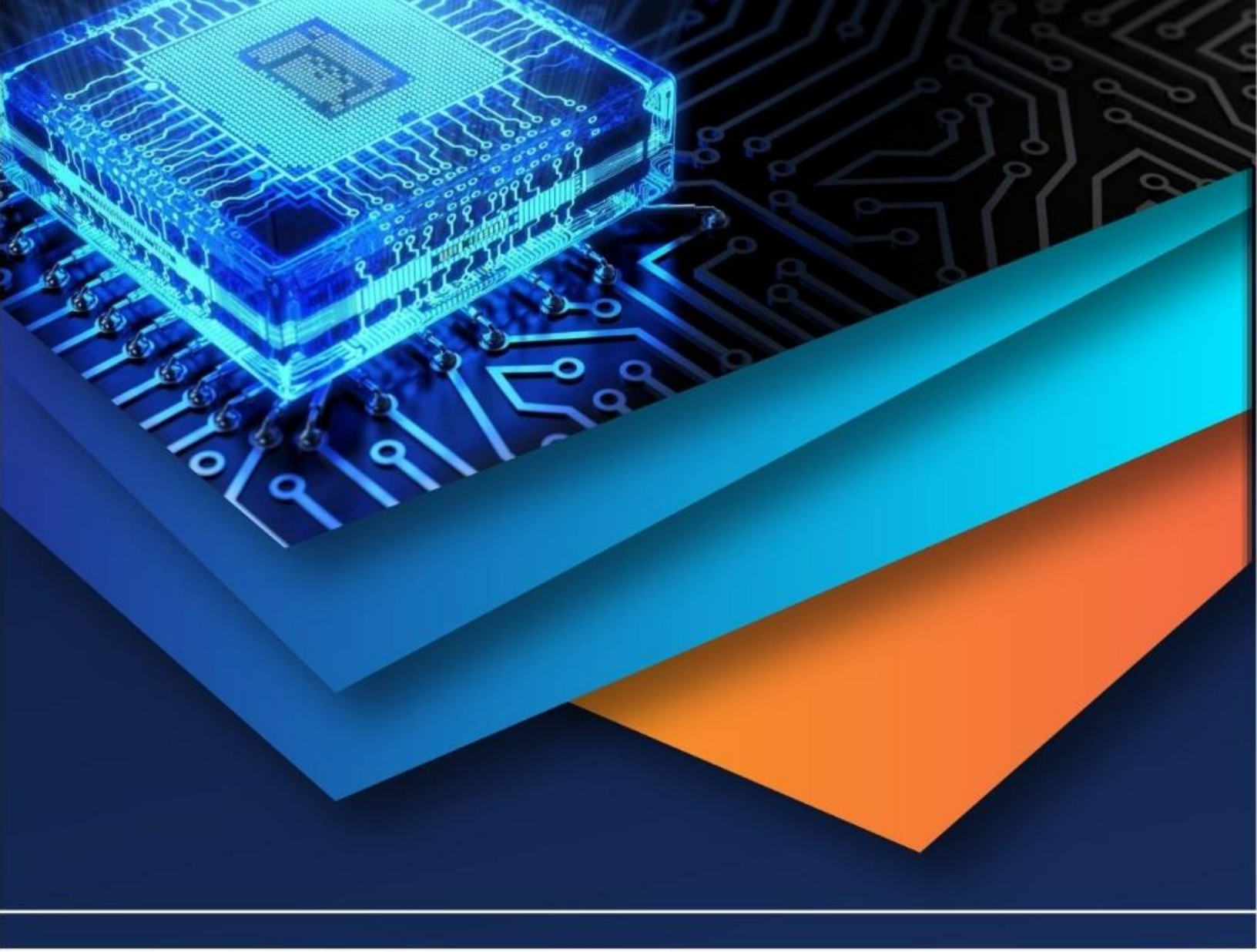
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

[1] G´omez-Luna, J., Hajj, I.E., Fernandez, I., Giannoula, C., Oliveira, G.F., Mutlu, O.: Benchmarking a new paradigm: An experimental analysis of a real processing- in-memory architecture. arXiv preprint arXiv:2105.03814 (2021)
[2] Lee, S., Kang, S.-h., Lee, J., Kim, H., Lee, E., Seo, S., Yoon, H., Lee, S., Lim, K., Shin, H., et al.: 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), pp. 43–56 (2021). IEEE
[3] Zou, X., Xu, S., Chen, X., Yan, L., Han, Y.: Breaking the von neumann bottleneck: architecture-level processing-in-memory technology. Science China Information Sciences 64(6), 160404 (2021)
[4] Raman, S.R.S., Nibhanupudi, S.T., Kulkarni, J.P.: Enabling in-memory compu- tations in non-volatile sram designs. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 12(2), 557–568 (2022)
[5] Jun, H., Cho, J., Lee, K., Son, H.-Y., Kim, K., Jin, H., Kim, K.: Hbm (high band- width memory) dram technology and architecture. In: 2017 IEEE International Memory Workshop (IMW), pp. 1–4 (2017). IEEE
[6] Hadidi, R., Asgari, B., Mudassar, B.A., Mukhopadhyay, S., Yalamanchili, S., Kim, H.: Demystifying the characteristics of 3d-stacked memories: A case study for hybrid memory cube. In: 2017 IEEE International Symposium on Workload Characterization (IISWC), pp. 66–75 (2017). IEEE
[7] Xie, Y.: Modeling, architecture, and applications for emerging memory technolo- gies. IEEE design & test of computers 28(1), 44–51 (2011)
[8] Asifuzzaman, K., Verdejo, R.S., Radojkovi´c, P.: Enabling a reliable stt-mram main memory simulation. In: Proceedings of the International Symposium on Memory Systems, pp. 283–292 (2017)

[9] Wong, H.-S.P., Lee, H.-Y., Yu, S., Chen, Y.-S., Wu, Y., Chen, P.-S., Lee, B., Chen,
[10] F.T., Tsai, M.-J.: Metal–oxide rram. Proceedings of the IEEE 100(6), 1951–1970 (2012)
[11] Thakkar, I.G., Pasricha, S.: Dyphase: A dynamic phase change memory architec- ture with symmetric write latency and restorable endurance. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37(9), 1760–1773 (2017)
[12] Sun, B., Liu, D., Yu, L., Li, J., Liu, H., Zhang, W., Torng, T.: Mram co- designed processing-in-memory cnn accelerator for mobile and iot applications arXiv preprint arXiv:1811.12179 (2018)
[13] Jung, S., Lee, H., Myung, S., Kim, H., Yoon, S.K., Kwon, S.-W., Ju, Y., Kim, M., Yi, W., Han, S., et al.: A crossbar array of magnetoresistive memory devices for in-memory computing. Nature 601(7892), 211–216 (2022)
[14] Imani, M., Gupta, S., Kim, Y., Zhou, M., Rosing, T.: Digitalpim: Digital-based processing in-memory for big data acceleration. In: Proceedings of the 2019 on Great Lakes Symposium on VLSI, pp. 429–434 (2019)
[15] Leitersdorf, O., Ronen, R., Kvatinsky, S.: Multpim: Fast stateful multiplication for processing-in-memory. IEEE Transactions on Circuits and Systems II: Express Briefs 69(3), 1647–1651 (2021)
[16] Peng, X., Liu, R., Yu, S.: Optimizing weight mapping and data flow for convo- lutional neural networks on rram based processing-in-memory architecture. In: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2019). IEEE
[17] Lu, A., Peng, X., Li, W., Jiang, H., Yu, S.: Neurosim simulator for compute-in- memory hardware accelerator: Validation and benchmark. Frontiers in artificial intelligence 4, 659060 (2021)
[18] Long, Y., Na, T., Mukhopadhyay, S.: Reram-based processing-in-memory archi- tecture for recurrent neural network acceleration. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26(12), 2781–2794 (2018)
[19] Lu, A., Peng, X., Luo, Y., Huang, S., Yu, S.: A runtime reconfigurable design of compute-in-memory based hardware accelerator. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 932–937 (2021). IEEE
[20] Kazemi, A., Sharifi, M.M., Zou, Z., Niemier, M., Hu, X.S., Imani, M.: Mimhd: Accurate and efficient hyperdimensional inference using multi-bit in-memory com- puting. In: 2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 1–6 (2021). IEEE
[21] Roy, S., Ali, M., Raghunathan, A.: Pim-dram: Accelerating machine learning workloads using processing in commodity dram. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 11(4), 701–710 (2021)
[22] Long, Y., Kim, D., Lee, E., Saha, P., Mudassar, B.A., She, X., Khan, A.I., Mukhopadhyay, S.: A ferroelectric fet-based processing-in-memory architecture for dnn acceleration. IEEE Journal on Exploratory Solid-State Computational Devices and Circuits 5(2), 113–122 (2019)
[23] Lee, W.J., Kim, C.H., Paik, Y., Park, J., Park, I., Kim, S.W.: Design of processing- “inside”-memory optimized for dram behaviors. IEEE Access 7, 82633–82648(2019)
[24] Gupta, S., Imani, M., Kaur, H., Rosing, T.S.: Nnpim: A processing in-memory architecture for neural network acceleration. IEEE Transactions on Computers 68(9), 1325–1337 (2019)
[25] Lin, C.-C., Lee, C.-L., Lee, J.-K., Wang, H., Hung, M.-Y.: Accelerate binarized neural networks with processing-in-memory enabled by risc-v custom instruc- tions. In: 50th International Conference on Parallel Processing Workshop, pp. 1–8 (2021)
[26] Olgun, A., Luna, J.G., Kanellopoulos, K., Salami, B., Hassan, H., Ergin, O., Mutlu, O.: Pidram: A holistic end-to-end fpga-based framework for processing- in-dram. ACM Transactions on Architecture and Code Optimization 20(1), 1–31 (2022)
[27] Drumond, M., Daglis, A., Mirzadeh, N., Ustiugov, D., Picorel, J., Falsafi, B., Grot, B., Pnevmatikatos, D.: Algorithm/architecture co-design for near-memory processing. ACM SIGOPS Operating Systems Review 52(1), 109–122 (2018)
[28] Park, N., Ryu, S., Kung, J., Kim, J.-J.: High-throughput near-memory processing on cnns with 3d hbm-like memory. ACM Transactions on Design Automation of Electronic Systems (TODAES) 26(6), 1–20 (2021)
[29] Kwon, Y., Rhu, M.: Beyond the memory wall: A case for memory-centric hpc sys- tem for deep learning. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 148–161 (2018). IEEE
[30] Ghose, S., Boroumand, A., Kim, J.S., G´omez-Luna, J., Mutlu, O.: Processing- in-memory: A workload-driven perspective. IBM Journal of Research and Development 63(6), 3–1 (2019)
[31] Huang, Y., Zheng, L., Yao, P., Zhao, J., Liao, X., Jin, H., Xue, J.: A heterogeneous pim hardware-software co-design for energy-efficient graph processing. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 684–695 (2020). IEEE
[32] Liu, J., Zhao, H., Ogleari, M.A., Li, D., Zhao, J.: Processing-in-memory for energy-efficient neural network training: A heterogeneous approach. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO),
[33] pp. 655–668 (2018). IEEE
[34] Wang, Y., Chen, W., Yang, J., Li, T.: Towards memory-efficient allocation of cnns on processing-in-memory architecture. IEEE Transactions on Parallel and Distributed Systems 29(6), 1428–1441 (2018)
[35] Jiao, B., Zhu, H., Zhang, J., Wang, S., Kang, X., Zhang, L., Wang, M., Chen, C.: Computing utilization enhancement for chiplet-based homogeneous processing- in-memory deep learning processors. In: Proceedings of the 2021 on Great Lakes Symposium on VLSI, pp. 241–246 (2021)
[36] Jeon, W., Lee, J., Kang, D., Kal, H., Ro, W.W.: Pimcaffe: Functional evaluation of a machine learning framework for in-memory neural processing unit. IEEE Access 9, 96629–96640 (2021)
[37] Hosseini, M.S., Ebrahimi, M., Yaghini, P., Bagherzadeh, N.: Near volatile and non-volatile memory processing in 3d systems. IEEE Transactions on Emerging Topics in Computing 10(3), 1657–1664 (2021)
[38] Ahn, J., Yoo, S., Mutlu, O., Choi, K.: Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. ACM SIGARCH Computer Architecture News 43(3S), 336–348 (2015)
[39] Pattnaik, A., Tang, X., Jog, A., Kayiran, O., Mishra, A.K., Kandemir, M.T., Mutlu, O., Das, C.R.: Scheduling techniques for gpu architectures with processing-in-memory capabilities. In: Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, pp. 31–44 (2016)
[40] Zhang, J., Zha, Y., Beckwith, N., Liu, B., Li, J.: Meg: A riscv-based system emula- tion infrastructure for near-data processing using fpgas and high-bandwidth mem- ory. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 13(4), 1–24 (2020)
[41] Gu, P., Xie, X., Ding, Y., Chen, G., Zhang, W., Niu, D., Xie, Y.: ipim: Programmable in-memory image processing accelerator using near-bank

architec- ture. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 804–817 (2020). IEEE
[42] Ghose, S., Hsieh, K., Boroumand, A., Ausavarungnirun, R., Mutlu, O.: Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions. arXiv preprint arXiv:1802.00320 (2018)
[43] Ke, L., Gupta, U., Cho, B.Y., Brooks, D., Chandra, V., Diril, U., Firoozshahian, A., Hazelwood, K., Jia, B., Lee, H.-H.S., et al.: Recnmp: Accelerating personalized recommendation with near-memory processing. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 790–803 (2020). IEEE
[44] Zhou, Z., Li, C., Wei, X., Wang, X., Sun, G.: Gnnear: Accelerating full-batch training of graph neural networks with near-memory processing. In: Proceed- ings of the International Conference on Parallel Architectures and Compilation Techniques, pp. 54–68 (2022)
[45] Nider, J., Mustard, C., Zoltan, A., Ramsden, J., Liu, L., Grossbard, J., Dashti, M., Jodin, R., Ghiti, A., Chauzi, J., et al.: A case study of {Processing-in-Memory} in {off-the-Shelf} systems. In: 2021 USENIX Annual Technical Conference (USENIX ATC 21), pp. 117–130 (2021)
[46] Giannoula, C., Fernandez, I., G´omez-Luna, J., Koziris, N., Goumas, G., Mutlu, O.: Towards efficient sparse matrix vector multiplication on real processing- in-memory architectures. ACM SIGMETRICS Performance Evaluation Review 50(1), 33–34 (2022)
[47] Kang, H., Gibbons, P.B., Blelloch, G.E., Dhulipala, L., Gu, Y., McGuffey, C.: The processing-in-memory model. In: Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures, pp. 295–306 (2021)
[48] Zhou, M., Imani, M., Kim, Y., Gupta, S., Rosing, T.: Dp-sim: A full-stack simula- tion infrastructure for digital processing in-memory architectures. In: Proceedings of the 26th Asia and South Pacific Design Automation Conference, pp. 639–644 (2021)
[49] Xu, S., Chen, X., Wang, Y., Han, Y., Qian, X., Li, X.: Pimsim: A flexible and detailed processing-in-memory simulator. IEEE Computer Architecture Letters 18(1), 6–9 (2018)
[50] Xie, X., Gu, P., Huang, J., Ding, Y., Xie, Y.: Mpu-sim: A simulator for in-dram near-bank processing architectures. IEEE Computer Architecture Letters 21(1), 1–4 (2021)
[51] Sanchez, D., Kozyrakis, C.: Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. ACM SIGARCH Computer architecture news 41(3), 475–486 (2013)
[52] Kim, Y., Yang, W., Mutlu, O.: Ramulator: A fast and extensible dram simulator. IEEE Computer architecture letters 15(1), 45–49 (2015)
[53] Yu, C., Liu, S., Khan, S.: Multipim: A detailed and configurable multi-stack processing-in-memory simulator. IEEE Computer Architecture Letters 20(1), 54– 57 (2021)
[54] Oliveira, G.F., G´omez-Luna, J., Orosa, L., Ghose, S., Vijaykumar, N., Fernandez, I., Sadrosadati, M., Mutlu, O.: Damov: A new methodology and benchmark suite for evaluating data movement bottlenecks. IEEE Access 9, 134457–134502 (2021)
[55] Xu, S., Chen, X., Qian, X., Han, Y.: Tupim: a transparent and universal processing-in-memory architecture for unmodified binaries. In: Proceedings of the 2020 on Great Lakes Symposium on VLSI, pp. 199–204 (2020)
[56] Xu, S., Chen, X., Wang, Y., Han, Y., Li, X.: 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, pp. 140–145 (2019)
[57] Singh, G., Chelini, L., Corda, S., Awan, A.J., Stuijk, S., Jordans, R., Corpo- raal, H., Boonstra, A.-J.: Near-memory computing: Past, present, and future. Microprocessors and Microsystems 71, 102868 (2019)
[58] Awan, A.J., Vlassov, V., Brorsson, M., Ayguade, E.: Node architecture implica- tions 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, pp. 237–246 (2016)
[59] Hoste, K., Eeckhout, L.: Microarchitecture-independent workload characteriza- tion. IEEE micro 27(3), 63–72 (2007)
[60] Anghel, A., Vasilescu, L.M., Jongerius, R., Dittmann, G., Mariani, G.: An instrumentation approach for hardware-agnostic software characterization. In: Proceedings of the 12th ACM International Conference on Computing Frontiers,
[61] pp. 1–8 (2015)
[62] Jongerius, R., Anghel, A., Dittmann, G., Mariani, G., Vermij, E., Corporaal, H.: Analytic multi-core processor model for fast design-space exploration. IEEE Transactions on Computers 67(6), 755–770 (2017)
[63] Xu, L., Zhang, D.P., Jayasena, N.: Scaling deep learning on multiple in-memory processors. In: Proceedings of the 3rd Workshop on Near-Data Processing (2015)
[64] Choe, H., Lee, S., Nam, H., Park, S., Kim, S., Chung, E.-Y., Yoon, S.: Near-data processing for differentiable machine learning models. arXiv preprint arXiv:1610.02273 (2016)
[65] Argollo, E., Falc´on, A., Faraboschi, P., Monchiero, M., Ortega, D.: Cotson: infras- tructure for full system simulation. ACM SIGOPS Operating Systems Review 43(1), 52–61 (2009)
[66] Li, B., Yan, B., Li, H.: An overview of in-memory processing with emerging non- volatile memory for data-intensive applications. In: Proceedings of the 2019 on Great Lakes Symposium on VLSI, pp. 381–386 (2019)
[67] Yu, S., Sun, X., Peng, X., Huang, S.: Compute-in-memory with emerging nonvolatile-memories: Challenges and prospects. In: 2020 Ieee Custom Integrated Circuits Conference (cicc), pp. 1–4 (2020). IEEE
[68] Yu, C., Liu, S., Khan, S.: Multipim: A detailed and configurable multi-stack processing-in-memory simulator. IEEE Computer Architecture Letters 20(1), 54– 57 (2021)
[69] Umesh, S., Mittal, S.: A survey of spintronic architectures for processing-in- memory and neural networks. Journal of Systems Architecture 97, 349–372 (2019)



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY