SCIENCE CHINA
Information Sciences

• **Supplementary File** •

CLEAR: A Full-Stack Chip-in-Loop Emulator for Analog RRAM based Computing-in-Memory System

Ruihua Yu[1], Wenqiang Zhang[1], Bin Gao[1*], Yiwen Geng[1], Peng Yao[1], Yuyi Liu[1], Qingtian Zhang[1], Jianshi Tang[1], Dong Wu[1], Hu He[1], Ning Deng[1], He Qian[1] & Huaqiang Wu[1]

[1]*School of Integrated Circuits, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing , China*

Appendix A Compiler Optimization

As shown in Fig. A1a, the optimization of the model mainly includes two parts, one is the combination and splitting of nodes, and the other is the optimization of the critical path. The specific optimization processes are as follows. After compiler optimization, the address information will be added into the intermediate representation (IR) as shown in Fig. A1c, which can decide the running backends (real chip or analog computing model) of each CIM-friendly layer in deep neural network models.

Appendix A.1 Node combination

The fusion and splitting of nodes include two cases. The first is to combine or split the layer according to the finest-grained scheduling instructions of the actual hardware. Sometimes due to the design of the hardware, the hardware instructions cannot directly give a command to a certain function layer, but can control a group of layers. It is necessary to combine layers that cannot be executed separately. Besides, for those operations that cannot be completed by hardware instructions at one time, they need to be split into a combination of some basic operations. For instance, the convolution (CONV) operation can be split into a combination of VMM as shown in Fig. A1b. The second is to fuse different layers for simplifying the model based on equivalent conversion. For example, the BatchNormalization layer and CONV layer can be fused by changing the parameters of the CONV layer. The new weight of CONV layer after fusion can be expressed as $w^{'} = w\beta/\sqrt{var}$, where w, beta and var represent the original weight, variance of IFM and learning parameters respectively.

Appendix A.2 Critical path reforming

When there are remaining hardware resources after all layers are mapped once, these resources can be re-allocated to improve the utilization and throughput. In the inference phase, CONV layers consume the most of the time. The throughput of inference can be maximized by reallocating XBs for CONV layers. Generally, the throughput of layer-pipeline CIM chip is always limited by the most computing-insentive layer in the longest path, which have the most layers. The longest path is defined as the critical path. Assuming that there are N CONV layers in the critical path, the weight of different CONV layers are mapped to different number of XBs, so we can compute the time for implementing each CONV operation according to the input size of each CONV layer respectively as $T_1, ..., T_L$. If available XBs of critical path are more than the XBs that are used in initial allocation of all layers on the critical path, we can improve the throughput by minimizing $max\{T_1X_1^0/X_1, ..., T_LX_L^0/X_L\}$ where X_l^0 and X_l represents the initial-allocated and re-allocated number of XBs for l-th CONV layer with critical path reforming method (Algorithm A1), respectively. For example, as shown in Fig. A2a, there is a 3-layer CNN model with two CONV layers and one FC layer. And we use a simplified chip architecture with 8 arrays (Fig. A2b) to explain the Algorithm A1. Firstly, we can allocate the array for each CONV or FC layer in the model accordding to the weight kernel size (iteration #0 in Fig. A2b). After that, there are some arrays unoccupied. And we can find the most time-consuming layer among them (CONV1, CONV2, FC). We mainly use $T_l^0 = t_{data} + t_{XB}$ to calculate the consumed time of each layer on chip, where t_{data} represents the data transmission time from one array to another array. t_{XB} represents the calculating time of analog computing for current feature map, and the layer with maximum will be selected to allocate more resources. So when the CONV1 is selected as the most time-consuming layer, one more array can be allocated to CONV1 (iteration #1 in Fig. A2b). The optimization will repeat this process until all arrays in the chips are allocated to at least one layer (iteration #5 in Fig. A2b).

* Corresponding author (email: gaob1@tsinghua.edu.cn)

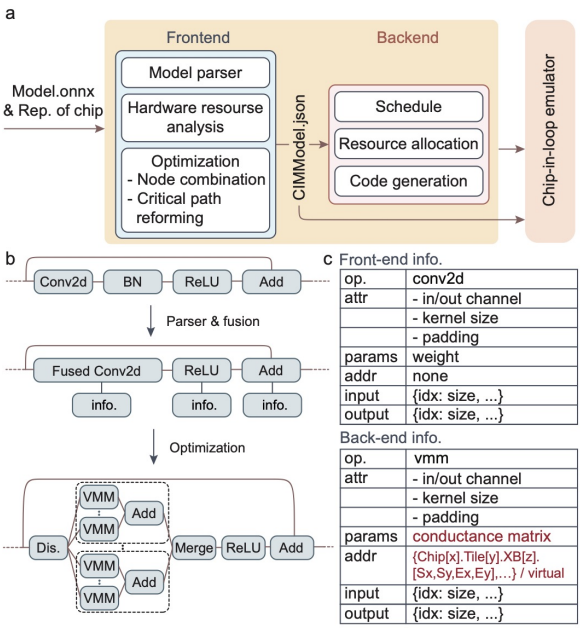

Figure A1 (a) The architecture of compiler,(b) workflow of the frontend in compiler and (c) the frontend and backend informaton of emulation-oriented IR

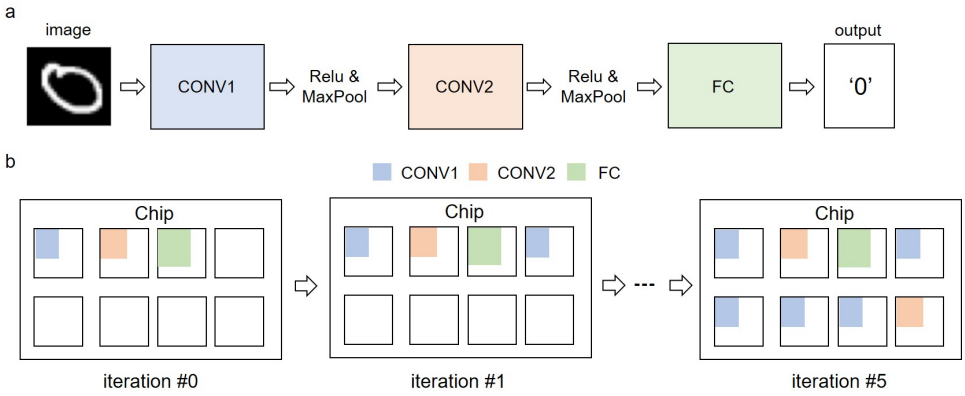

Figure A2 (a) A 3-layer CNN model for image classification and (b) the weight placement of 3-layer CNN model on chip during the iteration of optimization method.

Appendix B Emulation-oriented IR of CIM paradigm

After the model is optimized, we use a new data structure to represent the model in order to map the algorithm model to the hardware. The data format plays the role of emulation-oriented IR during the compilation process. In addition to the basic information such as the input, output and weight data dimension, precision, and node fusion and splitting relationships of each layer, emulation-oriented IR has defined an address attribute to determine the specific location of the corresponding hardware for each layer in the model. The basic expression form of some attributes in emulation-oriented IR is shown in TABLE B1.

Operation and OpType in the table give the operation types supported in emulation-oriented IR and the corresponding detailed operations, respectively. Address gives the description format of hardware location for some common operations. Generally, after the parser and optimization, the address information does not appear in the emulation-oriented IR graph, and its value is NONE. When the compiler backend allocates resources according to hardware constraints, the address information in the emulation-oriented IR will correspond to the actual hardware location. When describing the hardware, we analogize the three-dimensional coordinates of the design space, and define the three-level architecture of the hardware as a three-dimensional hardware space, using x, y, and z to represent the chip-level, Tile-level, and XB-level positions, respectively. During hardware analysis, hardware resources are numbered according to the above three dimensions. Then the three dimensions are combined to obtain the coordinates corresponding to each actual hardware. This position is uniquely determined by the values of x, y, and z. It should be noted that for the CONV or FC layer, after determining the position of XB, the row and column coordinates of the weight mapped in XB need to be given, as shown in the

Algorithm A1 Critical path reforming method

Input: Neural network, Array size ($H \times W$), Dataset, On-chip resource (P);
Output: Re-allocated resources of each layer: X_l;
Calculate initial allocation of each layer: $X_l = X_l^0 = (C_{out} \times C_{in} \times K \times K)/(H \times W)$;
Calculate the size of input feature maps of each layer: IFM^l;
Calculate initial execution time of each layer: $T_l^0 = t_{data} + t_{XB}$;
Calculate initial available resource for critical path: $M = P - \sum_{l=1}^{L} X_l^0$;
while $M > 0$ **do**
  $i = argmax[T_1 X_1^0/X_1, \ldots, T_L X_L^0/X_L]$;
  $X_i = X_i + X_i^0$;
  Calculate available resource for critical path:$M = P - \sum_{l=1}^{L} X_l$;
end while
return $X_l, l \in (1, L)$

Table B1 Emulation-oriented IR of CIM paradigm

Operation	Description	Address	OpType
VMM	Conv2d/MatMul can be splitted into VMM operations in the XBs	• {Chip[x].Tile[y].XB[z].[Sx,Sy,Ex,Ey],. . . }	• Conv2d • MatMul
Add	Add the results from XBs	• {Chip[x].Tile[y].Adder[z],. . . } • virtual	• add
Activation	Element-wise operations to calculate the activation function	• {Chip[x].Tile[y].Activation[z],. . . } • virtual	• Relu
Pooling	Compute the average/max of a subset of input data	• {Chip[x].Tile[y].Pooling[z],. . . } • virtual	• MaxPool • AveragePool
Flatten	Flattens the 3D data from the Tiles into a vector	• {Chip[x].Controller[y],. . . } • virtual	• Flatten
Dispatch	Split the input data of specify layer to different Tiles	• {Chip[x].Controller[y],. . . } • virtual	• Dispatch
Merge	Concatenate the outputs of different Tiles	• {Chip[x].Controller[y],. . . } • virtual	• Merge

Table B1, where [Sx, Sy, Ex, Ey] represent the starting row and column position and the ending row and column position respectively.The rectangular area formed by the coordinates is the position where the weight is mapped to XB. In addition, some operations do not require all the three coordinates information, such as a Tile-level controller, which can perform Flatten operations, but only have two coordinates, which are x and y. It means that the device like Tile-level controller can control all z devices prefixed with x and y, which are similar to the concept of lines in three-dimensional space.

The backend of the compiler can use the address information as the input information of the hardware interface function. As there are different operations in different network models, a certain kind of hardware cannot fully support all operators. For those layers that cannot be executed in existing hardware, we define their address attributes as virtual, indicating that this operation is executed outside of the actual hardware. In the scheduling module, the layer with the virtual tag will be shielded, so that only the layer performed on the actual hardware will own its corresponding hardware instruction codes.

Appendix C Analog computing model

The accuracy drop of CIM chips is mainly induced by the nonidealities in analog computing. Thus, a detailed analog computing model should be developed. As shown in Fig. C1a, basically, the input data is sliced to multi single-bit vectors and then is transformed to voltages by DAC. When the voltages are applied to the array, the output currents are quantized by ADC and the output data is obtained with shift-adder circuits. The analog computing model includes two parts: device and array model (Fig. C1b) and peripheral circuit model (Fig. C1c).

For the device and array in the inference phase, the computing results are mainly influenced by read and write noise and IR-drop, which can be modeled as $I_{out} = V_{in} F_{Row}(G_{ori} + N(G_{ori}))F_{Col}$, where the F_{Row} and F_{Col} are the row and column IR-drop factors, respectively [1], the $N(G_{ori})$ is a noise factor related to original G. For the update phase of device and array, the next conductance state after the open-loop update without verify is modeled as

$$G_{next} = \begin{cases} G_{ori} e^{(-N/a)} + (b + G_{min})(1 - e^{(-N/a)}), \Delta G > 0 \\ G_{ori} e^{(N/a)} + (G_{max} - b)(1 - e^{(N/a)}), \Delta G < 0 \end{cases} \tag{C1}$$

where N is the update pulse number, G_{min} and G_{max} are the minimum and maximum of analog switching window, a and b are the nonlinear factors [2]. In the close-loop update with verify, the next conductance state is modeled as $G_{next} = G_{ori} + \Delta G + N(G_{ori} + \Delta G)$. For peripheral circuits the inference phase, the nonideal DAC and ADC can be

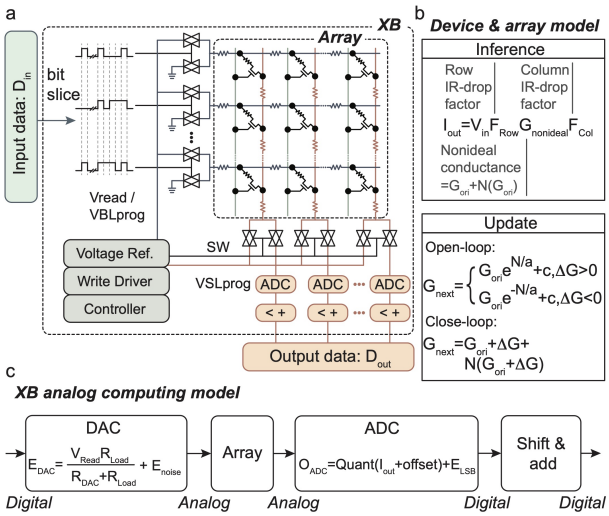

Figure C1 (a) Circuit blocks in XB, (b) device and array model in inference and updata phases and (c) XB analog computing model.

Table D1 The specifications of simulation architecture and referred chips

Parameter	Chip Specification for throughput optimization (Simulation)	4K Array[SCIS 21'] [4]	160K Chip[ISSCC 20'] [3]	16K System[Nature 20'] [5]
Array size	256 × 256	32 × 128	• Array1: 1568 × 100 • Array2: 100 × 10	128 × 16
Data bandwidth	1 Gb/s	-	-	-
ADC bits	8	-	• Array1: 2 • Array2: 8	8
DAC bits	1	-	• Array1: 1 • Array2: 1	1
ADC number per array	8	-	• Array1: 100 • Array2: 10	32
Memristor capacity	128M	4K	160K	16K

modeled as shown in Fig. C1c. The nonidealities of peripheral circuits in update phase can be integrated in device model, which is omitted in this work.

Appendix D Experimental Results

Appendix D.1 Experimental setup

We present three case studies: dataflow optimization, analog computing verification and model calibration. In the dataflow optimization case, we used four standard DNN models (VGG11, ResNet18, 34, 50) for IMAGENET to simulated on-chip throughput of the 128M RRAM chip when the array size is 256×256. In the analog computing verification case, we compare classification accuracies between native inference with FP32 and chip-in-loop inference of ResNet34 for CIFAR-10 using increasing gate voltage programing method with 100ns-width pulses. In the model calibration case, we use the 160K fully chip [3] to calibrate the circuit model with two-layer FC NN, and multi-chip system to calibrate the device and array model with five-layer CNN for MNIST classification. The specification of simulated architecture and referred chips are shown in Table D1.

Appendix D.2 Optimization of dataflow

The result of throughput of the ResNet18 is shown in Fig. D1a. It can be found that when the array size is small, the throughput is higher, and the change of row has a smaller impact on the data throughput rate than the change of column. Therefore, we expect to set the array to with height greater than width, which can achieve faster processing speed when facing networks of different sizes. At the same time, after adopting the proposed compilation scheme, the throughput rate can be greatly improved for different networks when the hardware resources are determined. Fig. D1b shows that the throughput of each model has been improved by at least several tens of times, and the ResNet18 has even reached a 211× increase comparing to those without adapting the compilation process.

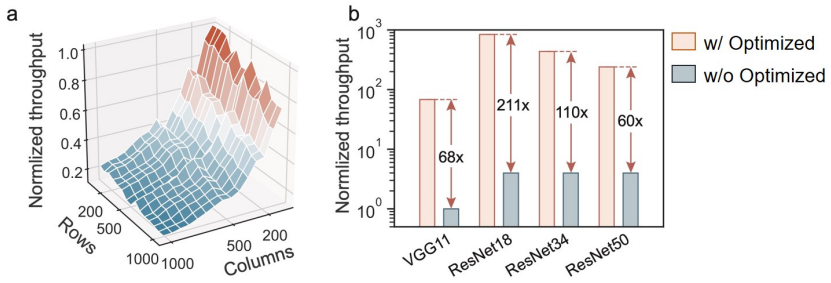

Figure D1 (a) The impacts of array size on the throughput and (b) throughput optimization with compiler

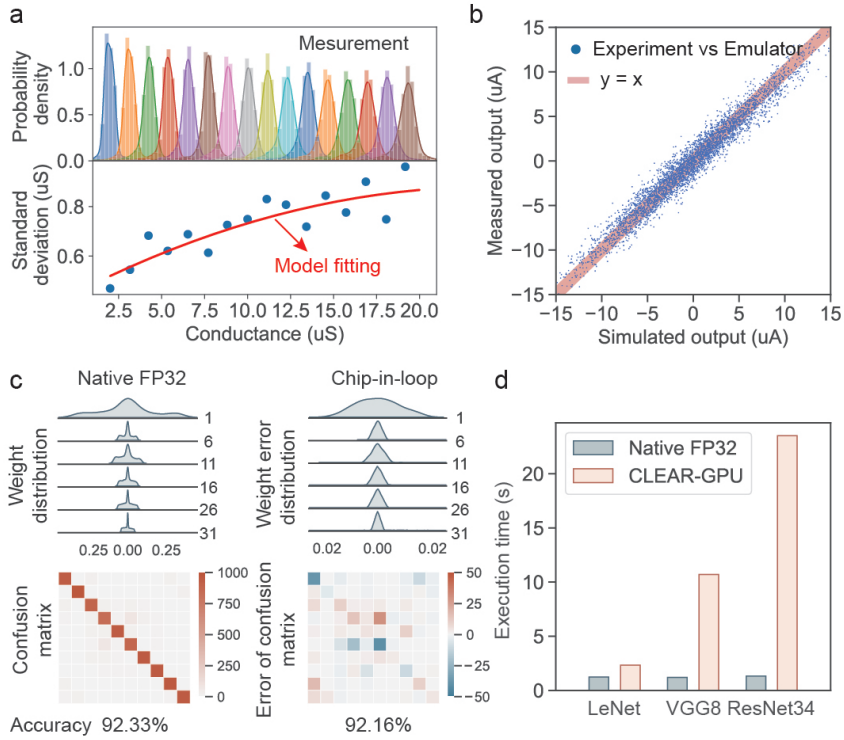

Figure D2 (a) Measured characteristics of RRAM device, (b) verification of array computing and (c) analog computing based DNN, (d) execution time of CLEAR.

Appendix D.3 Verification of analog computing based DNN

Many model-based simulation works have shown that the read and write noise of device will reduce the accuracy of RRAM based analog computing, hence, it is needed to verify the actual accuracy drop of DNN in the actual RRAM chip. We take two experiments: verification of the device model in simulation and verification of the function of analog computing based DNN. In the array computing experiment, the device model with Gaussian distribution in each conductance level is generated by the measured data in Fig. D2a. We can see that the simulated outputs of array computing are linear with measured outputs but still has deviated noise (Fig. D2b). In DNN inference, the accuracy of chip-in-loop inference with chip-aware training is slightly lower than native computing with FP32 (Fig. D2c). Thus, the read and write noise will slightly decrease the accuracy of analog computing based DNN. Further, the simulation time of CLEAR will increases as the model becomes larger (Fig. D2d).

Appendix D.4 Calibration of models in simulation

We calibrate the circuit model and device and array model with CLEAR. The simulated results are compared with the measured results of 160K fully chip in CLEAR based on the area, latency and power metrics (Fig. D3a). The simulated area is larger than the measured one due to the optimized chip layout. The simulated latency and power are slightly smaller the measured due to the simulated circuit model eliminates the power and latency of pad ring. In Fig. D3b, the simulated training process of hybrid training method is similar to the experimental training process with about 0.36% error in the test.

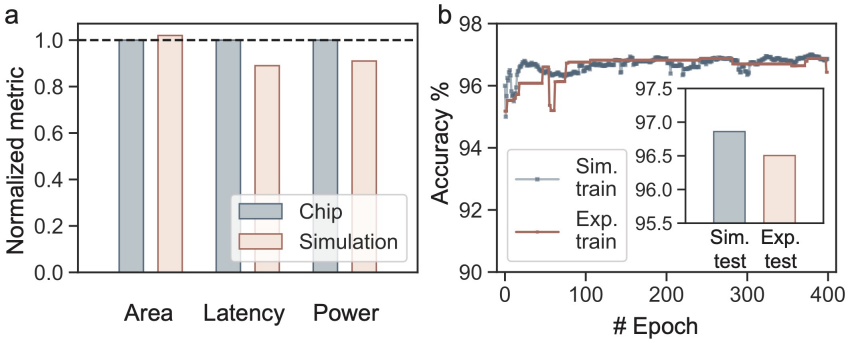

Figure D3 (a) Calibration of circuit model and (b) device and array model.

Table E1 Comparison with other works in the literature

Works	Compilation Optimization	Hardware System Support	Inference Model	Training	On-chip verification
This work	✓	✓	Circuit Model (High Simulation Precision)	✓	✓
PUMAsim [6]	✗	✗	Behaviour Model (Low Simulation Precision)	✗	✗
MNSIM [7] / MNSIM2.0 [8]	✗	✗	Behaviour Model (Low Simulation Precision)	✗	✗
NeuroSim [9] / NeuroSim2.0 [10]	✗	✗	Circuit Model (High Simulation Precision)	✓	✓

Appendix E Comparison with other works

We list a table (Table E1) to illustrate the differences between this work and former works about simulation platform of CIM chips. We mainly compare five representative aspects about hardware simulation tool, including compilation optimization, hardware system support, inference model, training, and on-chip verification. The compilation optimization refers to the model deployment optimization. When given a specific hardware architecture, the simulator can find an optimized placement and simply some connections of different layer if it supports the compilation optimization function.The hardware system support refers to whether the deployment flow of AI models on simulator and real hardware system can share some common software functional modules. To validate the proposed simulation model, we need to compare the results between the chips and simulator. The traditional way to get the results of hardware and simulator are two separate process due to the different interfaces of hardware and simulator. If we can use unified the workflow for hardware and simulator, the verification process will be much quicker and more convenient. The inference model refers to the which model does the simulator adopt. The circuit level model has higher simulation precision than behavior model because the circuit level model has more comprehensive details when calculating the results. The training refers to whether the tool supports simulation of on-chip training. The on-chip verification refers to whether the tool has been verified with experimental results from real chips. The comparison table shows that our proposed simulator has more abundant functions, which can improve the simulation efficiency. We introduce two functions, the compilation optimization and hardware system support, into our emulator, which are not supported in other works. With the compilation optimization, we can automatically place various AI models on hardware or simulator more reasonably. Meanwhile, with the design of unified deployment flow for simulator and hardware, we can support the hardware system test in the same workflow, which can reduce huge labor consuming to validate the proposed simulation model with fabricated chips.

References

1 Yan Liao, Bin Gao, Peng Yao, Wenqiang Zhang, Jianshi Tang, Huaqiang Wu, and He Qian. Diagonal matrix regression layer: Training neural networks on resistive crossbars with interconnect resistance effect. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(8):1662–1671, 2020.

2 Wei Wu, Huaqiang Wu, Bin Gao, Peng Yao, Xiang Zhang, Xiaochen Peng, Shimeng Yu, and He Qian. A methodology to improve linearity of analog rram for neuromorphic computing. In 2018 IEEE Symposium on VLSI Technology, pages 103–104. IEEE, 2018.

3 Qi Liu, Bin Gao, Peng Yao, Dong Wu, Junren Chen, Yachuan Pang, Wenqiang Zhang, Yan Liao, Cheng-Xin Xue, Wei-Hao Chen, Jianshi Tang, Yu Wang, Meng-Fan Chang, He Qian, and Huaqiang Wu. A fully integrated analog ReRAM based 78.4 TOPS/W compute-in-memory chip with fully parallel MAC computing. In 2020 IEEE international solid-state circuits conference, pages 500–502. IEEE, 2020.

4 Wenqiang Zhang, Bin Gao, Peng Yao, Jianshi Tang, He Qian, and Huaqiang Wu. Array-level boosting method with spatial extended allocation to improve the accuracy of memristor based computing-in-memory chips. Science China Information Sciences, 64(6):1–9, 2021.

5 Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J. Joshua Yang, and He Qian.

Fully hardware-implemented memristor convolutional neural network. Nature, 577(7792):641–646, 2020.

6 Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, et al. Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 715–731, 2019.

7 Lixue Xia, Boxun Li, Tianqi Tang, Peng Gu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, and Huazhong Yang. Mnsim: Simulation platform for memristor-based neuromorphic computing system. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(5):1009–1022, 2017.

8 Zhenhua Zhu, Hanbo Sun, Kaizhong Qiu, Lixue Xia, Gokul Krishnan, Guohao Dai,Dimin Niu, Xiaoming Chen, X Sharon Hu, Yu Cao, et al. Mnsim 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems. InProceedings of the 2020 on Great Lakes Symposium on VLSI, pages 83–88, 2020.

9 Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(12):3067–3080, 2018.

10 Xiaochen Peng, Shanshi Huang, Hongwu Jiang, Anni Lu, and Shimeng Yu. Dnn+neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memoryaccelerators for on-chip training.IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(11):2306–2319, 2020.