TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 0?/?? pp???-???
Volume 18, Number 3, June 2013

XB-Sim*: A Simulation Framework for Modelling and Exploration of ReRAM-based CNN Acceleration Design

Xiang Fei, Youhui Zhang*, and Weimin Zheng

Abstract: ReRAM-based neural network accelerators have the potential to outshine their digital counterparts in terms of computational efficiency and performance remarkably, while their design faces a number of specific challenges, including the imperfections of ReRAM device and the 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. It also includes an ReRAM-aware 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. Further, a batch processing mode of the massive calculations that are required to mimic ReRAM-crossbar circuits' behavior is proposed to fully employ the computational concurrency of the mapping strategy; on CPU/GPGPU, it can improve the simulation speed by up to 5.02× or 34.29× respectively. Within this framework, comprehensive architectural exploration and end-to-end evaluation have been achieved, which provides some insights for systemic optimization.

Key words: deep neural network, ReRAM, simulation, accelerator, processing in memory

1 Introduction

Resistive random access memory (ReRAM) is a multi-level memory device whose conductance can be programmed to any value within its lowest and highest bounds (high- and low-resistance states, abbreviated to HRS/LRS) theoretically [1]. The ReRAM-crossbar array has been proved to be a very efficient kernel component for accelerators of neural network (NN) computation, by many existing studies [2–6]. 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:

The ReRAM cell of each cross point can be programmed to store an element of a weight-matrix; 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 is 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 could achieve very high parallelism and be performed in a single time step.

ReRAM-based NN accelerators (abbreviated to RNA) have been studied for years and a variety of different architectures were proposed [2–7]. However, open tools, which enable designers to perform rapid modeling and end-to-end evaluation of RNA, are still missing.

• Xiang Fei, Youhui Zhang, and Weimin Zheng are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: feix16@mails.tsinghua.edu.cn, zyh02@tsinghua.edu.cn, zwm-dcs@mail.tsinghua.edu.cn

∗ To whom correspondence should be addressed.

Manuscript received: year-month-day; revised: year-month-day; accepted: year-month-day

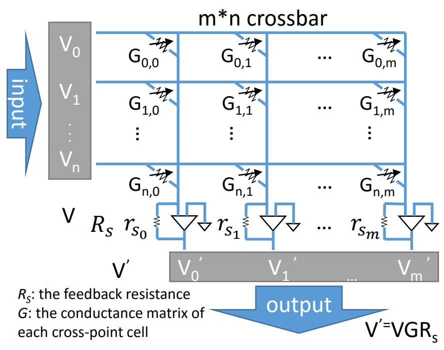


Fig. 1 The ideal crossbar

This paper proposes such an open source framework, XB-SIM*. Compared with existing studies of software tools that allow the modelling of ReRAM devices[8] or the simulation of the synaptic devices, crossbars[9] and even accelerator architectures [10, 11], our work is the only one that supports the complete end-to-end co-design process of RNA, composed of training a real DNN, deploying NN parameters onto hardware efficiently (a.k.a. mapping), and simulation.

Next, the behavioral simulation of XB-SIM* is more accurate. Due to the non-ideality of ReRAM and analog calculations, computation of ReRAM-based crossbar is not accurate [12] (Fig. 1 is the ideal state). Usually, the device variations allow the distribution of device conductivity values to conform to some kind of dynamic Gaussian distribution (it will be detailed in Section 2). Thus, when an ReRAM cell that stores an NN weight is accessed during simulation, a random number generation would be needed. Considering a real CNN with millions or more parameters, these operations and massive vector-matrix-multiplications may be a huge computational load. In order to reduce it, some studies (e.g., MNSIM[10]) ignore the ReRAM device variations during simulation, which results in a loss of precision. In contrast, our design proposes a parallel method to speed up the accurate simulation, which is essential to enable large-scale exploration of design space.

Last but not least, existing behavioral simulation tools have not been verified on a real chip, while we have done it and the error is very tiny.

In summary, XB-SIM* is a whole 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 get runtime information; it contains the following contributions:

(1) *A configurable clock-driven RNA simulator, plus the 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. Further, 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.* This paper proposes a batch processing mode to support multiple simulation kernels' concurrent processing 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 above mapping strategy.

(3) *Extensive end-to-end evaluations.* First, the batch processing can improve the simulation speed remarkably. Second, compared with full precision training, the error introduced by our training algorithm is very small. Third, 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 [13], which uses metal oxide layers as switching material sandwiched between electrodes. ReRAM has two types of variation: device-to-device and cycle-to-cycle [12, 14]. The first is reflected as parametric variation across multiple cells: If we program multiple ReRAM cells to a given conductance value, the really programmed conductance of different cells will follow some distribution model [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 [15]. Quite a few non-ideal situations, e.g., abrupt switching and the fluctuation during repeated cycles, will happen. Moreover, multiple cells show different behaviors. Therefore, some write-verify programming circuit is needed. Accordingly, due to the nature of device imperfection and the programming overhead (e.g., the number of reference voltages used for verification is always limited), the range of actual available conductance

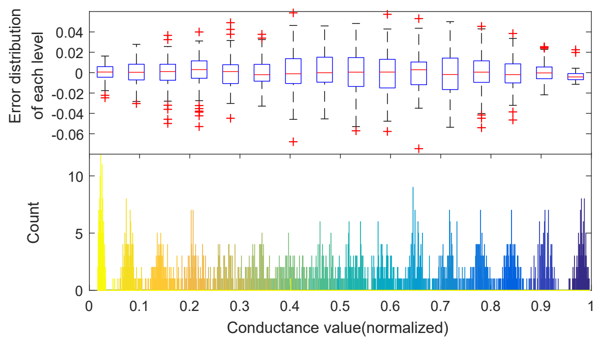


Fig. 2 Conductance value distributions. The data is derived from [20]. 32 cells have been set to 16 different conductance values. All cells have been read 20 times for each level and the read-out data is presented; thus there are 640 data at each level.

values is discrete. This is one of the reasons that the weight value that ReRAM can represent in existing RNA designs [16–19] is considered as discrete value (e.g., PRIME [16] assumes one cell can support 16-level and for ISAAC [17] and Reno [18] and PipeLayer [19], it is 4, 16, and 16 respectively). Another reason is that weight-quantization is beneficial to enhance the anti-noise ability of NN. Further, ReRAM cells are devices with non-linear V-I characteristics.

Besides device features, a real crossbar structure includes quite a few other non-ideal issues, like circuit parasitics, noise and IR-drop[6]. Thus, anyway, the actual available conductivity values are discrete, each of which 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 ADC/DAC consumption. Because the main body of the accelerator system is still digital circuits, to interact with the digital part, usually DACs/ADCs have to be used, 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, etc.

2.1.2 The ReRAM-aware training algorithm

There are two phases in NN computation: forward and backward(BP). The main difference between the ReRAM-aware training algorithm and the conventional one is the forward pass; the principle of customization is 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 [21]. The point is using 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 [22] introduces 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 takes into account all the above aspects.

2.2 Related Work

Some studies have proposed various RNA architectures, like PRIME [16], ISAAC [17], Reno [18], PipeLayer [19], etc. For training and mapping, [23] proposes an IR-drop compensation technique to overcome the adverse impacts of the wire resistance. [24] presents a training scheme to enhance the robustness by compensating the impact of device variations. [25] uses binarized NN training to lower the requirement for ReRAM representation ability.

From the aspect of simulation, none of the studies has presented an open simulation framework. NVSim[8] just models emerging non-volatile memory storage technologies. PIMSim[26] is a trace-based simulator of PIM architecture for traditional memory technology. NeuroSim[9] is a simulation framework for NVM-based array architectures of NN. Its architecture only supports a 2-layer multilayer perceptron neural network for MNIST handwritten dataset. MNSIM[10] is a simulation platform for the ReRAM-based neuromorphic system. But it only implements the simulation function. A system-level simulator[11] for ReRAM-based neuromorphic computing chips integrate network-on-chip (NoC) and ReRAM together and focuses on the simulation of spiking neural network (SNN), not DNN. Further, this work does not involve any training algorithm or network mapping. In addition, none of existing work has analyzed the computational burden of large-scale simulation of RNAs. We think part of the reason is that none of them support the end-to-end evaluation for real and large-scale CNNs and some, e.g., MNSIM[10], ignored the device variations.

This work is based on a preliminary study [22]. In particular, we design the concrete and optimized mapping method and propose a parallel acceleration technology to match the former.

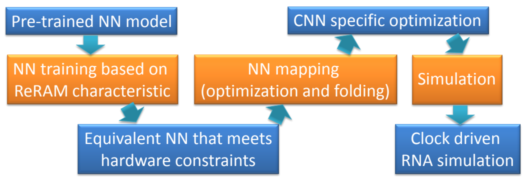

Pre-trained NN model
CNN specific optimization
NN training based on ReRAM characteristic
NN mapping (optimization and folding)
Simulation
Equivalent NN that meets hardware constraints
Clock driven RNA simulation

Fig. 3 Workflow of the framework

3 Framework

This section presents the workflow of the whole framework in Fig. 3. The initial model is a trained CNN from some existing deep learning development framework, used as the input for our ReRAM-aware training algorithm. The latter considers RNA-specific features, including NN parameters with limited precision, device variation, the AD/DA effect and I/O limitation. Then, it 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, then splits or merges the latter according to the size of a crossbar and assigns them onto simulated hardware processing units. During the procedure, some CNN-specific optimization would be adopted based on the available crossbar resources. In detail, according to the producer-consumer relationship between NN layers, we give an intuitive representation method to guide the optimal allocation of crossbar resources on the chip, and analyze the simulation concurrency of each crossbar.

Finally, a corresponding simulator is generated: Kernel components are networked ReRAM-crossbar-centric processing units (abbreviated to PUs), equipped with the control logic and data buffer to implement the optimized mapping; NN parameters have been also deployed onto crossbars. According to the above-mentioned concurrency analysis, the simulator is automatically parallelized. Thus, driven by input samples, we can achieve a parallel RNA simulation; the runtime statistics include the information of inference result, performance, power consumptions, etc., which can be used for end-to-end evaluation and optimization.

4 Simulator

Here we first 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 RNA and digital accelerator: The architectural level and circuit level

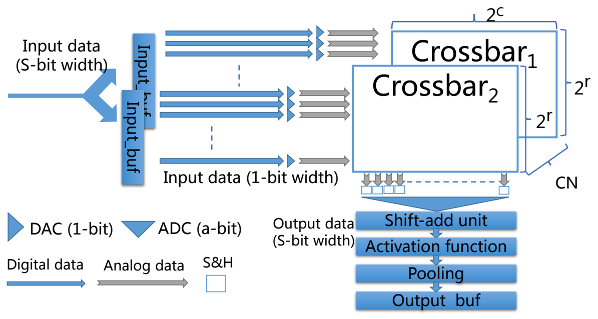


Fig. 4 Crossbars and peripheral circuits

designs are both proposed; the current implementation of the latter is substitutable, a.k.a., a reference design. Specifically, the behavior-level fast simulation method for the analog component is detailed. The simulation accuracy is verified by the comparison with a real chip.

4.1 The RNA Hierarchy

Without loss of generality, on the top level a CNN simulator can be regarded as composed of several layers, each of which just corresponds the convolutional (CONV) or fully-connected (FC) counterpart of the simulated object. Each layer is further composed of several PUs and 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. Currently, we adopt an in-house clock-driven NoC simulator, which is also substitutable.

4.2 Processing Unit

After referencing typical RNA designs [16, 17, 19], we present a common PU design (Fig.4) that includes the following key components and parameters:

(1) There are CN crossbars with the size of 2^r (rows) and 2^c (columns); each cell has 2^b levels.

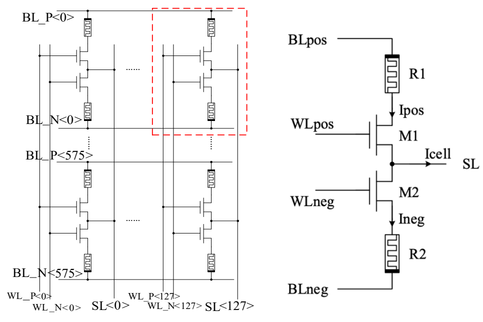
(2) DACs transform digital input signals of each crossbar-row into analog; the sensing resolution, d. To decrease this consumption, many designs (ISAAC [17], PRIME [16], pipelayer [19], etc.) convert the input signal of S-bit width (I/O precision) into a series of low-precision signals (d-bit width) and crossbars have to perform several (S/d) sequential operations, which is a time-for-space optimization strategy. The number of DAC is $2^r * CN$.

(3) ADCs convert the analog output of crossbar-columns into digital. To decrease consumption, a time-division multiplexing method is used: Each crossbar-column's output is fed into some shared ADC(s); the

latter converts output signals one by one. The number of ADC 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:
(5) Shift-Adder of each column. They are compatible with the above sequential operations of DACs, which respectively shift and add the partial sums of the input signals of d-bit width. (6) Activation function of each column and pooling function.
(7) Adder tree. When a large weight matrix is split into multiple small matrices to accommodate the crossbar dimension, it is necessary to add up the output of the corresponding columns of each crossbar. Section 6.1 will give a concrete example.
(8) Data buffers. There is a buffer between the activation and the pooling modules. Its size is $x*q*C$, where x is the width of image size, q is the stride of pooling operation ($q=1$ if no pooling operation) and C is the channel size. When feeding data into this buffer, the simulator determines whether the data entered is sufficient for a pooling operation; the threshold is $(x*(q-1)+q)*C$. If so, the pooling function will be triggered and send the result to the buffer for the next layer. Then, every $q*C$ data fed into the pooling buffer will trigger pooling function.

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



(a) 2T2R-Crossbar design (b) Crossbar cell design

Fig. 5 Crossbar design

4.3 The Crossbar

As a reference design, we adopt a typical 2T2R-crossbar design (Fig. 5(a)). One 'logic' cell for a weight value is composed of 2 ReRAMs and 2 transistors: Each stands for the positive or negative value respectively and WL_P/WL_N control whether the corresponding transistor is conduction. BL_P/BL_N are the input and their voltage difference stands for the input value, while SL is the output line. So 2 cells can represent a value of $2^{(b+1)}$ levels (suppose a cell can represent 2^b levels; 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 of 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. Further, 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 (like C++) conveniently and efficiently. So 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's equations: For a 2^r*2^c crossbar, more than $[2^r*2^c+2^r*(2^c-1)]$ voltage variables (the voltages of the input node and output node of each cell) and $3*2^r*2^c$ currents need to be solved. Moreover, since ReRAM cells are devices with non-linear V-I characteristics [39], the equations are not linear. So the simulation speed is very slow.

Therefore, like some existing studies (e.g. MNSIM[10]) we use behavior model computation to discard some unimportant factors: First, the influence of non-linear V-I characteristics is ignored, as the input signal of 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. Second, the inductances and capacitances of interconnect lines have little influence on ReRAM-based computing[27]; thus we also ignore them.

We simulate the crossbar behavior through the corresponding vector-matrix-multiplication directly, a.k.a., 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 (in contrast, MNSIM[34] ignored the variation): From the computational perspective, a random is generated according to the Gaussian distribution and added to the conductance value.

Our compromise method works very well. For verification, we compare our simulation result of crossbar with the corresponding result of circuit-level simulation of a real chip[*]: Our method only introduces 2.68% computation error. The comparison details are as following:

We carry out the circuit simulation of the crossbar with CADENCE Spectre. The size of the crossbar is $1152 * 128$, which leads to a $576 * 128$ 2T2R array (where two cells represent one weight value, with resistance R_pos and R_neg); the HRS (LRS) is $800k\Omega (50k\Omega)$[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 $87m\Omega$ for bit-line, $100m\Omega$ for source-line, and 1.16Ω for word-line, which are derived from the post-layout parameters (the processing technology is CMOS 130nm). 4 input voltages, 0.15/0.1/0.05/0V, are used, which means each time one 2-bit input value is fed(d=2). ReRAM device model is from[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 part of crossbar columns (source-lines) are working. In this verification, under different configurations, 4, 8, 16, 32, 64, 128 columns of equal intervals are enabled each time (in our actual reference design, the case of 4-column is used, a.k.a., $A = 4$, which is the same configuration as ISAAC).

Then, we compare the output currents of behavior model computation and the 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 4 columns each time, the error is much less, about 0.03%.

Note that, here we just present the computation paradigm of a single crossbar; the proposed parallel simulation mode will be detailed in Section 6.

[*]It is an MLP-enabled chip taped out under the 130nm process; the size of a crossbar is $1152 * 128$ with the 2T2R structure.

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 The I/O Limitation

Due to hardware resource limitation, the bit-width of signals between layers is usually fixed. Suppose a weight value is 2^b-level (weight precision) and there are 2^c rows of inputs, the full precision of the crossbar output is $(S + b + c)$ bits (computation precision): S is the I/O precision; thus, there are 2^c 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 some other 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. But the distribution of source data may be not even, which tends to be concentrated within a certain value range; thus we should extract S consecutive bits from the original data and the starting and ending range is configurable[28]. Our algorithm supports this method.

5.2 The Serial Input From DACs

The serial input (usually 1 or 2 bits 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 will be stored in S&H and processed by shift-adders with the following results. It 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. 5(b), R1 and R2 represent the positive and negative resistances respectively in one cell. 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, are working in the linear area, a.k.a., switched on. Let $V_{BLpos} - V_{SL} = V_{SL} - V_{BLneg} = V_{read}$, then we have $I_{cell} = I_{pos} - I_{neg} = V_{read} * (1/R1 - 1/R2)$ according to the Kirchhoff's law. Thus, the equivalent weight of the cell is $(1/R1 - 1/R2)$; we can adjust $R1$ and $R2$ to get any signed weight value.

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

The fixed-point adders are high-speed components while the working frequency of PU is very limited (for example, in ISAAC it is only 10MHz). Therefore, they can be multiplexed by time, making the overhead very small. FC layers can be mapped similarly. With 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 unbalance 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, whose ratio of storage to calculation amount is always 1.

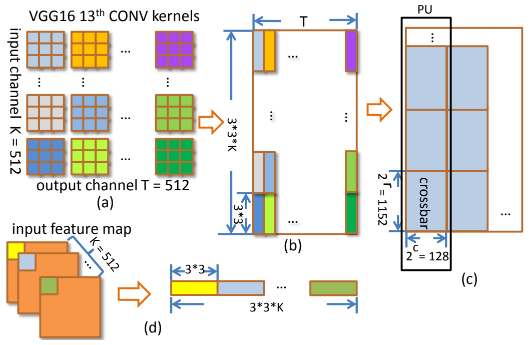


Fig. 6 Mapping weight to crossbars (the 13^{th} CONV layer of VGG16).

Thus, if the hardware is only able to hold all the parameters[†], the layer with the highest reuse degree would become the performance bottleneck, as its allocated resource(s) has to repeatedly process input data with shared weights in time-division multiplexing.

Accordingly, the mapper is also focused on the scalability of RNA when more resources are available. We propose an intuitive and quantitative method for the balanced distribution of on-chip resources.

There is a production-consumption relationship between two adjacent layers from the perspective of data flow. So we can define a *production-consumption ratio* for every layer; $C : 1$ means averagely every C calculations in the previous layer will trigger 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 is always ready to read.

2) For two adjacent CONV layers, CONV$_i$ and CONV$_{i+1}$, let q is the stride of the max-pooling layer between them ($q = 1$ if there is no max-pooling) and p is the stride of CONV$_{i+1}$, the ratio of CONV$_{i+1}$ is $(p * q)^2 : 1$.

3) For the two adjacent layers, CONV$_i$ and FC$_{i+1}$, let t is the output size of CONV$_i$, then the ratio of FC$_{i+1}$ is $t : 1$, no matter whether there is a max-pooling between them or not. The reason is that FC layer can consume all input data and produce all output data at once, while CONV layer has to produce output data one by one.

4) For the two adjacent FC layers, FC$_i$ and FC$_{i+1}$, the

[†]If one RNA cannot hold all the data, multiple chips have to be used. Our work is currently focused on a single chip, but its mapping principle can also be applied to multi-chip scenarios.

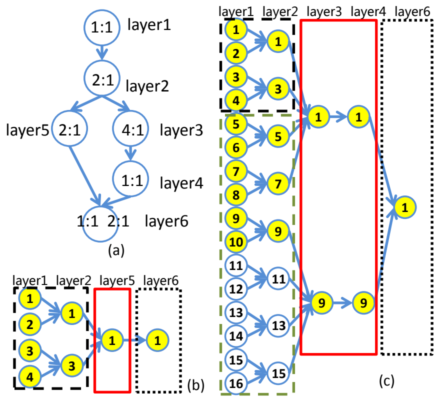


Fig. 7 (a) The NN contains two branches. (b) and (c) are ratio trees of two branches with resource-allocation priority in nodes. The bottleneck is layer 6.

ratio of the latter is always 1:1.

Based on the above analyses, we can design a *ratio tree* to represent the production-consumption relationships of a given NN intuitively, as well as the corresponding algorithm that gives the balanced allocation strategy of crossbar-resources.

Without loss of generality, Fig. 7(a) is the topology of an imaginary NN that contains two branches[‡]. Each circle represents one layer and the corresponding ratio is shown in it: Layer 6 has two ratios as it has two previous layers.

Fig. 7(b) and (c) are the ratio trees of the two branches respectively. In each figure, one column represents a layer (FC or CONV), and each node of a column represents the smallest crossbar set that can store this layer's all weights. It 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 layer 1&2 in the black dashed box of Fig. 7(b) is just the corresponding set in the black box of Fig. 7(c), as well as layer 6. Further, the nodes in red boxes belong to different branches and the nodes in the green dashed box of Fig. 7(c) do not appear in (b) because (b) has a smaller ratio and it doesn't need so many nodes.

Besides, the indegree for each node equals to the production-consumption ratio of the layer; the yellow nodes of each layer represent the crossbars that have been assigned to this layer, a.k.a., the number of replicas of the weight (abbreviated to NRW. Obviously, there must be at least one yellow node for each layer).

[‡]Modern neural networks may contain branches as well as branch fusion, such as the ResNet family.

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

```
1  Input: N = Total # of crossbars, ncb[i] = min # of crossbars required by layer $i$,
   r[i]=ratio of layer $i$, L = # 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 $Y \cup S_k > N$ then
7          break;
8      $Y = Y \cup S_k, i++$;
9  $NRW[i]$ = # of nodes of layer $i$ in $Y$;
10 $n[1] = NRW[1], n[j] = +\infty (j > 1)$;
11 for $j = 2; j \le L; j++$ do
12     $n[j]$ = min($n[j]$, float( $n$[ previous layer of $j$]) / $r[j]$ );
13 $batch\ number = \sum_{t=1}^{L} n[t] * ncb[t]$;
```

Thus, for a node, the prerequisite for its 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, this layer is the *bottleneck*.

Further, we give the pseudo code to obtain the balanced allocation strategy, and the average number of crossbars capable of running in each cycle (defined as *batch number*):

The 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, and then all the nodes on the next shortest path from another leaf (except those that have been assigned) and so on. For a given layer, the maximum number of its yellow nodes in all the ratio trees is its NRW, e.g., NRW of layer 1 is 10 in Fig. 7.

The calculation algorithm of *batch number* (line 10-13) is also evident: For layer j, this value is just the quotient of the NRW of its previous layer and its production-consumption ratio. If there are more than one previous layer, the minimum value is taken.

6.3 Parallel Simulation

Now we can also calculate the number of crossbars that could be simulated at the same cycle, a.k.a., *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 numbers* is 10.6 averagely. VGG16 needs at least 401 and 3416 crossbars respectively, and the *batch numbers* is 10.2 averagely. When more resources available, the *batch number* increases.

The first row of Table 3 shows *batch numbers* of

VGG for CIFAR-10 under different number of available crossbars[§].

To fully use the parallel resources of the state-of-the-art CPUs or GPGPUs, we propose the batch processing mode, reflected in the following aspects:
1) We gather S/d sequential input vectors into an input matrix, which converts the crossbar computation as matrix-matrix-multiplication. The sizes of the two matrixes are $(S/d) * 2^r$ and $2^r * 2^c$, respectively. S is the width of input signal, d is the width of low-precision signals and the crossbar scale is $2^r * 2^c$, as defined in Section 4.2. Note that, for crossbar and the peripheral circuits, we separate the 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), while the computation is just done in the batch processing mode. It's equivalent in effect.
2) All the crossbars that can run at the same time will be simulated synchronously in different threads: For the multi-core/multi-socket CPU(s), we use OpenMP for parallel execution, while for GPGPU, we use CUDA.

In summary, we increase the ratio of computation 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 code is open source in GitHub anonymously. The training algorithm is provided as some customized operations and script files in PyTorch[29], and the simulation is based on SystemC. The latter can be generated according to the user-provided description file of the target CNN, the mapping strategy and the simulated hardware constraints.

For parallel computing, the general workflow is: In each cycle the simulator will identify all the crossbars that can be handled synchronously, collect their data and call 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 host memory and GPGPU memory. APIs of Intel Math Kernel Library (MKL) based on OpenMP and cuBLAS/cuRAND on CUDA are employed respectively.

[§]Based on the area of the RNA chip in ISAAC[17], we estimate the number of crossbars under the same area; the upper limit is about 3000. Parameters of individual components of our PU design are in Section 7.1.

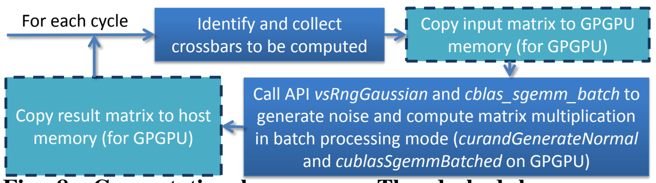

For each cycle
Identify and collect crossbars to be computed
Copy input matrix to GPGPU memory (for GPGPU)
Call API *vsRngGaussian* and *cblas_sgemm_batch* to generate noise and compute matrix multiplication in batch processing mode (*curandGenerateNormal* and *cublasSgemmBatched* on GPGPU)
Copy result matrix to host memory (for GPGPU)

Fig. 8 Computational process. The dashed boxes are omitted in CPU.

Table 1 Hardware Parameters (130nm CMOS)

ADC(8-bit)	26 mw	0.55 mm^2
DAC(1-bit)	0.06256 mw	2.656256E-06 mm^2
S&H	1.5625008E-04 mw	6.25E-07 mm^2
X-bar	43.2 mw	0.021312 mm^2
Shift-add	0.8 mw	0.00096 mm^2
Input-buf(2KB)	19.84 mw	0.0336 mm^2
Output-buf(256B)	3.68 mw	0.01232 mm^2
ReLU	51.2 uw	142.4 um^2
Max Pooling	6.4 mw	0.00384 mm^2

As a reference design, the hardware process is 130nm, consistent with the chip for verification. Note that, because the chip is not officially released, it is inconvenient for us to directly disclose its data. Instead, we use the following methods to obtain the information of area, power consumption of various hardware components (listed in Table 1). CACTI[30] is used to model energy and area for buffers, ADC's area and power are obtained from the ADC survey[31]; others are scaled based on ISAAC's.

ReRAM device model is from [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 ($A = 4$) and the crossbar size is $1152 * 128$. The processing latency of a crossbar with the ADC/DAC peripheral circuits is 100 ns. Currently, 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.

Table 2 Host Configurations

CPU Intel(R) Xeon E5-2680 v4@2.40GHz, 14-core		
OS Linux 4.4.0-87	SystemC 2.3.2	GCC 5.4.0
Memory DDR4-512GB	GPU Tesla P100-PCIE-12GB	
MKL 2019.0.3	CUDA 9.0	cuBLAS, cuRAND 9.0.176

7.2 Simulation Speed

For VGG for CIFAR-10 with different available resources (the first row of Table 3), we construct the corresponding simulators with the optimized mapping strategy and test the acceleration performance of the batch mode. Information of the simulation host are presented in Table 2.

For each configuration, i.e., different number of available crossbars, the following test cases are constructed: The baseline is simulating the whole 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; their *batch numbers* are as shown in first Row of Table 3. For each case, we also evaluate whether the introduction of device variation has an impact on performance or not. The input data contains 100 pictures whose size is $32 * 32$ with three channels.

From Table 3, we can see that the speedup of the batch mode with random number generation on the multi-core CPU server is from 3.00× to 5.02×, while on GPGPU the speedup is from 6.18× to 34.29×. If no random generation, the speedup on CPU is from 2.99× to 5.48× and on GPGPU it is from 5.43× to 11.91×. Anyway, the greater the size of the calculation, the greater the advantage of the batch mode. Especially for GPGPU, the more computation, the easier it is to perform the vast parallel computing power.

In addition, *batch number* determined by the mapping strategy is also directly related to the computational parallelism, so an optimized mapping strategy can not only make full use of RNA crossbars but also benefit simulation speed.

7.3 Evaluations of Training Algorithm

The NN bench is VGG16 [32] (a large CNN including $1.4 * 10^8$ parameters for ImageNet), VGG for CIFAR-10 (a medium-sized CNN including 15 CONV layers) and LeNet [33] (a small CNN) for MNIST.

Different training algorithms are compared. The method is 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 just considers the distribution of the actual available device conductance. *R(rounding)* is the enhanced method that takes the non-linearity of the distribution of I/O data into account for rounding. *S/A(shift/add)* takes the effect of shift/add operations into account further.

Table 4 Evaluations of Training Algorithm

NN	LeNet	VGG for CIFAR-10	VGG16
Full precision	99.26%	85.17%	69.10% / 88.90%
Q+N	99.30%	79.40%	46.08% / 71.34%
Q+N+R	99.31%	79.01%	45.97% / 71.44%
Q+N+R+S/A	99.32%	81.23%	65.64% / 86.96%

All generated CNNs are deployed onto the simulator to get the inference accuracies in Table 4: For small datasets, the corresponding CNNs have relatively sufficient ability. So 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 impact on accuracy because it alters the usual dot-product calculation paradigm. In addition, as the 8-bit I/O width is sufficient for general inference applications, the impact of rounding method is very small.

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 and the baseline configuration is $128 * 128$ crossbar, 5-bit I/O precision (i.e., the ADC resolution), and the others are the same as the default values. All results are in Fig. 9.

From Fig. 9(a), we can see that the crossbar scale does not affect NN accuracy but affects the hardware consumption. As the scale is smaller, the crossbar number increases, as well as the consumption of peripheral circuit (especially ADCs). When the scale is too large, the utilization of each crossbar decreases and the entire hardware consumption rises again.

In Fig. 9(b), 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), so increasing the resolution of ADC actually improves the calculation accuracy. The hardware consumption also increases because the resolutions of ADCs have risen (Here 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

Now we evaluate the relationship between the number of available crossbars and the performance of the simulated design. Using the proposed

Table 3 *Batch number* and Execution Time per Sample of Each Case. Unit: Second.

Total # of crossbars / *batch number*	45 / 10.6	600 / 577	1200 / 1197	1800 / 1784	2400 / 2394	3000 / 2981
1 CPU core+noise (baseline)	1460.0	891.5	840.1	826.7	819.3	819.9
8 CPU core+noise / speedup	486.0 / 3.00	224.9 / 3.96	180.3 / 4.65	184.9 / 4.47	163.1 / 5.02	163.4 / 5.01
GPGPU+noise / speedup	236.1 / 6.18	25.9 / 34.29	25.4 / 33.06	25.9 / 31.84	24.5 / 33.32	24.3 / 33.64
1 CPU core (baseline)	166.2	123.1	117.2	111.0	93.1	87.8
8 CPU core / speedup	55.5 / 2.99	25.6 / 4.79	23.8 / 4.92	20.2 / 5.48	18.9 / 4.90	18.2 / 4.80
GPGPU / speedup	30.5 / 5.43	10.3 / 11.91	10.84/10.81	11.1 / 9.98	10.5 / 8.79	10.5 / 8.32

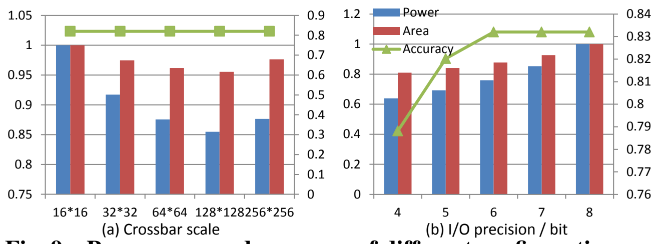


Fig. 9 Power, area and accuracy of different configurations (normalized)

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 $5\times$ to $64\times$ speedup with only $1.42\times$ to $15.33\times$ crossbars used: The case of initial configuration consumes 45 crossbars, which just meets the NN's minimum demand for weight storage.

Specifically, this CNN contains three $2 * 2$ max-pooling layers, each of which is located after several successive convolution layers; FC layers are behind all of the above 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 $2 * 2$, which limits the performance ratio of its front and successive layers to a maximum of 4; thus the total upper limit of speed-up is $64\times$, as we do not duplicate the FC layer.

8 Conclusion and Future Work

We propose an end-to-end tool chain for RNA design; 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.

On the other hand, now the reference design is based on the 130nm process (to match the verification chip) and the wire width is large, so the influence of factors such as IR-drop is small. As the process progresses, the effects of the unfavorable factors will gradually become more obvious, and extra techniques[24, 34] need to be introduced to overcome them in the next step. Further, 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 will be gradually taken into consideration, too. We intend to utilize the widely used NoC simulator (Booksim[35]) and the corresponding power model (ORION 2.0[36]) to obtain more detailed running data.

References

[1] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[2] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[3] G. Snider, R. Amerson, D. Carter, H. Abdalla, M. S. Qureshi, J. Leveille, M. Versace, H. Ames, S. Patrick, B. Chandler, A. Gorchetchnikov, and M. Ennio, "From synapses to circuitry: Using memristive memory to explore the electronic brain," *Computer*, vol. 44, no. 2, pp. 21–28, 2011.

[4] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen, and H. Yang, "Memristor-based approximated computation," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*. IEEE Press, 2013, pp. 242–247.

[5] B. Liu, M. Hu, H. Li, Z.-H. Mao, Y. Chen, T. Huang, and W. Zhang, "Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–6.

[6] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[7] B. Li, L. Xia, P. Gu, Y. Wang, and H. Yang, "Merging the interface: Power, area and accuracy co-optimization for rram crossbar-based mixed-signal computing system," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[8] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[9] P.-Y. Chen, X. Peng, and S. Yu, "Neurosim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *Electron Devices Meeting (IEDM), 2017 IEEE International*. IEEE, 2017, pp. 6–1.

[10] L. Xia, B. Li, T. Tang, P. Gu, P. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "Mnsim: Simulation platform for memristor-based neuromorphic computing system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7 2017.

[11] M. K. F. Lee, Y. Cui, T. Somu, T. Luo, J. Zhou, W. T. Tang, W.-F. Wong, and R. S. M. Goh, "A system-level simulator for rram-based neuromorphic computing chips," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, p. 64, 2019.

[12] S. R. Lee, Y.-B. Kim, M. Chang, K. M. Kim, C. B. Lee, J. H. Hur, G.-S. Park, D. Lee, M.-J. Lee, C. J. Kim, U.-I. Chung, I.-K. Yoo, and K. Kim, "Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory," in *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 2012, pp. 71–72.

[13] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[14] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[15] D. Ielmini, "Modeling the universal set/reset characteristics of bipolar rram by field-and temperature-driven filament growth," *IEEE Transactions on Electron Devices*, vol. 58, no. 12, pp. 4309–4317, 2011.

[16] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 2016, pp. 27–39.

[17] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 2016, pp. 14–26.

[18] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, and J. Yang, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[19] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA),2017 23rd IEEE Symposium on*. IEEE, 2017.

[20] P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, H.-S. P. Wong, and H. Qian, "Face classification using electronic synapses," *Nature Communications*, vol. 8, 2017.

[21] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *arXiv preprint arXiv:1712.05877*, 2017.

[22] H. Liu, J. Han, and Y. Zhang, "A unified framework for training, mapping and simulation of reram-based convolutional neural network acceleration," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 63–66, 2019.

[23] B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell, "Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems," in *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*. IEEE, 2014, pp. 63–70.

[24] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: variation-aware training for memristor x-bar," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[25] T. Tianqi, X. Lixue, L. Boxun, W. Yu, and Y. Huazhong, "Binary convolutional neural network on rram," in *Proceedings of 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017.

[26] S. Xu, Y. Wang, Y. Han, and X. Li, "Pimch: cooperative memory prefetching in processing-in-memory architecture," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 209–214.

[27] L. Xia, P. Gu, B. Li, T. Tang, X. Yin, W. Huangfu, S. Yu, Y. Cao, Y. Wang, and H. Yang, "Technological exploration of rram crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, 2016.

[28] P. Gysel, "Ristretto: Hardware-oriented approximation of convolutional neural networks," *arXiv preprint arXiv:1605.06402*, 2016.

[29] A. Paszke, S. Chintala, R. Collobert, K. Kavukcuoglu, C. Farabet, S. Bengio, I. Melvin, J. Weston, and J. Mariethoz, "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration, may 2017."

[30] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi,

"Cacti 6.0: A tool to model large caches," *HP Laboratories*, pp. 22–31, 2009.

[31] B. Murmann *et al.*, "Adc performance survey 1997-2016," *Online] http://www. stanford. edu/murmann/adcsurvey. html*, 2016.

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

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[34] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, and Y. Xie, "FPSA: A full system stack solution for reconfigurable reram-based NN accelerator architecture," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, 2019, pp. 733–747. [Online]. Available: https://doi.org/10.1145/3297858.3304048

[35] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. Dally, "Booksim interconnection network simulator," *Online, https://nocs. stanford. edu/cgibin/trac. cgi/wiki/Resources/BookSim*, 2016.

[36] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A power-area simulator for interconnection networks," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 20, no. 1, pp. 191–196, 2011.

Xiang Fei received his B.S. degree in computer science from Zhejiang University, Hangzhou, in 2016. He is currently a Ph.D. student in Department of Computer Science and Technology, Tsinghua University, Beijing. His research interests include computer architecture and neuromorphic computing.

Youhui Zhang received his B.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a professor in the 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 was born in 1946. He received the Master degree in computer science from Tsinghua University, Beijing, China. Currently he is a professor at the Department of Computer Science and Technology in Tsinghua University, Beijing, China. His research interests include high performance computing, network storage and parallel compiler.