XB-SIM*: A Simulation Framework for Modeling and Exploration of ReRAM-Based CNN Acceleration Design

Xiang Fei
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China; Beijing National Research Center for Information Science and Technology, Beijing 100084, China

Youhui Zhang
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China; Beijing National Research Center for Information Science and Technology, Beijing 100084, China

Weimin Zheng
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China; Beijing National Research Center for Information Science and Technology, Beijing 100084, China

Follow this and additional works at: https://tsinghuauniversitypress.researchcommons.org/tsinghua-science-and-technology

Part of the Computer Sciences Commons, and the Electrical and Computer Engineering Commons

Recommended Citation
XB-SIM*: A Simulation Framework for Modeling and Exploration of ReRAM-Based CNN Acceleration Design

Xiang Fei, Youhui Zhang*, and Weimin Zheng

Abstract: Resistive Random Access Memory (ReRAM)-based neural network accelerators have potential to surpass their digital counterparts in computational efficiency and performance. However, design of these accelerators faces a number of challenges including imperfections of the ReRAM device and a large amount of calculations required to accurately simulate the former. We present XB-SIM*, a simulation framework for ReRAM-crossbar-based Convolutional Neural Network (CNN) accelerators. XB-SIM* can be flexibly configured to simulate the accelerator’s structure and clock-driven behaviors at the architecture level. This framework also includes an ReRAM-aware Neural Network (NN) training algorithm and a CNN-oriented mapper to train an NN and map it onto the simulated design efficiently. Behavior of the simulator has been verified by the corresponding circuit simulation of a real chip. Furthermore, a batch processing mode of the massive calculations that are required to mimic the behavior of ReRAM-crossbar circuits is proposed to fully apply the computational concurrency of the mapping strategy. On CPU/GPGPU, this batch processing mode can improve the simulation speed by up to 5.02× or 34.29×. Within this framework, comprehensive architectural exploration and end-to-end evaluation have been achieved, which provide some insights for systemic optimization.

Key words: deep neural network; Resistive Random Access Memory (ReRAM); simulation; accelerator; processing in memory

1 Introduction

Resistive Random Access Memory (ReRAM) is a multi-level memory device with conductance that can be programmed to any value within its lowest and highest bounds (High-Resistance States (HRS) and Low-Resistance States (LRS)) theoretically[1]. In many existing studies[2–6], the ReRAM-crossbar array has been demonstrated as a highly efficient kernel component for accelerators of Neural Network (NN) computation. The uniqueness is that it integrates storage and computation in the same physical location (referred to as memory/computation co-localization) to avoid costly data movements. The ideal computing process is shown in Fig. 1.

Fig. 1 Illustration of ideal crossbar.
Thus, such a crossbar can occupy an NN weight-matrix. For an ideal crossbar, an input vector voltage $V_i$ is applied to the rows (word-lines) and multiplied by the conductance matrix of ReRAM cells $G_{ij}$. Based on Kirchhoff’s law, the resulting currents are summed across each column, which can be calculated by $I = GV$. That is, a vector-matrix-multiplication is completed in situ. This computation can achieve extremely high parallelism and be performed in a single time step.

ReRAM-based NN Accelerators (RNAs) have been studied for years and a variety of different architectures have been proposed\cite{2-7}. However, open tools, which enable designers to perform rapid modeling and end-to-end evaluation of RNA, are still missing.

We propose such an open source framework called XB-SIM\textsuperscript{*}. Compared with existing studies on software tools that allow the modeling of ReRAM devices\cite{8} or the simulation of synaptic devices, crossbars\cite{9}, and even accelerator architectures\cite{10,11}, our work is the only one that supports the complete end-to-end co-design process of RNA, which involves training a real Deep Neural Network (DNN), deploying NN parameters onto hardware efficiently (also called mapping), and simulation.

Then the behavioral simulation of XB-SIM\textsuperscript{*} improves in accuracy. Due to the non-ideality of ReRAM and analog calculations, computation of ReRAM-based crossbar is not accurate\cite{12} (Fig. 1 is the ideal state). Usually, the device variations allow the distribution of device conductivity values to conform to a type of dynamic Gaussian distribution (detailed in Section 2). Thus, when the ReRAM cell that stores an NN weight is accessed during simulation, a random number generation would be needed. Considering a real Convolutional Neural Network (CNN) with millions or more parameters, we find that these operations and massive vector-matrix-multiplications may result in a large computational load. To reduce this load, some studies (e.g., MNSIM\textsuperscript{10}) ignore the ReRAM device variations during simulation, which results in a loss of precision. In contrast, we propose a parallel method to speed up the accurate simulation, which is essential to enable large-scale exploration of design space. Finally, existing behavioral simulation tools have not been verified on a real chip, but we have done so and found a minimal error.

In summary, XB-SIM\textsuperscript{*} is an entire toolchain for designers to quickly construct the RNA architecture in the early stage of design and a real NN can be deployed on the design to obtain run-time information. This toolchain has the following contributions:

(1) A configurable clock-driven RNA simulator, ReRAM-aware NN training algorithm, and a CNN-oriented mapping mechanism. As a reference design, the correctness of the current simulated ReRAM-crossbar, i.e., the kernel analog component, has been verified by comparison with the corresponding circuit simulation of a real chip. Furthermore, the mapper proposes an intuitive resource allocation strategy for load balancing based on the feature of memory/computation co-localization and the structure of the target NN.

(2) Parallel acceleration of simulation. We propose a batch processing mode to support concurrent processing of multiple simulation kernels to make full use of the computing resources of multi-core CPUs and GPGPU. Specifically, this mode fully excavates the processing concurrency brought about by the aforementioned mapping strategy.

(3) Extensive end-to-end evaluations. Firstly, the batch processing can improve the simulation speed remarkably. Secondly, compared with full precision training, the error introduced by our training algorithm is minimal. Thirdly, the framework can complete the scalability test of RNA and evaluate the impact of various design factors on the performance.

2 Background and Related Work

2.1 Background

2.1.1 ReRAM basics

This paper focuses on a subset of ReRAM, called metal-oxide ReRAM\cite{13}, which uses metal oxide layers as switching material sandwiched between electrodes. ReRAM consists of two types: device-to-device and cycle-to-cycle\cite{12,14}. The first is reflected as parametric variation across multiple cells: If we program multiple ReRAM cells to a given conductance value, the actual programmed conductance of different cells will follow some distribution models\cite{12}. The second is a variation of one cell.

Specifically, during programming, the conductance change of a cell depends on the polarity, magnitude, and duration of the voltage input\cite{15}. Quite a few non-ideal situations, such as abrupt switching and fluctuation during repeated cycles, will happen. Moreover, multiple cells show different behaviors. Therefore, a write-verify programming circuit is needed. Accordingly, due to the nature of device imperfection and the programming
overhead (for example, the number of reference voltages used for verification is always limited), the range of actual available conductance values is discrete. This is one of the reasons that the weight value that ReRAM can represent in existing RNA designs\cite{16-19} is considered as a discrete value (e.g., PRIME\cite{16} assumes that one cell can support 16 levels and ISAAC\cite{17}, Reno\cite{18}, and PipeLayer\cite{19} can support 4, 16, and 16 levels, respectively). Another reason is that weight-quantization is beneficial to enhance the anti-noise ability of NN. Furthermore, ReRAM cells are devices with non-linear voltage-current ($V-I$) characteristics.

Besides device features, a real crossbar structure includes quite a few other non-ideal issues, such as circuit parasitics, noise, and IR-drop\cite{6}. Thus, the actual available conductivity values are discrete, and each can be considered to be distributed around the expected value in the form of a Gaussian distribution (Fig. 2).

Another problem of ReRAM-based accelerators is the Analog-to-Digital Converter/Digital-to-Analog Converter (ADC/DAC) consumption. As the main body of the accelerator system still consists of digital circuits, interaction with the digital part requires the use of DACs/ADCs, which may consume most of the hardware resources. Accordingly, how to set the DAC/ADC sensing resolution requires comprehensive consideration of network accuracy, hardware overhead, and other factors.

### 2.1.2 ReRAM-aware training algorithm

There are two phases in NN computation: forward and backward. The main difference between the ReRAM-aware training algorithm and the conventional one is the forward pass. The principle of customization aims to modify the forward pass to approximate the actual hardware inference procedure as much as possible. In order to make the output the same as actual ReRAM cells, the accessed weight value should be rounded to its nearest discrete value, that is, the nearest expected value (this method is also known as quantization which is often used in NN training\cite{21} and the point uses discrete value to present weight instead of continuous floating point). In addition, a random noise will be added to the discrete value according to the conductance distribution.

Some recent work\cite{22} introduced more RNA-specific features into the forward pass including the hardware I/O limitation and the effect of the serial input of DACs. Our proposal considers all the aforementioned aspects.

#### 2.2 Related work

Various RNA architectures have been proposed, such as PRIME\cite{16}, ISAAC\cite{17}, Reno\cite{18}, and PipeLayer\cite{19}. For training and mapping, Liu et al.\cite{23} proposed an IR-drop compensation technique to overcome the adverse impacts of wire resistance. Liu et al.\cite{24} presented a training scheme to enhance the robustness by compensating the impact of device variations. Tang et al.\cite{25} used binarized NN training to reduce the requirement for ReRAM representation ability.

From the aspect of simulation, none of the studies has presented an open simulation framework. NVSim\cite{8} only models emerging non-volatile memory storage technologies. PIMSim\cite{26} is a trace-based simulator of Processing-In-Memory (PIM) architecture for traditional memory technology. NeuroSim\cite{9} is a simulation framework for Non-Volatile Memory (NVM)-based array architectures of NN and its architecture only supports a two-layer multilayer perceptron neural network for MNIST handwritten dataset. MNSIM\cite{10} is a simulation platform for the ReRAM-based neuromorphic system. However, it only implements the simulation function. A system-level simulator\cite{11} for ReRAM-based neuromorphic computing chips integrated Network-on-Chip (NoC) and ReRAM and focused on the simulation of Spiking Neural Network (SNN), not DNN. Moreover, this work did not involve any training algorithm or network mapping. In addition, none of the existing work has analyzed the computational burden of large-scale simulation of RNAs. We believe that part of the reason is that none of them supports the end-to-end evaluation for real and large-scale CNNs and others, such as MNSIM\cite{10}, and the device variations are ignored.

Our work is based on a preliminary study\cite{22}. In particular, we design the concrete and optimized

![Fig. 2 Conductance value distributions. Data are derived from Ref. [20]. A total of 32 cells have been set to 16 different conductance values. All cells have been read 20 times for each level and the read-out data have been presented.](image-url)
mapping method and propose a parallel acceleration technology to match the former.

3 Framework

This section presents the workflow of the entire framework in Fig. 3. The initial model is a trained CNN from some existing deep learning development frameworks, which is used as the input for our ReRAM-aware training algorithm. The latter considers RNA-specific features including NN parameters with limited precision, device variation, Analog-to-Digital/Digital-to-Analog (AD/DA) effect, and I/O limitation. Then the training algorithm fine-tunes the original NN and outputs an equivalent model that meets the hardware constraints.

The output is further handled by the mapper that expands the convolution operations into dot-products, and then splits or merges the latter according to the size of a crossbar and assigns them to simulated hardware processing units. During the procedure, some CNN-specific optimizations would be adopted based on the available crossbar resources. In detail, according to the producer-consumer relationship between NN layers, we provide an intuitive representation method to guide the optimal allocation of crossbar resources on the chip, and then we analyze the simulation concurrency of each crossbar.

Finally, a corresponding simulator is generated: Kernel components are networked ReRAM-crossbar-centric Processing Units (PUs) equipped with the control logic and data buffer to implement optimized mapping. NN parameters have been also deployed onto crossbars. According to the aforementioned concurrency analysis, the simulator is automatically parallelized. Thus, driven by input samples, we can achieve a parallel RNA simulation. The run-time statistics include information on inference result, performance, power consumption, and others, which can be used for end-to-end evaluation and optimization.

4 Simulator

We firstly give the configurable, modular, and hierarchical design of the simulator. Then we focus on the PU composed of analog computation parts, which is the biggest difference between the RNA and digital accelerator. The architectural- and circuit-level designs are proposed. The current implementation of the latter is substitutable; i.e., a reference design. Specifically, the behavior-level fast simulation method for the analog component is detailed. The simulation accuracy is verified through comparison with a real chip.

4.1 RNA hierarchy

Without loss of generality, on the top level, a CNN simulator can be regarded as composed of several layers, each corresponding to the convolutional (CONV) or Fully-Connected (FC) counterpart of the simulated object. Each layer further consists of several PUs and is connected serially to each other with BUFFER modules. The number of crossbars in one PU is configurable.

Besides, all parts communicate with each other through an NoC at the implementation level. The latter is a mature technology. At present, we adopt an in-house clock-driven NoC simulator that is also substitutable.

4.2 Processing unit

After referencing typical RNA designs\cite{16,17,19}, we present a common PU design (Fig. 4) that includes the following key parameters:

1. CN crossbars with size of $2^r$ (rows) and $2^c$ (columns). Each cell has $2^b$ levels.

2. DACs transform the digital input signals of each crossbar row into analog. Sensing resolution is represented by $d$. To decrease this consumption, many designs (such as ISAAC\cite{17}, PRIME\cite{16}, and pipelayer\cite{19}) convert the input signal of S-bit width (I/O precision) into a series of low-precision signals ($d$-bit width) and crossbars have to perform several ($S/d$) sequential operations, which is a time-for-space optimization strategy. The number of DACs is $2^r \times CN$.

3. ADCs convert the analog output of crossbar columns to digital. To decrease consumption, a time-division multiplexing method is used. The output of each

![Fig. 3 Workflow of the framework.](image1)

![Fig. 4 Crossbars and peripheral circuits.](image2)
crossbar column is fed into some shared ADC(s), and the latter converts output signals one by one. The number of ADCs for each crossbar is \( A \).

(4) Sample and Hold (S&H) circuit is an analog device that samples the output of each column and holds its value for time-division multiplexing.

After the conversion of ADCs, some digital components are simulated for functional integrity including:

1. Shift-adder of each column. This component is compatible with the aforementioned sequential operations of DACs, which respectively shift and add the partial sums of the input signals of \( d \)-bit width.

2. Activation function of each column and pooling function.

3. Adder tree. When a large weight matrix is split into multiple small matrices to accommodate the crossbar dimension, adding up the output of the corresponding columns of each crossbar is necessary. Section 6.1 will provide a concrete example.

4. Data buffers. A buffer is placed between the activation and pooling modules. Its size is \( x \times q \times C \), where \( x \) is the width of the image size, \( q \) is the stride of pooling operation \( (q = 1 \text{ if no pooling operation exists}) \), and \( C \) is the channel size. When data are fed into this buffer, the simulator determines whether the data entered are sufficient for a pooling operation; the threshold is \( (x \times (q-1)+q) \times C \). If so, the pooling function is triggered and the result is sent to the buffer for the next layer. Then every \( q \times C \) data fed into the pooling buffer triggers the pooling function.

Another type of buffer is between layers, which provides data for the next CONV (FC) layer. The minimum size of the CONV buffer is \( K \times K \times C \) (i.e., the CONV scale) and the optimal size is \( x \times K \times C \). The threshold for a CONV operation is \( (x \times (K-1)+K) \times C \). Then every \( K \times C \) data fed into the CONV buffer triggers a CONV operation.

4.3 Crossbar

As a reference design, a typical 2T2R-crossbar design is adopted in this study (Fig. 5a). One “logic” cell for a weight value is composed of two ReRAMs and two transistors: Each stands for a positive or negative value, respectively, and WL_P/WL_N controls whether the corresponding transistor is conduction. BL_P/BL_N is the input, their voltage difference stands for the input value, and SL is the output line. Thus, two cells can represent a value of \( 2^{b+1} \) levels (suppose a cell can represent \( 2^b \) levels and some state-of-the-art work\[14\] can achieve \( 2^b=128 \)).

4.4 Simulation method and verification

XB-SIM* is a behavioral simulator. Thus, the information on the running performance and hardware overhead of its main components can be provided by developers as configuration parameters or through third-party assessment tools. Our implementation also provides such parameters of some key components (in Section 7) as a reference design. Furthermore, its main components are divided into two types: analog and digital. The latter’s processing procedure is accurate and can be simulated by software language (such as C++) conveniently and efficiently. Thus, the focus is how to quickly and accurately simulate the analog part, that is, crossbar(s).

Basically, the simulation method is a balance between precision and speed. The circuit-level simulation has to solve a large number of non-linear Kirchhoff equations. For a \( 2^r \times 2^c \) crossbar, more than \( [2^r \times 2^c + 2^r \times (2^c-1)] \) voltage variables (voltages of the input and output nodes of each cell) and \( 3 \times 2^r \times 2^c \) currents need to be solved. Moreover, because ReRAM cells are devices with non-linear \( V-I \) characteristics, the equations are not linear. Thus, the simulation speed is very slow.

Therefore, similar to some existing studies (e.g., MNSIM\[10\]), we use behavior model computation to discard some unimportant factors. Firstly, the influence of non-linear \( V-I \) characteristics is ignored, as the input signal of the crossbar is converted into a series of low-precision signals (usually it is 1-bit/2-bit binary voltage), in which the effect of non-linearity is very limited. Secondly, the inductances and capacitances of
interconnect lines have little influence on ReRAM-based computing\textsuperscript{[27]}; thus, we also ignore them.

We simulate the crossbar behavior through the corresponding vector-matrix-multiplication directly; in other words, the dot-product of the input voltage vector and the conductance matrix. It is necessary to note that when a conductance value is accessed, the specific ReRAM variation should be involved (by contrast, MNSIM\textsuperscript{[10]} ignored the variation). From the computational perspective, a random number is generated according to the Gaussian distribution and added to the conductance value.

Our compromise method works effectively. For verification, we compare the simulation results of the crossbar with the corresponding result of circuit-level simulation of a real chip, which is an MLP-enabled chip taped out under the 130 nm process. The size of a crossbar is $1152 \times 128$ with the 2T2R structure. Our method only introduces 2.68\% computation error. The comparison details are as follows.

We conduct the circuit simulation of the crossbar with CADENCE Spectre. The size of the crossbar is $1152 \times 128$, which leads to a $576 \times 128$ 2T2R array (where two cells represent one weight value, with resistance $R_\text{pos}$ and $R_\text{neg}$), and the HRS (LRS) is 800 k\textOmega (50 k\textOmega)\textsuperscript{[20]}. All of the above are consistent with real chip/devices.

For the verification case, each cell is set to represent a random weight value. To model the wire resistance, we use 87 m\textOmega for bit-line, 100 m\textOmega for source-line, and 1.16 \textOmega for word-line, which are derived from the post-layout parameters (the processing technology is CMOS 130 nm). We use four input voltages, namely, 0.15, 0.1, 0.05, and 0 V, which means that one 2-bit input value is fed ($d = 2$) each time. ReRAM device model is from Ref. [20].

Moreover, as mentioned in Section 2.1, to decrease the ADC consumption, a time-division multiplexing method is used, that is, each time only a part of the crossbar columns (source-lines) are working. In this verification, under different configurations, 4, 8, 16, 32, 64, and 128 columns of equal intervals are enabled each time (in our actual reference design, the four-column model is used, that is, $A = 4$, which is the same configuration as ISAAC\textsuperscript{[17]}).

Then we compare the output currents of behavior model computation and counterparts of circuit simulation. After many tests (random input and weight values), the maximum error of result is no more than 2.68\%, which happens when all the columns are enabled at the same time. For the case of four columns each time, the error is much less at approximately 0.03\%.

Here, we only present the computation paradigm of a single crossbar and the proposed parallel simulation mode is detailed in Section 6.

5 Training

Our training algorithm has considered the four RNA-specific factors (in Section 2.1.2), including NN parameters with limited precision, device variation, AD/DA effect, and I/O limitation. The first two are intuitive, so we mainly introduce the algorithm enhancements of the latter two.

5.1 I/O limitation

Due to hardware resource limitation, the bit-width of signals between layers is usually fixed. We suppose that a weight value is $2^k$-level (weight precision) and there are $2^e$ rows of inputs. The full precision of the crossbar output is $(S + b + c)$ bit (computation precision): $S$ is the I/O precision. Thus, there are $2^e$ additions and the operand of each addition is a product of $(S + b)$-bit width. The crossbar output has to be rounded to $S$ bit as it would be transmitted to another crossbar array(s) as input. This operation should be reflected by training algorithm, too.

Specifically, there are several rounding methods: The simplest one is direct linear-scaling, which converts the full range of the original values evenly into the range of the target values. However, the distribution of source data may be uneven, which tends to be concentrated within a certain value range; thus, we should extract $S$ consecutive bit from the original data, and the starting and ending ranges are configurable\textsuperscript{[28]}. Our algorithm supports this method.

5.2 Serial input from DACs

The serial input (usually 1 or 2 bit per time-step) and the corresponding shift-add operations have actually changed the dot-product computational paradigm. Each bit of the 8-bit input values is entered into the corresponding crossbar row serially and dot-product operations are performed. The partial sums of each column are stored in S\&H and processed by shift-adders with the following results. This process is also reflected in the enhanced training algorithm.
6 Mapper and Parallel Simulation

6.1 Conversion from weights to conductance

The method of converting weight values of the original NN into resistance values is given as follows.

As illustrated in Fig. 5b, $R_1$ and $R_2$ represent the positive and negative resistances in one cell, respectively. They together can represent any weight value of the NN weight matrix. During the computation procedure of a crossbar, the two MOSFETs, M1 and M2, work in the linear area; in other words, they are switched on. Let $V_{BL\text{pos}} - V_{SL} = V_{SL} - V_{BL\text{neg}} = V_{read}$, then we have $I_{cell} = I_{\text{pos}} - I_{\text{neg}} = V_{\text{read}} \times (1/R_1 - 1/R_2)$ according to the Kirchhoff’s law. Thus, the equivalent weight of the cell is $(1/R_1 - 1/R_2)$. We can adjust $R_1$ and $R_2$ to obtain any signed weight value.

After conversion, we should deploy cells onto fixed-size crossbars. Taking a CONV layer as an example, we unfold each of the CONV kernel matrices (Fig. 6a) to form a vector and map on the crossbar(s) from the bottom left to the top right (Fig. 6b). The input feature map is also expanded to the corresponding vectors (Fig. 6d). If the size of one crossbar is too small to accommodate all of the CONV kernels, then we distribute the kernels onto multiple crossbars (Fig. 6c). For example, after being unfolded, the 13th CONV layer of VGG16 has $512 \times 512 \times 3 \times 3$ parameters and each of them is 8 bit (it is widely believed that 8 bit is sufficient for inference with weight matrix). During the computation procedure of a CONV layer, the two 2T2R structures (4 cells) to represent an 8-bit value, $1=R_1=R_2$, and $V_{SL}=512$. The size of the crossbar is $1152 \times 128$ ($2^3=1152$ and $2^3=128$) with the 2T2R structure. Each cell has $2^b=4$ levels. Thus, we need two 2T2R structures (4 cells) to represent an 8-bit value, i.e., a total of 64 crossbars consisting of 16 rows and 4 columns. Furthermore, we generate the corresponding simulator configuration: The crossbars in one column are all in the same PU (Fig. 6c), which can simplify the design that the data in the layer are not transmitted through NoC. Instead, an adder tree is needed to add up the corresponding outputs of these crossbars.

The fixed-point adders are high-speed components while the working frequency of PU is limited (for example, it is only 10 MHz in ISAAC). Therefore, the adders can be multiplexed by time, making the overhead very small. FC layers can be mapped similarly. Through this approach, the minimum number of crossbars required for a CNN can be calculated on the premise of a given crossbar size.

6.2 Resource allocation

The memory/computation co-localization of RNA requires the chip to provide at least the amount of storage commensurate with the size of NN parameters. For CNN, there exists a severe imbalance between the storage requirement and computation amount. As we know, CONV layers reuse weight parameters and the reuse times of different layers vary greatly, while the other extreme is the FC layer, which has a ratio of storage to calculation amount that is always 1.

Thus, if the hardware is only able to hold all the parameters (if one RNA cannot hold all the data, then multiple chips have to be used and our work currently focuses on a single chip, but its mapping principle can also be applied to multi-chip scenarios), then the layer with the highest reuse degree would become the performance bottleneck, because its allocated resources have to repeatedly process input data with shared weights in time-division multiplexing.

Accordingly, the mapper also focuses on the scalability of RNA when additional resources are available. We propose an intuitive and quantitative method for the balanced distribution of on-chip resources. A production-consumption relationship exists between two adjacent layers from the perspective of data flow. Thus, we can define a production-consumption ratio for every layer; $T:1$ means that on average, every $T$ calculation in the previous layer triggers one calculation in the current layer.

1. The ratio of the first CONV layer (Layer 1) is always 1:1, as we assume that the input data are always ready to read.
2. For two adjacent CONV layers, CONV$_i$ and CONV$_{i+1}$, let $q$ be the stride of the max-pooling layer between them ($q=1$ if no max-pooling exists), $p$ is the stride of CONV$_{i+1}$ and the ratio of CONV$_{i+1}$ is $(p \times q) : 1$.
3. For the two adjacent layers, CONV$_i$ and FC$_{i+1}$,
let \( t \) be the output size of CONV\(_i\). Then the ratio of FC\(_{i+1}\) is \( t:1 \), regardless of whether max-pooling occurs between them or not. The reason is that the FC layer can consume all input data and produce all output data at once, while the CONV layer has to produce output data one by one.

(4) For the two adjacent FC layers, FC\(_i\) and FC\(_{i+1}\), the ratio of the latter is always 1:1.

Based on the preceding analyses, a ratio tree can be designed to intuitively represent the production-consumption relationships of a given NN as well as the corresponding algorithm that provides the balanced allocation strategy of crossbar resources.

Without loss of generality, Fig. 7a shows the topology of an imaginary NN that contains two branches (modern neural networks may contain branches and branch fusion, such as the ResNet family). Each circle represents one layer and the corresponding ratio is shown in it. Layer 6 has two ratios because it has two previous layers.

Figures 7b and 7c show the ratio trees of the two branches. One column represents a layer (FC or CONV), and each node of a column represents the smallest crossbar set that can store all the weights of this layer. This condition also implies that all nodes at the same layer have the same number of crossbars. Specifically, some nodes in the two trees are duplicated. The node set of Layers 1 and 2 in the black dashed box of Fig. 7b is only the corresponding set in the black box of Fig. 7c, as well as Layer 6. Furthermore, the nodes in red boxes belong to different branches and the nodes in the green dashed box of Fig. 7c do not appear in Fig. 7b, because Layer 5 in Fig. 7b has a smaller ratio and it does not need so many nodes.

Besides, the indegree for each node is equal to the production-consumption ratio of the layer. The yellow nodes of each layer represent the crossbars that have been assigned to this layer, specifically, Number of Replicas of the Weight (NRW; obviously, at least one yellow node must exist for each layer). Thus, for a node, the prerequisite for continuous calculation is that all of its precursor nodes (direct or indirect) have been assigned PU resources (i.e., colored yellow). Accordingly, if none of the nodes in a layer satisfies this condition, then this layer is the bottleneck. Furthermore, we provide the pseudo code to obtain the balanced allocation strategy, and the average number of crossbars that are capable of running in each cycle (defined as batch number).

Algorithm 1 of resource allocation is intuitive. All nodes on the shortest path from a leaf node (input node) to the root node (output node) have the same priority, then all the nodes on the next shortest path from another leaf (except those that have been assigned) have the same priority, and so on. For a given layer, the maximum number of its yellow nodes in all the ratio trees is its NRW, e.g., the NRW of Layer 1 is 10 in Fig. 7.

The calculation algorithm of batch number (Lines 10–13) is also evident: For Layer \( j \), this value is only the quotient of the NRW of its previous layer and its production-consumption ratio. If more than one previous layer exists, then the minimum value is taken.

### 6.3 Parallel simulation

We can also calculate the number of crossbars that could be simulated at the same cycle; this is called the batch number according to the data dependency relationship of the ratio tree. For example, VGG for CIFAR-10 (a medium-sized CNN including 15 CONV layers) needs 39 crossbars for CONV layers and 6 for FC layers at least, while the batch number is 10.6 on average. VGG16

---

**Algorithm 1 Balanced allocation strategy & batch number**

1. **Input**: \( N = \text{Total} \# \text{of crossbars}, ncb[i] = \text{min} \# \text{of crossbars required by layer} i, r[i]=\text{ratio of layer} i, L = \text{\# of layers};
2. **Output**: NRW & batch number;
3. \( Y = \emptyset, i = 1; \)
4. **while** True **do**
5. Find the shortest path \( S_k \) from leaf node \( i \) to the root node in any ratio tree \( k \);
6. If \# of crossbars of all nodes in \( Y \cup S_k \) is \( N \) then
7. \( Y = Y \cup S_k, i = +; \)
8. If NRW[\( i \)] is \# of nodes of layer \( i \) in \( Y \);
9. \( n[i] \) = NRW[\( i \)], \( n[f] = +\infty (j > 1); \)
10. **for** \( j = 2; j \leq L; j = + + \) **do**
11. \( n[f] = \text{min}(n[f], \text{float}(n[\text{previous layer of} f])/r[f]); \)
12. \( \text{batch number} = \sum_{i=1}^{L} n[i] \times ccb[i]; \)
needs at least 401 crossbars for CONV layers and 3416 crossbars for FC layers, and the batch number is 10.2 on average. The batch number increases when a large amount of resources are available.

The first column of Table 1 shows the batch number of VGG for CIFAR-10 under different numbers of available crossbars (based on the area of the RNA chip in ISAAC[17], the number of crossbars under the same area is estimated and the upper limit is approximately 3000). Parameters of individual components of our PU design are presented in Section 7.1.

To fully use the parallel resources of the state-of-the-art CPUs or GPGPUs, we propose the batch processing mode, which is reflected in the following aspects:

1) We gather $S/d$ sequential input vectors into an input matrix that converts the crossbar computation as matrix-matrix-multiplication. The sizes of the two matrices are $(S/d) \times 2^r$ and $2^r \times 2^c$, respectively. $S$ is the width of input signal, $d$ is the width of low-precision signals (sensing resolution), and the crossbar scale is $2^r \times 2^c$, as defined in Section 4.2. For the crossbar and peripheral circuits, we separate functional simulation from timing simulation, which is commonly used in behavioral simulation. Specifically, the timing and other run-time information including power consumption are counted by cycle (i.e., serially processed), whereas the computation is only performed in the batch processing mode. It’s equivalent in effect.

2) All the crossbars that can run at the same time are simulated synchronously in different threads. For the multi-core/multi-socket CPU(s), we use OpenMP for parallel execution, whereas for GPGPU, we use CUDA.

In summary, we increase the computation ratio to data amount and reduce the API calls in the batch processing mode.

7 Implementation and Evaluation

7.1 Implementation

We complete the XB-SIM* and all the codes are open sources (https://github.com/xb-sim-star/xb-sim-star) in GitHub. The training algorithm is provided as some customized operations and script files in PyTorch, and the simulation is based on SystemC. The latter can be generated according to the user-provided description file of the target CNN, mapping strategy, and simulated hardware constraints.

For parallel computing, the general workflow is as follows. In each cycle, the simulator identifies all the crossbars that can be handled synchronously, collects their data, and calls the corresponding API to compute them in the batch processing mode, as described in Fig. 8. On GPGPU, additional steps are required to transfer the raw and result data between the host memory and GPGPU memory. APIs of intel Math Kernel Library (MKL) based on OpenMP and cuBLAS/cuRAND on CUDA are employed.

As a reference design, the hardware process is 130 nm, which is consistent with the chip for verification. As the chip has not been officially released, it is inconvenient for us to directly disclose its data. Instead, we use the following methods to obtain the information of area and power consumption of various hardware components (listed in Table 2). CACTI[29] is used to model energy and area for buffers. The ADC area and power are obtained from the ADC survey[30] and others are scaled based on ISAAC.

ReRAM device model is from Ref. [20]. By default, the I/O (weight) precision is 8 bit (256-level), i.e., $S = 8$ and $b = 8$. The input of DAC is a series of 1-bit signals ($d = 1$), 32 crossbar-columns share one ADC ($A = 4$).

![Fig. 8 Computational process (the dashed boxes are omitted in CPU).](image)

Table 1 Batch number, execution time per sample, and speedup of each case.

<table>
<thead>
<tr>
<th>Total number of crossbars/batch number</th>
<th>Execution time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 CPU core+noise (baseline)</td>
<td>8 CPU core+noise (Case 1)</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------</td>
<td>---------</td>
</tr>
<tr>
<td>45 / 10.6</td>
<td>1460.0</td>
<td>486.0</td>
</tr>
<tr>
<td>600 / 577</td>
<td>891.5</td>
<td>224.9</td>
</tr>
<tr>
<td>1200 / 1197</td>
<td>840.1</td>
<td>180.3</td>
</tr>
<tr>
<td>1800 / 1784</td>
<td>826.7</td>
<td>184.9</td>
</tr>
<tr>
<td>2400 / 2394</td>
<td>819.3</td>
<td>163.1</td>
</tr>
<tr>
<td>3000 / 2981</td>
<td>819.9</td>
<td>163.4</td>
</tr>
</tbody>
</table>
Table 2  Hardware parameters (130 nm CMOS).

<table>
<thead>
<tr>
<th>Hardware component</th>
<th>Power (MW)</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC (8 bit)</td>
<td>26</td>
<td>0.55</td>
</tr>
<tr>
<td>DAC (1 bit)</td>
<td>0.062 56</td>
<td>2.656 256×10⁻⁶</td>
</tr>
<tr>
<td>S&amp;H</td>
<td>1.562 5008×10⁻⁴</td>
<td>6.25×10⁻⁷</td>
</tr>
<tr>
<td>X-bar</td>
<td>43.2</td>
<td>0.021 312</td>
</tr>
<tr>
<td>Shift-add</td>
<td>0.8</td>
<td>0.00096</td>
</tr>
<tr>
<td>Input-buf (2 KB)</td>
<td>19.84</td>
<td>0.0336</td>
</tr>
<tr>
<td>Output-buf (256 byte)</td>
<td>3.68</td>
<td>0.012 32</td>
</tr>
<tr>
<td>ReLU</td>
<td>0.0512</td>
<td>0.0001424</td>
</tr>
<tr>
<td>Max pooling</td>
<td>6.4</td>
<td>0.00384</td>
</tr>
</tbody>
</table>

...and the crossbar size is 1152 × 128. The processing latency of a crossbar with the ADC/DAC peripheral circuits is 100 ns. At present, we simulate a 2D-mesh NoC to connect all units, which utilizes the X-Y routing and “store and forward” switching without the support of virtual channel and multicast.

7.2 Simulation speed

For VGG for CIFAR-10 with different available resources (as shown in the first column of Table 1), we construct the corresponding simulators with the optimized mapping strategy and test the acceleration performance of the batch mode. Information on the simulation host are presented in Table 3.

For each configuration, i.e., a different number of available crossbars, the following test cases are constructed. The baseline simulates the entire design in the non-parallel mode on one CPU core. The others are executed in the batch mode on the multi-core CPU or GPGPU and their batch numbers are as shown in the first column of Table 1. For each case, we also evaluate whether the introduction of device variation influences the performance or not. The input data contain 100 pictures with size of 32 × 32 with three channels.

In Table 1, we can observe that the speedup of the batch mode with random number generation on the multi-core CPU server is from 3.00× to 5.02×, whereas on GPGPU, the speedup is from 6.18× to 34.29×. Without random generation, the speedup on CPU is from 2.99× to 5.48× and that on GPGPU is from 5.43× to 11.91×. The greater the calculation size, the greater the advantage of the batch mode. In particular, for GPGPU, the greater the computation, the easier the performance of vast parallel computing power.

In addition, batch number determined by the mapping strategy is also directly related to computational parallelism. Thus, an optimized mapping strategy not only makes full use of the RNA crossbars but also benefits simulation speed.

7.3 Evaluation of training algorithm

The NN bench is VGG16[31] (a large CNN that includes 1.4×10⁸ parameters for ImageNet), VGG for CIFAR-10 (a medium-sized CNN that involves 15 CONV layers), and LeNet[32] (a small CNN) for MNIST.

Different training algorithms are compared. The method aims to train one NN with the same RNA configuration under different algorithms. The difference lies in that various hardware constraints/features are introduced: \( Q + N \) (quantization+noise) stands for the ReRAM-aware training algorithm that only considers the distribution of the actual available device conductance. \( R \) (rounding) is the enhanced method that considers the nonlinearity of the distribution of I/O data for rounding. \( S/A \) (shift/add) further considers the effect of shift/add operations.

All generated CNNs are deployed onto the simulator to determine the inference accuracies in Table 4. For small datasets, the corresponding CNNs have sufficient ability. Thus, the effect of the enhanced training algorithm is not obvious. For the much larger ImageNet and CIFAR-10, the shift/add operation has a significant effect on accuracy, because it alters the usual dot-product

<table>
<thead>
<tr>
<th>NN</th>
<th>Top 1 accuracy (%)</th>
<th>Top 5 accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full precision</td>
<td>( Q+N )</td>
</tr>
<tr>
<td>LeNet</td>
<td>99.26</td>
<td>99.30</td>
</tr>
<tr>
<td>VGG for CIFAR-10</td>
<td>85.17</td>
<td>79.40</td>
</tr>
<tr>
<td>VGG16</td>
<td>69.10</td>
<td>46.08</td>
</tr>
<tr>
<td></td>
<td>Full precision</td>
<td>( Q+N )</td>
</tr>
<tr>
<td></td>
<td>88.90</td>
<td>71.34</td>
</tr>
</tbody>
</table>
calculation paradigm. In addition, as the 8-bit I/O width is sufficient for general inference applications, the effect of the rounding method is minimal.

7.4 Architecture exploration

7.4.1 Tradeoff

We evaluate the relationship between hardware consumption, NN accuracy, and some system configurations. Without loss of generality, the NN tested is VGG for CIFAR-10, the baseline configuration is $128 \times 128$ crossbar, I/O precision (i.e., ADC resolution) is 5 bit, and the others are the same as the default values. All results are presented in Fig. 9.

From Fig. 9a, we can see that the crossbar scale does not affect NN accuracy but affects the hardware consumption (power and area). With the increase of the crossbar scale, the crossbar number decreases, which leads to smaller chip area and the consumption of peripheral circuit (especially ADCs) also decreases, which results in lower chip power. But when the scale is larger than $128 \times 128$, the utilization of each crossbar decreases and the entire hardware consumption rises.

In Fig. 9b, the accuracy increases with the improvement of I/O precision. The reason is that the accuracy of the actual calculation results is greater than that of the preset I/O (please refer to Section 4.2); thus, increasing the resolution of ADC actually improves the calculation accuracy. The hardware consumption also increases because the resolutions of ADCs have risen (the ADC resolution is consistent with the I/O precision, and the former overhead is proportional to the square of the resolution).

7.4.2 Scalability

Here, we evaluate the relationship between the number of available crossbars and performance of the simulated design. Using the proposed mapping strategy, we assign additional crossbars to the bottleneck to improve the overall throughput. Results show that VGG for CIFAR-10 can achieve a super-linear acceleration ratio with the increase of resources applied to bottleneck layers. We can achieve $5 \times 64 \times 15.33 \times$ speedup by using only $1.42 \times 15.33 \times$ crossbars. The case of initial configuration consumes 45 crossbars, which only meets the NN’s minimum demand for weight storage.

Specifically, this CNN contains three $2 \times 2$ max-pooling layers, each of which located after several successive convolution layers. FC layers are behind all of the aforementioned layers. After the minimum resource requirements for each layer have been satisfied, extra units are assigned to these convolution layers. Note that the pooling is $2 \times 2$, which limits the performance ratio of its front and successive layers to a maximum of 4; thus, the total upper limit of speedup is $64 \times$ because we do not duplicate the FC layer.

8 Conclusion and Future Work

We propose an end-to-end tool chain for RNA design. The behavior of the simulation kernel has been verified by comparison with the corresponding circuit of a real chip. Moreover, a batch processing mode is proposed to speed up the RNA-specific simulation on the CPU/GPGPU hardware substrate. The reference design is now based on the 130 nm process (to match the verification chip) and the wire width is large; thus, the influence of factors, such as IR-drop, is small. As the process progresses, the effects of the unfavorable factors gradually become more obvious, and extra techniques need to be introduced to overcome them in the next step. Furthermore, because the running performance of the PU is low, the common NoC can meet the data communication requirements between PUs. As the process improves, NoC optimization is also gradually considered. We aim to utilize the widely used NoC simulator (Booksim) and corresponding power model (ORION 2.0) to obtain more detailed running data.
Acknowledgment

This work was supported in part by Beijing Academy of Artificial Intelligence (BAAI) (No. BAAI2019ZD0403), Beijing Innovation Center for Future Chip, Tsinghua University, and the Science and Technology Innovation Special Zone Project, China.

References


**Xiang Fei** received the BS degree in computer science from Zhejiang University, Hangzhou in 2016. He is currently a PhD student at Department of Computer Science and Technology, Tsinghua University, Beijing. His research interests include computer architecture and neuromorphic computing.

**Youhui Zhang** received the BS and PhD degrees in computer science from Tsinghua University, Beijing, China in 1998 and 2002, respectively. He is currently a professor at Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include computer architecture and neuromorphic computing. He is a member of CCF, ACM, and IEEE.

**Weimin Zheng** received the master degree in computer science from Tsinghua University, Beijing, China in 1982. Currently, he is a professor at Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include high performance computing, network storage, and parallel compiler. He is an academician of Chinese Academy of Engineering.