MNSIM 2.0: A Behavior-Level Modeling Tool for Processing-In-Memory Architectures

Zhenhua Zhu, Hanbo Sun, Tongxin Xie, Yu Zhu, Guohao Dai, *Member, IEEE*, Lixue Xia, Dimin Niu, Xiaoming Chen, *Member, IEEE*, Xiaobo Sharon Hu, *Fellow, IEEE*, Yu Cao, *Fellow, IEEE*, Yuan Xie, *Fellow, IEEE*, Huazhong Yang, *Fellow, IEEE*, and Yu Wang, *Fellow, IEEE*

***Abstract*—In the age of artificial intelligence (AI), the huge data movements between memory and computing units become the bottleneck of von Neumann architectures, i.e., the "memory wall" problem. In order to tackle this challenge, processing-in-memory (PIM) architectures are proposed, which perform *in-situ* computations in memory and give alternative solutions to boost the computing energy efficiency and performance. Because of the large-scale neural network (NN) algorithm models and the huge hardware design space, various factors affect computing accuracy and performance, bringing the need for efficient PIM modeling and evaluation tools. In this work, we propose a behavior-level modeling tool, *MNSIM 2.0*, to model the performance of PIM architectures efficiently. At the hardware level, *MNSIM 2.0* provides a hierarchical PIM modeling structure with flexible architecture configurability and components extensibility. Moreover, the first unified PIM memory array model is proposed for describing both digital and analog PIM. At the algorithm level, *MNSIM 2.0* supports the PIM-based NN computing accuracy simulation considering various architecture and device parameters. A PIM-oriented NN model training and quantization flow is also integrated to improve the performance gain brought by PIM. At the scheduling level, *MNSIM 2.0* adopts a universal scheduling description compatible with different scheduling strategies. Validation using fabricated PIM macros shows the relative modeling error rate of *MNSIM 2.0* is 3.8%–5.5%. Case studies show that *MNSIM 2.0* enables PIM design space explorations, influences analysis of device parameters, and architecture design insight discoveries.**

***Index Terms*—Hardware modeling tool, processing-in-memory (PIM), software–hardware co-optimization.**

Manuscript received 24 October 2022; revised 24 January 2023; accepted 21 February 2023. Date of publication 2 March 2023; date of current version 20 October 2023. This work was supported in part by the Beijing Innovation Center for Future Chips. The work of Zhenhua Zhu, Hanbo Sun, Tongxin Xie, Yu Zhu, Guohao Dai, Huazhong Yang, and Yu Wang was supported by the Tsinghua–Meituan Joint Institute for Digital Life. This article was recommended by Associate Editor K. Olcoz. *(Corresponding author: Yu Wang.)*

Zhenhua Zhu, Hanbo Sun, Tongxin Xie, Yu Zhu, Huazhong Yang, and Yu Wang are with the Department of Electronic Engineering, Tsinghua University, Beijing 100084, China (e-mail: zhuzhenh18@mails.tsinghua.edu.cn; yu-wang@mail.tsinghua.edu.cn).

Guohao Dai is with the Qing Yuan Research Institute, Shanghai Jiao Tong University, Shanghai 200030, China.

Lixue Xia is with the Department of Cloud Intelligence, Alibaba Group, Beijing 100022, China.

Dimin Niu and Yuan Xie are with the Computing Technology Laboratory, Alibaba DAMO Academy, Hangzhou 311121, China.

Xiaoming Chen is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100045, China.

Xiaobo Sharon Hu is with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA.

Yu Cao is with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85281 USA.

Digital Object Identifier 10.1109/TCAD.2023.3251696

I. Introduction

In THE past few years, convolutional neural networks (CNNs) have demonstrated powerful capabilities in many fields, e.g., object detection [1], face recognition [2], etc. In addition to the high computing accuracy, the amount of data and model size increase dramatically as the CNN models become more and more complex. In traditional von-Neumann architectures (e.g., CPU and GPU), the CNN computations cause massive data movements between memory and computing units, which consume more than 80% of the overall system energy and exacerbate the "memory wall" problem [3].

In order to tackle the "memory wall" problem, researchers have proposed novel processing-in-memory (PIM) architectures that perform computations ***inside*** memory [4], [5], [6], [7], [8], [9], [10]. Fig. 1 shows two mainstream PIM approaches, i.e., analog PIM and digital PIM. In the analog PIM [Fig. 1(a)], each CNN weight kernel is expanded to a vector and stored in one column of the memory array. The input feature map is also expanded in the same way and transformed into a voltage vector loaded to the word-line (WL) of the memory array. Then, the current of bit-line (BL) can represent the result of matrix–vector multiplications (MVMs) between weights and feature maps [4], [5], [6], [7], [8]. In the digital PIM [Fig. 1(b)], the weight kernels are stored in the memory array, whose memory cells are attached to simple computing units (e.g., logic gates). During NN computation, the feature map data are loaded bit-by-bit, and the attached computing units perform bit-serial multiplications. Afterward, multiple adder trees are used to merge the results of bit-serial multiplications, achieving massively parallel MVM operations [9], [10]. Both the analog and digital PIM architectures eliminate the weight data movements between memory and computing units, which are also the most time and energy-consuming parts of CNNs, bringing impressive performance gain.

Due to the huge PIM design space and large-scale NN models, SPICE simulation will take unacceptably long time to simulate the entire PIM architecture. For example, an 8-kB memristor array needs more than 10 min for SPICE simulation on one input vector [11], while typical NN models have MB-scale weights, requiring days to months of simulation time for PIM architectures. Besides, restricted by the evolving fabrication technologies, existing PIM chips mainly focus on the macro designs (e.g., 1-Mb memory arrays with their peripheral circuits). There exists an urgent need to evaluate

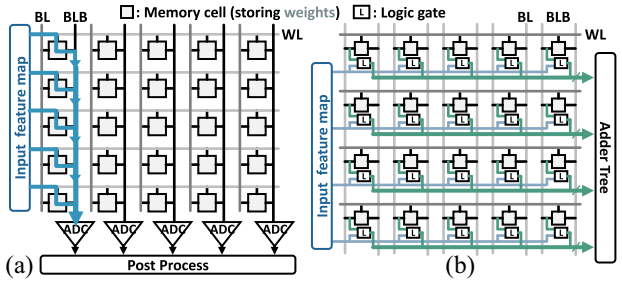


Fig. 1. Schematic view of (a) analog PIM and (b) digital PIM.

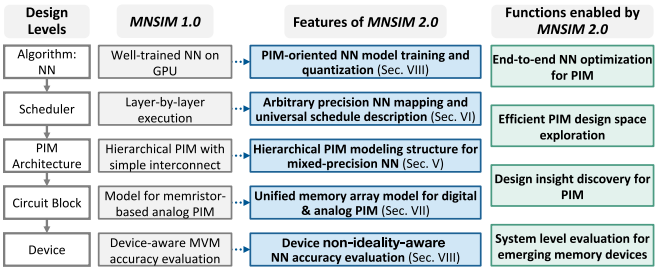


Fig. 2. New features and functions in *MNSIM 2.0* compared with *MNSIM* [11].

the performance of the entire PIM architecture based on these macro designs.

For the purpose of evaluating PIM architecture performance thoroughly and efficiently, the PIM modeling tool needs to be designed meticulously from the following perspectives.

1) *Algorithm Level:* Different datasets and CNN models should be supported. Besides, the analog-domain computing error and noise cause NN accuracy degradation in analog PIM architectures. Therefore, in order to evaluate and exploit the benefits of an NN model on PIM, PIM-oriented algorithmic optimizations are required.
2) *Schedule Level:* Algorithm mapping and scheduling bridge various NN models and PIM architectures, determining the data flow among computing units. Carefully designed scheduling optimization techniques conduce to improvements in hardware resource utilization and performance.
3) *Architecture Level:* In order to support different PIM architectures and NN models, a basic PIM architecture abstraction model is necessary. The architecture abstraction model should support computations with different data precision. It also should be easy to extend to be compatible with other computing components.
4) *Circuit Block Level:* A unified PIM memory array model is required for describing both analog and digital PIM macro designs, which is still an open research problem.
5) *Device Level:* Memory device is the basic computing and storage unit of PIM architectures. The PIM modeling tool should support different memory technologies (e.g., SRAM, memristor, etc.) and be able to analyze the impact of device parameters on performance and accuracy.

In this article, we first review existing representative PIM modeling tools and summarize these modeling tools from a cross-level perspective, i.e., from algorithm simulation support to memory device configurability analysis. Then, we propose *MNSIM 2.0*,[1] a behavior-level PIM modeling tool aiming at efficient PIM-based CNN accuracy and architecture performance simulation. Fig. 2 summarizes the new features of *MNSIM 2.0* and various functions enabled by *MNSIM 2.0*. The contributions of this article include the following.

1) *At the hardware level*, inspired by the conventional memory structure, we propose a hierarchical PIM modeling structure with flexible architecture configurability and component extensibility. The basic architecture of *MNSIM 2.0* provides support for mixed-precision NN operations, which can cover computations with different data precision. Furthermore, we design the first unified memory array model for both digital and analog PIM, offering the opportunity for a fair comparison of different digital and analog PIM architectures.
2) *At the algorithm and scheduling level*, we emulate PIM computing behaviors and build a PIM hardware-aware NN accuracy simulation flow in machine learning frameworks. In order to fully exploit the benefits of PIM and make the tool easier for users in different areas, we integrate a PIM-oriented NN model training and quantization flow in *MNSIM 2.0*. Besides, *MNSIM 2.0* adopts an arbitrary precision NN mapping and the universal scheduling description interface to support various algorithm models and scheduling strategies.
3) We use two fabricated PIM macros to validate the simulation results of *MNSIM 2.0*. The relative error rate is 3.8%–5.5%, showing good modeling accuracy. Multiple case studies are provided to demonstrate the functionalities of *MNSIM 2.0* from the algorithm level to the device level.

The remainder of this article is organized as follows. Section II introduces the basic knowledge of CNN and PIM. The overview of *MNSIM 2.0* is explained in Section III. Section IV provides the basic hierarchical PIM modeling structure used in *MNSIM 2.0*. Sections V–VII introduce the detailed modeling flow from the scheduling level, the architecture and circuit block level, and the algorithm level, respectively. Section VIII shows the experimental results, and Section IX concludes this article.

II. Preliminary

A. Convolutional Neural Network

CNNs are usually composed of three types of layers, i.e., the convolutional (CONV) layers, the pooling layers, and the fully connected (FC) layers. CONV layers perform convolution operations between 3-D feature map matrices F_i and convolution weight kernels W. In this article, we denote kernel size, input channel, and output channel as K, C_{in}, and C_{out}, respectively. In each CONV layer, the convolution kernels traverse the entire input feature map according to the sliding stride s. FC layers are similar to CONV layers, and these two types of layers can be expressed as MVMs. Pooling layers usually come after CONV layers, which execute subsampling operations in each sliding window. Two common types of pooling layers are max pooling and average pooling.

B. Processing-In-Memory Architecture

The PIM (also known as computing-in-memory, CIM) architectures mainly contain two categories, i.e., analog PIM and digital PIM.

[1]The code is available in https://github.com/thu-nics/MNSIM-2.0.git.

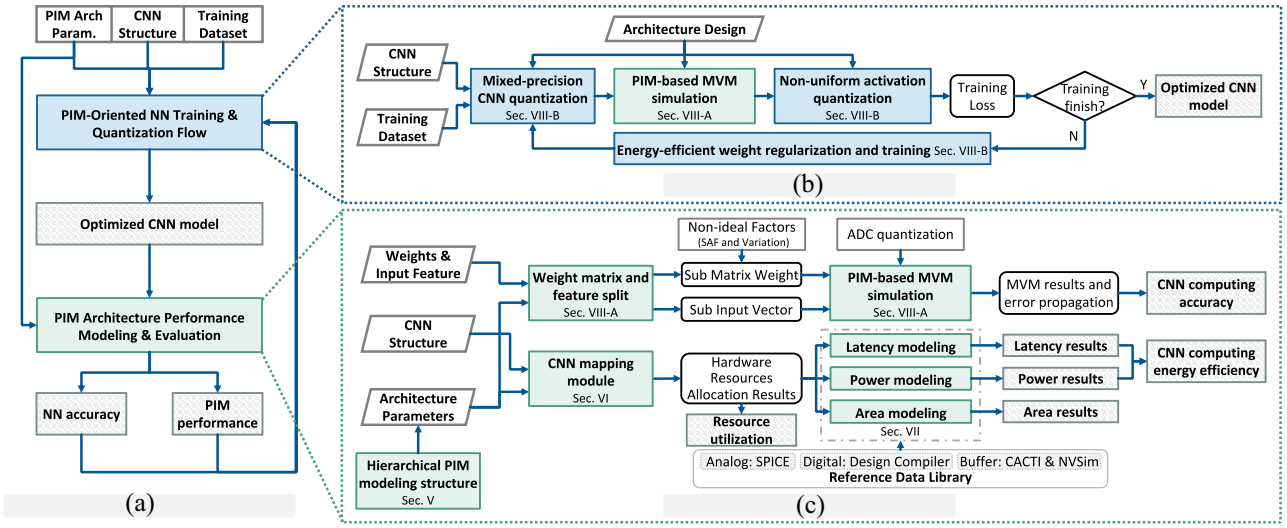


Fig. 3. (a) High-level depiction of *MNSIM 2.0*. (b) PIM-oriented NN training and quantization flow. (c) PIM architecture performance modeling and evaluation flow.

The analog PIM architectures perform MVMs in the analog domain, e.g., using conductances and voltages for computations. Memristor and SRAM are the two most common memory types used for analog PIM with a similar working principle. Here, we use the memristor-based PIM as an example. Memristor is a kind of nonvolatile memory that stores information using different resistance values. Typical memristors contain resistive random access memory (RRAM), magnetic RAM (MRAM), etc. Multiple memristors construct the crossbar structure, which is the basic computing unit of analog PIM. The input vector is represented by a voltage vector $\boldsymbol{V}$ loaded to the WL of crossbars, and the memristors are used to represent the matrix. Then, the MVM results are derived by the output current vector $\boldsymbol{I}$ from each BL. Some interfaces [e.g., analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and sensing amplifiers (SAs)] are required between PIM memory array and other digital circuits.

The analog PIM exploits MVM computing parallelism in the analog domain, causing the requirement of costly analog-digital interfaces. To reduce the interface overhead, digital PIM is gradually rising in recent years [9], [10]. The weights are stored in the memory array of digital PIM architectures. Each memory cell or each memory row is attached with simple digital computing units that perform the multiplication between the input bit and weights. The feature map is loaded bit-by-bit during the computation so that the digital computing units can be simplified to logic gates. Besides the memory array, a near-memory adder tree is added to complete the multiply-accumulate for input feature map with multiple bits

$$O_j = \sum_{b=0}^{B-1} 2^b \times \sum_{i=1}^{N} \left(A_{bj} \times W_{ji}\right) \tag{1}$$

where B is the bitwidth of the input feature map, A_{bj} is the bth bit of input A_j, and W_{ji} is the (i, j) value in the weight matrix. The multiplication between A_{bj} and W_{ji} can be expressed as Boolean logic operation (e.g., bitwise AND operation).

III. *MNSIM 2.0* Overview

Fig. 3(a) shows the high-level depiction of *MNSIM 2.0*, containing the PIM-oriented NN model training and quantization flow and the PIM architecture performance modeling and evaluation flow. These two parts form a closed feedback loop. The training and quantization flow utilizes the architecture modeling results to guide the CNN algorithm optimizations for PIM. Then, the optimized CNN models, which contain the information of network structure and weight parameters, are sent to the architecture performance modeling flow to assess the PIM-based NN computing accuracy and hardware metrics.

The PIM-oriented NN training and quantization flow [Fig. 3(b)] aims at tailoring the NN model to suit the computation pattern of PIM architectures from the perspectives of *accuracy*, *energy*, *latency*, and *area*. In the NN training phase, *MNSIM 2.0* splits the weight matrices according to the memory array size and introduces ADC quantization, which help to obtain a high-*accuracy* NN model for PIM architectures. In addition to the original training flow, we integrate an *energy*-efficient weight regularization method in *MNSIM 2.0*. It is designed to explore the relationship between weight values and hardware energy consumption by introducing an energy "loss" into the NN training loss function. In the NN quantization phase, *MNSIM 2.0* leverages two PIM-aware quantization techniques: the mixed-precision quantization method and the nonuniform activation quantization method. The former method performs the layerwise weights and input activations quantization based on weight storage overhead and MVM calculation volume. It can reduce hardware *area* and *latency* while maintaining comparable accuracy. The latter method performs the data distribution-aware nonuniform quantization method on output activations. It reduces the requirement for readout ADC resolution, achieving high *energy efficiency*.

The PIM architecture performance modeling and evaluation flow [Fig. 3(c)] evaluates PIM hardware performance and PIM-based CNN accuracy. In *MNSIM 2.0*, users can describe the specific PIM architecture by setting configuration parameters of the proposed hierarchical PIM modeling structure. For the hardware performance modeling part, we first use a CNN mapping module to allocate the hardware resource for the input CNN model. It will also construct the PIM-based CNN computing data flow considering different scheduling strategies. Then, the overall performance of PIM architecture is modeled according to the specific data flow,

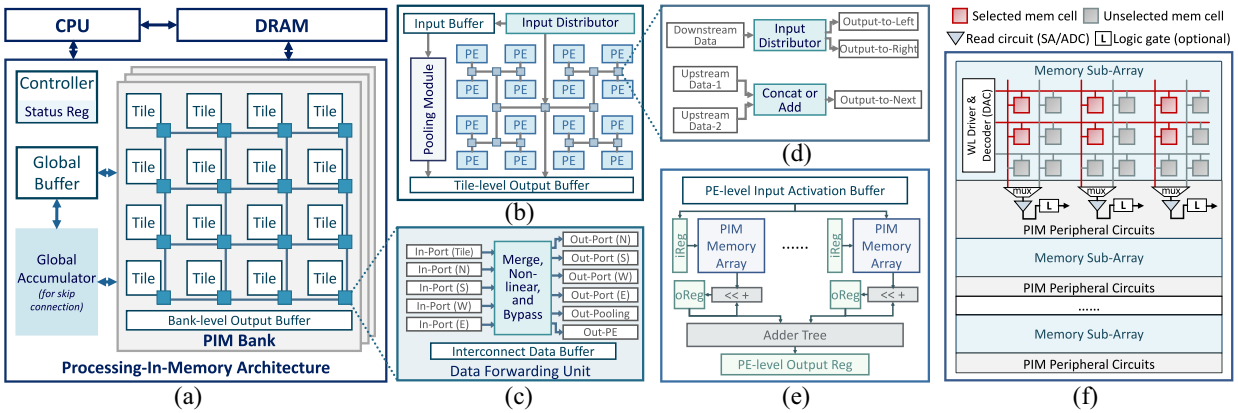


Fig. 4. Hierarchical PIM modeling structure used in *MNSIM 2.0*. (a) Global architecture and PIM bank. (c) Data forwarding unit in PIM bank. (e) PIM process element (PE). (f) PIM memory array.

resource utilization, and performance data of basic units. The performance data of basic units are obtained from the reference data library, which is built offline based on the logic synthesis and circuit simulation results. The CNN accuracy evaluation part is designed to assess the PIM-based NN accuracy by emulating the hardware computing behaviors. We build the PIM-based NN accuracy simulation flow in the following steps: 1) splitting weight matrices and activation vectors due to design parameters of the PIM memory array; 2) performing MVMs on the split data; 3) introducing circuit-/device-level nonideal factors and ADC quantization error; 4) generating the CONV/FC results; and 5) propagating errors to get the final accuracy.

TABLE I
ARCHITECTURE CONFIGURATION PARAMETERS OF *MNSIM 2.0* (R/W REPRESENTS READ AND WRITE)

	Variable Parameters		Variable Parameters
Device Level	Device Area	PE Level	Array Number
	Device R/W Power		ADC/DAC Number
	Device R/W Latency		Array Polarity
	Device Precision	Tile Level	PE Number
	Variation and SAF		Inter Tile Bandwidth
Array Level	Array Size		Intra-Tile Bandwidth
	Cell Type	Bank Level	Tile Number
	Wire RC		Inter Bank Bandwidth
	Technology Node		Intra-Bank Bandwidth
Interface Level	ADC/DAC Resolution		NoC Configuration
	ADC/DAC Sample Rate	Arch Level	Bank Number
	ADC/DAC Power		Buffer Configuration
	ADC/DAC Area		Pooling Structure

IV. HIERARCHICAL PIM MODELING STRUCTURE

MNSIM 2.0 provides a hierarchical PIM modeling structure for describing different analog/digital PIM architectures, as shown in Fig. 4. From a system perspective, the entire system is composed of CPU, DRAM, and PIM. Before the calculation starts, CPU generates NN mapping results, deploys weights to PIM memory, and initializes PIM controller. DRAM provides a large off-chip storage capability and is used to store the NN weights to be deployed and input images to be calculated. After the NN model is deployed, the PIM part is woken up by CPU and reads the input image from DRAM. PIM relies on its internal controller to manage the entire NN computing data/control flow. Besides, PIM uses the multilevel on-chip buffer for storing all the intermediate data during the computation, eliminating frequent data communications between PIM and DRAM. After the NN inference is completed, the results are returned to CPU.

For the PIM part, *MNSIM 2.0* uses a basic PIM architecture design to adapt to different architectures with flexible hardware configurability and component extensibility. We follow the hierarchical structure of conventional memory and describe the PIM architecture from five levels: the global architecture [Fig. 4(a)], PIM banks [Fig. 4(a)], PIM tiles [Fig. 4(b)], Processing Elements (PEs) [Fig. 4(e)], and PIM memory arrays [Fig. 4(f)]. In each level, we provide multiple default hardware modules with various configurable parameters (i.e., hardware module `class` and its `member variables`). The typical architecture configuration parameters are listed in Table I. Users can configure their architecture designs by setting these parameters. Furthermore, *MNSIM 2.0* also contains the interface for integrating new hardware modules.

The *global architecture* includes several PIM banks, the global buffer, the global accumulator, and the PIM controller. The global buffer and accumulator perform computations of elementwise sum layers in the skip connection network structure (e.g., ResNet [19]). The PIM controller manages the data/control flow of the entire PIM architecture. The functionality of the controller is determined by its internal status registers, which are configured by the host CPU according to NN model mapping and scheduling results.

In each *PIM bank*, an array of PIM memory tiles are organized and connected as a 2-D-mesh network-on-chip (NoC) structure. It is also easy to extend the mesh-based NoC to other interconnection structures like ring and torus. Each memory tile is adjacent to a data forwarding unit, a kind of computing capability enhanced router [Fig. 4(c)]. The data forwarding unit merges data from different tiles, performs activation functions of NN, and transfers the data to neighbor tiles within the same PIM bank. Different PIM banks use the bank-level output buffer and bus for data communications.

For the *PIM memory tile*, to reduce the complexity of control logic and data path, *MNSIM 2.0* specifies that each tile should only process one CNN layer. While for some large-scale layers, weight matrix splitting is required, and multiple tiles collaborate to complete the layer computations. According to the layer type, tiles can be configured as the MVM mode or the pooling mode. This is achieved by using an input distributor to distribute data to the PE array or

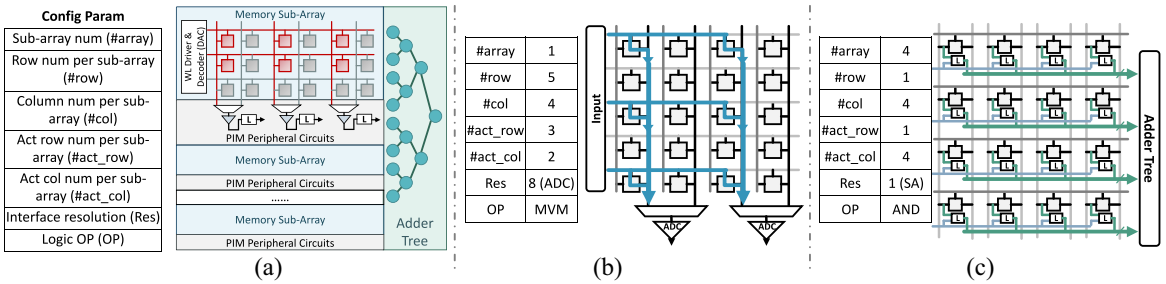


Fig. 5. Unified PIM memory array model. (a) Basic model. (b) Example of analog PIM. (c) Example of digital PIM.

the pooling module. The PEs in one tile are linked in an H-Treestructure to reduce the intratile interconnection overhead. Each connection node of the H-Treeis an inner-tile joint module, which manages the data movements and summations of PE results [Fig. 4(d)]. The pooling module consists of digital comparators and adders to implement the max pooling and average pooling functions.

PE mainly contains multiple PIM memory arrays with peripheral circuits. To support computations with different data precision, *MNSIM 2.0* leverages a reconfigurable multibit weights mapping method. Multiple memory arrays with low-precision devices are used to store the high-precision weights. Then, the MVM outputs of each PIM memory array are shifted and added to obtain the final results. Using different numbers of PIM memory arrays and configuring the shift bitwidth can realize the computations with different data precision.

PIM memory array carries out the core MVM computations of CNNs. It is the major difference between digital PIM and analog PIM. In order to support the modeling of both digital and analog PIM architectures, we propose a unified PIM memory array model in *MNSIM 2.0*, as shown in Fig. 5. In the unified PIM memory array model, we split one PIM memory array into multiple subarrays, and an adder tree is attached to the memory array for merging the partial results. Each subarray contains a separate set of peripheral circuits, including WL drivers, decoders, DACs (for analog PIM), readout circuits (high-resolution ADCs for analog PIM or SAs for digital PIM), and the logic gates required for digital PIM. The adjustable configuration parameters of the memory array are listed in Fig. 5(a). Fig. 5(b) shows an example of describing a 5×4 memristor crossbar for analog PIM. Here, we want to accumulate the current of different rows in one memory array, so the subarray number is set to one (i.e., without splitting the memory array). In this example, the memristor crossbar contains three DACs and two ADCs, and the activated row/column number per subarray is set to three/two, respectively. The resolution of the readout interface is set to eight, representing the 8-bit ADCs, and the in-memory computing operation is analog-domain MVMs. Furthermore, an example of a digital PIM memory array is demonstrated in Fig. 5(c). In this example, each row of the memory array reads the weight data in parallel and completes bitwise multiplications with the input activation bit. Therefore, the subarray number is set to the number of array rows (i.e., the row number per subarray equals one). Only 1-bit SAs are required for weights readout. And the memory cells are attached with AND gates for performing bitwise multiplications.

V. Mapping and Scheduling Module in *MNSIM 2.0*

The first step in evaluating PIM performance is to map and deploy the CNN model into PIM architecture. Then, we schedule the computation tasks among different computing units and analyze the data flow. *MNSIM 2.0* focuses on the mapping of mixed-precision NNs and provides a universal scheduling description interface to support different CNN models and scheduling strategies.

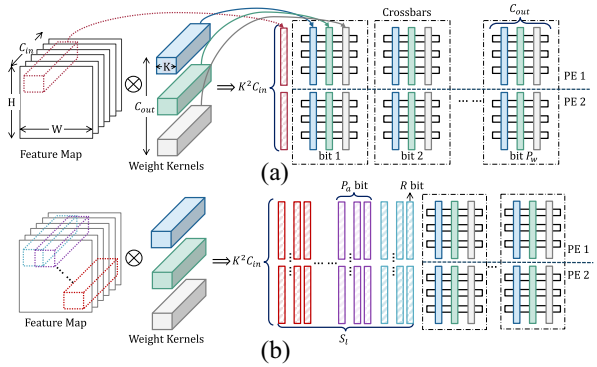


Fig. 6. CNN mapping strategy. K: Kernel size, C_{in}: Input channel, C_{out}: Output channel, S_l: Sliding times, P_w: Weight precision, P_a: Input precision, and R: DAC resolution. (a) CONV kernel mapping. (b) Input feature mapping.

A. CNN Model Mapping and Hardware Resource Allocation

Ideally, in PIM architectures, the 3-D CONV kernel is flattened into a 1-D column vector and mapped onto devices in one column of PIM memory arrays. Different kernels in one layer are placed in different columns of the same array. However, because of the limited device precision and array size, it is difficult to map a large-scale network layer in one PIM memory array. Therefore, we split the CONV kernels from two dimensions, i.e., weight precision and kernel size, to complete the reconfigurable mapping procedure for different precision data, as shown in Fig. 6(a). For the weight precision splitting, if the weight precision (e.g., 8-bit) is higher than the memory device precision (e.g., 1-bit), we use one PIM memory array to store a certain bit of all the data in the weight matrix. Therefore, we can flexibly deploy mixed-precision weights by using different numbers of arrays for different layers. For kernel size splitting, if the number of output channels and the kernel vector length exceed the array size, multiple PEs are needed in the horizontal and vertical direction, each of which processes a part of the CONV layer.

According to the mapping rules, the allocation results of PIM memory tiles can be modeled as (only shows CONV layers here, FC layers are similar)

$$\#\text{Tile}_{\text{CONV}} = \left\lceil \frac{\left\lceil \frac{C_{\text{in}}}{\lfloor \text{row}/K^2 \rfloor} \right\rceil \left\lceil \frac{C_{\text{out}}}{\text{col}} \right\rceil \left\lceil \frac{P_w}{\#xbar_{PE} \times P_m} \right\rceil}{\#PE_{\text{Tile}}} \right\rceil \tag{2}$$

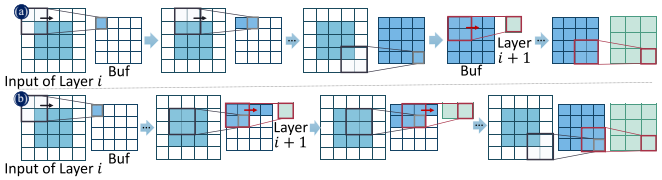


Fig. 7. Examples of: (a) layer-by-layer execution and (b) inner layer pipeline structure.

where *row*/*col* is the row/column number of the PIM memory array. P_w and P_m represent the precision of weights and memory devices. $\#xbar_{PE}$ and $\#PE_{\text{Tile}}$ are the number of PIM memory arrays per PE and PE number per tile, respectively.

The input feature mapping is shown in Fig. 6(b). Corresponding to the expansion of CONV kernels, the 3-D input feature block is converted to a vector, which will be transformed into voltage pulse signals through WL drivers or DACs and loaded onto the WLs of arrays. Different from the spatial expansion in multiple arrays of CONV kernels, the input feature blocks in each sliding window need to be loaded sequentially in different computation cycles. Besides, if the precision of the feature map is higher than the DAC's resolution, the input needs to be split and loaded into WLs in multiple cycles.

B. Universal Scheduling Description Interface

At the scheduling level, *MNSIM 2.0* is dedicated to analyzing when to start the calculation for each operation in CNNs. The calculation start time depends on the data dependencies among different CNN layers and the scheduling strategy. To be specific, assume that we want to calculate the output pixel $(r, c)_{i+1}$ of the layer $(i+1)$ (i.e., the data in the rth row and the cth column of the output feature map). We can analyze the calculation start condition for this output from two aspects.

1) *Data Dependency:* The coordinate of the last input that the layer i needs to provide: $(r_d, c_d)_i$.
2) *Scheduling Strategy:* The coordinate of the last output that the layer $(i+1)$ needs to complete: $(r_s, c_s)_{i+1}$.

Denote the calculation start time, completion time, and results transfer latency of the output $(r, c)_i$ are $T_{\text{start}}^{(r,c)_i}$, $T_{\text{end}}^{(r,c)_i}$, and $L_{\text{trans}}^{(r,c)_i}$, respectively. Then, the calculation start time can be modeled as

$$T_{\text{start}}^{(r,c)_{i+1}} = \max\left(T_{\text{end}}^{(r_d,c_d)_i} + L_{\text{trans}}^{(r_d,c_d)_i},\ T_{\text{end}}^{(r_s,c_s)_{i+1}}\right). \tag{3}$$

Users can leverage different scheduling stratiges in *MNSIM 2.0* by describing the relationship between $(r, c)_{i+1}$ and $\{(r_d, c_d)_i,\ (r_s, c_s)_{i+1}\}$. As default, *MNSIM 2.0* takes two strategies into consideration: 1) the vanilla layer-by-layer execution and 2) the inner layer pipeline structure.

The vanilla layer-by-layer execution is shown in Fig. 7(a); each layer starts calculations until the previous layer completes all the calculations (e.g., the convolution kernel has traversed the entire input feature). So the strategy can be described as

$$\begin{aligned}(r_d, c_d)_i &= (H_i, W_i)\\ (r_s, c_s)_{i+1} &= (r-1, c-1)_{i+1}\end{aligned} \tag{4}$$

where H_i and W_i are the height and width of the output feature map of layer i, respectively.

In the inner layer pipeline structure, each layer starts its calculation once the required input data are ready. Fig. 7(b) demonstrates a simple example of the inner layer pipeline structure. Here, we consider two consecutive layers with the convolution kernel size of 2×2. In the second step of Fig. 7(b), layer i generates the intermediate data that can be used to perform the convolution of layer $(i+1)$. Then, the layer $(i+1)$ starts the computation, rather than waiting for the layer i to complete all the convolutions. Therefore, the strategy can be described as

$$\begin{aligned}(r_d)_i &= \begin{cases}\min(H_i, K_i + s_i \times (r-1) - p_i), & \text{CONV}\\ H_i, & \text{FC}\end{cases}\\ (c_d)_i &= \begin{cases}\min(W_i, K_i + s_i \times (c-1) - p_i), & \text{CONV}\\ W_i, & \text{FC}\end{cases}\\ (r_s, c_s)_{i+1} &= (r-1, c-1)_{i+1}\end{aligned} \tag{5}$$

where K_i is the kernel size, and s and p represent the size of stride and padding, respectively.

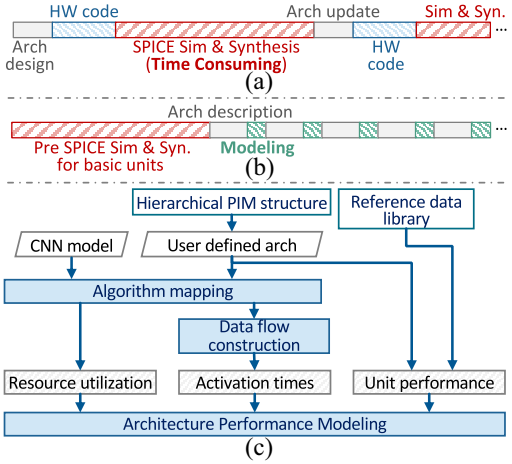


Fig. 8. (a) Original PIM hardware design and simulation flow. (b) Modeling flow of *MNSIM 2.0*. (c) Detailed hardware performance modeling flow used in *MNSIM 2.0*.

VI. Hardware Performance Modeling

A. Modeling Flow Overview

The original PIM hardware design and simulation flow [Fig. 8(a)] spends long time on SPICE simulation and logic synthesis to derive the architecture performance, which is time-consuming and becomes the bottleneck of architecture iterative optimization. *MNSIM 2.0* moves these time-consuming parts to the tool design phase [Fig. 8(b)]. During the iterative architecture optimization, we only need to perform efficient hardware modeling without the necessity of circuit simulation and synthesis. Therefore, *MNSIM 2.0* can evaluate the performance of large-scale PIM architectures in several seconds, enabling efficient architecture design space exploration.

Fig. 8(c) depicts the detailed hardware performance modeling flow of *MNSIM 2.0*. First, users provide the CNN model and describe the PIM architecture using the proposed hierarchical PIM modeling structure. Second, *MNSIM 2.0* maps the CNN model onto the given PIM architecture and allocates the hardware resources. Third, the data flow in PIM is constructed according to the scheduling strategy. The constructed data flow reflects the activation times and working time of each computing unit (e.g., the working clock cycles of each PIM memory array). Fourth, based on the reference data library, we use the performance LUT and approximate fitting functions to obtain the performance data of each computing

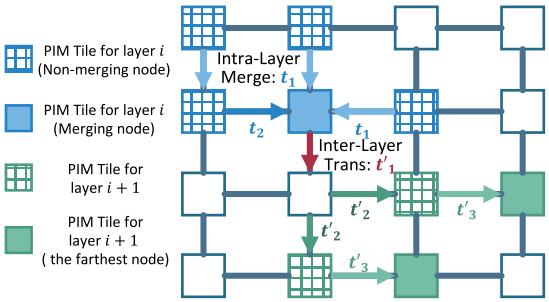


Fig. 9. Example of the tile interconnection graph model.

unit. Finally, the performance of the entire PIM architecture is evaluated due to the resource utilization, working time of computing units, and performance of each computing unit.

B. Intertile NoC Performance Analysis

In PIM architectures, multiple PIM memory tiles are required for processing the large-scale NN layers. The NoC structure is designed for intertile data communications. In order to describe the communication relationship among different tiles and model the NoC performance, we construct the tile interconnection graph model in *MNSIM 2.0*. The tile interconnection graph model takes the following basic assumptions.

1) We assume each layer has a merging node tile as a "data harbor" to the next layer. The blue tile in Fig. 9 shows an example of the merging node tile, and the red arrow indicates the first hop of the interlayer transmission.
2) Other tiles transmit their outputs to the merging node tile via NoC. The outputs of the tiles are merged hop-by-hop during the data transmission, as the blue arrows in Fig. 9 show. Different NoC wires transmit data concurrently.
3) The final results of one layer are transferred from its merging node tile to the tiles belonging to the next layer (green arrows in Fig. 9). The data transfer time depends on the maximum Manhattan distance from the merging node tile to tiles of the next layer.

Based on these assumptions, the merging node is selected by

$$M_N \in \arg\min_{m\in\{\text{Tile}\}_i}\left(\max_{n\in\{\text{Tile}\}_i}(|x_m-x_n|+|y_m-y_n|) + \max_{p\in\{\text{Tile}\}_{i+1}}\left(|x_m-x_p|+|y_m-y_p|\right)\right) \quad (6)$$

where (x_m, y_m) is the coordinate of the tile m, $\{\text{Tile}\}_i$ is the tile set of the layer i. M_N represents the merging node tile that achieves the minimum summation of the distance from the farthest tile in this layer and the distance from the farthest tile in the next layer. On the basis of the tile interconnection graph model, *MNSIM 2.0* evaluates the NoC performance in two simulation modes: 1) hop-based NoC modeling and 2) time-slice-based NoC simulation.

1) Hop-Based NoC Modeling: It uses the number of data transfer hops among different tiles for NoC evaluation. To reduce the simulation complexity, we model the data transfer time between adjacent tiles (i.e., one hop) according to the transferred data volume and NoC bandwidth. According to the tile interconnection graph model, the transfer latency of layer i can be calculated as

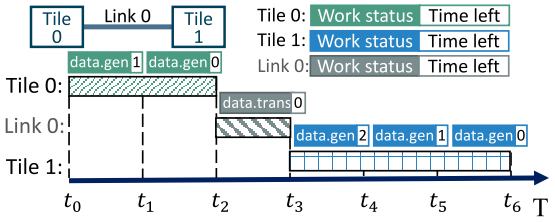


Fig. 10. Example of the time-slice-based NoC simulation.

TABLE II
COMPARISON OF NOC SIMULATION RESULTS

	Hop-based	Time-slice-based (time slice: ns)				
		1	5	10	50	100
Runtime (s)	6.3	1610.7	348.8	158.5	32.4	17.5
Latency (ms)	3.30	3.90	3.88	3.74	3.75	4.06

$$L_{\text{trans},i} = \sum_{d=0}^{D_{\text{inter}}}\frac{\text{DV}_d}{\text{BW}} + \sum_{d=0}^{D_{\text{intra}}}\left(\frac{\text{DV}_d}{\text{BW}} + t_{\text{merge},d}\right) \quad (7)$$

where D_{intra} and D_{inter} are the Manhattan distance from the farthest tile to the merging node in this layer and next layer, respectively. DV is the transferred data volume and BW is the bandwidth. Besides the communication time, the intralayer transfer latency includes the merging time ($t_{\text{merge},d}$), which is determined by the number of tiles to be merged at once.

2) Time-Slice-Based NoC Simulation: It divides the entire timeline into a number of time slices (e.g., each time slice is $2ns$). The status of each tile and the link between adjacent tiles is tracked from two aspects. One is the work status of each tile/link, including data generation, data transfer, waiting, and idle. The other is the remaining time to complete the current operation for the tile/link, which is measured by the number of time slices. During the simulation, we traverse the time slice in chronological order and update the work status and remaining time of each tile and link, as shown in Fig. 10.

Compared with the hop-based NoC modeling, the time-slice-based NoC simulation provides more accurate simulation results when the time slice is extremely "narrow" (e.g., 1 ns), while the overhead is the significant increase in simulation time. Besides, when the time slice is shorter than the clock cycle, the ground truth value of the NoC latency can be obtained from time-slice-based simulation. A simple NoC analytical example of running VGG [20] on an RRAM-based analog PIM is demonstrated in Table II, showing the runtime and simulation results of the two modes. For the NoC power and area estimation, we refer to the method used in [21] and [22].

C. Intratile Performance Analysis

The PIM memory tile can be configured into two operation modes, i.e., MVM mode for CONV and FC layers and Pooling mode for Pooling layers, as shown in Fig. 11. The entire data flow within the PIM tile is composed of three parts.

1) *Input Data Flow (Blue-Striped Arrow):* In the MVM mode, the input data flow starts from the data forwarding unit, goes through the `H-Tree`to PEs, and writes the input activation buffer. In the pooling mode, the input data flow bypasses the `H-Tree`and ends with the pooling buffer.

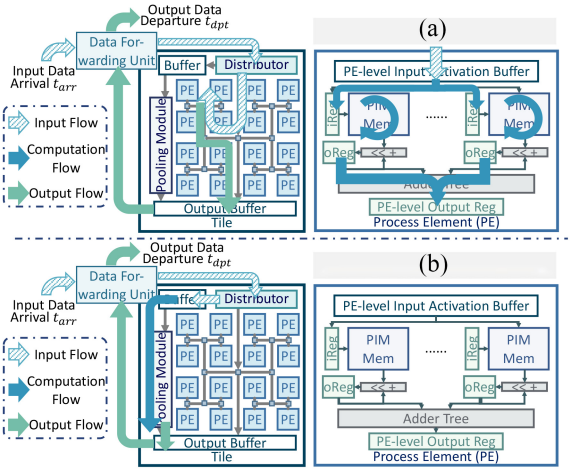


Fig. 11. Data flow of the PIM tile. (a) MVM Mode (CONV & FC). (b) Pooling Mode (Pooling Layer).

2) *Computation Data Flow (The Cyan-Blue Arrow):* For the MVM mode, the input feature is first read from the buffer and then sent to the input register (iReg). On account of the restricted DAC resolution and WL/BL parallelism, the PIM array needs to keep activated for multiple cycles to complete the calculation of the high-precision input feature

$$\text{mul} = \left\lceil \frac{P_a}{R_{\text{DAC}}} \right\rceil \times \left\lceil \frac{K^2 \min\left(\left\lfloor \frac{\text{row}}{K^2} \right\rfloor, C_{\text{in}}\right)}{\text{Parallel}_r} \right\rceil \times \left\lceil \frac{\min(C_{\text{out}}, \text{col})}{\text{Parallel}_c} \right\rceil. \quad (8)$$

In (8), the first term represents that multiple cycles are required for loading the high precision activation (P_a) by the low-resolution DAC (R_{DAC}). The second and the third terms reflect the limited WL/BL computation parallelism (Parallel_r, Parallel_c) causes more calculation cycles. In each cycle, several input activation bits are loaded to the DAC of the PIM memory array. Then, the analog MVM values or digital logic results generated by the PIM memory array are converted to the digital form by ADCs or SAs. The digital outputs are shifted and added with the previous temporary results and then stored in the output register. After *mul* cycles, the adder tree merges all the arrays' results in this PE. For the pooling mode, the computation latency is quite simple. It is determined by the kernel size of the pooling layer and the number of computing resources in the pooling module.

3) *Output Data Flow (The Green Arrow):* The output data flow is similar to the input data flow. The results of PEs are first merged by the joint module in `H-Tree`, then the merged results are sent to the output buffer. Data forwarding units read data from the output buffer and send them to other tiles.

D. Area, Power, and Latency Models for Basic Units

1) PIM Memory Array: MNSIM 2.0 provides the device description interface in Table I to model different device technologies. For the area estimation, users can provide information from three aspects: 1) array area; 2) device area; and 3) technology node, as shown in

$$\text{Xbar\_Area} = \begin{cases} \text{User Given Value} & \\ \text{col} \times \text{row} \times \text{Device\_Area} & \\ \text{col} \times \text{row} \times 120F^2 & \text{SRAM} \\ \text{col} \times \text{row} \times 3(W/L+1)F^2 & \text{1T1R} \\ \text{col} \times \text{row} \times 4F^2 & \text{0T1R} \end{cases} \quad (9)$$

where $1T1R$ and $0T1R$ represent the MOSFET-accessed memristor and cross-point memristor structure, W/L is the transistor technology parameter, and F is the technology node. Similarly, for the latency estimation, users can provide the array-level latency or the device read latency. For the latter one, we model the latency of PIM memory array by introducing the RC delay caused by interconnection wires. As for the power consumption model, *MNSIM 2.0* uses the percentage of bit-0/1 in the feature map and weights to estimate the equivalent power.

2) Analog and Digital Conversion Interface: In *MNSIM 2.0*, the design of ADCs/DACs is mainly customized by users. User needs to provide the circuit parameters shown in Table I. *MNSIM 2.0* also includes some state-of-the-art DAC and ADC designs with different resolutions as default values.

3) Digital Circuits: For non-MVM computing circuits, *MNSIM 2.0* synthesizes the digital circuit modules at TSMC 65-nm technology node by the Synopsys Design Compiler. The digital circuit modules include address decoders, adder trees, pooling modules, data forwarding units, etc. For other technology nodes, we get the modeling results converted from 65-nm simulation results using the scaling-down estimation method [23].

4) Buffer Design: For the on-chip buffer, *MNSIM 2.0* uses CACTI [18] and NVSIM [24] to get reference data (area, power, bandwidth, etc.) of different memory technologies (i.e., SRAM, DRAM, Nonvolatile Memories). After users give the buffer configurations (e.g., buffer size), *MNSIM 2.0* uses the method of fitting and looking up table through reference data to estimate the buffer read/write overhead.

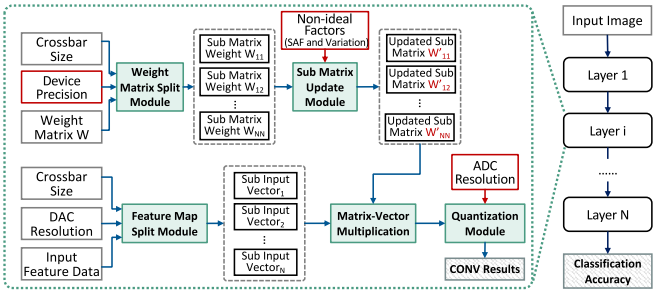


Fig. 12. Accuracy evaluation of PIM-based CNN inference.

VII. PIM-Based CNN Evaluation and Optimizations

A. PIM-Based CNN Accuracy Evaluation Flow

The PIM-based CNN accuracy evaluation flow is illustrated in Fig. 12, which emulates PIM computing behaviors and is implemented within the PyTorch framework. First, considering the size of PIM memory array, the precision of memory devices, and the DAC resolution, we split the weight matrix and input feature into submatrices and subvectors, as mentioned in Section V-A. Second, for each submatrix, device nonideal factors are introduced to update the values.

MNSIM 2.0 takes three major nonideal factors of memory devices into consideration, i.e., stuck-at-faults (SAFs), limited on/off ratio, and resistance variations. These nonideal factors are modeled as additive noises and mask matrices applied to weight matrices. We also retain the interface for describing other nonideal factors. Third, MVMs are performed between the updated submatrices and subvectors. The MVMs results are quantized according to the ADC resolution. Finally, the quantized MVM results are merged into the CONV results and propagated to the next layers to get the final CNN accuracy.

B. PIM-Oriented NN Training and Quantization

1) PIM-Oriented NN Training: When deploying an original well-trained NN model on PIM architecture, the circuit-/device-level nonideal factors (red box in Fig. 12) induce significant accuracy loss. In order to improve the PIM-based CNN computing accuracy, *MNSIM 2.0* integrates a PIM-oriented NN training flow. In the modified NN training flow, the forward propagation utilizes the PIM-based CNN accuracy evaluation flow to calculate the loss term, while the backward propagation remains similar as the original training flow without much modifications. Besides, the analog PIM architectures perform MVMs in the analog domain, where the energy consumption is closely related to the weights/activations distribution. *MNSIM 2.0* integrates an energy-efficient weight regularization method, which adjusts the distributions of weights and activations to reduce energy consumption [25]. To be specific, if we use the voltage and conductance to represent input activations and weights, then the updated loss function considering energy consumption is

$$\text{Loss}_{\text{new}} = \text{Loss}_o + \sum_{l=0}^{L-1} f\left(X_l^2, W_l\right) \tag{10}$$

where Loss_o is the original loss function, L is the total CNN layer number, and $f(\cdot)$ represents an elementwise product function. X_l and W_l represent the activations and weights of layer l.

2) PIM-Oriented NN Quantization: In CNN algorithms, different layers have different quantization sensitives [26]. In order to maintain the accuracy, some layers' weights and activation must be kept in high precision while others can be quantized into low bit width. In PIM architectures, the precision of weights and activation will affect the storage usage and latency according to (2) and (8). Therefore, *MNSIM 2.0* leverages the layerwise mixed-precision quantization to reduce storage burden and latency with little accuracy loss. Besides, existing work has demonstrated that high-resolution ADCs occupy the major energy consumption of PIM architectures, and their resolution has a crucial impact on computing accuracy [25]. In order to reduce the ADC energy consumption, *MNSIM 2.0* integrates a PIM-based nonuniform activation method, which optimizes the quantization range and determines the ADC quantization steps according to the distribution of output activations. Thus, the ADCs can pay more attention to the interval with more data. As a result, the ADC resolution can be reduced by 2-bit with lower power consumption.

VIII. Experimental Results

A. Experiment Setup

We use four typical CNNs as benchmarks, i.e., LeNet [28], modified VGG-8, VGG-16 [20], and ResNet-18 [19]. All experiments are evaluated on the CIFAR-10 and ImageNet datasets [29], [30]. The memristor and SRAM data we used refer to existing papers [10], [27], [31]. The driver, DAC, SA, and ADC data come from [5], [32], [33], [34], [35], [36]. The digital parts are synthesized at 65 nm with 500 MHz using Synopsys Design Compiler. We scale the synthesized results to other technology nodes according to the scaling equations [23].

TABLE III
Configuration Parameters of Two Representative PIM Macros: SRAM-Based Digital PIM (ISSCC 22-11.7 [10]) and RRAM-Based Analog PIM (ISSCC 20-33.2 [27])

	[ISSCC 22-11.7]	[ISSCC 20-33.2]
(#array, #row, #col)	(32, 16, 64)	(1, 784, 100)
(#act_row, #act_col)	(1, 64)	(784, 100)
(Res, Op)	(1, AND)	(8, MVM)

TABLE IV
Comparisons Between Chip Measurement Results and Simulation Results of *MNSIM 2.0*

	[ISSCC 22-11.7]	*MNSIM 2.0*	Error
Area (mm^2)	0.030	0.034	13.3%
Performance ($GOPS$)	16.00	16.22	1.4%
Energy efficiency ($TOPS/W$)	27.38	28.23	3.1%
	[ISSCC 20-33.2]	*MNSIM 2.0*	Error
Area (mm^2)	3.77	3.50	-7.2%
Latency (ns)	51.10	53.38	4.5%
Energy efficiency ($TOPS/W$)	78.40	74.44	-5.1%

B. Validation of Simulation Results

In order to validate the simulation results of *MNSIM 2.0*, we consider two fabricated PIM chips and use the device and circuits parameters from these PIM macros in *MNSIM 2.0*.

1) We use the Dynamic Logic-based SRAM PIM macro proposed in [10] as an example of *digital PIM*. It contains a 32-kb array that is split into 32 compartments, each of which has 16 WLs and 64 BLs. In each cycle, one WL is activated and performs logic operation with the 1-bit input signal.
2) We use the fully integrated analog RRAM PIM macro proposed in [27] as an example of *analog PIM*. It contains two RRAM arrays for computing two FC layers. For simplicity, we focus on the 784 × 100 RRAM array designed for the first layer, whose ADC resolution is eight. When performing MVM operations, all rows and columns are activated.

To evaluate these designs, we configure parameters of the PIM memory array model as shown in Table III. The comparison results are shown in Table IV. Compared with chip measurement results, the relative error rate of *MNSIM 2.0* is only 3.8% and 5.5% for SRAM-based digital PIM and RRAM-based analog PIM, respectively, showing acceptable simulation accuracy of *MNSIM 2.0*. Besides these two mentioned PIM macros, other PIM designs can also be modeled by setting different configuration parameters in Fig. 5.

TABLE V
SIMULATION TIME COMPARISONS (UNIT: *s*)

	MNSIM 2.0		SPICE	
	HW	NN accuracy	#Mem array	Sim time
LeNet	3.1	3.2	48	5.35 per PIM memory array
VGG-8	4.5	48.5	1272	
VGG-16	6.3	53.2	2040	
ResNet-18	5.5	37.3	1968	

TABLE VI
ACCURACY AND PERFORMANCE RESULTS OF RESNET-18 ON DIFFERENT DATASETS

	Accuracy (%)	Area (mm^2)	Latency (ms)	Energy (mJ)
CIFAR-10	89.80 (TOP1)	459.96	3.28	1.3
ImageNet	89.08 (TOP5)	831.21	79.4	47.3

TABLE VII
ACCURACY AND PERFORMANCE RESULTS @ CIFAR-10

	Accuracy (%)	Area (mm^2)	EDP ($ms \times mJ$)
ResNet-18	89.8	459.96	4.264
Auto search	89.2	67.34	1.67

C. Simulation Time

Table V shows the simulation time of *MNSIM 2.0* and SPICE. Because of the extremely long SPICE simulation time, we use the simulation time of one PIM memory array (256 × 256) and the number of memory arrays for comparisons. Results show that the entire architecture modeling time of *MNSIM 2.0* is similar to the SPICE simulation time of only one PIM memory array. Considering that large-scale NNs require thousands of PIM memory arrays, *MNSIM 2.0* is more efficient for the design space exploration.

D. Algorithm-Level Case Study

Thanks to the flexible CNN description interface and the proposed PIM-based CNN optimization flow, *MNSIM 2.0* provides the opportunity for evaluating different datasets and CNN models on PIM. Table VI shows the accuracy and hardware performance of PIM-based ResNet-18 on CIFAR-10 and ImageNet datasets. Here, we use an RRAM-based analog PIM with 256 × 256 crossbars and four ADCs as the hardware instance. The larger scale of ImageNet causes high latency and energy consumption on PIM.

We also integrate *MNSIM 2.0* with an open-sourced PIM and NN co-exploration tool, i.e., *Gibbon* [37]. By using *MNSIM 2.0* as the performance evaluator, the results of the automated neural architecture search are shown in Table VII. The co-exploration results bring 7.37× and 2.55× reduction in area and energy-delay-product, respectively.

E. Scheduling-Level Case Study

MNSIM 2.0 supports different scheduling strategies by analyzing the calculation start condition of output pixels in each layer. Here, we use the built-in layer-by-layer execution and inner layer pipeline to demonstrate the influence on the performance of different scheduling strategies.

Fig. 13 shows an example of the inner layer pipeline data flow. The dots in different rows represent the computing time of different layers, revealing the high computing parallelism.

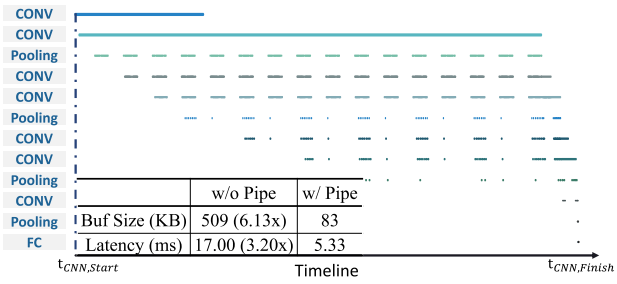


Fig. 13. VGG-8 data flow using the inner layer pipeline.

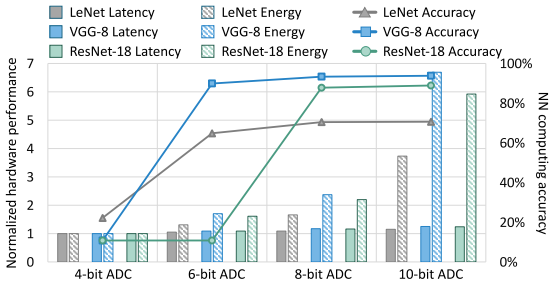


Fig. 14. Normalized latency, energy, and accuracy results under different ADC resolutions. NN accuracy baseline: LeNet 70.6%, VGG-8 93.5%, and ResNet-18 91.2%.

The table in Fig. 13 shows the inner layer pipeline structure reduces the buffer size and latency by 6.13× and 3.2×.

F. Architecture-Level Case Study

Because of the fast evaluation speed, *MNSIM 2.0* enables early-stage PIM architecture design space exploration. Here, we use two examples to demonstrate the design space exploration results given by *MNSIM 2.0*.

1) Tradeoff Between Accuracy and Hardware Performance: One primary computing deviation source of the analog PIM architecture is the quantization error of ADCs. The quantization error mainly depends on two factors: 1) the ADC resolution and 2) the activated row number (i.e., input parallelism). These two factors also influence the hardware performance significantly, so the optimal architecture should be carefully studied.

For the ADC resolution, Fig. 14 shows the CNN accuracy and hardware performance under different ADC configurations. We use the analog PIM architecture with 256 × 256 RRAM crossbars, and 256 rows are activated at the same time. The results demonstrate that the CNN computing power drops dramatically when the ADC resolution becomes lower. But ADCs with low resolution may destroy the CNN function. Besides, different CNN models have different tolerance capabilities for quantization errors. Thus, the ADC resolution, together with the NN model, should be selected meticulously to achieve the tradeoff between accuracy and hardware performance.

For the input parallelism, the computation parallelism is enhanced as the number of activated rows increases, resulting in low latency. However, the higher input parallelism also leads to higher ADC quantization error since the precision of accumulated current becomes large. Fig. 15 shows the CNN accuracy and latency under different input parallelisms. Here, we use 256 × 256 RRAM crossbars with 1-bit DACs and

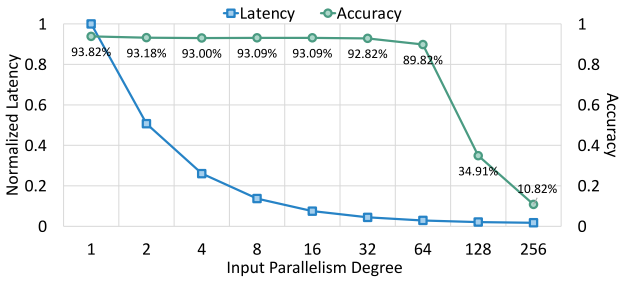


Fig. 15. Normalized Latency and accuracy under different input parallelisms (CNN model: VGG-16, 256 × 256 crossbars with 1-bit DACs and 4-bit ADCs).

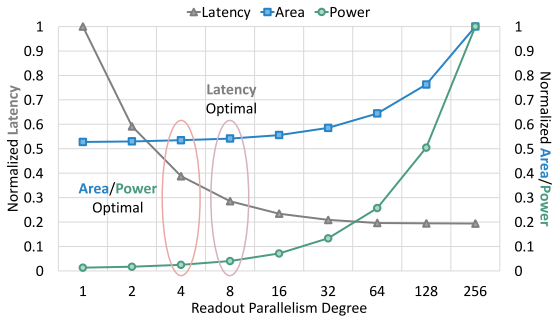


Fig. 16. Normalized Latency, Area, and Power results under different readout parallelisms (CNN model: VGG-16, 256 × 256 crossbars with 256 1-bit DACs and 8-bit ADCs).

4-bit ADCs. Theoretically, 16 rows can be activated simultaneously without ADC quantization error when using 4-bit ADC designs. Thanks to the inherent error tolerance features of NN, the maximum input parallelism with little accuracy loss reaches 32, significantly reducing the latency compared with the input parallelism of one. However, when the input parallelism becomes larger (e.g., 128), the huge ADC quantization error causes unacceptable accuracy loss. Under this circumstance, a higher ADC resolution is required.

2) Tradeoff Among Latency, Area, and Power: Due to the place and routing limitations, multiple WLs/BLs usually share one DAC/ADC in memory array. Interface sharing reduces the hardware area and power consumption at the cost of increased latency. Fig. 16 shows the latency, area, and power changing trend with respect to (w.r.t) readout parallelism (i.e., the number of ADCs attached to one array). We can extract two optimal points from Fig. 16. One is the power and area optimal point, which can reduce the latency by 60% with little additional overhead. And the other is the latency optimal point. It achieves more aggressive optimized latency results (∼70% reduction), but the overhead (e.g., power increased by 16 W) is not negligible.

3) Area and Power Breakdown: We use the area/power optimal design in Fig. 16 to analyze the area and power breakdown. The results are shown in Fig. 17. The PIM memory arrays occupy the most area, and ADCs spend the most power budget. Further optimization should be done for ADCs to improve energy efficiency in future work.

G. Circuit Block-Level Case Study

At the circuit block level, the major contribution of *MNSIM 2.0* is providing a unified memory array model for

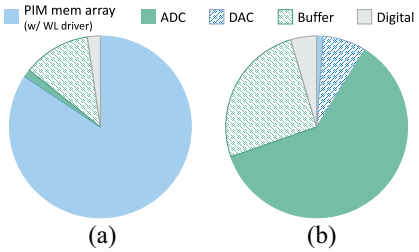


Fig. 17. (a) Area and (b) power breakdown of one typical PIM.

TABLE VIII
COMPARISONS BETWEEN ANALOG PIM AND DIGITAL PIM ON VGG-8 @ CIFAR-10

	Analog PIM	Digital PIM
Latency (ms)	5.33	4.44
Area (mm^2)	705	510
Energy(mJ)	3.83	1.04
Accuracy	93.36%	93.54%

TABLE IX
PIM-BASED NN COMPUTING ACCURACY (%) UNDER DIFFERENT VARIATION LEVELS @ CIFAR-10

	0%	1%	5%	10%	20%
LeNet	69.63	68.91	62.18	26.63	22.82
VGG-8	93.36	93.09	92.72	90.91	71.82
VGG-16	93.09	93.45	89.73	91.18	27.00

both digital and analog PIM, which helps to make a fair comparison among different PIM architectures. Table VIII shows the performance comparisons between analog PIM and digital PIM when executing VGG-8 on the CIFAR-10 dataset. Here, we use the device and circuits parameters extracted from the PIM device and macro designs [10], [31]. For the latency, the analog PIM and the digital PIM show similar performance. These small performance differences are mainly influenced by the metrics of memory devices and interface circuits. However, analog PIM architectures suffer from significant weaknesses in terms of area, energy, and accuracy. In the analog PIM architecture, high-resolution ADCs are required for high CNN accuracy, which account for ∼80% of the total area and energy consumption. Besides, the ADC quantization error and other device-level analog noise cause MVM results deviation, resulting in the NN accuracy loss.

H. Device-Level Case Study

Emerging memory technologies like memristor have shown great potential in storage density and *in-situ* computing capability. Despite these virtues, the device parameters and non-ideal factors may cause unexpected accuracy loss. Because of the memory device description interface of *MNSIM 2.0*, researchers can easily analyze the influence of various device parameters. Here, we consider three kinds of device parameters.

1) Resistance Variations: They mean the deviation between the actual and ideal resistance values. *MNSIM 2.0* models the variation as a Gaussian noise added to the resistance and uses the noise variance to describe the intensity of the resistance variation. Table IX provides the PIM-based NN accuracy under different noise variances. The results demonstrate that VGG is more tolerant to variations than LeNet.

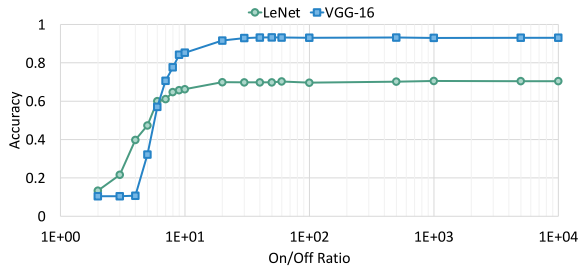


Fig. 18. PIM-based NN accuracy under different on/off ratio configurations (i.e., R_{on}/R_{off}) @ CIFAR-10.

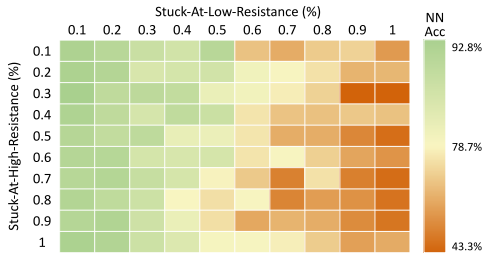


Fig. 19. PIM-based NN accuracy under different SAF probabilities (CNN model: VGG-8 @ CIFAR-10).

2) On/Off Ratio: It is the ratio of the high resistance state to the low resistance state. The high on/off ratio makes the data stored in the memory array more closely to the ideal value while mitigating the effects of resistance variations. Fig. 18 shows the NN accuracy under different on/off ratios, indicating the on/off ratio > 50 conduces to maintain high accuracy.

3) Stuck-At-Fault: It refers to a damaged memory device that can not be written correctly. Fig. 19 shows the heat map of computing accuracy *w.r.t* SAF probabilities. Compared with the low-resistance faults, the increased probability of the high resistance faults does not cause a significant decrease in accuracy. This is due to the fact that sparse NNs have a higher proportion of bit-0, and the high resistance faults (i.e., stored data becoming 0) have a lower impact on accuracy.

IX. Conclusion and Future Work

In this article, we propose *MNSIM 2.0*, a behavior-level modeling tool for both digital and analog PIM architectures. The proposed tool can help architecture and algorithm designers to fast evaluate the performance of PIM architectures and enable the design space exploration, device parameters influence analysis, and design insight discoveries. In the future, we will add more algorithm mapping strategies and circuit noise models to support other AI algorithms and PIM circuit modules.

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 91–99.

[2] G. Hu et al., "When face recognition meets with deep learning: An evaluation of convolutional neural networks for face recognition," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, 2015, pp. 142–150.

[3] M. M. S. Aly et al., "The N3XT approach to energy-efficient abundant-data computing," *Proc. IEEE*, vol. 107, no. 1, pp. 19–48, Jan. 2019.

[4] P. Chi et al., "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 27–39.

[5] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 14–26.

[6] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined ReRam-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 541–552.

[7] P. Yao et al., "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[8] J. Yue et al., "15.2 A 2.75-to-75.9TOPS/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong CIM with simultaneous computation and weight updating," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 64, 2021, pp. 238–240.

[9] Y.-D. Chi et al., "16.4 An 89TOPS/W and 16.3TOPS/mm^2 all-digital SRAM-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 64, 2021, pp. 252–254.

[10] B. Yan et al., "A 1.041-Mb/mm^2 27.38-TOPS/W signed-INT8 dynamic-logic-based ADC-less SRAM compute-in-memory macro in 28nm with reconfigurable bitwise operation for AI and embedded applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 65, 2022, pp. 188–190.

[11] L. Xia et al., "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1009–1022, May 2018.

[12] M.-Y. Lin et al., "DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[13] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[14] W. Zhang et al., "Design guidelines of RRAM based neural-processing-unit: A joint device-circuit-algorithm analysis," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[15] A. BanaGozar et al., "CIM-SIM: Computation in memory simuiator," in *Proc. 22nd Int. Workshop Softw. Compilers Embedded Syst.*, 2019, pp. 1–4.

[16] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2019, pp. 1–4.

[17] Q. Zheng et al., "PIMulator-NN: An event-driven, cross-level simulation framework for processing-in-memory-based neural network accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 12, pp. 5464–5475, Dec. 2022.

[18] S. J. E. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[21] G. Krishnan, S. K. Mandal, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao, "Interconnect-aware area and energy optimization for in-memory acceleration of DNNs," *IEEE Design Test*, vol. 37, no. 6, pp. 79–87, Dec. 2020.

[22] S. K. Mandal, R. Ayoub, M. Kishinevsky, and U. Y. Ogras, "Analytical performance models for NoCs with multiple priority traffic classes," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 1–21, 2019.

[23] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, Jun. 2017.

[24] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[25] H. Sun, Z. Zhu, Y. Cai, X. Chen, Y. Wang, and H. Yang, "An energy-efficient quantized and regularized training framework for processing-in-memory accelerators," in *Proc. 25th Asia South Pac. Des. Autom. Conf. (ASP-DAC)*, 2020, pp. 325–330.

[26] Z. Zhu et al., "A configurable multi-precision CNN computing framework based on single bit RRAM," in *Proc. 56th ACM/IEEE Des. Autom. Conf. (DAC)*, 2019, pp. 1–6.
[27] Q. Liu et al., "33.2 A fully integrated analog ReRAM based 78.4TOPS/W compute-in-memory chip with fully parallel MAC computing," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 500–502.
[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[29] "CIFAR-10—Object recognition in images." Kaggle. 2014. [Online]. Available: https://www.kaggle.com/c/cifar-10
[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[31] W. Wu, H. Wu, B. Gao, N. Deng, and H. Qian, "Suppress variations of analog resistive memory for neuromorphic computing by localizing V_o formation," *J. Appl. Phys.*, vol. 124, no. 15, 2018, Art. no. 152108.
[32] L. Kull et al., "28.5 A 10b 1.5GS/s pipelined-SAR ADC with background second-stage common-mode regulation and offset calibration in 14nm CMOS FinFET," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2017, pp. 474–475.
[33] H. Chen, X. Zhot, Q. Yu, F. Zhang, and Q. Li, "A >3GHz ERBW 1.1GS/s 8b two-sten SAR ADC with recursive-weight DAC," in *Proc. IEEE Symp. VLSI Circuits*, 2018, pp. 97–98.
[34] K. D. Choo, J. Bell, and M. P. Flynn, "27.3 area-efficient 1GS/s 6b SAR ADC with charge-injection-cell-based DAC," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2016, pp. 460–461.
[35] B. Nasri, S. P. Sebastian, K.-D. You, R. RanjithKumar, and D. Shahrjerdi, "A 700 μW 1GS/s 4-bit folding-flash ADC in 65nm CMOS for wideband wireless communications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2017, pp. 1–4.
[36] M. Shafique and R. Sharma, "Comparative study of sense amplifiers for SRAM," *Int. J. Eng. Res. Technol.*, vol. 4, no. 32, pp. 1–4, 2016.
[37] H. Sun et al., "Gibbon: Efficient co-exploration of NN model and processing-in-memory architecture," in *Proc. Des. Autom. Test Eur. Conf. Exhibit. (DATE)*, 2022, pp. 867–872.

Zhenhua Zhu received the B.S. degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2018, where he is currently pursing the Ph.D. degree.

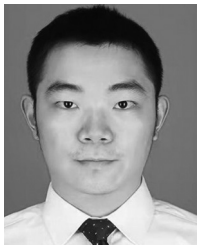
His research mainly focuses on processing-in-memory, near-memory-computing, computer architecture, and software–hardware co-optimization.

Hanbo Sun received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Electronic Engineering.

His research interests focus on processing-in-memory systems.

Tongxin Xie is currently pursuing the B.S. degree with the Department of Electronic Engineering, Tsinghua University, Beijing, China.

His research interests focus on processing-in-memory.



Yu Zhu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2021, where he is currently pursuing the Ph.D. degree with the Department of Electronic Engineering.

His research interests include in-memory logic, in/near-memory computing, and architecture design of graph computing.

Guohao Dai (Member, IEEE) received the B.S. and Ph.D. (Hons.) degrees from Tsinghua University, Beijing, China, in 2014 and 2019, respectively.

He is joining Shanghai Jiao Tong University, Shanghai, China, as an Associate Professor. His research mainly focuses on large-scale sparse graph computing, heterogeneous hardware computing, and emerging hardware architecture.

Dr. Dai has received the Best Paper Award in ASP-DAC 2019 and the Best Paper Nomination in DAC 2022 and DATE 2018.

Lixue Xia received the B.S. and Ph.D. (Hons.) degrees in electronic engineering from Tsinghua University, Beijing, China, in 2013 and 2018, respectively.

His research mainly focuses on energy-efficient hardware computing system design and neuromorphic computing system based on emerging nonvolatile device.

Dimin Niu received the Ph.D. degree in computer science from Pennsylvania State University, State College, PA, USA, in 2013.

He is a Research Scientist with the Computing Technology Laboratory, Alibaba DAMO Academy, Hangzhou, China. His current research interests include computer architecture, memory architecture, storage system, process-in-memory, and domain-specific architectures.

Xiaoming Chen (Member, IEEE) received the B.S. and Ph.D. (Hons.) degrees in electronic engineering from Tsinghua University, Beijing, China, in 2009 and 2014, respectively.

He is currently an Associate Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He has published more than 100 papers in top-tier conferences and journals. His research interests include computer-aided design for integrated circuits and advanced computer architecture design.

Dr. Chen was a recipient of the 2015 European Design and Automation Association Outstanding Dissertation Award, the 2018 DAMO Academy Young Fellow Award, and the ASP-DAC'22 Best Paper Award.

Xiaobo Sharon Hu (Fellow, IEEE) received the B.S. degree from Tianjin University, Tianjin, China, the M.S. degree from the Polytechnic Institute of New York, Brooklyn, NY, USA, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1989.

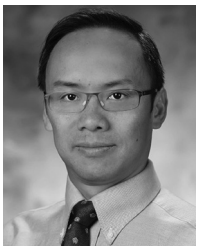
She is currently a Professor with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA. Her research interests include computing with beyond-CMOS technologies, low-power system design, and cyber–physical systems.

Prof. Hu received the NSF CAREER Award in 1997, the Best Paper Award from DAC in 2001, and the ACM/IEEE ISLPED in 2018. She served as the General Chair for DAC'18. She is the Editor-in-Chief of ACM TODAES and a Fellow of ACM.

Yu Cao (Fellow, IEEE) received the B.S. degree in physics from Peking University, Beijing, China, in 1996, and the M.A. degree in biophysics and the Ph.D. degree in electrical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 1999 and 2002, respectively.

He worked as a Summer Intern with Hewlett-Packard Labs, Palo Alto, CA, USA, in 2000, and IBM Microelectronics Division, East Fishkill, NY, USA. in 2001. After working as a Postdoctoral Researcher with the Berkeley Wireless Research Center, Berkeley, he is currently a Professor of Electrical Engineering with Arizona State University, Tempe, AZ, USA. He has published numerous articles and two books on nano-CMOS modeling and physical design. His research interests include neural-inspired computing, hardware design for on-chip learning, and reliable integration of nanoelectronics.

Dr. Cao was a recipient of the 2022 Best IP Award at Design, Automation and Teste in Europe Conference, the 2022 Distinguished Lecturer of the IEEE Circuits and Systems Society, the 2020 Intel Outstanding Researcher Award, the 2012 Best Paper Award at IEEE Computer Society Annual Symposium on VLSI, the 2010, 2012, 2013, 2015, 2016, and 2021 Top 5% Teaching Award, Schools of Engineering, Arizona State University, the 2009 ACM SIGDA Outstanding New Faculty Award, the 2009 Promotion and Tenure Faculty Exemplar, Arizona State University, the 2009 Distinguished Lecturer of IEEE Circuits and Systems Society, the 2008 Chunhui Award for outstanding oversea Chinese scholars, the 2007 Best Paper Award at International Symposium on Low Power Electronics and Design, the 2006 NSF CAREER Award, the 2006 and 2007 IBM Faculty Award, the 2004 Best Paper Award at International Symposium on Quality Electronic Design, and the 2000 Beatrice Winner Award at International Solid-State Circuits Conference. He served as Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and on the technical program committee of many conferences.



Yuan Xie (Fellow, IEEE) received the B.S. degree from Tsinghua University, Beijing, China, and the M.S. and Ph.D. degrees from the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, in 2002.

He was with IBM Microelectronics Division's Worldwide Design Center, Burlington, NJ, USA; Pennsylvania State University, State College, PA, USA; AMD Research, Beijing; and University of California at Santa Barbara, Santa Barbara, CA, USA.

Dr. Xie is a recipient of the NSF CAREER Award and the IEEE Computer Society Edward J. McCluskey Technical Achievement Award in 2020. He is a Fellow of ACM and AAAS.

Huazhong Yang (Fellow, IEEE) received the B.S. degree in microelectronics and the M.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993, and 1998, respectively.

He joined the Department of Electronic Engineering, Tsinghua University in 1993, where he has been a Professor since 1998. He has been in charge of several projects, including projects sponsored by the national science and technology major project, 863 program, NSFC, and several international research projects. He has authored and coauthored over 500 technical papers, seven books, and over 180 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain-inspired computing.

Prof. Yang was awarded the Distinguished Young Researcher by NSFC in 2000, the Cheung Kong Scholar by the Chinese Ministry of Education (CME) in 2012, the Science and Technology Award First Prize by China Highway and Transportation Society in 2016, and the Technological Invention Award First Prize by CME in 2019. He has also served as the Chair for the Northern China ACM SIGDA Chapter Science 2014, the General Co-Chair for ASPDAC'20, a Navigating Committee Member for AsianHOST'18, and a TPC Member for ASP-DAC'05, APCCAS'06, ICCCAS'07, ASQED'09, and ICGCS'10.

Yu Wang (Fellow, IEEE) received the B.S. and Ph.D. (Hons.) degrees from Tsinghua University, Beijing, China, in 2002 and 2007.

He is currently a tenured Professor with the Department of Electronic Engineering, Tsinghua University. His research interests include brain-inspired computing, parallel circuit analysis, application-specific acceleration and power/reliability-aware circuit, and system design methodology. He has authored and coauthored more than 350 papers in refereed journals and conferences.

Dr. Wang has received the Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA 2017, and ISVLSI 2012, the Best Poster Award in HEART 2012, and 11 Best Paper Nominations, such as DAC22, ICT18, DATE18, DAC17, ASPDAC16, ASPDAC14, and ASPDAC12, two in ASPDAC10, ISLPED09, and CODES09. He is a recipient of the Alexander von Humboldt Fellowship in 2019, the DAC Under-40 Innovators Award in 2018, and the IBM X10 Faculty Award in 2010. He served as the TPC Chair for ICFPT 2019 and 2011 and ISVLSI 2018, the Finance Chair for ISLPED 2012–2016, the Track Chair for DATE 2017–2019 and GLSVLSI 2018, the Program Committee Member for leading conferences in these areas, including top EDA conferences, such as DAC, DATE, ICCAD, and ASP-DAC, and top FPGA conferences, such as FPGA and FPT. He currently serves as the Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, *ACM Transactions on Design Automation of Electronic Systems*, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.