SCIENCE CHINA
Information Sciences

June 2021, Vol. 64 160404:1–160404:10
https://doi.org/10.1007/s11432-020-3227-1

• PROGRESS •
Special Focus on Near-memory and In-memory Computing

# Breaking the von Neumann bottleneck: architecture-level processing-in-memory technology

Xingqi ZOU[1,2], Sheng XU[3*], Xiaoming CHEN[1,4*], Liang YAN[1,4] & Yinhe HAN[1,4*]

[1]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190*, China;*
[2]*Zhejiang Lab, Hangzhou* 311121*, China;*
[3]*Department of Computer Technology and Information, Anhui Normal University, Wuhu* 241002*, China;*
[4]*University of Chinese Academy of Sciences, Beijing* 101408*, China*

Received 31 December 2020/Revised 9 March 2021/Accepted 23 March 2021/Published online 27 April 2021

**Abstract** The "memory wall" problem or so-called von Neumann bottleneck limits the efficiency of conventional computer architectures, which move data from memory to CPU for computation; these architectures cannot meet the demands of the emerging memory-intensive applications. Processing-in-memory (PIM) has been proposed as a promising solution to break the von Neumann bottleneck by minimizing data movement between memory hierarchies. This study focuses on prior art of architecture level DRAM PIM technologies and their implementation. The key challenges and mainstream solutions of PIM are summarized and introduced. The relative limitations of PIM simulation are discussed, as well as four conventional PIM simulators. Finally, research directions and perspectives are proposed for future development.

**Keywords** processing-in-memory (PIM), von Neumann bottleneck, memory wall, PIM simulator, architecture-level PIM

**Citation** Zou X Q, Xu S, Chen X M, et al. Breaking the von Neumann bottleneck: architecture-level processing-in-memory technology. Sci China Inf Sci, 2021, 64(6): 160404, https://doi.org/10.1007/s11432-020-3227-1

## 1 Introduction

In recent years, with the development of artificial intelligence, big data, and cloud computing applications, computer architecture has shifted from computing-intensive to memory-intensive. Memory-intensive applications have two characteristics: large datasets and poor data locality (discrete and random memory access). The conventional von Neumann architecture, with a separated memory-computing structure, is difficult to meet the demand of these applications because of the "memory wall" problem or von Neumann bottleneck. The memory wall problem is mainly summed up in three aspects: (1) data movement between memory arrays and the processing unit results in non-negligible latency; (2) data movement in memory hierarchies is greatly limited by bandwidth; (3) high energy consumption, such as the power consumption of moving data between computing and off-chip memory units is 100 times more than floating-point computing [1].

To overcome such problems, processing-in-memory (PIM) technology is proposed and considered as a promising solution to break the von Neumann bottleneck. The key idea of technology is to bring memory and computing closer instead of separating them, thus, improving the efficiency of data movement.

The PIM technology can be studied from different perspectives. There have been several studies from the perspective of devices [2–7] and circuits [8–12]; however, in this paper, we mainly focus on PIM technology based on architecture level. The architecture level PIM is application-oriented and can execute a variety of operations to accelerate memory-intensive applications. Adopting PIM architecture in emerging technology has revolted conventional computing architectures; nevertheless, it has brought a series of architecture issues such as data coherence, compatibility, transparency, and simulation.

This paper summarizes architecture-level DRAM PIM technology and simulation technology. Section 2 provides a brief introduction to the background of PIM technology. Section 3 introduces existing PIM

* Corresponding author (email: xusheng2019@ahnu.edu.cn, chenxiaoming@ict.ac.cn, yinhes@ict.ac.cn)

 

technology. Section 4 introduces existing PIM simulation technology. Section 5 summarizes challenges faced by PIM technology, as well as potential prospects for future development.

## 2 Background

PIM technology was proposed in the late 1960s. Kautz [13] and Stone [14] in Stanford Research Institute, brought the logic-in-memory concept to the world. They combined memory arrays with logical processing units to implement the earliest PIM system. Although many researchers have presented various PIM methods, only a few PIM systems have been adopted and are not commercialized due to manufacturing incompatibility manufacturing between memory and logic devices [15]. In recent years, the research on PIM architecture has become popular in academia and industry, mainly due to the following reasons: (1) Advanced integrated process technology, such as 2.5D and 3D integrated packaging process, has been proposed so that memory and logic processing units can be integrated. (2) With the prosperity of big data, artificial intelligence, and other applications, data-intensive applications have been constantly emerging. The conventional computing architecture is limited by latency, low bandwidth, and high power consumption; thus, new solutions are needed; (3) with the emergence of new devices and materials, some novel PIM methods have been realized at the device level.

Processing-in-memory, also known as logic-in-memory or near-data processing, can be divided into the device level, circuit level, and architectural level.

Device-level PIM ensures devices implement in-situ computing. Nonvolatile memory (for example, RRAM [2–5], PCM [6], Flash [7]) benefits from the intrinsic properties of device-level PIM for computation, such that multiplication is realized according to Ohm's law and addition is realized according to Kirchhoff's current law. Hence, memory arrays achieve multiply-and-accumulate (MAC) operations.

Circuit-level PIM modifies peripheral circuits to directly implement computing within memory subarrays. For instance, a circuit-level PIM based on SRAM array stores network weights in SRAM cells, and peripheral circuits implement XNOR operations or binary MAC operations [8,9]. A circuit-level PIM based on DRAM modifies peripheral circuits and uses a charge-sharing mechanism between DRAM cells to achieve bulk bitwise operations [10–12].

Architectural-level PIM integrates computing logic circuits or processing cores into memory chips to achieve a variety of operations. In recent years, with the development of 2.5D and 3D integration technology, manufacturing incompatibility challenge between memory and logic devices has been tackled. In 3D-stacking memory, multilayer DRAM is integrated with the bottom logical layer through a through-silicon-via (TSV) process. Architecture-level PIM technology based on DRAM is realized by integrating some computing resources into memory. Such PIM has a high data bandwidth, which can alleviate the "von Neumann bottleneck". PIM reduces data movement between processor and memory, thereby greatly improving memory access and energy efficiency. Currently, 3D stacking DRAM is commercially produced, such as Micron's hybrid memory cube (HMC) [16], AMD, and Hynix's high bandwidth memory (HBM) [17]. At the same time, some companies have proposed PIM technology based on DRAM, such as UPMEM [18].

HMC is a typical PIM technology that integrates simple logic with memory to perform an uninterrupted "atomic operation". The atomic operation is a read-modify/calculate-write operation for data in the same memory address. According to HMC specification 2.1 [19], HMC supports four types of atomic operations: simple addition, comparison, Boolean operations, and bitwise operations. Although HMC supports a limited number of computing types, Samsung, Intel, and Micron have formed an HMC alliance to further standardize and promote the HMC protocol interface, making it easy for researchers to expand and design new PIM architectures. In recent years, researchers have proposed a series of architecture-level DRAM-based PIM designs [20–22]. In a PIM module, not only simple logic circuits but also one or more processing cores, referred to as a PIM processor in this article, can be integrated [15]. Figure 1 shows the conventional computing architecture and PIM architecture. Unlike the conventional computing architecture, the use of a PIM processor in new architecture creates a series of issues and challenges. In a PIM system, both the host processor and PIM processor can handle data, leading to data coherence issues. The host processor needs to interact with a PIM module for data and instructions, causing compatibility issues. The availability of different PIM solutions makes it difficult to design general PIM instructions and programming models, resulting in transparency issues. A lengthy design period thwarts PIM system hardware tape-out verification. To expedite research and development in this regard, most

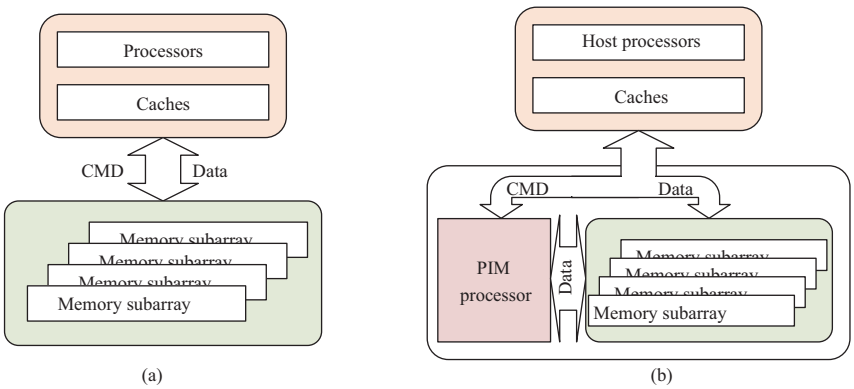


**Figure 1** (Color online) (a) Conventional computing architecture and (b) PIM architecture.

PIM verification has shifted to simulation. Simulator designs, on the other, present several challenges. These issues and challenges are thoroughly discussed in Sections 3 and 4.

## 3 Open research issues of PIM technology

### 3.1 Data coherence issues

In PIM architecture, because both the host processor and PIM processor can process data, it is necessary to maintain data coherence among the host processor, PIM processor, and cache.

In early PIM architecture, the non-cacheable approach was adopted, such as the PEI system proposed by Seoul University in 2015 with a write-through strategy [23]. Each write request from the host processor triggers a write-through to memory, which writes the data required by the PIM module from the cache back into memory. However, with the development of PIM technology, the amount of data processed by the PIM module has increased. The non-cacheable approach greatly increases memory access and consumes bandwidth; as a result, it cannot meet the needs of PIM architecture.

Several studies have been conducted on the coherence of conventional multi-core architectures, such as coherence protocols MESI [24] and MOESI [25]. The basic method they adopted is setting a status tag in each cache line. The system monitors change in the cache line through the broadcast of the coherence bus. The system changes the state of the cache line if the processor accesses the cache line. However, conventional coherence protocols do not adapt to PIM structures because broadcast coherence messages consume bandwidth and increase memory wall issues.

Based on the conventional multi-core coherence protocol, which is a large overhead for PIM systems, researchers have proposed a "coarse-grained" coherence scheme [26–28]. The coarse-grained increases the granularity of conventional coherence protocols. Only minor changes to conventional coherence protocols are required, such as changing from a cache line size (typically 64 bytes) to a page size (typically 4 kB) or more. The coarse-grained coherence scheme maintains a global lock table in memory. When the PIM processor issues an access request, a table entry is created in the global lock table to indicate that this memory region is accessed by the PIM processor. Each time the processor issues an access request, it needs to detect whether its address is "locked". Only when it is not locked, that is, when the data have not been modified by the PIM processor, can the next operation be performed. The coarse-grained coherence scheme increases the granularity of data management and reduces the number of coherence synchronization messages, which reduces the access bandwidth usage.

However, coarse-grained coherence schemes have some drawbacks. In the schemes, to prevent the PIM processor from accessing the wrong data when the PIM processor issues an access request, all data are triggered in the management granularity to be written back from the cache to memory. Both the host and PIM processor have to wait for the data to be written back, resulting in a significant latency and power consumption. If the host and PIM processor access different data within the same data management granularity, a write-back command is also triggered. This causes the data ping-pong phenomenon between cache and memory, also resulting in latency and power consumption.

LazyPIM [29] is a PIM coherence approach based on speculative execution proposed by Amirali Boroumand of Carnegie Mellon University. It periodically records and compares read/write data by

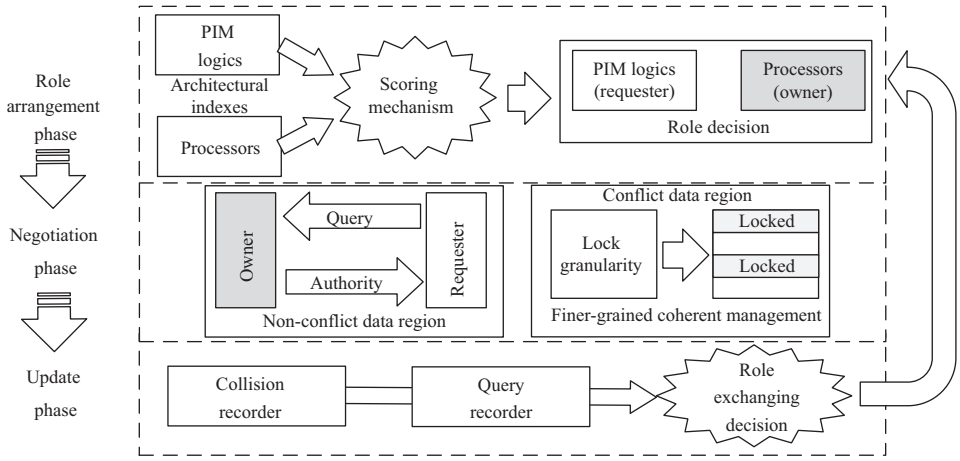


**Figure 2** Overview of the CuckooPIM execution.

the host processor and PIM processor. Before data are synchronized, the PIM processor performs speculative execution based on existing data. The result of execution will not be submitted until the data comparison is completed and no data conflicts are confirmed. To achieve this, LazyPIM includes a "Conflict Detect Hardware" component in its scheme, which is used to synchronize cache line read/write states between the host processor and PIM processor at regular intervals and to detect whether there is a data coherence conflict between them. When conflict detect hardware detects that the speculative execution is using incorrect data (modified by the host processor), the modified data will be written back and the PIM processor will roll back to the last "Check Point" state and re-execute until there are no more data conflicts. The speculative execution strategy of LazyPIM reduces system blocking caused by the coarse-grained coherence approach and improves the performance of the PIM system. LazyPIM can reduce the latency caused by data blocking in the system; however, this approach is highly dependent on the characteristics of the dataset and application.

To solve the problems of data ping-pong and system blocking caused by coarse-grained PIM coherence, a data coherence method CuckooPIM [30] based on the "Owner-Requester relationship" is proposed. Figure 2 shows the execution diagram of the CuckooPIM framework. When a PIM task starts, CuckooPIM enters the role arrangement phase. At this moment, CuckooPIM first periodically collects run-time information from both the host processor and in-memory processor and then decides on a role decision based on a scoring mechanism. The data owner then holds the paramount data authority. The data requester has to ask the owner for access permissions. CuckooPIM also proposes a local fine-grained data management approach to avoid frequent data flushing when the data region is frequently accessed by both sides or when deciding the role arrangement is difficult due to close scores. CuckooPIM collects the parameters that indicate the efficiency of execution at intervals during the updating phase. If the previous role judgment is considered insufficient to keep up with the current execution, the roles will be switched between the host and PIM processors. CuckooPIM dynamically determines data ownership, which is beneficial to PIM-based systems. CuckooPIM is a criticality-aware and less-blocking coherence mechanism. Experiments show that CuckooPIM can effectively avoid unnecessary data movements and lock-content stalls, achieving a 1.68× speedup on average over coarse-grained coherence.

### 3.2 Compatibility issues

Prefetching technology can improve the cache hit rate by taking advantage of the data locality of memory access. The improvement of the prefetching algorithm increases the number of instructions executed per cycle. However, there are compatibility issues in PIM architecture. Owing to data coherence, a part of the prefetched cached data needs to be written back to memory during the PIM processor's execution to ensure that the PIM processor can access new data. Therefore, data are prefetched-write and back-read, resulting in frequent data movement between the cache and PIM, increasing bandwidth occupation and system energy consumption. This phenomenon is called overlock because it generates additional lock-contention stalls [31]. The reason for this is that existing cache prefetch mechanisms such as GHB [32] or AMPM [33] only work according to the access history of the host processor, without considering the access behavior of the PIM module and the data write back issue.

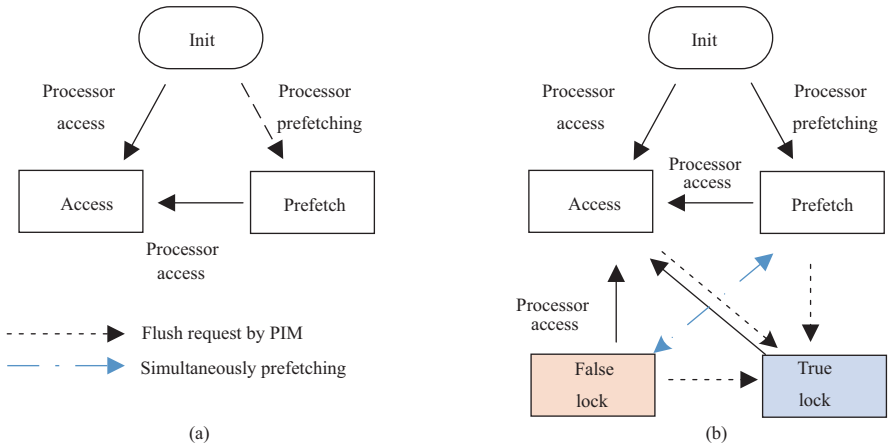


**Figure 3** (Color online) Prefetching state machines of traditional (a) AMPM and (b) PIMCH.

Motivated by these problems, a PIM-aware prefetching scheme, PIMCH [31], is proposed. PIMCH enables cooperative memory prefetching that considers the memory access hinting of both the host processor and PIM processor. Therefore, PIMCH could avoid overlock prefetching decisions that may cause unnecessary data movement between memory hierarchies. Figure 3 shows the state machine of PIMCH. Based on the AMPM state machine, two states representing data sharing are added to determine whether the current prefetch request of the host processor will cause a system block. First, the PIM processor takes advantage of the prefetching function to only obtain a result without sending a request. Then, the confidences of the host processor and PIM processor are calculated. If the PIMCH unit predicts that the PIM predictions have higher confidence, which means that prefetching the target data may cause overlock, the host processor will then abandon the generated prefetching requests and change the state of the cache line to false lock. False lock means that the current data are considered to be used first by the in-memory processors before being accessed by the host processor. This mechanism can prevent unnecessary data movement between memory hierarchies. On the other hand, if the PIMCH unit is aware that the confidence of the host processor prediction is higher than that of the in-memory processors, the PIMCH unit will not change the state and continue generating the prefetching request. Therefore, the state false lock can be changed to processor prefetch as a result. Our full system simulation shows that PIMCH improves performance by 43.3% on average, reduces off-chip traffic by 16.4%, and successfully prevents up to 84.01% mis-prefetching predictions.

### 3.3 Transparency issues

With the development of architecture-level PIM technology, researchers have proposed a variety of PIM solutions. Different memory-processing units can be simple logic circuits, that is, special processing units for "read-change-write" atomic operations or multiple processors [15]. The diversity of memory processing units makes it difficult to design general PIM instruction and programming models. Therefore, when running a new application on PIM architecture, a dynamical analysis of programs and datasets is needed. Then, the system decides which instructions, called PIM kernels, are executed in PIM. When PIM kernels are decided, the conventional PIM solution is to manually annotate the source code, modifying the compiler, or applying new programming models to designate PIM kernels. For example, Tesseract [34] rewrites source codes using a dedicated programming model on some graph-processing applications and gains performance and energy improvements, which does not apply to other PIM applications. Moreover, the requirements of source code modification and compiler severely limit the benefits of PIM [35]. However, this technique is not general and is not suitable for other PIM architectures and applications.

To address the transparency issue, researchers propose TUPIM [35], short for a transparent and universal PIM, a PIM architecture that can execute unmodified binaries, allowing virtually all applications to benefit from PIM [35]. TUPIM makes the following contributions. (1) The architectures are completely transparent to software developers. (2) An effective and low-cost PIM kernel selection method is proposed to determine suitable instructions that should be executed in the memory. (3) The instruction-level PIM parallelism method is proposed based on speculative execution to ensure the correct execution of instructions in PIM. To achieve these functions, TUPIM divides the entire PIM execution process into three

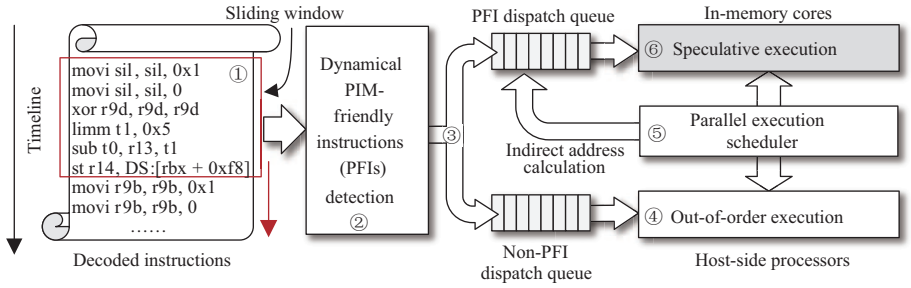


**Figure 4** (Color online) Top-level execution diagram of TUPIM.

steps: extract, dispatch, and execute, as shown in Figure 4. When a binary is being decoded in TUPIM, the decoded instructions are temporarily stored in an instruction buffer and await processing in the subsequent pipeline stages. TUPIM detects PIM-friendly instructions (PFIs) using an approach such as a sliding window ①. Then, TUPIM identifies PFIs ② and classifies them into two groups ③. Instructions to be executed on host processors, named non-PFIs, will be executed on host CPUs by TUPIM as the same way as it has in the past ④. Instructions that possibly benefit from PIM, named candidate PFIs (cPFIs), will be executed in the memory cores speculatively ⑤. After the identification of cPFIs, TUPIM starts another round of checks to determine when the cPFIs are suitable for PIM executions based on run-time factors. If not, they are then sent back to the host processors ⑥. Otherwise, TUPIM performs a speculative execution for the PFIs in the memory [35]. Through this mechanism, TUPIM can reduce the correlation between the host processor and PIM processor, increase the concurrency of instructions, and improve the efficiency of PIM architecture. At the same time, because PFIs are dynamically extracted from the original instruction sequence, TUPIM does not need to make any modifications to the existing program source code, compiler, or operating system. Therefore, this scheme achieves a "transparent" PIM architecture and improves the universality of PIM architecture. Experiments show that TUPIM is 2.2 times faster and 15.7% less energy consumption than conventional CPU-only executions on average.

## 4 PIM simulator

PIM integrates computing resources within the memory, changing conventional architecture, affecting program compilation, the instruction set, inter-module communication protocol, workload offloading, data coherence, and so on. It is difficult for existing simulators to achieve a comprehensive simulation of the PIM system. To address this problem, some universities or research institutions have developed their simulators for PIM research, such as SMCSim [27], CLAPPS [36], Ramulator [37,38] and PIMSim [39].

### 4.1 SMCSim

SMCSim was proposed in 2016 to meet the requirements of complicated computing in HMC. SMCSim implemented the HMC module and PIM processor design based on the GEM5 simulator and completed a series of automated scripts. SMCSim is an open-source simulator on GitHub. A common PIM computing unit is configured in the logic layer of the HMC in SMCSim, which is essentially a common processing core that supports the ARM instruction set, using the Cortex-V7 processor model of the ARM architecture in GEM5 [40]. For the control of the PIM processor, SMCSim configures the host processor as a single-core ARM Cortex-V7/8 processor. SMCSim implements a programming model for PIM and includes part of the benchmark assembly. SMCSim realizes PIM execution based on ELF file; PIM execution based on ELF files is implemented. However, SMCSim has difficulty flexibly configuring and universally simulating PIM architecture. For example, rather than designing a dedicated PIM instruction set, SMCSim assumes that codes execute as ELF files on a PIM processor, making it difficult to simulate PIM instruction sets, data coherence, multi-core, multithreading, etc.

### 4.2 CLAPPS

CLAPPS is based on SystemC and integrates the latest HMC specifications. All HMC technical details are implemented, including communication protocol, serialization module, deserialization module, link control, Internet network, and vault control, and a set of PIM interfaces are proposed. CLAPPS uses

SystemC as the frontend to describe the behavior of the computing unit and the HMC simulator as the backend to complete a series of simulations of the HMC module. CLAPPS, based on SystemC, fully customizes the logic of the PIM processor and the instruction control flow to achieve a high degree of core simulation freedom. However, the biggest problem with CLAPPS is that it is incompatible with system-level and application-level simulations. Although CLAPPS allows users to configure PIM modules and control instructions with a high degree of flexibility, these instructions are difficult to associate with higher-level applications. Moreover, CLAPPS is not open source, and its specific implementation and effect are unknown.

### 4.3 Ramulator-PIM

Ramulator-PIM is developed from Ramulator, which is a well-known DRAM simulator that is established by Carnegie Mellon University. Ramulator enables the realization of a cycle-accurate DRAM simulation without sacrificing speed. The main difference between Ramulator and other simulators is that it provides a template for memory simulation depending on what the simulator can customize to meet the needs. This advantage enables the simulation of different kinds of DRAM standards (e.g., DDR3/4, HBM, HMC), making it feasible to simulate PIM architecture [41]. To perform the design space exploration of DRAM-based PIM architectures, Ramulator-PIM needs to work with ZSim [41], which is also a widely-known open-source simulator; it is regarded as a new computing system that includes host CPU cores and general-purpose PIM cores. PIM cores are placed into a 3D-stacked DRAM logic layer to offer computing ability. As for this simulation framework, Ramulator-PIM can perform simulations for general-purpose PIM-cores to compare the performance of different applications.

Memory traces generated by ZSim are used by Ramulator-PIM to perform trace analysis in Ramulator. To make ZSim compatible with Ramulator simulation, ZSim has made some modifications generating memory traces that are adapted to Ramulator. Two kinds of traces generated by ZSim are introduced in Ramulator-PIM. Specifically, host CPU cores are simulated with filtered memory traces; on the other hand, unfiltered traces are used to perform PIM cores' simulations. Ramulator-PIM enables the selection of out-of-order and in-order cores for both host CPU cores and PIM cores. Especially, Ramulator-PIM has been modified to reduce the overhead introduced by off-chip links in the simulation of PIM cores embedded in the logic layer of 3D-stacked memory. Most importantly, Ramulator-PIM is sufficiently fast with good extensibility, benefiting from the speed of Ramulator, which is 2.5× faster than the next simulator according to evaluations.

### 4.4 PIMSim

The PIMSim simulator is developed based on the GEM5 simulator, comprising modules such as workload offloading, PIM instruction loading, and cache coherence. The structure of the simulator is shown in Figure 5. A configurable PIM architecture-level simulator is implemented. PIMSim supports the definition of PIM computing instruction sets, instruction control flows, PIM coherence protocols of different granularities, and power simulation of PIM structures.

The frontend of PIMSim is a workload offloading module that identifies and distributes PIM instructions. The choice of a PIM kernel determines the performance of the system. PIMSim provides four flexible PIM annotation styles in the source code and a corresponding compiler that enables users to specify which parts of an application should run in memory without having a significant impact on the programming model. PIMSim also supports feedback to determine whether PIM instructions are dynamically executed in PIM computing units, during which a user can insert a custom monitor to monitor specified metrics, such as the number of instructions per cycle (IPC) or memory access strength, and use them as the feedback to dynamically adjust their application partitioning policies.

At the same time, PIMSim enables PIM computing units to be configured as processors or simple logic. If configured as simple logic, a black-box model of the computing unit is designed to enable rapid configuration and simulation of circuit logic. PIMSim supports a variety of simulation modes, considering both analog accuracy and simulation speed compromise: the fast simulation mode only simulates primary models, such as the host processor, memory, the PIM processor, and simple lock-based PIM coherent methods; the instrumentation-driven mode of PIMSim provides an efficient tool that uses Pin as a frontend to feed real-time traces to the simulator by instrumenting efficiently; the Full-System simulation mode of PIMSim simulates the details and peripherals of the PIM system based on the GEM5 full system mode. The PIMSim simulator is now open-source on GitHub.

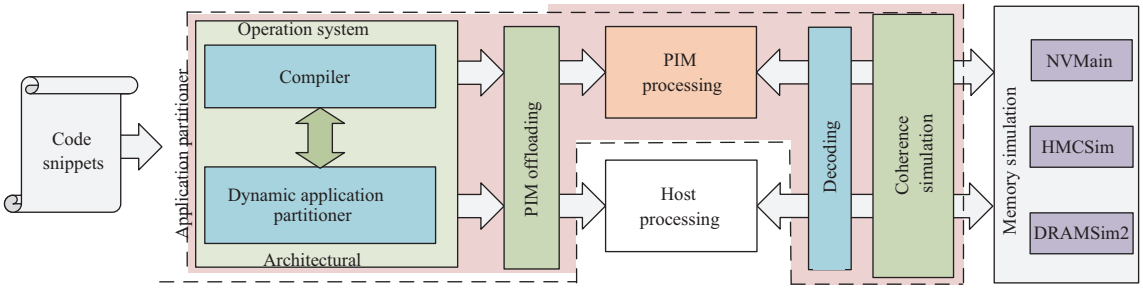


**Figure 5** (Color online) Top-level architectural diagram of PIMSim [39] @Copyright 2019 IEEE.

## 5 Future and challenges

The PIM technology is considered as a promising method for overcoming the von Neumann bottleneck, but it still faces numerous challenges. In the future, PIM architecture may achieve high performance through new integration technologies, such as Chiplet [42] and Xtacking [43].

One of the challenges in PIM architecture is that only a few simple sequential processors can be integrated because of thermal problems. The cooling of the underlying logical layer is negatively affected by HMC based on TSV processing. However, the Chiplet technology is scale-out for horizontal integration, allowing higher performing computing units to be integrated into PIM.

There are numbered connections between the memory and logic layers in the TSV process, limiting the overall performance of PIM. The Xtacking technology integrates logic chips with memory chips through a bonding process, with a connection pitch of a few microns between logic and memory. Compared to the TSV process, the Xtacking technology can achieve a greater number of interconnects with smaller parasitic effects. As a result, the logic and memory layers can be more closely integrated, increasing bandwidth between the logic and memory layers and, thus, improving the overall performance of the PIM system.

In the future, with the development of new integration technologies, PIM will have higher performance but also more challenges, necessitating more advanced PIM solutions. Furthermore, based on new integration technologies, the key issues that must be addressed are coherence issues, compatibility issues, transparency issues, and limitations of simulation technologies.

## 6 Summary

This paper provides an overview of PIM technologies. It also discusses the issues these technologies face, such as data coherence, compatibility, and transparency, and introduces current research and solutions to these issues. At the same time, PIM simulation technologies are introduced, as are four types of PIM simulators. Finally, the prospects and challenges of architecture level DRAM PIM technology are discussed.

**Acknowledgements** This work was supported by National Key R&D Program of China (Grant No. 2018YFA0701500), Zhejiang Lab (Grant No. 2019KC0AB010), Key Research Program of Frontier Sciences, CAS (Grant No. ZDBS-LY-JSC012), Strategic Priority Research Program of CAS (Grant No. XDB44000000), Youth Innovation Promotion Association CAS, Beijing Academy of Artificial Intelligence (BAAI), Anhui Natural Science Foundation (Grant No. 2008085QF330), and Research Program of Anhui Normal University (Grant No. 751968).

**References**

1 Mittal S. A survey of ReRAM-based architectures for processing-in-memory and neural networks. Mach Learn Knowl Extr, 2018, 1: 75–114

2 Chen L R, Li J W, Chen Y R, et al. Accelerator-friendly neural-network training: learning variations and defects in RRAM crossbar. In: Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, 2017. 19–24

3 Chen W H, Li K X, Lin W Y, et al. A 65 nm 1 Mb nonvolatile computing-in-memory ReRAM macro with sub-16 ns multiply-and-accumulate for binary DNN AI edge processors. In: Proceedings of IEEE International Solid-State Circuits Conference, San Francisco, 2018. 494–496

4 Cai F, Correll J M, Lee S H, et al. A fully integrated reprogrammable memristor-CMOS system for efficient multiply-accumulate operations. Nat Electron, 2019, 2: 290–299

5 Yao P, Wu H, Gao B, et al. Fully hardware-implemented memristor convolutional neural network. Nature, 2020, 577: 641–646

6 Burr G W, Shelby R M, Sidler S, et al. Experimental demonstration and tolerancing of a large-scale neural network (165000 synapses) using phase-change memory as the synaptic weight element. In: Proceedings of IEEE International Electron Devices Meeting, 2015. 3498–3507

7 Guo X, Bayat F M, Bavandpour M, et al. Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology. In: Proceedings of IEEE International Electron Devices Meeting (IEDM), San Francisco, 2017. 1–4

8 Jiang Z, Yin S, Seo J S, et al. XNOR-SRAM: in-bitcell computing SRAM Macro based on resistive computing mechanism. In: Proceedings of the 2019 on Great Lakes Symposium on VLSI, 2019. 417–422

9 Valavi H, Ramadge P J, Nestler E, et al. A 64-Tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute. IEEE J Solid-State Circ, 2019, 54: 1789–1799

10 Seshadri V, Lee D, Mullins T, et al. Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Boston, 2017. 273–287

11 Li S, Niu D, Malladi K T, et al. DRISA: a DRAM-based reconfigurable in-situ accelerator. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Boston, 2017. 288–301

12 Angizi S, Fan D. ReDRAM: a reconfigurable processing-in-DRAM platform for accelerating bulk bit-wise operations. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Westminster, 2019. 1–8

13 Kautz W H. Cellular logic-in-memory arrays. IEEE Trans Comput, 1969, 18: 719–727

14 Stone H S. A logic-in-memory computer. IEEE Trans Comput, 1970, 19: 73–78

15 Singh G, Chelini L, Corda S, et al. Near-memory computing: past, present, and future. Microprocessors Microsyst, 2019, 71: 102868

16 Jeddeloh J, Keeth B. Hybrid memory cube new DRAM architecture increases density and performance. In: Proceedings of Symposium on VLSI Technology (VLSIT), 2012

17 Dong U L, Kyung W K, Kwan W K, et al. 25.2 A 1.2 V 8 Gb 8-channel 128 GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29 nm process and TSV. In: Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, 2014. 432–433

18 Devaux F. The true processing in memory accelerator. In: Proceedings of IEEE Hot Chips 31 Symposium (HCS), Cupertino, 2019. 1–24

19 Consortium. Hybrid memory cube specification 2.1, 2015

20 Zhuo Y, Wang C, Zhang M, et al. GraphQ: scalable PIM-based graph processing. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium, 2019. 712–725

21 He M, Song C, Kim I, et al. Newton: a DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning. In: Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Athens, 2020. 372–385

22 Boroumand A, Zheng H, Mutlu O, et al. CoNDA: efficient cache coherence support for near-data accelerators. In: Proceedings of ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA), Phoenix, 2019. 629–642

23 Ahn J, Yoo S, Mutlu O, et al. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In: Proceedings of ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, 2015. 336–348

24 Cheng L, Muralimanohar N, Ramani K, et al. Interconnect-aware coherence protocols for chip multiprocessors. In: Proceedings of the 33rd International Symposium on Computer Architecture (ISCA), Boston, 2006. 339–351

25 Baer J L, Wang W H. On the inclusion properties for multi-level cache hierarchies. In: Proceedings of the 15th Annual International Symposium on Computer Architecture, Honolulu, 1988. 73–80

26 Imani M, Gupta S, Rosing T. Ultra-efficient processing in-memory for data intensive applications. In: Proceedings of the 54th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, 2017. 1–6

27 Azarkhish E, Rossi D, Loi I, et al. Design and evaluation of a processing-in-memory architecture for the smart memory cube. In: Proceedings of International Conference on Architecture of Computing Systems. Berlin: Springer, 2016

28 Farmahini-Farahani A, Ahn J H, Morrow K, et al. NDA: near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In: Proceedings of IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, 2015. 283–295

29 Boroumand A, Ghose S, Patel M, et al. LazyPIM: an efficient cache coherence mechanism for processing-in-memory. IEEE Comput Arch Lett, 2017, 16: 46–50

30 Xu S, Chen X, Wang Y, et al. 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 (ASPDAC'19). New York: Association for Computing Machinery, 2019. 140–145

31 Xu S, Wang Y, Han Y, et al. PIMCH: cooperative memory prefetching in processing-in-memory architecture. In: Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, 2018. 209–214

32 Nesbit K J, Smith J E. Data cache prefetching using a global history buffer. IEEE Micro, 2005, 25: 90–97

33 Ishii Y, Inaba M, Hiraki K. Access map pattern matching for high performance data cache prefetch. J Instruction-Level Parallelism, 2011, 13: 499–500

34 Ahn J, Hong S, Yoo S, et al. A scalable processing-in-memory accelerator for parallel graph processing. In: Proceedings of 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, 2015. 105–117

35 Xu S, Chen X, Han Y, et al. TUPIM: a transparent and universal processing-in-memory architecture for unmodified binaries. In: Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI'20). New York: Association for Computing Machinery, 2020. 199–204

36 Oliveira G F, Santos P C, Alves M A Z, et al. A generic processing in memory cycle accurate simulator under hybrid memory cube architecture. In: Proceedings of 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Pythagorion, 2017. 54–61

37 Kim Y, Yang W, Mutlu O. Ramulator: a fast and extensible DRAM simulator. IEEE Comput Arch Lett, 2016, 15: 45–49

38 Singh G, Gómez-Luna J, Mariani G, et al. NAPEL: near-memory computing application performance prediction via ensemble learning. In: Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, 2019. 1–6

39 Xu S, Chen X, Wang Y, et al. PIMSim: a flexible and detailed processing-in-memory simulator. IEEE Comput Arch Lett, 2019, 18: 6–9

40 Binkert N, Beckmann B, Black G, et al. The GEM5 simulator. SIGARCH Comput Archit News, 2011, 39: 1–7

41 Sanchez D, Kozyrakis C. ZSim: fast and accurate microarchitectural simulation of thousand-core systems. SIGARCH Comput Archit News, 2013, 41: 475–486

42 Coudrain P, Charbonnier J, Garnier A, et al. Active interposer technology for chiplet-based advanced 3D system architectures. In: Proceedings of 2019 IEEE 69th Electronic Components and Technology Conference (ECTC), Las Vegas, 2019. 569–578

43 Shen X, Xia Z, Yang T, et al. Hydrogen source and diffusion path for Poly-Si channel passivation in Xtacking 3D NAND flash memory. IEEE J Electron Dev Soc, 2020, 8: 1021–1024