An Overview of Processing-in-Memory Circuits for Artificial Intelligence and Machine Learning

Donghyuk Kim, *Graduate Student Member, IEEE*, Chengshuo Yu, *Graduate Student Member, IEEE*, Shanshan Xie, *Graduate Student Member, IEEE*, Yuzong Chen, Joo-Young Kim, *Senior Member, IEEE*, Bongjin Kim, *Senior Member, IEEE*, Jaydeep P. Kulkarni, *Senior Member, IEEE*, and Tony Tae-Hyoung Kim, *Senior Member, IEEE*

***Abstract*— Artificial intelligence (AI) and machine learning (ML) are revolutionizing many fields of study, such as visual recognition, natural language processing, autonomous vehicles, and prediction. Traditional von-Neumann computing architecture with separated processing elements and memory devices have been improving their computing performances rapidly with the scaling of process technology. However, in the era of AI and ML, data transfer between memory devices and processing elements becomes the bottleneck of the system. To address this data movement issue, memory-centric computing takes an approach of merging the memory devices with processing elements so that computations can be done in the same location without moving any data. Processing-In-Memory (PIM) has attracted research community's attention because it can improve the energy efficiency of memory-centric computing systems substantially by minimizing the data movement. Even though the benefits of PIM are well accepted, its limitations and challenges have not been investigated thoroughly. This paper presents a comprehensive investigation of state-of-the-art PIM research works based on various memory device types, such as static-random-access-memory (SRAM), dynamic-random-access-memory (DRAM), and resistive memory (ReRAM). We will present the overview of PIM designs in each memory type, covering from bit cells, circuits, and architecture. Then, a new software stack standard and its challenges for incorporating PIM with the conventional computing architecture will be discussed. Finally, we will discuss various future research directions in PIM for further reducing the data conversion overhead, improving test accuracy, and minimizing intra-memory data movement.**

***Index Terms*— Artificial intelligence, machine learning, processing-in-memory, neural networks.**

Manuscript received December 20, 2021; revised February 16, 2022; accepted March 10, 2022. Date of publication March 17, 2022; date of current version June 13, 2022. This article was recommended by Guest Editor H. H.-C. Iu. *(Corresponding author: Tony Tae-Hyoung Kim.)*

Donghyuk Kim and Joo-Young Kim are with the Electrical Engineering Department, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea.

Chengshuo Yu is with the Electrical and Electronic Engineering Department, Nanyang Technological University, Singapore 639798, and also with the Institute of Microelectronics, A*STAR, Singapore 138634.

Shanshan Xie and Jaydeep P. Kulkarni are with the Electrical and Computer Engineering Department, The University of Texas at Austin, Austin, TX 78712 USA.

Yuzong Chen is with the Electrical and Computer Engineering Department, National University of Singapore, Singapore 119077.

Bongjin Kim is with the Electrical and Computer Engineering Department, University of California at Santa Barbara, Santa Barbara, CA 93106 USA.

Tony Tae-Hyoung Kim is with the Electrical and Electronic Engineering Department, Nanyang Technological University, Singapore 639798 (e-mail: thkim@ntu.edu.sg).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2022.3160455.

Digital Object Identifier 10.1109/JETCAS.2022.3160455

I. Introduction

EMERGING applications such as machine learning and artificial intelligence have increased interest in low power hardware accelerators for processing parallel data in neural networks. Multiple-and-Accumulate (MAC) operation is a key arithmetic function in neural networks. Typical computing architecture named von-Neumann architecture consists of separated processing elements and memory. To execute MAC operation, huge amount data need to be transferred between processing elements and memory through interconnect channels with large loading. It is well-known that this extremely frequent data communication consumes very high power, which is a challenge in energy-efficient edge computing systems. Processing-in-memory (PIM) architectures have been reported to overcome the above bottleneck called memory wall [1]–[11]. In PIM architectures, each processing element has a computing circuit and a memory, reducing the frequency of the data transfer from/to external memory. Since the power-hungry data transfer is minimized, the PIM architecture can improve the energy efficiency by orders of magnitude. To this end, we investigate the state-of-the-art PIM research works based on the memory type: SRAM-based PIM, DRAM-based PIM, and ReRAM-based PIM.

SRAM has become the most popular selection compared to other candidates for implementing PIM macro [3]–[5], [12]–[20], thanks to the simple operation mode and the mature technology. However, the area of the SRAM cell is larger than other candidates like DRAM and ReRAM, which results in the lower memory density.

DRAM-based PIM is an attractive solution to accelerate large-sized machine learning models with its large memory capacity, yet the high density of DRAM cells deters the realization of the DRAM-based PIM. However, its feasibility has been proved with the first fabricated PIM chip on 3-d stacked DRAMs. In addition, many approaches [21]–[28] apply in-memory processing on a different level of logic integration to mitigate the high density of DRAM cells.

ReRAM-based PIM has become increasingly attractive in energy-efficient accelerators for edge computing where batteries or energy harvesting devices are the primary power sources [29]–[36]. Particularly, edge computing devices process data rarely and mostly stay in the standby mode, which makes ReRAM-based PIM with moderate performance a promising solution for edge computing. However, ReRAM technologies

TABLE I
SUMMARY OF STATE-OF-THE-ART SRAM-BASED PIMS

Paper	Year	Tech.	Bitcell	Function	Key Feature
6T ML Classifier	JSSC17	130nm	Standard 6T	MUL	Compact Bitcell
Split-6T PIM BNN	SOVC19	28nm	Split WL 6T	XNOR	Compact Bitcell
Charge-based PIM	JSSC19	65nm	8T+1C	XNOR	Less Variation
CONV-SRAM	JSSC19	65nm	10T	MUL	Wide Dynamic Range
C3SRAM PIM	JSSC20	65nm	8T+1C	XNOR	Less Variation
XNOR-SRAM	JSSC20	65nm	12T	XNOR	Wide Dynamic Range
8T SRAM PIM	CICC20	65nm	8T	MUL	Diff. Read Bitline
T8T SRAM PIM	JSSC20	55nm	T8T	MUL	Large Signal Margin
7nm SRAM PIM	JSSC21	7nm	Foundry 8T	AND	Foundry Cell
6T SRAM 8b PIM	ISSCC21	28nm	6T+SILMC	MUL	High Precision

are not mature yet, and the ReRAM-based PIM needs to overcome various design issues for wider employment.

The software stack is a key enabler for the adoption of PIM as a new mainstream device that could potentially outperform the conventional von-Neumann computer in usability and performance. PIM, unlike conventional memory devices, is no longer a passive device because it can perform logic operations. It implies that a fundamental change in the software stack is necessary for optimization of a real PIM system. Thus, we must revisit the whole software stack, including the application, framework, run-time, and driver. This paper discusses the challenges in modifying the software stack for PIM, such as offloading executions, data mapping, scheduling, and cache coherence.

The remainder of this paper is organized as follows. Section II introduces state-of-the-art analog and digital SRAM PIM design. Section III reviews various DRAM PIM architectures. In Section IV, we review ReRAM-based PIMs. Section V discusses the software stack for PIM and the challenges in adopting PIM, followed by the future works and trends in Section VI. Finally, Section VII concludes the article.

II. SRAM PIM

This section introduces the conventional SRAM background, including read/write operation and overall architecture, and SRAM-based PIM macros. First, the SRAM has the most intuitive operation theory and mature manufacturing technology, which reduces the difficulty of building a PIM macro and makes it become the most popular candicate for constructing an artificial neural networks accelerator. Then, several SRAM-based works are introduced with tradeoff analysis for solving the challenges in the transition from the pure memory array to the PIM macro. Note that the discussed works include both analog and digital domains for providing a complete knowledge framework. Analog PIM macro shows high energy/area efficiency performance while expressing limitations in flexibility and classification accuracy of the artificial neural networks. On the other hand, digital PIM macro demonstrates low efficiency and throughput but adept in avoiding physical variations. The summary of state-of-the-art SRAM-based PIMs is presented in Table I.

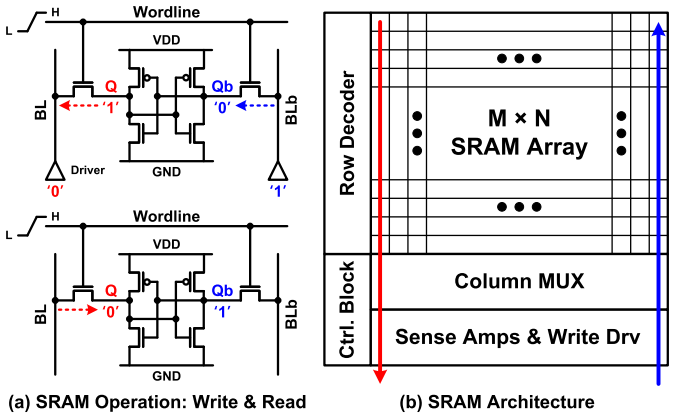


Fig. 1. SRAM background: (a) Operation; and (b) Architecture.

A. Background of SRAM Operation and Architecture

Fig. 1(a) shows the write and read operation of the conventional 6T SRAM cell. The six transistors form two cross-coupled inverters and a pair of access switches. The written data is loaded to bitlines before turning on the wordline. Then, we turn on the access transistors to let the data in the bitline pair go to the storage node of the SRAM cell. Note that the path of writing '0' limits the SRAM write operation since the NMOS access transistor has a higher ability for passing low voltage than high voltage. Therefore, the access transistor needs to be stronger than the PMOS transistor to lower Q below the threshold voltage of the inverters inside the SRAM cell and guarantee the success writing of '0'. For the SRAM write operation, the wordline is turned on after pre-charging the bitline pair. Then, one bitline decreases based on the SRAM storage data. In the end, an output is generated through a sense amplifier that amplifies the voltage difference of the bitline pair. The sample SRAM architecture, shown in Fig. 1(b), comprises a memory array, row decoders, column multiplexers, sense amplifiers, write drivers, and a controller. During write operation, write drivers send to written data to the selected bitline pairs while the unselected bitlines are pre-charged to the high voltage before activating the selected wordline. Only one cell is selected during read operation, and the corresponding bitline pair is connected to a sense amplifier through a column multiplexer.

B. Analog SRAM-Based Processing-in-Memory

A standard 6T SRAM cell [12] can build a memory macro for processing MAC operations, as shown in Fig. 2(a). Note that a binary input is applied to a wordline (WL), and a binary weight is stored in the SRAM internal node when the standard 6T SRAM operates as MAC mode. The signal applied to WL has two patterns: 1) A DC low voltage for input zero; 2) A short positive pulse for input one. One binary multiplication is performed in one SRAM cell right after the input signal is applied to WL, and the corresponding multiplication result is reflected in a bitline pair (BL and BLb). The multiplication results of all the bitcells in one column are accumulated and contribute to voltage drops in both bitlines. Note that the

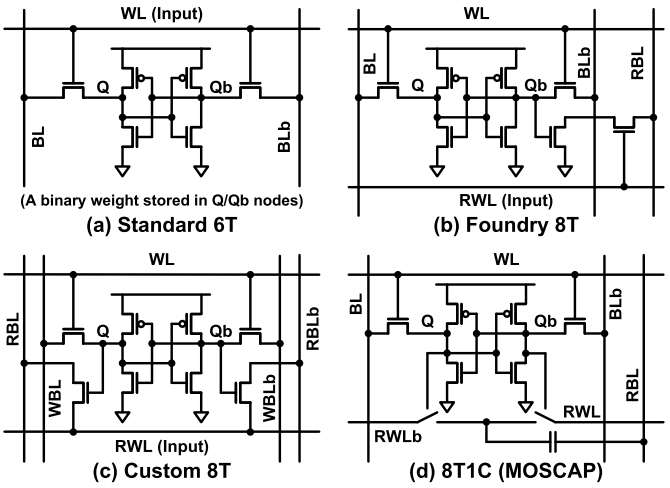


Fig. 2. SRAM-based PIM cells.

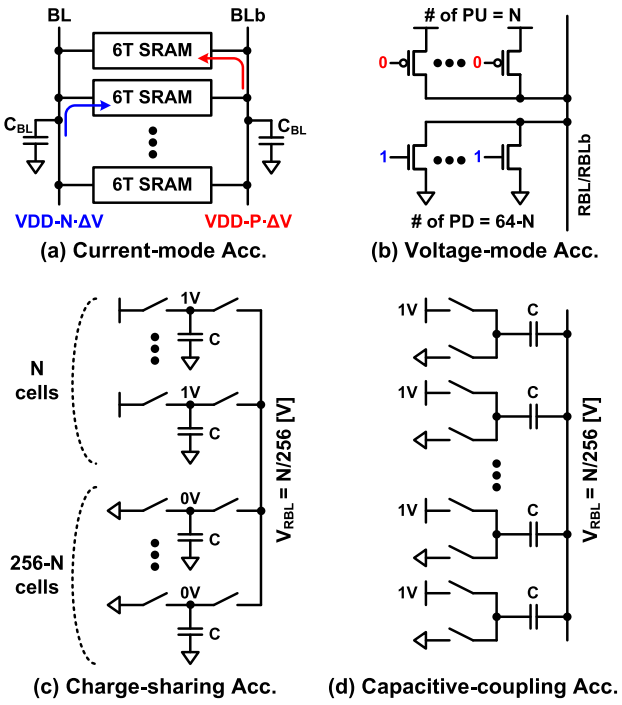


Fig. 3. SRAM-based accumulation mode.

standard 6T SRAM still suffers from a disturbance issue due to the shared write and read path.

Recently, several different bitcells have been proposed to solve the intrinsic drawback of the conventional 6T SRAM when processing MAC operations. In Fig. 2(b), A foundry 8T SRAM cell [13] successfully solves the read disturbance issue. Two additional NMOS transistors are added to standard 6T with a series connection for creating an independent read bit-line (RBL) discharging path to the ground. The multiplication result depends on the weight value stored in an SRAM internal node (Qb) and the input value represented by the voltage level of a read wordline (RWL). However, the foundry 8T only supports a single-ended read operation with a larger bitcell area. A custom 8T SRAM cell [14] was developed to improve the diversity of MAC operation (i.e., weight '±1' multiply input '1/0') compared to the foundry 8T (i.e., weight '1/0' multiply input '1/0'), as shown in Fig. 2(c). This design still suffers from limited dynamic range and non-idealities due to the current-based accumulation scheme. A differential voltage-mode SRAM-based PIM cell [16] has been proposed using two CMOS inverters and one XNOR upon the standard 6T SRAM to improve the dynamic range. However, the increased cell size and huge non-idealities remain to be solved. An 8T SRAM with one metal-oxide-metal capacitor (MOMCAP) [4] is proposed to minimize the residual analog non-idealities thanks to the contribution of the embedded passive capacitors, as shown in Fig. 2(d). It also decouples MAC operation for eliminating SRAM disturb issue. Nevertheless, the area overhead of the bitcell is significant due to the extra two transistors and a capacitor.

Fig. 3 describes the popular operation types of analog SRAM-based PIM macros: current-mode, voltage-mode, charge-sharing, and capacitor-coupling. In current-mode (Fig. 3(a)), the element-wise binary multiplication results from each 6T SRAM cells in the same column are accumulated with the representation of generating or not generating discharge current and lead to an aggregated voltage drop in bitline pair [3], [12]–[15]. Note that a limited dynamic range needs to be set to guarantee the linearity of accumulation results. Voltage-mode SRAM-based MAC operation scheme [5], [16] is proposed to significantly improve the dynamic range (i.e., rail-to-rail). A shared RBL is driven by parallel pull-up or pull-down paths depending on the multiplication results of each bitcells, as shown in Fig. 3(b). However, the voltage-mode method suffers from residual non-linearity and variation issues. Passive capacitors have been used to implement charge-sharing [4] and capacitive-coupling [17] approaches for minimizing non-linearity issues and variations due to analog computations. As illustrated in Fig. 3(c) and 3(d), the charge-sharing scheme needs one more switch in each cell and takes one more cycle for accumulation operation compared to the capacitive-coupling scheme. Although the charge-domain offers higher energy efficiency and throughput performances, it still suffers from larger area overhead due to the implementation of capacitors. Moreover, the charge injection issue needs to be carefully considered when designing a charge-domain PIM macro.

C. Digital SRAM-Based Processing-in-Memory

Digital PIM architecture is developed to solve the issues in analog PIM works such as analog-to-digital/digital-to-analog converter (ADC/DAC) overhead and PVT variation induced non-linearity computation. The computation in digital PIM architecture is entirely digital to eliminate the effects of any physical variation, and the data conversion is no longer needed.

Fig. 4 describes the digital PIM macro [18], which can support reconfigurable inputs, weights, and outputs. Each PIM cell consists of a 6T SRAM, an XNOR gate, and a 1-bit full-adder. The PIM cells can be stacked together to construct 1-16b unit column MAC based on the required computation precision. Note that different functions are assigned depending

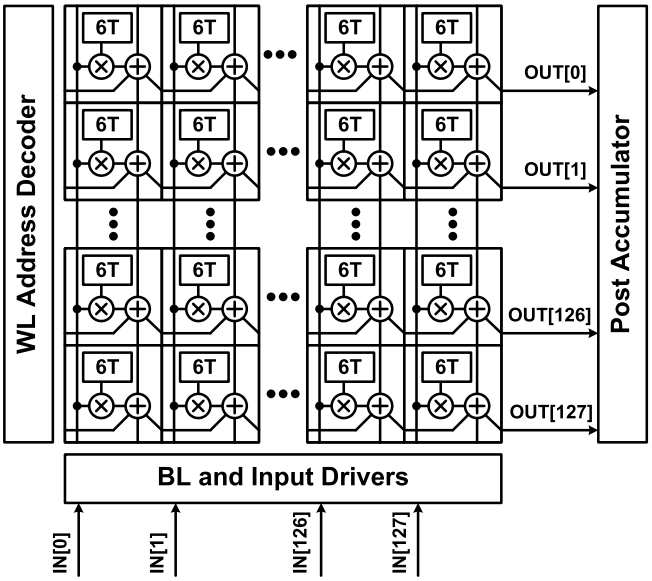


Fig. 4. Digital SRAM-based processing-in-memory macro [18].

TABLE II
SUMMARY OF STATE-OF-THE-ART DRAM-BASED PIMS

Paper	Year	Level	Function	Key Feature
AMBIT [23]	MICRO'2017	Cell	AND,OR,NOT	TRA,DCC
DRISA [24]	MICRO'2017	Cell	NOR,ADD	3T1C-NOR,1T1C-NOR/MIX,Adder
Newton [21]	MICRO'2020	Bank	MAC	Multiplier, Adder Tree
HBM-PIM [22]	ISCA'2017	Bank	SIMD	SIMD (Multiplier, Adder)
Neurocube [26]	ISCA'2016	3D	FC, Conv	MAC Units
Tetris [27]	ASPLOS'2017	3D	FC, Conv	PE Array, Data Reuse
iPIM [28]	ISCA'2020	3D	SIMD	SIMD Unit, SIMB ISA

on the location of the PIM cell within the unit column. However, the digital PIM macro suffers from the hardware redundancy caused by the memory and computation blocks, resulting in large unit PIM cells and low memory density.

The latest digital PIM macro [19] implements 256x 4bit weights in each column and produces 64x 12bit MAC outputs. Each PIM cell comprises a fused 6T SRAM and a 2-input NOR gate and processes binary multiplication results. Then, the accumulation operation is processed in the dedicated adder tree. The weight precision can be further extended to 8-16bit using multiple macros while spending more area.

III. DRAM PIM

This section introduces the conventional architecture and operation of DRAM as well as various PIM architectures and their implementations. The DRAM architecture has been developed in focusing on cell density, where each cell has a simple structure of a single transistor and a capacitor. Although the structural simplicity of the memory cells inspires interesting ideas in integrating logic for DRAM-based PIMs, its tight physical constraint has been a major challenge. Many previous researches tackle different levels of DRAM architecture to address this issue. Recent DRAM-based PIM architectures are summarized in Table II. As illustrated in Fig. 5, we differentiate the DRAM-based PIMs into three categories based on the level of logic integration: cell-level, bank-level, and 3-d level. First, DRAM cell-level PIM integrates low level transistor

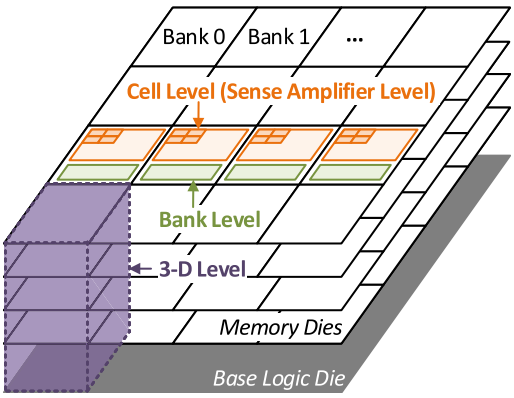


Fig. 5. Different levels of DRAM PIM architectures.

logic with bitline sense amplifiers to conduct bulk bitwise operations, taking advantages of the memory bank's whole internal bandwidth. Second, the bank-level PIM integrates high level processing logic after the column decoder in each bank. This technique cannot use the maximum internal bandwidth as compared to cell-level PIMs, but it does open up the possibility of DRAM-based PIMs [21], [22] by utilizing a larger logic area. Third, 3-d level PIM utilizes 3-d stacked memory with the base logic die, such as Hybrid memory cube (HMC). It integrates the compute logic die to the stacked memory dies where two entities are interconnected by through-silicon via (TSV), providing energy-efficient and high-bandwidth communication between them. However, due to the strict physical and timing constraints of 3-d stacked dies, the realization of 3-d level PIM remains a challenge.

A. Background of DRAM Architecture and Operation

A DRAM chip consists of the memory cells to store charges as well as the control logic and data I/O circuitry to support its operation. It consists of multiple DRAM banks, each of which is made up of stacks of DRAM mats. They are the basic 2-dimensional array structure of DRAM cells where each of the cells, composed of a single transistor and a capacitor, stores a bit value. In order to read/write cell values in a mat, the row decoder receives a row address and selects a single wordline. Then, the transistors of every DRAM cell connected to the wordline are activated and the values are read/write from/to the capacitors. To elaborate further, the capacitor in each DRAM cell begins to share the charge with the bitline, which is pre-charged to half V_{DD} when the transistor is activated. The charge sharing induces little voltage variations on the bitlines, and the bitline sense amplifiers amplify the variations to recognizable logic levels. Once an entire row is amplified, the column decoder specifies one or more bitlines to transfer the corresponding data to the IO pads.

B. Cell-Level Processing-in-Memory

The cell-level PIM integrates logic at the bit-line sense amplifiers to execute bulk bitwise operations on multiple rows while maximizing internal memory bandwidth. However, logic integration in a DRAM cell with an extremely narrow pitch

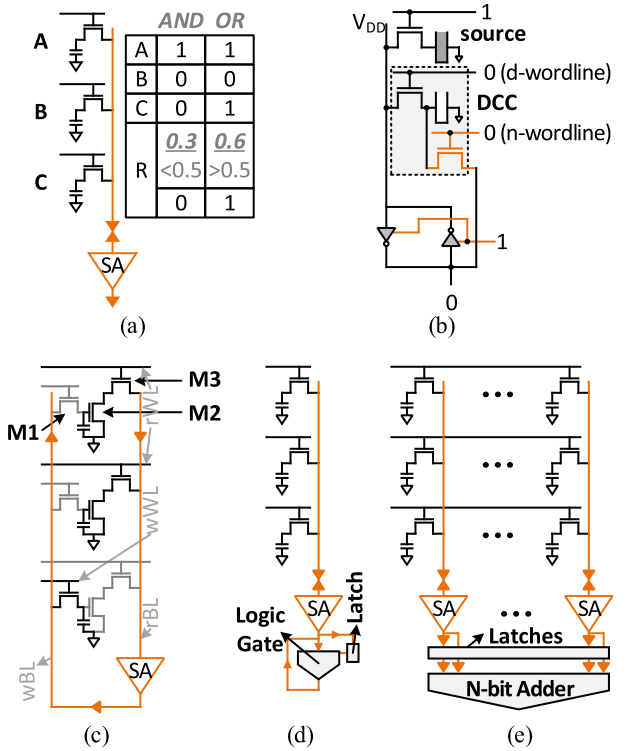


Fig. 6. (a) AMBIT TRA, (b) AMBIT DCC, (c) DRISA 3T1C-NOR, (d) DRISA 1T1C-NOR/MIX, and (e) DRISA 1T1C-ADDER.

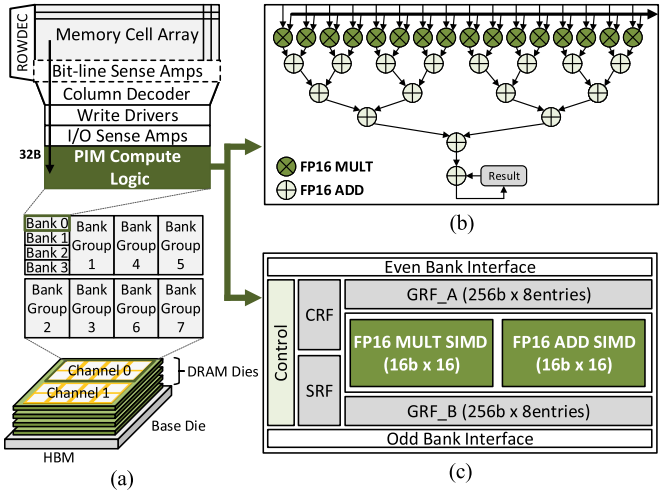


Fig. 7. (a) Bank-level PIM organization, (b) Newton compute logic, and (c) HBM-PIM compute unit.

may not be feasible. The pitch of a single cell is already optimized for only a single transistor and a capacitor, adding more transistors to it can be challenging. To address this difficulty, previous research has focused on executing simple gate-level operations (i.e., AND, OR, NOR, and NOT) without explicitly implementing logic gates.

AMBIT [23] proposes the triple row access (TRA) to execute AND and OR operations without adding any transistors to the sense amplifiers, as illustrated in Fig. 6(a). TRA only requires modifying the control logic of DRAM to activate three wordlines simultaneously. If three wordlines are activated, three cells sharing the same bitline will share their charges. The result of the TRA becomes 1 if the number of 1s in the three cells are more than or equivalent to 2. $R = AB + BC + CA = C(A + B) + \overline{C}(AB)$ represents the Boolean expression of the TRA, where A,B and C are values in three cells and R is the result. By presetting C to 1 or 0, TRA enables to execute either OR or AND operation on two rows. In addition, AMBIT proposes a row of dual-contact cells (DCCs) to execute NOT operation. Each DCC contains an additional pair of a wordline and a transistor to move the inverted value of the sense amplifier to the cell, as illustrated in Fig. 6(b). However, the actual implementation of DCC may not be feasible as it requires one more wordline and a transistor to fit the pitch of a DRAM cell.

DRISA [24], on the other hand, proposes three modified DRAM cell architectures: 3T1C-NOR, 1T1C-NOR/MIX, and 1T1C-ADDER, as shown in Fig. 6(c), 6(d), and 6(e), respectively. 1T1C-NOR/MIX and 1T1C-ADDER integrate latches with simple logic such as NOR or other logic gates and a parallel adder, respectively, below sense amplifiers. Even with these simple logic gates, it is difficult to integrate them within the DRAM cell pitch. However, 3T1C-NOR addresses this difficulty by using the early DRAM cell design of 3T1C cell, which has two separated wordlines for each cell for both read and write operation. It can naturally perform NOR operation without any modification to the cell design. If two M3 transistors are enabled by activating the two corresponding wordlines, two M2 transistors sharing the same bitline turns into the NOR gate. In addition, DRISA proposes transistor-level shifter circuits under the bitline sense amplifiers for data movement to the adjacent bitlines, allowing for more sophisticated computations, such as selection, multiplication, and addition. As the cell-level PIM can be optimized further for its large input operand size of an entire row, AMBIT and DRISA adopt the method of RowClone-FPM (Fast Parallel Mode) [25]. In addition, they modify the row decoders and drivers to enable multi-row activation to speed up the execution. AMBIT also proposes a fused complex command primitive of activate-activate-precharge (AAP) to reduce the number of required commands, thus reducing the total latency.

C. Bank-Level Processing-in-Memory

Aforementioned, the cell-level PIM is the best in utilizing the memory bank's whole internal bandwidth; however, it is the most challenging due to the severe area constraint, where the cell pitch continues to decrease as the DRAM technology advances. Two feasible bank-level PIMs, Newton [21] and HBM-PIM [22], solve this difficulty by integrating processing logic after column decoder and selector, allowing the logic to benefit the entire width of the cell array, as shown in Fig. 7(a). They also exploit the bank parallelism that enables multiple banks at the same time to compensate for the internal bandwidth loss from not utilizing the entire row. Especially, HBM-PIM proves it by fabricating the first PIM chip ever in HBM using a 20nm DRAM process.

Maximizing the benefit of PIM, Newton exclusively focuses on memory-bound deep learning models such as recommendation systems (e.g., Facebook's DLRM [37]) and language models (e.g., Google's BERT [38] and OpenAI's GPT [39]),

and thus proposes a fixed data flow accelerator that computes matrix-vector multiplication effectively. Fig. 7(b) illustrates the overall architecture of Newton in a single DRAM die. Each bank includes 16 multipliers, 16 adders in a reduction tree, and a 16-bit accumulator register. Two input operands of the multipliers come from the memory cell array after the column selector and the global buffer, which broadcasts an input vector to all memory banks. In addition, Newton proposes customized PIM commands, such as GWRITE, G_ACT, COMP, and READRES, to exploit the bank parallelism and reduce the latency. Newton can load the global buffer with the input vector data using GWRITE command. G_ACT command activates multiple banks. COMP command simultaneously executes in-bank MAC operations using the multipliers and adder tree in all banks. Newton can read all the results from the banks using READRES command.

On the other hand, HBM-PIM proposes the programmable compute unit (PCU) to support the complex computational requirements of AI applications. HBM-PIM integrates a PCU block per two banks, where they share the PCU block with two distinct IO sense amplifiers. Fig. 7(c) illustrates the block diagram of the PCU, consisting of an interface unit, execution unit, and register group. The interface unit receives control and data signals from the memory's command controller. The execution unit includes single-instruction-multiple-data (SIMD) fashioned 16 FP16 multipliers and adders. The register group includes the command register file (CRF), general-purpose register file (GRF), and scalar register file (SRF). In addition, the PIM controller is integrated to support the programmability of the PCU as well as the seamless integration with the host. The PIM controller determines the mode between the normal and PIM. It asserts PIM mode if an activation command of the specific row address comes in. If the PIM mode is asserted, PCU executes the PIM instructions pre-stored in the CRF. For the following instructions, the program counter of the CRF increments by one per every DRAM's read command.

D. 3-d Processing-in-Memory

Unlike bank-level PIM (e.g., Newton and HBM-PIM), where the logic is placed next to the memory cell arrays in a memory die, the 3-d PIM involves both the logic die in the bottom and the stacked memories on top. The 3-d PIM is more sophisticated than HBM, allowing for more energy-efficient data communications between the logic die and the memory dies. However, due to the strict physical and time constraints among 3-d stacked dies, realizing 3-d PIM can be challenging. Only simulation is used to evaluate all of the proposed works: Neurocube [26], Tetris [27], and iPIM [28]. These works explore accelerating deep neural network computations with the 3-d PIM using hybrid memory cube (HMC). They exploit the vault structure of the HMC, where the HMC stack is partitioned vertically into sixteen vaults, as illustrated in Fig. 8(a).

As illustrated in Fig. 8(b), Neurocube puts a programmable neurosequence generator (PNG) and processing element (PE) on the logic die per each vault. The PNG manages the data access sequences within the vault using the vault controller and

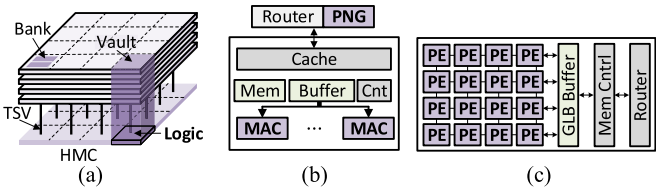


Fig. 8. (a) Hybrid memory cube, (b) Neurocube PE block diagram, and (c) Tetris PE array block diagram.

sends data to the PE. Each PE contains a row of multiple MAC units to compute the data in parallel. A 2-d mesh network-on-chip connects all PEs to enable flexible data mappings and operations inter-vault communications. On the other hand, Tetris focuses on the input data reuse of the neural network models. As illustrated in Fig. 8(c), Tetris integrates a PE array structure with a global buffer per vault to maximize the data reuse while maintaining the base architecture from Neurocube.

iPIM explores on boosting the effective compute bandwidth and reducing the energy spent from data movements via TSV by combining the 3-d PIM and the bank-level PIM. The iPIM architecture decouples the control module, which locates on the logic die, from the execution units, which locates on the memory dies. By separating the control and execution, iPIM gains abundant bank-level bandwidth to maximize the parallel execution of processing engines on the memory dies. The iPIM architecture places one iPIM control core in the logic die of each vault, while it places one process group (PG) in the memory die of each vault, next to the bank. The iPIM control core manages flexible intra/inter-vault data communication and executes instruction decoding with the support of the single-instruction-multiple-bank (SIMB) instruction set architecture (ISA). On the other hand, PG includes PEs to perform memory-bound operations near the bank to maximize the parallel execution.

IV. ReRAM PIM

This section introduces the background of ReRAM and how the MAC operation is implemented in ReRAM-based PIM architectures. Design challenges associated with ReRAM PIMs and state-of-the-art ReRAM PIM works for the MAC operation are discussed. In addition, we describe three new directions for ReRAM PIMs to motivate future research.

A. ReRAM Basics

ReRAM is one type of emerging non-volatile memory under active research in both academia and industry [29]. It has good read and write performance, low programming voltages, as well as good scalability and compatibility to the CMOS fabrication process. For digital applications, one ReRAM device can be programmed to either a high-resistance-state (HRS) or a low-resistance-state (LRS) for storing a binary value. Because of the analog nature of ReRAM, some works also employed ReRAM for analog storage to improve memory capacity. For example, Chang *et al.* [30] and Lin *et al.* [31] implemented ReRAM-based ternary content addressable memory by using a middle-resistance-state to represent "don't care".

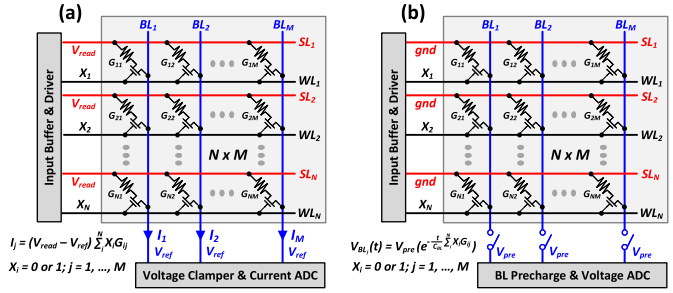


Fig. 9. ReRAM-based PIM architectures for MAC based on: (a) Current-mode sensing and (b) Voltage-mode sensing.

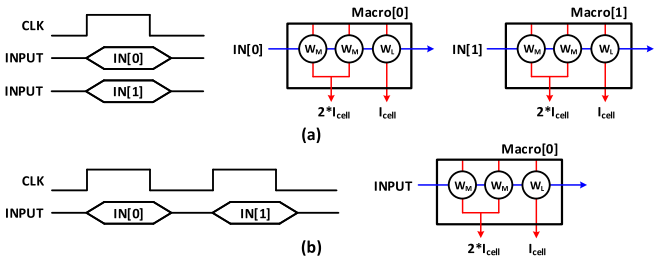


Fig. 10. Multiplication of 2-bit inputs and 2-bit weights in digital ReRAM-based PIM: (a) One cycle and multiple macros and (b) Multiple cycles and one macro.

Wu *et al.* [32] used the ReRAM device as an approximate accumulator by applying multiple reset pulses to gradually change the ReRAM state from LRS to HRS. In addition, analog ReRAM has also been utilized in PIM for neural networks with good error tolerance [34], [35], [40].

B. Multiply-and-Accumulate in ReRAM-Based PIM

Fig. 9 illustrates how the MAC operation is performed in a general ReRAM-based PIM architecture. Based on the output sensing mode, ReRAM-based PIM architectures for MAC can be classified into current-mode and voltage-mode. For both types of ReRAM PIMs, the weight in a neural network layer is stored as the conductance (denoted by Gij for the ith row and jth column) of the ReRAM device, while a binary input (0 or 1) is applied through the word-line (WL). As shown in Fig. 9(a), current-mode ReRAM PIM requires the input driver to drive every source-line (SL) to a read voltage V_{read}, and every bit-line (BL) is clamped to a reference voltage V_{ref} by the voltage clamper. The multiplication between one input feature and one weight can be represented by the current through one ReRAM bit-cell. Hence, the dot product between the input vector and the weight matrix can be performed in the analog domain by accumulating ReRAM bit-cell currents from the same column. Finally, each column's current is converted to a digital value by a current ADC. For voltage-mode ReRAM PIM shown in Fig. 9(b), all SLs are grounded, while all BLs are precharged to V_{pre} and left floating. The discharge rate of the BL capacitance C_{BL} is proportional to the dot product between the input vector and the weight matrix. Finally, the resulting BL voltage is converted to a digital value by an ADC.

There exist different trade-offs between current-mode and voltage-mode ReRAM PIMs. Similar to a normal ReRAM memory access, ReRAM PIMs based on current-mode sensing have a larger sensing margin and therefore a faster sensing speed. However, the static current must be present for the entire sensing period, leading to poor energy efficiency compared with voltage-mode sensing that only consumes dynamic power. Due to ReRAM device non-idealities, the sensing margin is normally a more critical design consideration for the target neural network accuracy. Hence, most ReRAM PIM works are based on current-mode sensing [34], [41]–[44].

C. Multi-Bit Multiplication in ReRAM-Based PIM

Although the analog nature of ReRAM allows it to store a multi-bit weight for improved memory capacity [34], [35], [40], most ReRAM PIM works employ digital ReRAM with two distinct states: HRS and LRS [41]–[45]. This is because: (1) ReRAM-based PIM shows worse MAC accuracy compared to DRAM and SRAM counterparts because of large ReRAM device variations and offset current caused by ReRAM devices in HRS; (2) Controlling ReRAM with more resistance states requires more complicated write-verification circuit, leading to considerable overhead [33], [40]. In general, for PIM architectures based on digital ReRAM, the multiplication of multi-bit inputs and weights can be implemented in two different ways named 'Parallel-Input-Parallel-Weight (PIPW)' and 'Serial-Input-Parallel-Weight (SIPW)' [42]. Consider an example of multiplication between 2-bit input and 2-bit weight, in PIPW, two macros store the same weight and the 2-bit input (IN[0]: LSB and IN[1]: MSB) is applied to the macros as shown in Fig. 10(a). Here, each macro receives either IN[0] or IN[1]. The 2-bit weight is implemented with three ReRAM devices, two for MSB and one for LSB. Therefore, the BL current for MSB is 2× of the BL current for LSB. These two BL currents are merged to generate a multiplication result. In SIPW, a multi-bit input is applied to a macro bit by bit over multiple cycles as shown in Fig. 10(b). To merge the currents over multiple cycles, currents from previous cycles need to be stored in binary-weighted capacitors.

Both PIPW and SIPW suffer from large BL current distributions on the read path. To improve the MAC accuracy in presence of ReRAM device variations, extra circuits are needed for merging the split currents from multiple columns, multiple macros, or multiple cycles. Various MAC strategies have been reported to address this challenge [41]–[44]. Table III shows an example of multiplication using one-bit input and a ternary weight [41]. The ternary weight is realized by two arrays (one positive array and one negative array). The weight in the positive array generates '+1 (LRS)' and '0 (HRS)' while that in the negative array produces '−1 (LRS)' and '0 (HRS)'. The final ternary multiplication result is obtained by combining the currents from both positive and negative arrays.

D. Design Challenges in ReRAM PIMs

1) ADC/DAC Overhead: Unlike normal ReRAM where only one row is activated at a time, and sense amplifiers produce the binary comparison result, ReRAM PIM may require DACs for multi-bit inputs and ADCs for multi-bit MAC outputs. A sample power and area breakdown of a

TABLE III
MULTIPLY WITH TERNARY WEIGHT IN RERAM

		Positive Array			Negative array		
Input (IN)	Ternary Weight	Weight (W)	IN×W	I_{MC}	Weight (W)	IN×W	I_{MC}
0	+1	+1(LRS)	0	0	0(HRS)	0	0
1		+1(LRS)	+1	I_{LRS}	0(HRS)	0	I_{HRS}
0	0	0(HRS)	0	0	0(HRS)	0	0
1		0(HRS)	0	I_{HRS}	0(HRS)	0	I_{HRS}
0	-1	0(HRS)	0	0	-1(LRS)	0	0
1		0(HRS)	0	I_{HRS}	-1(LRS)	-1	I_{LRS}

ReRAM-based PIM macro was reported in [34]. The ReRAM array size is 1152 × 128. 2-bit DACs and 8-bit ADCs are considered in the breakdown evaluation. Regarding power consumption, DACs and ADCs consume 24% and 61% of the total power, respectively, which are dominant compared to that of the ReRAM array. Moreover, ADCs also occupy a majority of the area when the required output precision is relatively high (e.g. 91% of the total area at 8-bit output precision). To mitigate ADC energy overhead, the number of ADCs can be reduced by reducing the number of MAC results that are generated at the same time. However, this will increase the number of cycles for processing the same amount of MAC results, degrading the performance of the PIM system. Even though various ADC techniques such as zero-skipping and column ADCs have been developed for better power and area efficiency in SRAM-based PIM [14], [15], the power and area overheads of ADCs are still challenging in ReRAM-based PIM systems.

2) Accuracy Degradation Due to BL Sensing: Besides data conversion, ReRAM PIMs face various challenges such as large BL current distribution and overlap in BL current for different MAC values due to large ReRAM device variations. These problems can significantly degrade the MAC accuracy. For example, consider the ReRAM-based PIM macro in [41] where the maximum MAC value from each column is 9. According to the multiplication mechanism in Table III, the smallest BL current for the MAC value of '+1' occurs when only one WL is turned on in the positive array ('1L0H', one ILRS and no IHRS). However, the maximum BL current for the same MAC value of '+1' is '1L8H' (one ILRS and eight IHRS) where 9 WLs are turned on in the positive array. The selected ReRAM devices in the HRS state will generate I_{HRS} even though the computed MAC value has no difference, leading to a large BL current distribution for the same MAC value. This BL current distribution will degrade the sensing margin, and therefore reduce the MAC accuracy. Such BL sensing challenges must be addressed to improve the MAC accuracy in ReRAM PIMs.

E. State-of-the-Art ReRAM-Based PIMs for MAC Operations

Table IV summarizes recent ReRAM-based PIM prototypes for MAC. From the table, we can observe the following trends: (1) The MAC precision gradually increases for more recent ReRAM-based PIM works to satisfy the accuracy requirement of more complex machine learning algorithms and datasets.

TABLE IV
SUMMARY OF STATE-OF-THE-ART RERAM PIMS FOR MAC

Paper	Tech	Type①	Capacity	Max Precision②	Algorithm	Dataset (Accuracy)	Efficiency (TOPS/W)③
[41], ISSCC'18	65nm	(C, D)	1 Mb	(1b, ternary, 3b)	FCN	MNIST (96.2%)	19.2
[42], ISSCC'19	55nm	(C, D)	1 Mb	(2b, 3b, 4b)	CNN	CIFAR-10 (88.52%)	21.9
[43], ISSCC'20	22nm	(C, D)	2 Mb	(4b, 4b, 11b)	CNN	CIFAR-100 (66.4%)	28.93
[34], ISSCC'20	130nm	(C, A)	158.8 Kb	(1b, 3b, 8b)	FCN	MNIST (94.4%)	78.4
[35], ISSCC'20	130nm	(V, A)	64 Kb	(1b, analog, 1b)	RBM	MNIST (N.A.)	158
[44], ISSCC'21	22nm	(C, D)	4 Mb	(8b, 8b, 14b)	CNN	CIFAR-100 (67.5%)	11.91
[45], ISSCC'21	40nm	(V, D)	64 Kb	(8b, 8b, 20b)	CNN	ImageNet (54.5%)	56.67

FCN: Fully-connected network. CNN: Convolutional neural network. RBM: Restricted Boltzmann machine.
① The type of each work is expressed in the format of (Sensing mode, ReRAM type). The sensing mode is current-mode (C) or voltage-mode (V), while the ReRAM type is analog (A) or digital (D).
② The max precision is expressed in the format of (Input, Weight, Output). E.g. (1b, ternary, 3b) means "1-bit input, ternary weight, 3-bit output".
③ Energy efficiency when the ReRAM PIM macro operates at the max precision.

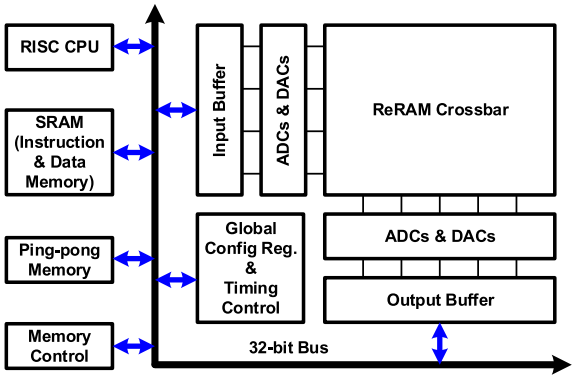


Fig. 11. Block diagram of ReRAM coprocessor [46].

This requires designers to propose more advanced circuit techniques for MAC operations at higher precisions. (2) Most ReRAM-based PIM works for MAC focused on accelerating neural networks, but many other types of machine learning algorithms remained largely untapped and could potentially open new PIM research directions. (3) Digital ReRAM is more preferred compared with analog ReRAM due to large ReRAM device variations. However, with the advancement of device characteristics, the analog ReRAM can become more favored in future ReRAM-based PIMs to reduce the area cost of storing weight data.

F. New Directions for ReRAM PIM Research

1) ReRAM Coprocessor: Besides designing a single ReRAM PIM macro, recent research also starts to focus on integrating the ReRAM PIM macro in a complete computing system. Fig. 11 illustrates the architecture of a fully integrated reprogrammable ReRAM coprocessor for MAC [46]. It consists of a reduced instruction set computer (RISC) processor, multiple SRAMs, a memory controller, and a ReRAM PIM macro with the mixed-signal interface. The RISC processor controls the ReRAM PIM macro through the shared 32-bit bus. Inside the ReRAM PIM macro, a pulse-modulated voltage signal represents the input for each ReRAM crossbar row, and each crossbar column can produce current to represent the vector-matrix multiplication in the analog domain. Inside the mixed-signal interface, one enable-control block and

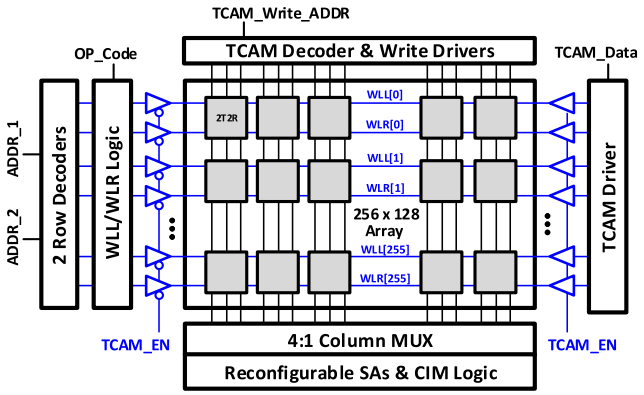


Fig. 12. 2T2R ReRAM macro for versatile PIM functions [49].

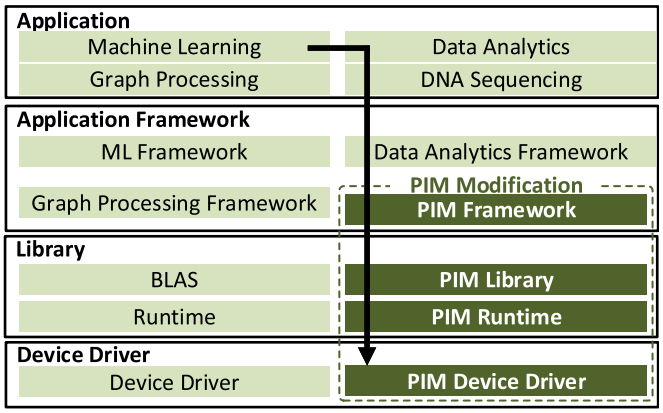


Fig. 13. PIM software stack.

one DAC-select block determine the operation mode of the ADCs/DACs.

2) Security and Reliability Considerations: Apart from the increasing compute precision of MAC, some recent works start to focus on other design considerations such as security and reliability to facilitate the wide adoption of ReRAM PIMs. For example, Li *et al.* [47] proposed a secure ReRAM PIM macro with an embedded XOR cipher for lightweight encryption. The same group also introduced a more reliable ReRAM PIM macro by designing on-chip write-verify and ADC reference generation circuits [48].

3) ReRAM PIM for Versatile Functions: Even though MAC is one of the most common tasks accelerated by PIM, modern data-intensive applications require many other functions such as Boolean logic and search operations. Various ReRAM-based cell structures such as 4T2R and 2T2R are reported to support versatile PIM operations on the same array [49], [50]. Fig. 12 illustrates a 2T2R ReRAM macro for multiple PIM operations [49]. It consists of a ReRAM array, decoders, drivers, reconfigurable sense amplifiers, and PIM logic. This ReRAM PIM macro supports search, Boolean logic and dot product operations. In addition, to mitigate the pseudo-write effect on the ReRAM device during PIM operations, several design techniques are proposed to reduce the stress voltage across the ReRAM device.

V. PIM Software Stack

This section introduces a modified software stack for PIM and the challenges in incorporating PIM with the conventional computer architecture such as CPU or GPU. The main objective of the PIM software stack is to run applications on PIM hardware and further optimize them for PIM. However, there are challenges ahead: offloading executions for PIM, data mapping, scheduling of the kernel executions, and cache coherence. We must overcome these challenges to adopt PIM into the system successfully.

A. PIM Software Stack

Fig. 13 illustrates the PIM software stack with the four modifications: PIM framework, PIM library, PIM runtime, and PIM device driver. First, the PIM framework is a high-level framework with custom operations. For example, the PIM framework for AI workloads includes a wide range of operations from simple operations (i.e., addition and multiplication) to complex operations (i.e., convolutional layer, fully connected layer, batch normalization, and activation functions) as its custom operations. Second, the PIM library is a set of low-level routines that is explicitly called by these custom operations. For the AI workloads, the PIM library includes the basic linear algebra subprograms (BLAS) optimized for PIM, which provides low-level operations for major AI workloads (i.e., vector-vector operations, matrix-vector operations, and matrix-matrix operations). Third, the PIM runtime runs three tasks: 1) optimizes the PIM framework to decide what operations to offload to the PIM hardware and generates PIM instructions, 2) manages the memory usage by the PIM instructions and the operand data, and 3) configures the PIM kernel with the generated PIM instructions. Fourth, the PIM device driver allocates the memory space for PIM operations. HBM-PIM [22] proposes a complete software stack for PIM to accelerate machine learning workloads. It includes custom TensorFlow operations, PIM BLAS library, PIM runtime, and PIM device driver to cover the whole stack. In addition, it proposes a multi-core fashioned programming model that maximally utilizes the PIM execution units.

B. PIM Offloading Execution

Identifying what operations are suitable for PIM is essential since only a few types of operations can benefit from PIM. We need to distinguish PIM-friendly operations from the rest and assign them to PIM. Different strategies are required depending on how PIM logic is designed in memory. We categorized it into two. First, identifying PIM-friendly operations is simple for a specialized PIM logic since it can only execute specific operations. For example, Newton [21] supports a fixed data flow with a row of 16 multipliers followed by a reduction adder tree. This type of architecture can execute the memory-bounded matrix-vector multiplications the best and proves it by the achievements in speed up and power savings. Second, identifying PIM-friendly operations is more challenging for the PIM logic with general-purpose cores. Typically,

PIM copes with memory-intensive operations that have low data locality. It implies that the memory-intensive operations request tremendous data from memory but hardly reuse them. Rather than leaving identifying memory-intensive operations to programmers' manual work, Boroumand *et al.* [51] proposes an efficient tool flow with four criteria to identify PIM-friendly operations: 1) if it consumes the most energy out of all functions in the workload, 2) if its data movement consumes a significant fraction of the total workload energy, 3) if the last cache misses per kilo instruction (MPKI) is greater than 10, and 4) if data movement is the single most significant component of the function's energy consumption. To evaluate the effectiveness, they demonstrate the tool flow in analyzing primary Google consumer workloads, including the Chrome browser, TensorFlow mobile, video playback, and video capture.

C. Data Mapping

The data mapping scheme is a key factor in reducing the data access latency. All levels of PIM architecture can exploit spatial and temporal locality in mapping data. For example, a DRAM-based bank-level PIM can apply this concept. First, the bank-level PIM benefits from locating data in the bank where they are being computed. Otherwise, memory access from a PIM core to its neighbor bank burdens the global data bus. Potentially, if more PIM cores request data from their neighbor bank, it will cause a memory bottleneck. Second, the bank-level PIM benefits from a sequential memory access pattern, which guarantees the shortest data read latency within a bank in DRAM. This access pattern reads data of the same row without additional delays, whereas accessing data of a different row address requires additional delays with DRAM's pre-charge and activation commands. Furthermore, previous research works on data mapping in DRAM-based PIM and ReRAM-based PIM are introduced.

Reference [52] proposes a new programmer-transparent data mapping mechanism that uses consecutive address bits for memory. The mechanism co-locates offloaded code and data in the same PIM computation unit by exploiting predictability in the memory access patterns out of offloaded code blocks. The authors explore various applications to attain predictability, including the backward propagation of the AI workload. The result shows that 85% of all offloaded code blocks have a fixed offset between access addresses, generating a predictable access pattern. With the predictable access pattern, the authors prove that using consecutive address bits reduces internal data movement overhead and maximizes the benefit of PIM.

Reference [53] proposes an optimized weight mapping and dataflow in computing convolutional neural networks on ReRAM-based PIM. Unlike the traditional mapping scheme, which unrolls all of the 3D kernels into a large matrix, the proposed mapping scheme divides the 3D kernel of $K \times K \times D$ into multiple $1 \times 1 \times D$ matrices, where K is the height and width, and D is the channel, then places them into different processing elements (PEs) of PIM. This kernel partitioning decreases the size of the matrix, allowing for more freedom in mapping various shapes of the kernels in convolutional layers. The most notable distinction is that the proposed scheme exploits the internal data movement between PEs to maximize both weight and input data reuse, resulting in increased throughput and energy efficiency in convolutional neural network computation.

D. PIM Execution Scheduling

The earlier scheme of identifying PIM-friendly operations is insufficient for PIM to incorporate with the host processors. It only decides what operations to offload based on the analytical energy and memory models in the static compile time. However, the scheduling of the kernel executions further optimizes the utilization of both PIM and the host processors and enables the concurrent executions.

Reference [52] introduces two issues in scheduling executions for combined GPU and 3-d stacked memory PIM architecture, called GPU-PIM. First, the load imbalance occurs between GPU and PIM with the static offloading scheme. For example, if a large number of executions that offloaded to PIM are queued, GPU has to wait until PIM completes everything on the queue. Second, the offloading executions do not consider the utilization of the receive (RX) or transmit (TX) channels. It is possible that one channel is overloaded with one-sided executions that only use the channel, while the other is left underutilized. They propose a dynamic offloading aggressiveness control to address the issue. It decides final calls on offloading executions to PIM based on runtime information without any programmer's intervention. The new mechanism tracks two metrics: the number of pending offloading requests and the bandwidth utilization rate of both RX and TX channels. It simply sets threshold values for both metrics and stops offloading executions to PIM if the metrics reach their threshold values.

Reference [54] proposes a concurrent scheduling mechanism for GPU-PIM architecture. The scheduling mechanism uses three metrics to optimize executions: kernel dependency information, affinity prediction model, and execution time prediction model. The kernel dependence information determines which kernels can execute in parallel. It is obtained by read-after-write (RAW) dependencies across the kernels by profiling the whole application's kernel execution. The affinity prediction model is a logistic regression model that determines which computation cores can execute the kernels. The execution time prediction model is a linear regression model that predicts the execution time of a kernel on each computation core. To achieve high accuracy in training, both affinity and execution time models use metrics of kernel-level analysis such as memory intensity, kernel parallelism, and shared memory intensity.

E. Cache Coherence

Cache coherence protocols manage discrepancy on the shared data between different cores in a multi-processor computer system. When multiple cores are working simultaneously on the same data, they can read the proper data value unless any of them modifies the data. The cache coherence protocols

guarantee that none of the cores read invalid or stale data value.

Many research on PIM use the conventional cache coherence protocols to mitigate this issue between the host processor and PIM. Cache bypass is used in GraphPIM [55] and HBM-PIM. It forces a portion of the memory region uncacheable, allowing all memory requests to bypass the cache hierarchy and go straight to memory. In addition, write-through, message passing, and write-back are used in [52], [56], and [57], respectively, between PIM cores and CPU caches. However, the conventional approaches do not scale, and thus degrade the benefit of PIM if more data are shared through the narrow off-chip bandwidth between PIM and the host.

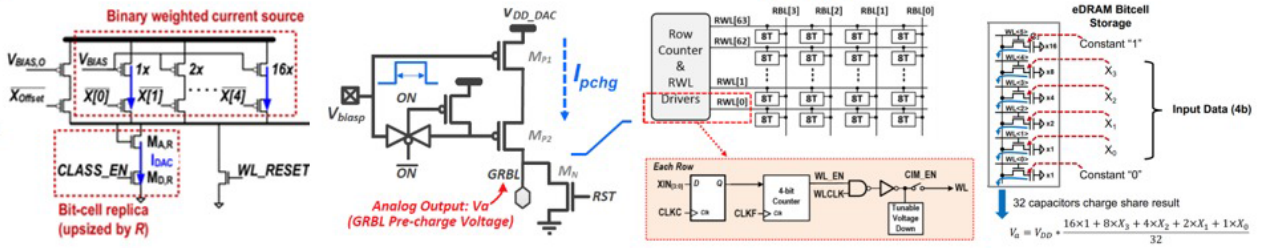
Regarding the issue, [58] proposes coherence for near data accelerators (CoNDA) to manage the cache coherence between the CPU and DRAM-based 3-d PIM. CoNDA optimizes the mechanism for updating the coherence request by letting PIM always execute on optimistic execution mode. During the optimistic execution mode, PIM stops issuing any coherence request and only tracks the memory accesses information such as addresses of all PIM read, write, and CPU write. Thus, none of the modified data is written to the memory. After PIM completes optimistic execution mode, it manages the coherence issue. It looks at the tracking information and only manages the coherence of the updated shared data during the execution to eliminate the unnecessary requests. As a result, CoNDA reduces the significant overhead in cache coherence requests and thus achieves performance speed up.

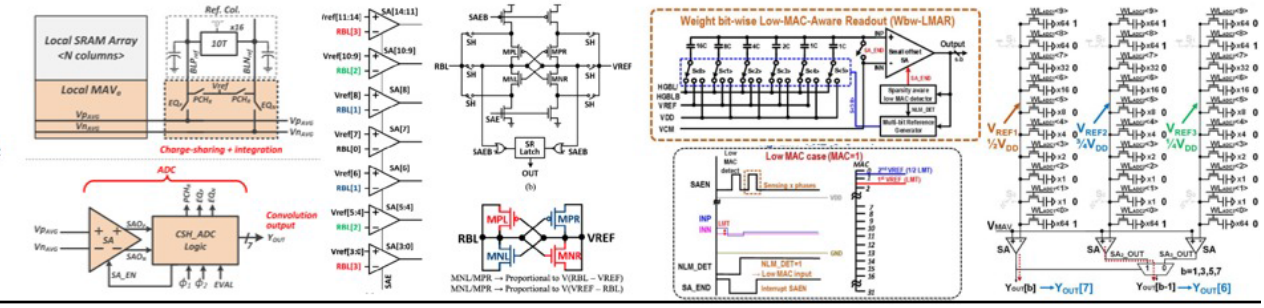
VI. Research Direction

A. Data Converter Overhead

In compute-in-memory designs, digital inputs need to be converted into analog domain to achieve in-memory computation. This conversion is realized with a DAC. Reference [12] presented a current DAC topology to convert the 5-bit digital input into current through binary-weighted PMOS current sources. This current is converted into a wordline voltage through the up-sized replica transistor and a diode-connected NMOS transistor. Reference [3] implemented a digital-to-time converter and a time-to-analog converter to embed digital inputs into an ON signal pulse-width. This ON signal is used to charge the global bitline with a constant current source. Therefore, according to the pulse-width, different digital input values can be represented by an analog voltage on the global bitline. Reference [13] applied multiple unit pulses on the RWL for discharging the bitline voltage to convert the digital input into the analog domain. Compared with the pulse-width method, the multiple RWL pulses approach shows better linearity, and the amount of discharge is closer to the expected result. In this design, digital components, such as D flip-flop, 4-bit counter, and several logic gates for each row are used to realize the multiple RWL pulses. The 4-bit input are loaded into the counter, and the RWL pulses are generated until the counter reaches 0. Reference [59] repurposed the inherent embedded dynamic random-access memory (eDRAM) for the DAC implementation. Initially, 4-bit digital data is loaded into the eDRAM bitcells, and the number of bitcells for each bit position is binary-scaled. Next, all WLs are asserted to perform charge share operation along the entire eDRAM column for generating the analog voltage. Compared with other types of data converter in the CIM design, this design fully utilizes the existing resource (i.e., 1T1C eDRAM bitcell, peripheral circuit) inside the memory array. Moreover, these components can be configured as CIM engines and can be configured as memory storage in a non-CIM mode. After the MAC operation, an analog-to-digital converter also needs to be implemented to convert the final analog computation results into the digital domain, which is usually the last step of the compute-in-memory design. Reference [3] presented an integrating ADC using a charge-sharing-based integrator, StrongARM latch-type sense amplifier, and logic block. This design replicates the 10T SRAM columns for charge-share and integration purposes, mitigating the process and temperature variation effect on the results. Reference [13] developed a 4-bit Flash ADC in the SRAM-CIM design, where 15 references voltage levels and 15 comparators are used. The reference voltage is achieved using the resistive diode ladder and changing the voltage input to the ladder. However, one dedicated sense amplifier design consists of 4 PMOS, 4 NMOS, 6 passgates, 2 NOR gates, and an SR latch, which incurs significant area overhead on the CIM macro. [8] presented a weight bitwise low-mac aware readout scheme to improve energy efficiency. In this design, multibit MAC is read out through the switch-capacitor voltage amplifier. Compared with the conventional readout scheme, this design included an extra phase to detect if the analog input voltage is above or lower than the threshold voltage, and the threshold voltage is determined by the MAC values distribution. If the MAC result is less than the threshold voltage, then the switch-capacitor voltage sense amplifier only required two phases to generate the final digital output, resulting in a higher readout rate. Reference [59] repurposed the 1T1C eDRAM bitcells for 8-bit SAR ADC implementation, significantly reducing the area overhead due to the extra arithmetic circuit because the 1T1C eDRAM bitcells can be either used as normal memory storage or CIM computation. In the in-eDRAM SAR ADC design, the reference voltage is generated from the charge-share operation between multiple 1T1C eDRAM bitcells. The comparison between the reference voltages and MAC analog voltage is implemented with the existing sense amplifier in the peripheral circuit. It is worth noting that many prior designs introduce extra circuit components to the original memory array. For example, [3] added an 8:1 MUX, 3 2:1 MUX, and extra transistors, [13] realized the flash ADC by adding 15 dedicated sense amplifiers and [12] implemented more than 32 extra transistors to achieve the current DAC design. To address the problem, [59] reconfigured the eDRAM bitcells and inherent charge share operation to achieve the data converter functionality. While compute-in-memory does reduce the von-Neumann bottleneck caused by repeatedly reading from and writing to the memory array from the processor unit, on-chip computation remains a concern. Since the size and organization of a memory array are optimized for high-performance and high-density purposes, extra circuit components added into

TABLE V
DATA CONVERTER COMPARISON

Digital-to-Analog Converter				
	JSSC 17' [12]	**ISSCC 18' [3]**	**ISSCC 20' [13]**	**ISSCC 21' [59]**
Component	WLDAC	GBL_DAC	PWM DAC	In-eDRAM DAC
Topology	Current DAC	Digital-to-time & time-to-analog converter	Multiple RWL pulses	Voltage DAC
Extra Components	Extra transistors	8:1 MUX, 2:1 MUX, extra transistors	Flip-flop, 4-b counter, logic gate (each row)	Analog buffer
Schematic				

Analog-to-Digital Converter				
	ISSCC 18' [3]	**ISSCC 20' [13]**	**ISSCC 20' [8]**	**ISSCC 21' [59]**
Component	CSH_ADC	4b ADC	Wbw-LMAR	In-eDRAM ADC
Topology	Integrating ADC	Flash ADC	Switch-capacitor Converter	SAR ADC
Extra Components	Integrator, sense amplifier, logic Block	Sense amplifier x15, resistive diode ladder	Voltage sense amplifier, capacitor x32, low-MAC-aware logic block	SAR logic block
Schematic				

the memory array will cause degradation on memory design parameters and cause unintended area/power overhead.

B. Test Accuracy

In order to reduce the area overhead, one can maximize the existing peripheral circuit reuse to realize the essential components. In addition, one can also reduce the bit precision of DAC and ADC to reduce the area and power overhead. Previously, [60] reported a binary neural network (BNN) implementation, which helps to speed up the design and eliminates the data conversion steps in the conventional compute-in-memory design. While BNNs are compact and efficient, they suffer a 3% accuracy loss on the CIFAR-10 dataset [61]. In recent compute-in-memory design, convolution neural networks became the mainstream [3], [8], [12], [13], [59]. Table VI summarizes the test accuracy from the recent compute-in-memory demonstrations [3], [7], [8], [10], [43], [59], [62]–[64]. For the MNIST dataset, the range of test accuracy is 97%∼99.63%, depending on the design and the ML algorithm. In recent years, most of the designs validate the design with the CIFAR10 dataset instead of the MNIST dataset. The test accuracy for this dataset in different designs is about 80.1%∼92.52% and the accuracy drop from software accuracy (ideal case) is 0.55%∼1.39%.

Among all CIM approaches, the non-linearity in the analog compute-in-memory design topology is one of the main sources of accuracy degradation compared with the software (ideal) accuracy. To address the accuracy loss challenges in such CIM designs, [65] retrain the network using the gradient-blocking technique for quantization which is presented in [66]. In this design, the test accuracy was dropped to 90.3% (81.6%) with a calibrated (uncalibrated) multiplicative digital-to-analog converter, while the floating-point software accuracy is 91.9% on CIFAR-10 dataset. However, after 3 epochs of retraining, the test accuracy increases back to 91.6% (86.2%) with calibrated (uncalibrated) multiplicative digital-to-analog converter, which is only 0.3% lower than the software accuracy.

C. Intra-Memory Data Movement

Compared with conventional MAC computation on CPU, in-memory computation minimizes the need for off-chip data movement and reduces data movement energy. However, at the same time, it also introduces data duplication and extra intra-memory data movement, which can dominate the

TABLE VI
MACHINE LEANING ALGORITHM, TEST ACCURACY, SOFTWARE ACCURACY AND ACCURACY DROP COMPARISON

Paper	Year	ML Algorithm	Dataset	Test Accuracy	Software accuracy	Difference
CONV-SRAM [3]	ISSCC 18	LeNet-5	MNIST	98%	-	-
T8T SRAM-CIM [10]	ISSCC 19	ResNet20	MNIST	99.52%	-	-
ReRAM-nvCIM [43]	ISSCC 20	ResNet-18	MNIST	98.80%	-	-
7nm CIM SRAM [13]	ISSCC 20	VGG9	MNIST	99.63%	-	-
T8T SRAM-CIM [10]	ISSCC 19	ResNet20	CIFAR10	90.42%	-	-
ReRAM-nvCIM [43]	ISSCC 20	ResNet-18	CIFAR10	88.52%	89.68%	1.16%
TWT SRAM-CIM [7]	ISSCC 20	ResNet20	CIFAR10	91.94%	-	-
7nm CIM SRAM [13]	ISSCC 20	VGG9	CIFAR10	88.90%	-	-
6T SRAM CIM [8]	ISSCC 20	ResNet20	CIFAR10	92.02%	92.57%	0.55%
DARAM CIM [64]	ISSCC 21	ResNet18	CIFAR10	91.20%	-	-
CIM NN Processor [63]	ISSCC 21	ResNet18	CIFAR10	92.52%	-	-
eDRAM-CIM [59]	ISSCC 21	ResNet50	CIFAR10	91.08%	92.47%	1.39%
TWT SRAM-CIM [7]	ISSCC 20	ResNet20	CIFAR100	67.79%	-	-
6T SRAM CIM [8]	ISSCC 20	ResNet20	CIFAR100	67.89%	68.40%	0.51%
COMPAC [62]	JSSC 20	AlexNet	ImageNet	52.10%	54.20%	2.10%
CIM NN Processor [63]	ISSCC 21	ResNet18	ImageNet	66.31%	-	-

energy consumption due to peripheral circuit usage. Hence it is critical to explore the design CIM topology from intra-memory data movement perspective and investigate efficient computational dataflow strategies. In a conventional im2col-based algorithm, data needs to be duplicated and converted from a two-dimensional matrix into a one-dimensional column matrix, which introduces a significant amount of redundancy in the memory array. Although the conventional compute-in-memory approach benefits from high throughput, data are duplicated for every cycle in the im2col algorithm, which occupies the cache and memory bandwidth for other resources.

A matrix lowering scheme has been proposed to reduce the memory overhead and accelerate the convolution computation without accuracy degradation [67]. This algorithm addresses the intra-memory traffic issues in current CIM design, enables training or inference for a given size of memory and improves memory sub-system efficiency to improve computation. By using the presented algorithm, one can apply DNN computation inside a memory-constrained environment. In addition, a direct convolution algorithm in static random-access memory is implemented as an all-in-one approach without extra data duplication by recycling data between neighboring SRAM bit-cells [65]. In this way, memory bandwidth can be utilized for other resources, which relaxes the constraint for other applications since the data inside the CIM engine is reused for multiple computation cycles. In addition, this design allows parallel computation for multiple MAC products every clock cycle, which significantly increases the throughput and efficiency while keeping the intra-memory data movement to a minimum.

VII. CONCLUSION

To conquer the memory bottleneck in processor-centric design of modern computing systems, the emerging memory-centric computing paradigms such as processing-in-memory architecture is especially effective for data-intensive AI and ML workloads. To this extent, this paper introduces an overview of PIM architecture for AI and ML technologies. The state-of-the-art PIM research works are investigated based on the memory types: SRAM-based PIM, DRAM-based PIM, and ReRAM-based PIM. For each PIM type, we comprehensively discuss the basic operation of the memory, circuit component designs, macro designs, and entire architectures as well as the challenges. To make the PIM hardware be widely adopted in the future, there is a need for more advanced software stack in incorporating PIM into a computing system, allowing PIM to be easily exploited by end-users. Lastly to ensure its reliability, further research on reducing the overhead in data converter, improving test accuracy of ML models, and minimizing intra-memory data movement are crucial.

ACKNOWLEDGMENT

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ, USA.

REFERENCES

[1] K. Ando *et al.*, "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, Apr. 2018.

[2] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, "A multi-functional in-memory inference processor using a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 53, no. 2, pp. 642–655, Feb. 2018.

[3] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 217–230, Jan. 2019.

[4] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute," *IEEE J. Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, Jun. 2019.

[5] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, Jun. 2020.

[6] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42 pJ/decision 3.12 TOPS/W robust in-memory machine learning classifier with on-chip training," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2018, pp. 490–492.

[7] J.-W. Su *et al.*, "A 28 nm 64 Kb inference-training two-way transpose multibit 6T SRAM compute-in-memory macro for AI edge chips," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 240–242.

[8] X. Si *et al.*, "A 28 nm 64 Kb 6T SRAM computing-in-memory macro with 8b MAC operation for AI edge chips," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 246–248.

[9] W.-S. Khwa *et al.*, "A 65 nm 4 Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2018, pp. 496–498.

[10] X. Si *et al.*, "A twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 396–398.

[11] J.-M. Hung, C.-J. Jhang, P.-C. Wu, Y.-C. Chiu, and M.-F. Chang, "Challenges and trends of nonvolatile in-memory-computation circuits for AI edge devices," *IEEE Open J. Solid-State Circuits Soc.*, vol. 1, pp. 171–183, 2021.

[12] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[13] M. E. Sinangil *et al.*, "A 7-nm compute-in-memory SRAM macro supporting multi-bit input, weight and output and achieving 351 TOPS/W and 372.4 GOPS," *IEEE J. Solid-State Circuits*, vol. 56, no. 1, pp. 188–198, Jan. 2021.

[14] C. Yu, T. Yoo, T. T.-H. Kim, K. C. T. Chuan, and B. Kim, "A 16 K current-based 8T SRAM compute-in-memory macro with decoupled read/write and 1-5bit column ADC," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Mar. 2020, pp. 1–4.

[15] C. Yu, K. T. Chuan Chai, T. T.-H. Kim, and B. Kim, "A zero-skipping reconfigurable SRAM in-memory computing macro with binary-searching ADC," in *Proc. IEEE 47th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2021, pp. 131–134.

[16] H. Kim, Q. Chen, and B. Kim, "A 16 K SRAM-based mixed-signal in-memory computing macro featuring voltage-mode accumulator and row-by-row ADC," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2019, pp. 35–36.

[17] Z. Jiang, S. Yin, J.-S. Seo, and M. Seok, "C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism," *IEEE J. Solid-State Circuits*, vol. 55, no. 7, pp. 1888–1897, Jul. 2020.

[18] H. Kim, T. Yoo, T. T.-H. Kim, and B. Kim, "Colonnade: A reconfigurable SRAM-based digital bit-serial compute-in-memory macro for processing neural networks," *IEEE J. Solid-State Circuits*, vol. 56, no. 7, pp. 2221–2233, Jul. 2021.

[19] Y.-D. Chih *et al.*, "An 89 TOPS/W and 16.3 TOPS/mm^2 all-digital SRAM-based full-precision compute-in memory macro in 22 nm for machine-learning edge applications," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2021, pp. 252–254.

[20] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and trends of SRAM-based computing-in-memory for AI edge devices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 5, pp. 1773–1786, May 2021.

[21] M. He *et al.*, "Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 372–385.

[22] S. Lee *et al.*, "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 43–56.

[23] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 273–287.

[24] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable *in-situ* accelerator," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 288–301.

[25] V. Seshadri *et al.*, "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2013, pp. 185–197.

[26] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 380–392, 2016.

[27] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3D memory," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2017, pp. 751–764.

[28] P. Gu *et al.*, "IPIM: Programmable in-memory image processing accelerator using near-bank architecture," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 804–817.

[29] H.-S. P. Wong *et al.*, "Metal–oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

[30] M.-F. Chang *et al.*, "A 3T1R nonvolatile TCAM using MLC ReRAM with sub-1ns search time," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2015, pp. 318–319.

[31] C.-C. Lin *et al.*, "A 256b-wordlength ReRAM-based TCAM with 1ns search-time and 14× improvement in wordlength-energyefficiency-density product using 2.5T1R cell," in *IEEE ISSCC Dig. Tech. Papers*, Jan. 2016, pp. 136–137.

[32] T. F. Wu *et al.*, "Brain-inspired computing exploiting carbon nanotube FETs and resistive RAM: Hyperdimensional computing case study," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2018, pp. 492–493.

[33] T. F. Wu *et al.*, "A 43pJ/cycle non-volatile microcontroller with 4.7 μs shutdown/wake-up integrating 2.3-bit/cell resistive RAM and resilience techniques," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 226–227.

[34] Q. Liu *et al.*, "A fully integrated analog ReRAM based 78.4 TOPS/W compute-in-memory chip with fully parallel MAC computing," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 500–501.

[35] W. Wan *et al.*, "A 74 TMACS/W CMOS-RRAM neurosynaptic core with dynamically reconfigurable dataflow and *in-situ* transposable weights for probabilistic graphical models," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 498–499.

[36] S. Yu, H. Jiang, S. Huang, X. Peng, and A. Lu, "Compute-in-memory chips for deep learning: Recent trends and prospects," *IEEE Circuits Syst. Mag.*, vol. 21, no. 3, pp. 31–56, 3rd Quart., 2021.

[37] M. Naumov *et al.*, "Deep learning recommendation model for personalization and recommendation systems," 2019, *arXiv:1906.00091*.

[38] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[39] T. B. Brown *et al.*, "Language models are few-shot learners," 2020, *arXiv:2005.14165*.

[40] J.-H. Yoon, M. Chang, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, and A. Raychowdhury, "A 40 nm 100 Kb 118.44 TOPS/W ternary-weight computein-memory RRAM macro with voltage-sensing read and write verification for reliable multi-bit RRAM operation," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2021, pp. 1–2.

[41] W.-H. Chen *et al.*, "A 65 nm 1 Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2018, pp. 494–495.

[42] C.-X. Xue *et al.*, "A 1 Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 388–389.

[43] C.-X. Xue *et al.*, "A 22 nm 2 Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 244–245.

[44] C.-X. Xue *et al.*, "A 22 nm 4 Mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7 TOPS/W for tiny AI edge devices," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2021, pp. 246–247.

[45] J.-H. Yoon, M. Chang, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, and A. Raychowdhury, "A 40 nm 64 Kb 56.67 TOPS/W read-disturb-tolerant compute-in-memory/digital RRAM macro with active-feedback-based read and *in-situ* write verification," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2021, pp. 404–405.

[46] J. M. Correll *et al.*, "A fully integrated reprogrammable CMOS-RRAM compute-in-memory coprocessor for neuromorphic applications," *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 6, pp. 36–44, 2020.

[47] W. Li, S. Huang, X. Sun, H. Jiang, and S. Yu, "Secure-RRAM: A 40 nm 16 kb compute-in-memory macro with reconfigurability, sparsity control, and embedded security," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2021, pp. 1–2.

[48] W. Li, X. Sun, H. Jiang, S. Huang, and S. Yu, "A 40 nm RRAM compute-in-memory macro featuring on-chip write-verify and offset-cancelling ADC references," in *Proc. IEEE 47th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2021, pp. 79–82.

[49] Y. Chen, L. Lu, B. Kim, and T. T.-H. Kim, "Reconfigurable 2T2R ReRAM architecture for versatile data storage and computing in-memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 12, pp. 2636–2649, Dec. 2020.

[50] Y. Chen, L. Lu, B. Kim, and T. T.-H. Kim, "A reconfigurable 4T2R ReRAM computing in-memory macro for efficient edge applications," *IEEE Open J. Circuits Syst.*, vol. 2, pp. 210–222, 2021.

[51] A. Boroumand *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2018, pp. 316–331.

[52] K. Hsieh *et al.*, "Transparent offloading and mapping (TOM) enabling programmer-transparent near-data processing in GPU systems," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 204–216, 2016.

[53] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on processing-in-memory architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 4, pp. 1333–1343, Apr. 2020.

[54] A. Pattnaik *et al.*, "Scheduling techniques for GPU architectures with processing-in-memory capabilities," in *Proc. Int. Conf. Parallel Architectures Compilation*, Sep. 2016, pp. 31–44.

[55] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling instruction-level PIM offloading in graph computing frameworks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 457–468.

[56] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 105–117.

[57] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 336–348.

[58] A. Boroumand *et al.*, "CoNDA: Efficient cache coherence support for near-data accelerators," in *Proc. 46th Int. Symp. Comput. Archit.*, Jun. 2019, pp. 629–642.

[59] S. Xie, C. Ni, A. Sayal, P. Jain, F. Hamzaoglu, and J. P. Kulkarni, "eDRAM-CIM: Compute-in-memory design with reconfigurable embedded-dynamic-memory array realizing adaptive data converters and charge-domain computing," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2021, pp. 248–250.

[60] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," 2016, *arXiv:1602.02830*.

[61] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, Jun. 2019.

[62] A. Sayal, S. Fathima, S. T. Nibhanupudi, and J. P. Kulkarni, "COMPAC: Compressed time-domain, pooling-aware convolution CNN engine with reduced data movement for energy-efficient AI computing," *IEEE J. Solid-State Circuits*, vol. 56, no. 7, pp. 2205–2220, Jul. 2021.

[63] J. Yue *et al.*, "A 2.75-to-75.9 TOPS/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong CIM with simultaneous computation and weight updating," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2021, pp. 238–240.

[64] Z. Chen, X. Chen, and J. Gu, "A 65 nm 3T dynamic analog RAM-based computing-in-memory macro and CNN accelerator with retention enhancement, adaptive analog sparsity and 44 TOPS/W system energy efficiency," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2021, pp. 240–242.

[65] J. N. Rohan and J. P. Kulkarni, "Realizing direct convolution in memory with systolic-RAM," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2021, pp. 1–3.

[66] C. N. Coelho *et al.*, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Mach. Intell.*, vol. 3, no. 8, pp. 675–686, 2021.

[67] M. Cho and D. Brand, "MEC: Memory-efficient convolution for deep neural network," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 815–824.

Donghyuk Kim (Graduate Student Member, IEEE) received the B.S. degree in electrical and computer engineering from the University of Washington, Seattle, USA, in 2020. He is currently pursuing the M.S. degree in electrical engineering with the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea.

His research interests include energy-efficient processing-in-memory architecture for machine learning and database and deep neural network accelerators.

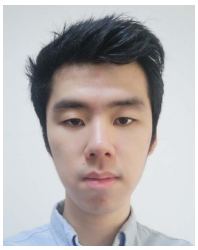
Chengshuo Yu (Graduate Student Member, IEEE) received the B.S. degree in electronic engineering from Feng Chia University, Taiwan, in 2019. He is currently pursuing the Ph.D. degree with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

His research interests include mixed signal SRAM/DRAM-based in-memory computing and time-domain hardware accelerator.

Shanshan Xie (Graduate Student Member, IEEE) received the B.S. degree in electrical and computer engineering (ECE) from Worcester Polytechnic Institute (WPI), Worcester, MA, USA, in 2018. She is currently pursuing the M.S. and Ph.D. degrees in electrical and computer engineering with The University of Texas at Austin (UT Austin), Austin, TX, USA.

She was an Intern with Analog Devices, Wilmington, MA, USA; and Texas Instrument Corporation, Dallas, TX, USA, where she was involved in programmable gain instrumentation amplifier, ECG heart rate monitor, CAN isolation products. Her research interests include mix-signal design for compute-in-memory techniques, machine learning accelerator, and annealing processor.

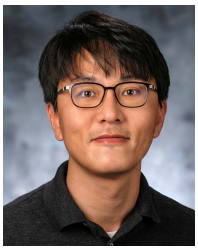
Miss. Xie has received the 2020 Cadence Women in Technology Scholarship, the 2021–2022 IEEE Student Travel Grant Award, and the 2021–2022 IEEE Solid State Circuit Society (SSCS) Predoctoral Achievement Award.



Yuzong Chen received the B.Eng. degree in electrical and electronic engineering from Nanyang Technological University, Singapore, in 2019. From 2019 to 2021, he was a Project Officer with the Centre for Integrated Circuits and Systems (CICS), Nanyang Technological University. He is currently a Research Assistant with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. His research interests include low-power circuit design and computing in-memory hardware.

Joo-Young Kim (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2005, 2007, and 2010, respectively.

He is currently an Assistant Professor with the School of Electrical Engineering, KAIST. He is also the Director of the AI Semiconductor Systems Research Center. Before joining KAIST, he was a Senior Hardware Engineering Lead at Microsoft Azure, Redmond, WA, USA, working on hardware acceleration for its hyper-scale big data analytics platform named Azure Data Lake. He was also one of the initial members of the Catapult Project at Microsoft Research, Redmond, where he deployed a fabric of field-programmable gate arrays (FPGAs) in datacenters to accelerate critical cloud services, such as machine learning, data storage, and networking. His research interests span various aspects of hardware design, including VLSI design, computer architecture, FPGA, domain-specific accelerators, hardware/software co-design, and agile hardware development. He was a recipient of the 2016 IEEE Micro Top Picks Award, the 2014 IEEE Micro Top Picks Award, the 2010 DAC/ISSCC Student Design Contest Award, the 2008 DAC/ISSCC Student Design Contest Award, and the 2006 A-SSCC Student Design Contest Award. He serves as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.



Bongjin Kim (Senior Member, IEEE) received the B.S. and M.S. degrees from POSTECH, Pohang, South Korea, in 2004 and 2006, respectively, and the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA, in 2014.

He was with Rambus, Sunnyvale, CA, USA, where he was a Senior Staff Member and worked on the research of high-speed serial link circuits and microarchitectures. He was a Post-Doctoral Research Fellow with Stanford University, Stanford, CA, USA. From 2006 to 2010, he was with Samsung Electronics, Yongin, South Korea, where he performed research on clock generators for high-speed serial links and clock generators. He also worked as a Research Intern with Texas Instruments, Dallas, TX, USA; IBM TJ Watson Research, Yorktown Heights, NY, USA; and Rambus, from 2012 to 2014. He was an Assistant Professor with Nanyang Technological University, Singapore, from 2017 to 2020. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering (ECE), University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include memory-centric computing devices, circuits, and architectures, hardware accelerators, alternative computing, and mixed-signal circuit design techniques and methodologies. His research works appeared at top integrated circuit design and automation conference proceedings and journals, including ISSCC, VLSI Symposium, CICC, ESSCIRC, ASSCC, ISLPED, DATE, ICCAD, the IEEE JOURNAL OF SOLID-STATE CIRCUITS, and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.

Dr. Kim was a recipient of the Prestigious Doctoral Dissertation Fellowship Award based on his Ph.D. research works, the International Low Power Design Contest Award from ISLPED, and the Intel/IBM/Catalyst Foundation Award from CICC.

Jaydeep P. Kulkarni (Senior Member, IEEE) received the B.E. degree in electronics/electrical engineering from the University of Pune, India, in 2002, the M.Tech. degree in electronics/electrical engineering from the Indian Institute of Science (IISc), Bengaluru, India, in 2004, and the Ph.D. degree in electronics/electrical engineering from Purdue University, West Lafayette, IN, USA, in 2009.

From 2009 to 2017, he was with Intel Circuit Research Lab, Hillsboro, OR, USA, where he worked on energy-efficient integrated circuit technologies. He is currently an Assistant Professor in electrical and computer engineering at The University of Texas at Austin, a fellow of the AMD Endowed Chair in Computer Engineering, and a fellow of the Silicon Labs Chair in Electrical Engineering. He has filed 36 patents and published over 100 articles in peer-reviewed journals and conferences. His current research is focused on machine learning hardware accelerators, in-memory computing, emerging nano-devices, hardware security, heterogeneous/3-D integration, and cryogenic computing.

Dr. Kulkarni is a Senior Member of the U.S. National Academy of Inventors (NAI). He has received the Best M.Tech. Student Award from IISc, the Intel Foundation Ph.D. Fellowship Award, the SRC Best Paper and Inventor Recognition Awards, the Purdue Outstanding Doctoral Dissertation Award, the seven Intel Divisional Recognition Awards, the 2015 IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS Best Paper Award, the SRC Outstanding Industrial Liaison Award, the Micron Foundation Faculty Awards, the Intel Rising Star Faculty Award, and the NSF CAREER Award. During his tenure at Intel Labs, he has served as an Industrial Distinguished Lecturer for IEEE Circuits and Systems Society and an Industrial Liaison for SRC, NSF programs. He has served as the TPC Co-Chair and the General Co-Chair for 2017 and 2018 ISLPED, respectively. He is also serving as the Chair for IEEE Solid State Circuits Society and Circuits and Systems Society, Central Texas Joint Chapter. He serves as an Associate Editor for IEEE SOLID-STATE CIRCUITS LETTERS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS. He serves as a Guest Editor for IEEE MICRO and IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS. He is serving as a Distinguished Lecturer for the IEEE Solid State Circuit Society and the IEEE Electron Device Society.

Tony Tae-Hyoung Kim (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, South Korea, in 1999 and 2001, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Minnesota, Minneapolis, MN, USA, in 2009.

From 2001 to 2005, he was with Samsung Electronics, Hwasung, South Korea, where he performed the research on the design of high-speed SRAM memories, clock generators, and IO interface circuits. From 2007 to 2009, he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA; and Broadcom Corporation, Edina, MN, USA, where he performed the research on circuit reliability, low-power SRAM, and battery-backed memory design. In 2009, he joined Nanyang Technological University, Singapore, where he is currently an Associate Professor. He has authored or coauthored over 190 journal articles and conference papers and holds 17 U.S. and Korean patents registered. His current research interests include computing-in-memory for machine learning, ultra-low power circuits and systems for smart edge computing, low-power and high-performance digital, mixed-mode, and memory circuit design, variation-tolerant circuits and systems, and emerging memory circuits for neural networks.

Dr. Kim has served on numerous conferences as a committee member. He has received the IEEE ISSCC2019 Student Travel Grant Award; the Best Demo Award at APCCAS2016; the Low Power Design Contest Award at ISLPED2016; the Best Paper Awards at 2014 and 2011 ISOCC; the AMD/CICC Student Scholarship Award at IEEE CICC2008; the Departmental Research Fellowship from the University of Minnesota in 2008; the DAC/ISSCC Student Design Contest Award in 2008; the Samsung Humantech Thesis Award in 2008, 2001, and 1999; and the *ETRI Journal* Paper of the Year Award in 2005. He was the Chair of the IEEE Solid-State Circuits Society Singapore Chapter in 2015–2016. He is the Chair-Elect/Secretary of the IEEE Circuits and Systems Society VSATC. He serves as a Corresponding Guest Editor for the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS, a Guest Editor for the IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, and an Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE ACCESS, and the *IEIE Journal of Semiconductor Technology and Science*.