MNSIM 2.0: A Behavior-Level Modeling Tool for Memristor-based Neuromorphic Computing Systems

Zhenhua Zhu[1*], Hanbo Sun[1*], Kaizhong Qiu[1*], Lixue Xia[2], Gokul Krishnan[3], Guohao Dai[1], Dimin Niu[2], Xiaoming Chen[4], X. Sharon Hu[5], Yu Cao[3], Yuan Xie[6], Yu Wang[1], Huazhong Yang[1]
[1]Dept. of EE, BNRist, Tsinghua University, [2]Alibaba Group, [3]Arizona State University, [4]Institute of Computing Technology, Chinese Academy of Sciences, [5]University of Notre Dame, [6]University of California, Santa Barbara
yu-wang@tsinghua.edu.cn

ABSTRACT

Memristor based neuromorphic computing systems give alternative solutions to boost the computing energy efficiency of Neural Network (NN) algorithms. Because of the large-scale applications and the large architecture design space, many factors will affect the computing accuracy and system's performance. In this work, we propose a behavior-level modeling tool for memristor-based neuromorphic computing systems, MNSIM 2.0, to model the performance and help researchers to realize an early-stage design space exploration. Compared with the former version and other benchmarks, MNSIM 2.0 has the following new features: 1. In the algorithm level, MNSIM 2.0 supports the inference accuracy simulation for mixed-precision NNs considering non-ideal factors. 2. In the architecture level, a hierarchical modeling structure for PIM systems is proposed. Users can customize their designs from the aspects of devices, interfaces, processing units, buffer designs, and interconnections. 3. Two hardware-aware algorithm optimization methods are integrated in MNSIM 2.0 to realize software-hardware co-optimization.

ACM Reference Format:
Zhenhua Zhu, Hanbo Sun, Kaizhong Qiu, Lixue Xia, Gokul Krishnan, Guohao Dai, Dimin Niu, Xiaoming Chen, X. Sharon Hu, Yu Cao, Yuan Xie, Yu Wang, Huazhong Yang. 2020. MNSIM 2.0: A Behavior-Level Modeling Tool for Memristor-based Neuromorphic Computing Systems . In *Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3407647

1 INTRODUCTION

In the past few years, Convolutional Neural Networks (CNNs) have demonstrated powerful capabilities in many fields. However, in addition to the high classification accuracy, the amount of network parameters and computations increase dramatically as the CNN models become more and more complex, which makes CNNs cause high energy consumption and long computation time.

Existing work has shown that emerging memristors (*e.g.*, RRAM, Resistive Random Access Memory) and memristor-based Processing-In-Memory (PIM) architectures can improve CNN computing performance and energy efficiency by over 100× compared with both CMOS ASIC and GPU solutions [2, 16]. The performance gain of PIM systems comes from the feature that memristor crossbar structures can perform the Matrix-Vector-Multiplications (MVMs) in memory and eliminate the weight data movements between memory and computing units in traditional von Neumann architectures, which are also the most time and energy consuming part in CNNs.

In memristor-based PIM systems, due to the huge architecture design space and large-scale algorithm models, the SPICE simulation will take an unacceptable long time for simulating the entire system [20]. To overcome this challenge, researchers have proposed several behavior-level simulators or modeling frameworks to evaluate the performance within a short time [6, 12, 15, 20, 21]. MNSIM [20] is a behavior-level simulation platform, which provides a hierarchical hardware structure abstraction to model different PIM accelerators. Although the processing elements can be quickly evaluated in MNSIM, it lacks simulation of other digital parts in PIM systems. Besides, the accuracy simulation of MNSIM is only applicable to MVM of a single memristor crossbar without considering the accuracy of the entire CNN model. NeuroSim [15] and XPESim [21] are two circuits-level modeling tools to benchmark the performance of PIM systems. They all provide detailed circuits-level evaluations from the aspects of device technologies, non-ideal parameters, and network topology. The target users of these tools are mainly device researchers. But in the algorithm and architecture level, these two tools lack flexibility in supporting various CNN structures and different architecture designs. DL-RSIM [12] and PytorX [6] are designed for analyzing the CNN inference accuracy in PIM systems. DL-RSIM focuses on estimating the accuracy with the consideration of device non-ideal factors and data mapping strategy onto low precision devices. PytorX not only evaluates computing accuracy, but also settles down these non-ideal effects from hardware and software approaches. Nevertheless, hardware performance evaluation is missed in these tools.

In this paper, we propose MNSIM 2.0* on the basis of MNSIM to model the CNN computing accuracy and hardware performance (*i.e.*, area, power, energy, and latency) in behavior-level. MNSIM 2.0 is developed for memristor-based PIM architecture designers and CNN algorithm researchers who want to fast evaluate the CNN accuracy and hardware performance of their architecture and algorithm model design. The contributions of this paper include:

*: All authors contributed equally to this work.

GLSVLSI '20, September 7–9, 2020, Virtual Event, China

ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3407647

*The code is avaliable in https://github.com/thu-nics/MNSIM-2.0.git

(1) **In the algorithm level**, MNSIM 2.0 can simulate the inference accuracy of mixed-precision CNNs considering data splitting strategy and memristor non-ideal factors. MNSIM 2.0 also integrates two hardware aware CNN optimization strategies, *i.e.*, PIM overhead aware mixed-precision CNN quantization method and energy efficient non-uniform activation quantization method, to optimize CNN models for PIM systems.

(2) **In the hardware level**, we propose a hierarchical modeling structure to describe the configurable mixed-precision CNN computing architecture. In MNSIM 2.0, users can make use of this hierarchical structure to customize their architecture designs from device to module interconnection. Furthermore, MNSIM 2.0 provides an inner-layer pipeline structure as the default configuration to evaluate computation latency and energy consumption. The pipeline structure leverages the CNN data flow characteristics to improve the computing parallelism and reduce computing latency.

(3) We propose a general CNN mapping and schedule module, which provides connections and interface between various algorithm models and architecture designs. It also conduces to explore the architecture design space.

(4) Case studies of MNSIM 2.0 are presented, which help us to analyze the trade-off between accuracy and hardware performance merits under different hardware configurations and CNN models.

2 PRELIMINARY

2.1 Convolutional Neural Network

In the algorithm level, MNSIM 2.0 aims to evaluate the CNN algorithm. 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 which can be expressed as:

$$F_o(x,y,z)=f\Big(\sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\sum_{k=1}^{C_{in}} F_i(x+i,y+j,k)w_z(i,j,k)\Big) \tag{1}$$

where 3-dimensional matrices F_o and F_i represent the input and output feature maps, respectively; w_z is the z^{th} 3-dimensional convolution kernel with the size of $K \times K \times C_{in}$; K is the kernel size and C_{in} is the number of input channels. The number of kernels is equal to the output channel number as denoted as C_{out}. $f(\cdot)$ is a nonlinear activation function. The computations in FC layers are similar to those in CONV layers.

2.2 Memristor Basics

Memristor is a kind of non-volatile memory which stores information in the form of different resistance values. Crossbar is an area-efficient memristor connection structure, which also has the capability of combining the storage and computing together. In PIM systems, the input vector is represented by a voltage vector $\boldsymbol{V}$ applied to the word-line (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 bit line (BL):

$$i_{out,k} = \sum_{j=1}^{N} g_{k,j} v_{in,j} \tag{2}$$

where $v_{in,j}$, $i_{out,k}$ represent the element of the input voltage vector $\boldsymbol{V}$ and the output current vector $\boldsymbol{I}$; $g_{k,j}$ is the conductance of the (i,j) cell. Because memristor crossbar performs MVMs in the analog domain, some interfaces (*e.g.* Anong-to-Digital Converters (ADCs), Digital-to-Analog Converters (DACs)) are required between crossbars and peripheral circuits.

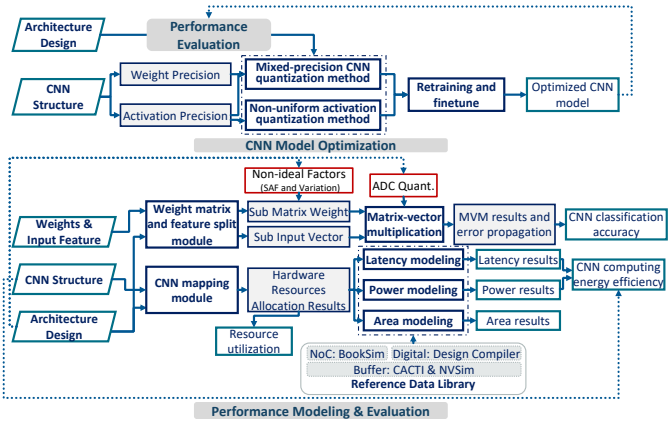

Figure 1: The overview of MNSIM 2.0. Up: CNN model optimization flow; Down: Performance modeling flow

3 MNSIM 2.0 OVERVIEW

The overview of MNSIM 2.0 is shown in Figure 1, which consists of the CNN model optimization flow and the performance modeling and evaluation flow. These two parts form a closed feedback loop, *i.e.*, the CNN model optimization flow utilizes modeling results generated by the modeling and evaluation flow to guide the CNN quantization strategy, then the optimized algorithm parameters are sent to the modeling and evaluation flow to further assess the performance of PIM systems.

The CNN model optimization flow is designed to realize PIM hardware-aware quantization methods rather than modify the network structure (*e.g.*, layer number and connections). The model optimization flow contains the mixed-precision CNN quantization method and the non-uniform activation quantization method. The former performs the layer-wise weights and input activation quantization based on PIM computing performance, which can reduce the storage overhead and latency while maintaining comparable accuracy. The latter method aims at introducing the non-uniform quantization method into the output activation of each layer to reduce the requirement for ADC resolution.

The modeling and evaluation flow contains two parts: hardware performance modeling (Section 5) and CNN accuracy estimation (Section 6). For the hardware performance modeling part, it makes use of CNN structures and the architecture design to model the hardware performance. Firstly, the algorithm mapping module completes the hardware resource allocation in terms of the architecture parameters. It also determines the computing units utilization and their connection relationship. Then, the specific data flow is constructed with considerations of CNN structure, unit connections, and Network-on-Chip (NoC) structure. Finally, the overall performance of the PIM system is estimated according to the specific data flow, resource utilization, module performance, and reference data obtained from other simulators. For the accuracy estimation part, it takes the exact weights and input features (given by the CNN model optimization flow) and the architecture design as inputs. The classification accuracy of the entire CNN model is obtained by four

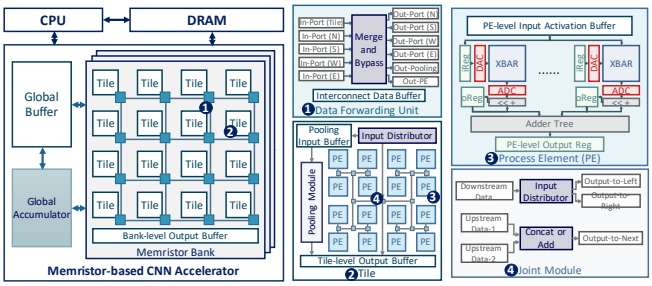

Figure 2: Left: The architecture design used in MNSIM 2.0; Right: Details of the Data Forwarding Unit, the Memristor Tile, the Process Element (PE), and the Joint Module

steps: splitting data, introducing non-ideal factors and quantization error, merging MVM results, and propagating error.

4 HIERARCHICAL STRUCTURE FOR PIM

MNSIM 2.0 provides a hierarchical modeling structure and a basic architecture design for describing different architectures, which are shown in Figure 2. From the system point of view, the proposed PIM system is composed of CPU, DRAM, and memristor-based CNN accelerator. In our design, the CPU is responsible for controlling the weights writing in the model deployment phase and waking up the accelerator in the inference phase. DRAM stores weight parameters and input features during the deployment phase and the inference phase, respectively. When the model is deployed, the CNN accelerator reads the input feature map from DRAM after waking up and uses the on-chip buffer for storing all the intermediate data during the computation. After the accelerator completes inference computation, the results are returned to the CPU. It should be noted that MNSIM 2.0 mainly simulates the inference computation. The simulation of CNN training will be supported in our future work.

The memristor-based CNN accelerator includes several memristor banks, the global buffer, and the global accumulator. The global buffer and accumulator are responsible for calculating the element sum layer in the bypass network structure. For the memristor bank, we propose a basic architecture which supports mixed-precision CNN computing and can be used to describe other existing architecture designs (*e.g.*, PRIME[2] and ISAAC[16]) with a few modifications. From the high level to the low level, the hierarchical structures of this architecture include memristor banks, memristor tiles, processing elements (PEs), and memristor crossbars.

In each **memristor bank**, an array of memristor tiles are organized and connected in a way similar to NoC. To reduce the complexity of control logic and data path, we specify each tile will only process one layer of CNN, while for some large-scale layers, matrix splitting and multiple tiles will be needed. Different memristor banks are connected with buses and use the bank level output buffer for data communications.

One **memristor tile** is adjacent to a data forwarding unit, which receives data from other tiles, merges (*i.e.*, add or concatenate) them, and outputs the result to the local tile or other tiles. According to the layer type, tiles can be configured as pooling mode or MVM mode, which are realized by the pooling module and the process elements (PEs) array. The memristor PEs in one tile are linked as an H-Tree structure to reduce the intra tile interconnection overhead. Each connection node of the H-Tree is a joint module, which manages the data forwarding and summations of PE results.

Table 1: Architecture Configuration Parameters of MNSIM 2.0 (R/W represents Read and Write, ADDA means ADC/DAC)

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

PE mainly contains multiple memristor crossbars with peripheral circuits (*i.e.*, drivers, DACs, and ADCs). To solve the limited device precision problem and to support mixed-precision algorithms, multiple low precision memristor crossbars in one PE are used to store the high precision weight values. For example, eight 1-bit memristor crossbars are required for storing 8-bit CONV kernels. Besides, if the weights are signed value, the crossbar number will be doubled (Positive and Negative Crossbars) in order to store the signed weights in the positive conductance. Computing results of different crossbars are merged together by shifter and adder tree.

The detailed architecture configuration parameters are listed in Table 1. Users can configure their own architecture design from different levels. For example, in ISAAC chip [16], it has 16 Tiles and each Tile contains 12 In-Situ Multiply Accumulates (IMAs). There are eight 128×128 memristor crossbars. In order to describe it with our hierarchical structure, we can set the Bank Number to one, regard IMA as PE in our design, and configure the parameters of Table 1 to the values given by ISAAC.

5 HARDWARE PERFORMANCE MODELING

5.1 CNN Model Mapping and Hardware Resource Allocation

Ideally, in memristor-based PIM CNN accelerators, the 3D CONV kernel is flattened into 1D column vector and mapped onto devices in one column of memristor crossbars. Different kernels in one layer are placed in different columns of the same crossbar. However, on account of the limited device precision and crossbar size, it is difficult to map a large-scale network layer in one crossbar. Therefore, we split the CONV kernels from two dimensions, *i.e.*, weight precision and kernel size, to complete the mapping procedure, as shown in Figure 3(a). In the weight precision splitting, if the weight precision is higher than the number of memristor resistance levels, we use several crossbars in one PE to store some bits of CONV kernels. For example, in Figure 3(a), we assume the memristor is 1-bit/cell while the weights' precision is P_w bit, then we need P_w crossbars in one PE for storage. Therefore, we can flexibly deploy mixed-precision CNN by using different numbers of crossbars for different layers. In the kernel size splitting, if the number of output channels and the kernel vector length exceeds the crossbar size, multiple PEs are needed in the horizontal and vertical direction

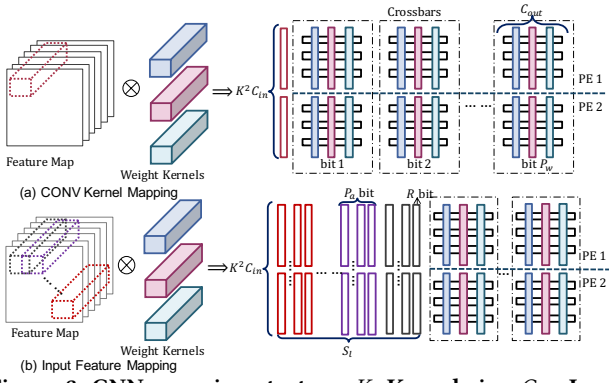

Figure 3: CNN mapping strategy. K: Kernel size, C_{in}: Input channel, C_{out}: Output channel, S_l: Sliding times, P_w: Weight precision, P_a: Input precision, R: DAC resolution

respectively, each of which processes a part of the CONV layer. According to the CONV kernel mapping strategy, we can get the tile allocation results for CONV layers in Equation 3:

$$\#Tile_{CONV} = \left\lceil \frac{\left\lceil \frac{C_{in}}{\lfloor xbar_r/K^2 \rfloor} \right\rceil \left\lceil \frac{C_{out}}{xbar_c} \right\rceil \left\lceil \frac{P_w}{\#xbar/PE} \right\rceil}{\#PE/Tile} \right\rceil \quad (3)$$

where $xbar_r, xbar_c$ are the row/column number of memristor crossbar, P_w represents the weight precision. Similarly, denoting the the length of the input/output feature in FC layers as $Length_{in/out}$, we can get the tile number used in FC layers:

$$\#Tile_{FC} = \left\lceil \frac{\left\lceil \frac{Length_{in}}{xbar_r} \right\rceil \left\lceil \frac{Length_{out}}{xbar_c} \right\rceil \left\lceil \frac{P_w}{\#xbar/PE} \right\rceil}{\#PE/Tile} \right\rceil \quad (4)$$

The input feature mapping is shown in Figure 3(b). Corresponding to the expansion of CONV kernels, the 3D input feature block is converted to a vector, which will be transformed into voltage pulse signals through DACs and loaded onto the WLs of crossbars. Different from the spatial expansion in multiple crossbars 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 feature map is higher than the DAC's resolution, the input needs to be split and loaded onto the WLs in multiple cycles.

5.2 Data Flow Construction

MNSIM 2.0 utilizes the inner-layer pipeline scheme to increase computing parallelism. The basic idea of it is to start computation once the required input data of this layer are ready, rather than waiting for the previous layer to complete all the calculations. If we want to calculate the output $(r, c)_i$ in layer i, the coordinate of the last output that the previous layer needs to provide is:

$$r_{req} = \begin{cases} \min(H_{i-1}, K_{i-1} + s_{t,i-1} \times (r-1) - p_{i-1})) & \text{CONV} \\ H_{i-1} & \text{FC} \end{cases} \quad (5)$$

$$c_{req} = \begin{cases} \min(W_{i-1}, K_{i-1} + s_{t,i-1} \times (c-1) - p_{i-1}) & \text{CONV} \\ W_{i-1} & \text{FC} \end{cases} \quad (6)$$

where H_{i-1} and W_{i-1} are the height and width of the output feature map of layer $i-1$, and s_t, p represent stride size and padding, respectively. According to Equation 5~6, when the calculation of the previous layer $(i-1)$'s output $(r_{req}, c_{req})_{i-1}$ is finished, the next layer can start to calculate the $(r, c)_i$ output.

As mentioned before, if the size of one layer exceeds the amount of parameters that one memristor tile can store, multiple tiles are

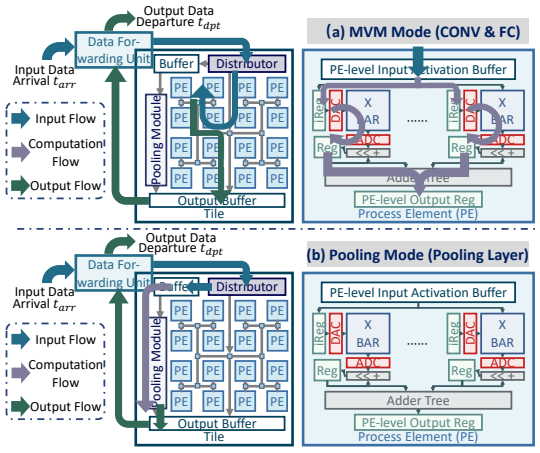

Figure 4: The data path of the memristor tile

required and their results need to be merged together before transferring to the next layer. The 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 Figure 4. The entire data flow within the memristor tile is composed of three parts:

1. Input data flow (the cyan-blue arrow): For the MVM mode, the input data flow starts from the data forwarding unit and goes through the H-tree to PEs, then the input data are written to the input activation buffer. For the pooling mode, the input data flow bypasses the H-tree structure and ends to the pooling buffer.

2. Computation data flow (the violet arrow): For the MVM mode, the input activation 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 crossbar needs to keep activated for multiple cycles to complete the calculation of the input activation:

$$mul = \left\lceil \frac{P_a}{Res_{DAC}} \right\rceil \left\lceil \frac{K^2 \min(\left\lfloor \frac{xbar_r}{K^2} \right\rfloor, C_{in})}{Parallel_r} \right\rceil \left\lceil \frac{\min(C_{out}, xbar_c)}{Parallel_c} \right\rceil \quad (7)$$

In Equation 7, the first term represents that multiple cycles are required for loading the high precision activation (P_a) to the low resolution DAC (Res_{DAC}), the second and the third terms show the limited WL/BL computation parallelism ($Parallel_r, Parallel_c$) increases the computation latency. In each cycle, several input activation bits are loaded to the DAC of the crossbar, then the crossbar calculation starts. After that, the analog values generated by crossbars are converted to the digital form by ADCs. The temporary results are shifted and added with the previous computation step's results and are stored in the output reg. After *mul* cycles, the adder tree merges all the crossbars' results in this PE.

3. Output data flow (the green arrow): The output data flow is similar to the input data flow. The results of PEs are firstly merged by the joint module, then the merged PE results and pooling results are sent to the output buffer. Data forwarding units read data from output buffer and send them to other tiles.

5.3 Area, Power, and Latency Models

5.3.1 Memristor Crossbar. Because the device parameters and characteristics are various, MNSIM 2.0 provides the device description interface in Table 1 to model different device technologies. For the area estimation, users can provide information from three aspects: crossbar area, device area, and technology node, as shown

in Equation 8:

$$Xbar\_Area = \begin{cases} User`s\ Given\ Value & \\ xbar_c \times xbar_r \times Device\_Area & \\ xbar_c \times xbar_r \times 3(W/L+1)F^2 & 1T1R \\ xbar_c \times xbar_r \times 4F^2 & 0T1R \end{cases} \quad (8)$$

where $1T1R$ and $0T1R$ represent MOSFET-accessed and cross-point structure, W/L is the transistor technology parameter, and F is memristor technology node. Similarly, for the latency estimation, users can directly provide the crossbar-level latency or give the device read latency. For the latter one, we model the crossbar latency by introducing the RC delay caused by interconnection wires. In the power consumption model, MNSIM 2.0 uses the percentage of each value in feature map and weights to estimate the equivalent resistance and voltage (pulse number or amplitude). For example, Equation 9 shows the calculation of equivalent resistance:

$$1/R_{eq} = \prod_{i=0}^{L-1} \alpha_i / R_i \quad (9)$$

where L is the number of resistance level, α_i is the value percentage, and R_i is the resistance of each level. If the device precision is 1-bit, then $R_{0/1}$ is the low/high resistance state and $\alpha_{0/1}$ is the percentage of bit-0/bit-1.

5.3.2 Analog and Digital Conversion Interface. In MNSIM 2.0, the design of ADCs/DACs is mainly customized by users. The user needs to provide the converter performance parameters shown in Table 1. Besides, MNSIM 2.0 also includes some state-of-the-art DAC and ADC designs with different resolutions as default values.

5.3.3 Digital Circuits. In PIM systems, many digital modules are responsible for the CNN computations except the MVM part. MNSIM 2.0 synthesizes the digital circuits modules at TSMC 65*nm* technology node by Synopsys® Design Compiler. The digital circuits 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.

5.3.4 Buffer Design. As shown in Figure 2, we use on-chip buffer to store the intermediate data. MNSIM 2.0 use CACTI [18] and NVSIM [4] to get reference data (area, power, bandwidth, etc.) of different memory technologies (*i.e.*, SRAM, DRAM, Non-Volatile 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.

5.3.5 NoC Simulation. Because the data flow in memristor bank is static and fixed during CNN computation, the tile interconnection graph model of MNSIM 2.0 is constructed in terms of the mapping strategy and the tile allocation result. For each layer, we specify one tile for merging other tiles' results. Then, the NoC communication is divided into two parts: from each tile to its merging tile in the same layer and from one merging tile to tiles in the next layer. We use Manhattan distance to describe these two parts and estimate the NoC latency. For the power and area estimation, we refer to the method used in [9, 13].

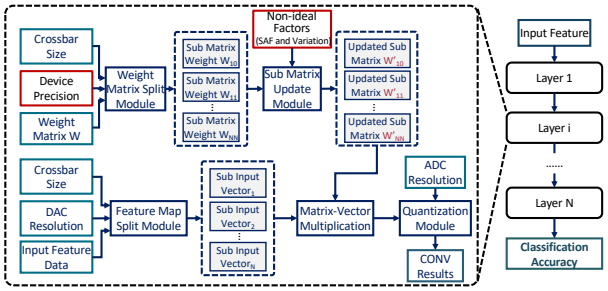

Figure 5: Accuracy evaluation of PIM-based CNN inference

6 ACCURACY ESTIMATION AND CNN MODEL OPTIMIZATION

6.1 CNN Accuracy Estimation Flow

The accuracy evaluation flow of PIM systems is illustrated in Figure 5, which is implemented within the PyTorch framework. Firstly, considering the crossbar size, memristor precision, and DAC resolution, we split the weight matrix and feature data into sub-matrices and sub-vectors, as mentioned in Section 5.1. Secondly, for each sub matrices, non-ideal factors are introduced to update the sub-matrix values. MNSIM 2.0 takes two major non-ideal factors of memristors in to consideration, *i.e.*, Stuck-At-Faults (SAFs) and resistance variations, other non-ideal factors will be updated in the the future. Thirdly, MVMs are performed between updated sub-matrices and sub vectors. Then, 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 later layers to get the final classification accuracy.

6.2 Hardware-Aware CNN Optimization

In MNSIM 2.0, we integrate two hardware-aware CNN optimization methods: mixed-precision CNN quantization method [22] and non-uniform quantization method [17].

6.2.1 Mixed-Precision CNN Quantization. In CNN algorithms, different layers have different quantization sensitives [22]. In order to maintain the accuracy, some layers' weights and activation must be kept in high precision (*e.g.*, 8-bit) while others can be quantized into low bit width. In PIM systems, the precision of weights and activation will affect the storage usage and latency according to Equation 3,4,7. Therefore, layer-wise quantization gives the opportunity to reduce storage burden and latency with little accuracy loss. MNSIM 2.0 uses greedy quantization strategy, which sets higher accuracy loss threshold for layers with more weights and sliding windows, to achieve greater performance improvement.

6.2.2 Non-Uniform Activation Quantization. Existing work has demonstrated that high resolution ADCs occupy the major energy consumption of PIM systems and their resolution has a crucial impact on computing accuracy [17]. In order to reduce the ADC energy consumption, MNSIM 2.0 integrates a PIM-based non-uniform activation method, which contains the quantization range optimization, high-precision scale implementation, and non-uniform ADC quantization. Compared with traditional uniform quantization, non-uniform quantization considers the data distribution and pays more attention to the interval with more data. As a result, ADC resolution can be reduced by 2-bit with lower power consumption.

Table 2: Latency (L, ms), power (P, W), and accuracy (A, %) under different ADC resolutions (R). B is algorithm baseline.

	LeNet			VGG-8			ResNet18		
R	L	P	A	L	P	A	L	P	A
B	-	-	76.27	-	-	91.64	-	-	91.18
4	0.26	0.34	10.81	10.45	9.67	10.81	2.82	11.97	10.81
6	0.28	0.54	49.45	11.56	15.44	53.91	3.06	19.33	10.81
8	0.29	1.55	72.72	12.32	43.67	90.18	3.22	55.32	87.81
10	0.28	2.62	76.63	12.29	73.76	91.45	3.20	93.68	88.91

7 CASE STUDIES

7.1 Experiment Setup

In our experiments, we use three typical CNNs as benchmarks, *i.e.*, LeNet [11], modified VGG-8 [8], and ResNet18 [5]. All experiments are evaluated on the Cifar-10 dataset [7]. The memristor we used refers to [19]. The ADC and DAC data come from [1, 3, 10, 14, 16]. The digital parts are modeled at 65nm with 500MHz.

7.2 Trade-off between Accuracy and Hardware Performance

The mainly computing error in memristor based CNN accelerators is caused by the quantization error of ADCs. Table 2 shows the CNN accuracy and hardware performance under different ADC configurations. The results demonstrate that the CNN computing power drops dramatically when the ADC resolution becomes lower. However, ADCs with lower resolution also bring higher quantization error, which will destroy the CNN function. Besides, experiment results show that different network models have different tolerance to the quantization error. Therefore, architecture designers should analyze the trade-off and select the appropriate ADC resolution according to the algorithm models and hardware performance constraints concurrently.

7.3 Trade-off among Latency, Area, and Power

Due to the place and routing limitations, multiple WLs/BLs usually share one DAC/ADC in crossbars. The area and power can be reduced by sharing interfaces but the latency increases. Figure 6 shows the latency, area and power changing trend *w.r.t* readout parallelism (*i.e.*, the number of ADCs that can be activated simultaneously in one crossbar). The higher the parallelism is, the more ADCs are activated at the same time, which reduces the computing latency at the cost of high area and power. We can extract two optimal points from the results. One is the power and area optimal point, which can reduces the latency by 70% with little additional overhead. And the other is the latency optimal point. It achieves more aggressive optimized latency results (~ 80%), but the overhead (*e.g.*, power increased by 25W) is not negligible in some cases.

8 CONCLUSION AND FUTURE WORK

In this paper, we propose MNSIM 2.0, a behavior-level modeling tool for memristor-based PIM systems, which consists of the CNN model optimization flow and the accuracy and hardware performance modeling flow. The proposed tool can help architecture and algorithm designers to fast evaluate the performance of PIM systems and realize an early-stage design space exploration. In the future, we will integrate MNSIM 2.0 with other circuits-level simulators to achieve more accurate simulation results.

9 ACKNOWLEDGEMENTS

This work was supported by National Key Research and Development Program of China (No. 2017YFA0207600), National Natural Science Foundation of China (No. 61832007, 61622403, 61621091, U19B2019), Beijing National Research Center for Information Science and Technology (BNRist), and Beijing Innovation Center for Future Chips. Chen's work was supported by the Beijing Academy of Artificial Intelligence under Grant BAAI2019QN0402. Dai's work was supported by China Postdoctoral Science Foundation (No. 2019M660641).

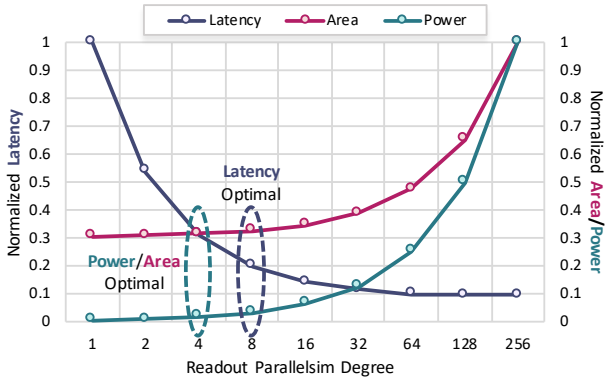

Figure 6: Normalized Latency, Area, and Power results under different readout parallelisms (CNN model: ResNet18)

REFERENCES

[1] H. Chen et al. 2018. A >3GHz ERBW 1.1GS/S 8B Two-Sten SAR ADC with Recursive-Weight DAC. In *VLSI-Circuits, 2018.* 97–98.
[2] P. Chi et al. 2016. PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In *ISCA, 2016.*
[3] K. D. Choo, J. Bell, and M. P. Flynn. 2016. 27.3 Area-efficient 1GS/s 6b SAR ADC with charge-injection-cell-based DAC. In *ISSCC, 2016.* 460–461.
[4] X. Dong et al. 2012. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE TCAD* (2012).
[5] K. He et al. 2016. Deep Residual Learning for Image Recognition. In *CVPR, 2016.*
[6] Z. He et al. 2019. Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping. In *DAC, 2019.* 1–6.
[7] Kaggle et al. 2014. CIFAR-10 - Object Recognition in Images. website. (2014). https://www.kaggle.com/c/cifar-10.
[8] S. Karen et al. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Computer Science* (2014).
[9] G. Krishnan et al. 2020. Interconnect-Aware Area and Energy Optimization for In-Memory Acceleration of DNNs. *IEEE Design and Test* (2020), 1–1.
[10] L. Kull et al. 2017. 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 *ISSCC, 2017.* 474–475.
[11] Y. LeCun et al. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE, 1998.* 2278–2324.
[12] M. Lin et al. 2018. DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning. In *ICCAD, 2018.* 1–8.
[13] Sumit K. Mandal et al. 2019. Analytical Performance Models for NoCs with Multiple Priority Traffic Classes. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 52 (Oct. 2019), 21 pages.
[14] B. Nasri et al. 2017. A 700 μW 1GS/s 4-bit folding-flash ADC in 65nm CMOS for wideband wireless communications. In *ISCAS, 2017.* 1–4.
[15] X. Peng et al. 2019. DNN+NeuroSim: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators with Versatile Device Technologies. In *IEDM, 2019.* 32.5.1–32.5.4.
[16] A. Shafiee et al. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-situ Analog Arithmetic in Crossbars. In *ISCA, 2016.*
[17] H. Sun et al. 2020. An Energy-Efficient Quantized and Regularized Training Framework For Processing-In-Memory Accelerators. In *ASPDAC, 2020.* 1–6.
[18] S. J. E. Wilton and N. P. Jouppi. 1996. CACTI: an enhanced cache access and cycle time model. *JSSC, 1996* 31, 5 (1996), 677–688.
[19] W. Wu et al. 2018. Suppress variations of analog resistive memory for neuromorphic computing by localizing Vo formation, In ISCAS, 2017. *Journal of Applied Physics* 124, 15.
[20] L. Xia et al. 2018. MNSIM: Simulation Platform for Memristor-Based Neuromorphic Computing System. *TCAD, 2018* (2018).
[21] W. Zhang et al. 2019. Design Guidelines of RRAM based Neural-Processing-Unit: A Joint Device-Circuit-Algorithm Analysis. In *DAC, 2019.* 1–6.
[22] Z. Zhu et al. 2019. A Configurable Multi-Precision CNN Computing Framework Based on Single Bit RRAM. In *DAC, 2019.* 1–6.