# RRAM-Based Analog Approximate Computing

Boxun Li, *Student Member, IEEE*, Peng Gu, *Student Member, IEEE*, Yi Shan, Yu Wang, *Senior Member, IEEE*, Yiran Chen, *Member, IEEE*, and Huazhong Yang, *Senior Member, IEEE*

***Abstract*—Approximate computing is a promising design paradigm for better performance and power efficiency. In this paper, we propose a power efficient framework for analog approximate computing with the emerging metal-oxide resistive switching random-access memory (RRAM) devices. A programmable RRAM-based approximate computing unit (RRAM-ACU) is introduced first to accelerate approximated computation, and an approximate computing framework with scalability is then proposed on top of the RRAM-ACU. In order to program the RRAM-ACU efficiently, we also present a detailed configuration flow, which includes a customized approximator training scheme, an approximator-parameter-to-RRAM-state mapping algorithm, and an RRAM state tuning scheme. Finally, the proposed RRAM-based computing framework is modeled at system level. A predictive compact model is developed to estimate the configuration overhead of RRAM-ACU and help explore the application scenarios of RRAM-based analog approximate computing. The simulation results on a set of diverse benchmarks demonstrate that, compared with a x86-64 CPU at 2 GHz, the RRAM-ACU is able to achieve 4.06–196.41× speedup and power efficiency of 24.59–567.98 GFLOPS/W with quality loss of 8.72% on average. And the implementation of hierarchical model and X application demonstrates that the proposed RRAM-based approximate computing framework can achieve >12.8× power efficiency than its pure digital implementation counterparts (CPU, graphics processing unit, and field- programmable gate arrays).**

***Index Terms*—Approximate computing, neural network, power efficiency, resistive random-access memory (RRAM).**

## I. Introduction

POWER efficiency has become a major concern in modern computing system design [1]. The limited battery capacity urges power efficiency of hundreds of giga floating point operation per second per watt (GFLOPS/W) for mobile embedded systems to achieve the desirable portability and performance [2]. However, the highest power efficiency of contemporary CPU and graphics processing unit (GPU) systems is only ∼10 GFLOPS/W, which is expected not to substantially improve in the predictable scaled technology node [3], [4]. As a result, researchers are looking for alternative architectures and technologies to achieve further performance and efficiency gains [5].

Approximate computing provides a promising solution to close the gap of power efficiency between present-day capabilities and future requirements [6]. Approximate computing takes advantage of the characteristic that many modern applications, ranging from signal processing, pattern recognition to computer vision, are able to produce results with acceptable quality even if many computation are executed imprecisely [7]. This tolerance of imprecise computation can be leveraged for substantial performance and efficiency gains and has inspired a wide range of architectural innovations [1], [8].

Recent work in approximate computing mainly focuses on hardware design of the basic computing elements, such as approximate adders and logics [9]–[11]. These techniques have adequately demonstrated the benefit of approximate computing, but the fixed functionality and low-level design stage limit the further improvement of performance and efficiency. Moreover, these techniques are all based on the traditional CMOS technology, despite of the circumstance that the innovations of device technology have offered a great opportunity for radically different forms of architecture design and can significantly promote the performance and efficiency of computing systems [12].

Our objective is to use the emerging metal-oxide resistive random-access memory (RRAM) devices to design a reconfigurable approximate computing framework with both power efficiency and computation generality. The RRAM device (or the memristor) is one of the promising innovations that can advance Moore's Law beyond the present silicon roadmap horizons [13]. RRAM devices are able to support a large number of signal connections within a small footprint by taking advantage of the ultraintegration density. And more importantly, RRAM devices can be used to build resistive cross-point structure [14], also known as the RRAM crossbar array, which can naturally transfer the weighted combination of input signals to output voltages and realize the matrix–vector multiplication with incredible power efficiency [15], [16].

To realize this goal, the following challenges must be overcome: first of all, an architecture, from the basic processing unit to a scalable framework, is required to provide an efficient

Manuscript received August 22, 2014; revised January 6, 2015 and March 21, 2015; accepted May 20, 2015. Date of publication June 15, 2015; date of current version November 18, 2015. This work was supported in part by the 973 Project under Grant 2013CB329000, in part by the National Natural Science Foundation of China under Grant 61373026 and Grant 61261160501, in part by the Brain Inspired Computing Research, Tsinghua University, under Grant 20141080934, in part by the Tsinghua University Initiative Scientific Research Program, and in part by the Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions, National Science Foundation, under Grant CNS-1253424 and Grant ECCS-1202225. This paper was recommended by Associate Editor D. Chen.

B. Li, P. Gu, Y. Wang, and H. Yang are with the Department of Electronic Engineering, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: yu-wang@tsinghua.edu.cn).

Y. Shan is with the Baidu Research—Institute for Deep Learning, Baidu Inc., Beijing 100085, China (e-mail: shanyi@baidu.com).

Y. Chen is with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261 USA (e-mail: yic52@pitt.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2445741

hardware implementation for RRAM-based analog approximate computing. Second, from the perspective of software, a detailed configuration flow is demanded to program the hardware efficiently for each specific application. Finally, a comprehensive analysis of the system performance and major tradeoffs is needed to explore the application scenarios of RRAM-based analog approximate computing.

In this paper, we explore the potential of RRAM-based analog approximate computing. The main contributions of this paper include the following.

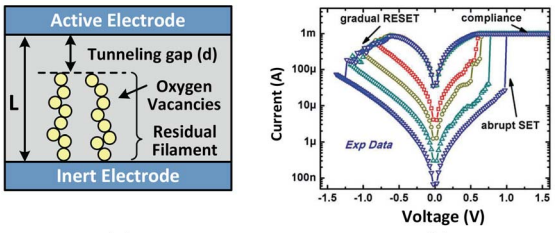
1) We propose a power efficient RRAM-based approximate computing framework. The framework is scalable and is integrated with our programmable RRAM-based approximate computing units (RRAM-ACUs), which work as universal approximators. Simulation results show that the RRAM-ACU offers less than 1.87% error for six common complex functions.
2) A configuration flow is proposed to program RRAM-ACUs. The configuration flow includes three phases: a) a training scheme customized for RRAM-ACU to train its neural approximator; b) a parameter mapping scheme to convert the parameters of a trained neural approximator to appropriate RRAM resistance states; and c) a state tuning scheme to tune RRAM devices to target states.
3) The proposed RRAM-based computing system is modeled at system level to estimate the system performance and explore the major tradeoffs and application scenarios of RRAM-based analog approximate computing. Particularly, a predictive compact model is developed to evaluate the configuration overhead of RRAM-ACU.
4) A set of diverse benchmarks are used to evaluate the performance of RRAM-based approximate computing. Experiment results demonstrate that, compared with a x86-64 CPU at 2 GHz, our RRAM-ACU provides power efficiency of 249.14 GFLOPS/W and speedup of 67.29× with quality loss of 8.72% on average. And the implementation of hierarchical model and X (HMAX) application demonstrates that the proposed RRAM-based approximate computing framework is able to support large scale applications under different noisy conditions, and can achieves >12.8× power efficiency improvements than the CPU, GPU, and field- programmable gate array (FPGA) implementation counterparts.

The rest of this paper is organized as follows. Section II provides the basic background knowledge. Section III introduces the details of the proposed RRAM-based approximate computing framework. The configuration flow and modeling method are depicted in Section IV and V, respectively. Experimental results of different benchmarks are presented in Section VI. Finally, Section VII concludes this paper.

## II. Preliminaries

### A. RRAM Characteristics and Device Model

The RRAM device is a passive two-port elements based on $TiO_x$, $WO_x$, $HfO_x$ [17] or other materials with variable resistance states. The most attractive feature of RRAM devices is that they can be used to build resistive cross-point structure, which is also known as the RRAM crossbar array. Compared with other nonvolatile memories like flash, the RRAM crossbar array can naturally transfer the weighted combination of input signals to output voltages and realize the matrix–vector multiplication efficiently by reducing the computation complexity from $O(n^2)$ to $O(1)$. And the continuous variable resistance states of RRAM devices enable a wide range of matrices that can be represented by the crossbar. These unique proprieties make RRAM devices and the RRAM crossbar array promising tools to realize analog computing with great efficiency.




(a) (b)

Fig. 1. (a) Physical model of the $HfO_x$-based RRAM. The RRAM resistance state is determined by the tunneling gap distance $d$, and $d$ will evolve due to the filed and thermally driven oxygen ion migration. (b) Typical DC I–V bipolar switching curves of $HfO_x$ RRAM devices reported in [18].

Fig. 1(a) demonstrates a model of the $HfO_x$-based RRAM device [18]. The structure is a resistive switching layer sandwiched between two electrodes. The conductance is exponentially dependent on the tunneling gap distance ($d$) as

$$\mathrm{I} = \mathrm{I}_0 \cdot \exp\left(-\frac{d}{d_0}\right) \cdot \sinh\left(\frac{\mathrm{V}}{\mathrm{V}_0}\right). \tag{1}$$

The ideal resistive crossbar-based analog computing requires both linear I–V relationship and continuous variable resistance states. However, nowadays RRAM devices can not satisfy these requirements perfectly. Therefore, we introduce the practical characteristics of RRAM devices in this section.

1) The I–V relationship of RRAM devices is nonlinear. However, when V is very small, an approximation can be applied as $\sinh(\mathrm{V}/\mathrm{V}_0)$–$(\mathrm{V}/\mathrm{V}_0)$. Therefore, the voltages applied on RRAM devices should be limited to achieve an approximate linear I–V relationship [19].
2) As shown in Fig. 1(b), the SET process [from a high-resistance state (HRS) to a low-resistance state (LRS)] is abrupt while the RESET process (the opposite switching event from LRS to HRS) is gradual. The RESET process is usually used to achieve multiple resistance states [20].
3) Even in the RESET process, the RRAM resistance change is stochastic and abrupt. This phenomenon is called "variability." The RRAM variability can be approximated as a lognormal distribution and can make the RRAM device miss the target state in the switching process.

In this paper, we use the $HfO_x$-based RRAM device for study because it is one of the most mature materials explored [17]. The analytical model is put into the circuit with Verilog-A [18], [21]. We use H-simulation program with integrated circuit emphasis (HSPICE) to simulate the circuit performance and study the device and circuit interaction issues for RRAM-based approximate computing.

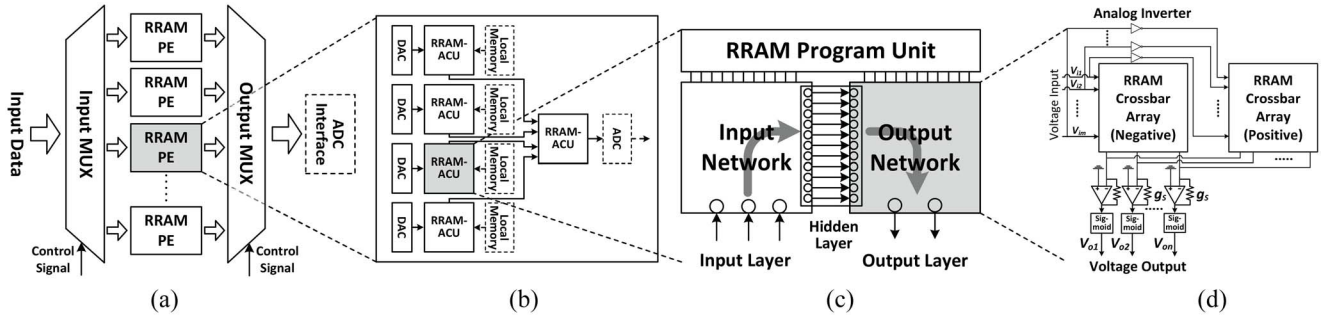



Fig. 2. Overview of the hardware architecture of RRAM-based analog approximate computing. (a) and (b) RRAM approximate computing framework. (c) and (d) RRAM-ACU.

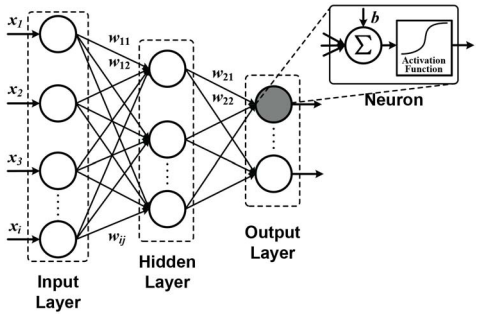



Fig. 3. Three-layer feedforward neural network with one hidden layer.

### *B. Neural Approximator*

Fig. 3 illustrates a simple model of a three-layer feedforward artificial neural network with one hidden layer. The computation between neighbor layers of the network can be expressed as

$$y_j = f_j\left(\sum_{i=1}^{n} w_{ij} \cdot x_i + b_j\right) \tag{2}$$

or

$$\vec{y} = f\left(W \cdot \vec{x} + \vec{b}\right) \tag{3}$$

where $x_i$ is the value of node $i$ in the input (hidden) layer, and $y_j$ represents the result of node $j$ in the hidden (output) layer. $w_{ij}$ is the connection weight between $x_i$ and $y_j$. $b_j$ is an offset. $f_j(x)$ is an activation function, e.g., sigmoid function

$$f(x) = \frac{1}{1+e^{-x}}. \tag{4}$$

It has been proven that a universal approximator can be implemented by a three-layer feedforward network with one hidden layer and sigmoid activation function [22], [23]. Table I gives the maximum errors of the approximations of six common functions by this method based on the MATLAB simulation. The mean square errors (MSEs) of approximations are less than $10^{-6}$ after the network training algorithm completes[1] [24]. The neural approximator offers less than 1.87% error for the six common complex functions. This precision level is able to satisfy the requirements of many approximate computing applications [1].

[1]Theoretically, the network's accuracy shall increase with the network size. However, it is usually more difficult to train a network with a bigger size. The network may easily fall into a local minima, instead of the global optimal solution, and thus sometimes provide a worse result [24].

TABLE I
MAXIMUM ERRORS (%) OF NEURAL APPROXIMATORS

| Function | Nodes in the Hidden Layer | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 5 | 10 | 15 | 20 | 25 |
| $x_1 \cdot x_2 \cdot x_3$ | 22.79 | 1.10 | 0.68 | 0.28 | 0.34 | 0.27 |
| $x^{-1}$ | 9.53 | 0.25 | 0.20 | 0.14 | 0.10 | 0.05 |
| $sin(x)$ | 10.9 | 0.05 | 0.07 | 0.05 | 0.07 | 0.06 |
| $log(x)$ | 7.89 | 0.21 | 0.13 | 0.14 | 0.12 | 0.14 |
| $exp(-x^2)$ | 20.27 | 0.04 | 0.03 | 0.05 | 0.03 | 0.04 |
| $\sqrt{x}$ | 13.76 | 1.87 | 1.19 | 1.43 | 0.35 | 0.49 |

## III. RRAM-BASED ANALOG APPROXIMATE COMPUTING

Fig. 2 demonstrates an overview of the hardware implementation of RRAM-based analog approximate computing. In this section, we will introduce this framework from the basic RRAM-ACU to the scalable RRAM-based approximate computing framework.

### *A. RRAM-Based Approximate Computing Unit*

Fig. 2(c) and (d) shows the proposed RRAM-ACU. The RRAM-ACU is based on an RRAM hardware implementation of a three-layer network (with one hidden layer) to work as a universal approximator. The mechanism is as follows.

As described in (2)–(4), the neural approximator can be conceptually expressed as: 1) a matrix–vector multiplication between the network weights and input variations and 2) a sigmoid activation function.

For the matrix-vector multiplication, this basic operation can be mapped to the RRAM crossbar array illustrated in Fig. 4. The output of the crossbar array can be expressed as

$$\mathrm{V}_{oj} = \sum_{k} \mathrm{V}_{ik} \cdot c_{kj} \tag{5}$$

where, for Fig. 4(a), $c_{kj}$ can be represented as

$$c_{kj} = -\frac{g_{kj}}{g_s} \tag{6}$$

and for Fig. 4(b)

$$c_{kj} = \frac{g_{kj}}{g_s + \sum_{l=1}^{N} g_{kl}} \tag{7}$$

where $g_{kj}$ is the RRAM conductance state in the crossbar array. And $g_s$ represents the conductivity of the load resistance.

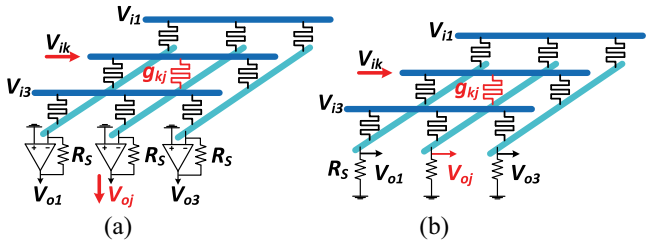



Fig. 4. Two implementations of RRAM crossbar arrays. (a) With and (b) without Op Amps.

Both two types of crossbar array are efficient to realize matrix–vector multiplication by reducing the computation complexity from $O(n^2)$ to $O(1)$.

The latter one, which does not require Op Amps, consumes less power and can be smaller in size. However, there are some drawbacks with the latter implementation when building multilayer networks.

1) $c_{kj}$ not only depends on the corresponding $g_{kj}$, but also depends on all the RRAM devices in the same column. It is difficult to realize a linear one-to-one mapping between the network weight $w_{ij}$ and the RRAM conductance $g_{ij}$. Although previous work proposed some approximate mapping algorithms, the computation accuracy is still a problem [25].
2) The parameters of neighbor layers will influence each other through $R_S$. Voltage followers or buffer amplifiers are demanded to isolate different circuit stages and guarantee the driving force [26], [27]. The size and energy savings compared with the first type implementation will be wasted.

The first implementation can overcome these drawbacks. Op Amps can enhance the output accuracy, make $c_{kj}$ linearly depend on the corresponding $g_{kj}$, and isolate neighbor layers. So we choose the first implementation to build RRAM-ACU.

Since both $R$ (the load resistance) and $g$ (the conductance states of RRAM devices) can only be positive, two crossbar arrays are needed to represent the positive and negative weights of a neural approximator, respectively, with the help of analog inverters [28] as shown in Fig. 6.

The practical weights of the network can be expressed as

$$w_{kj} = R \cdot \left(g_{kj(\text{postive})} - g_{kj(\text{negative})}\right). \tag{8}$$

We also note that the polarities of the terminals of the RRAM devices in two crossbar arrays should be set to opposite directions. This technique is aimed to make the resistance state deviations caused by the currents passing through the paired RRAM devices cancel each other [29]. We refer to this technique as RRAM pairing and it is shown in Fig. 6.

The sigmoid activation function can be generated by the circuit described in [30] and a complete feedforward network without hidden layer is accomplished.

Finally, by combining two networks together, a three-layer feedforward network unit is realized. As described in Section II-B, this network can work as a universal approximator to perform approximated computation. And a basic RRAM approximate computing unit is accomplished.

### B. RRAM-Based Approximate Computing Framework

The overview of the proposed RRAM approximate computing framework is shown in Fig. 2(a) and (b). The building blocks of the framework are the RRAM processing elements (RRAM PEs). Each RRAM PE consists of several RRAM-ACUs to accomplish algebraic calculus. Each RRAM PE is also equipped with its own digital-to-analog converters (DACs) to generate analog signals for processing. In addition, the RRAM PE may also have several local memories, e.g., analog data stored in form of the resistance states of RRAM devices, or digital data stored in the dynamic random access memory or static random access memory. Both use and type of local memory depends on the application requirement and we will not limit and discuss its implementation in detail in this paper. On top of that, all the RRAM PEs are organized by two multiplexers with round-robin algorithm.

In the processing stage, the data will be injected into the platform sequentially. The input multiplexers will deliver the data into the relevant RRAM PE to perform approximate computing. The data will be fed into the RRAM PE in digital format and the DACs in each RRAM PE will convert the date into analog signals. Each RRAM PE may work under low frequency but a group of RRAM PEs can work in parallel to achieve high performance. Finally, the output data will be transmitted out from the RRAM PE by output multiplexer for further processing, e.g., be converted back into digital format by a high-performance analog-to-digital converter (ADC).

The framework is scalable and the user can configure it according to individual demand. For example, for tasks requiring power efficiency, it is better to choose low power Op Amps to form the RRAM-ACUs and each RRAM PE may work in a low frequency. On the other hand, high speed Op Amps, analog to digital (AD)/digital to analogs (DAs) and even hierarchical task allocation architecture will be preferred for high-performance applications.

## IV. Configuration Flow for RRAM-ACU

The RRAM-based analog approximate computing hardware requires a configuration flow to get programmed for each specific task. In this section, we discuss the detailed configuration flow for the proposed RRAM-ACUs. The flow is illustrated in Fig. 5. It includes three phases to solve the following problems.

1) *Training Phase:* How to train a neural approximator in an RRAM-ACU to learn the required approximate computing task?
2) *Mapping Phase:* The parameters of a trained approximator can NOT be directly configured to the RRAM-ACU. We need to map these parameters to appropriate RRAM resistance states in the RRAM crossbar array.
3) *Tuning Phase:* After we achieve a set of RRAM resistance states for an approximate computing task, how to tune the RRAM devices accurately and efficiently to the target states?

All these phases will be introduced in detail in the following sections.

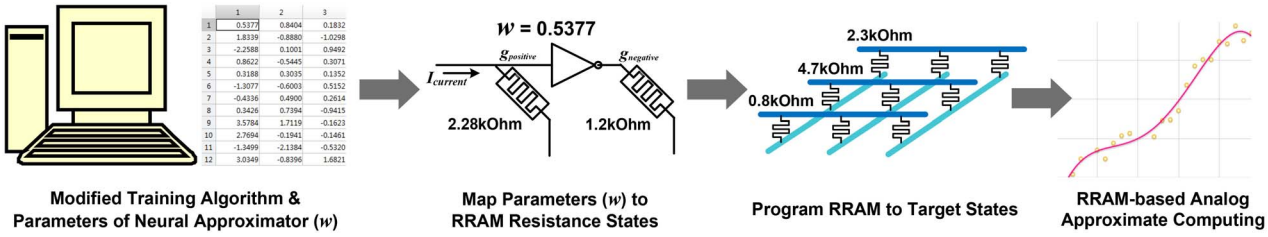



Fig. 5. Configuration flow for RRAM-ACU. The flow includes three phases: 1) training scheme customized for RRAM-ACU to train the neural approximator; 2) parameter mapping scheme to convert the parameters of a trained neural approximator to appropriate RRAM resistance states; and 3) RRAM state tuning scheme to tune RRAM devices to target states efficiently.

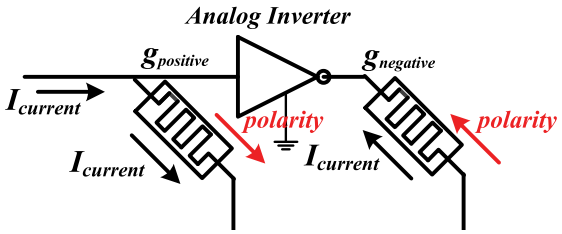



Fig. 6. RRAM pairing technique.

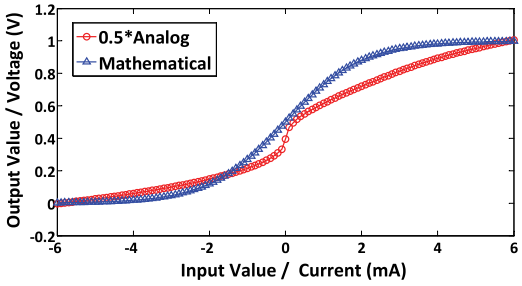



Fig. 7. Comparison between the mathematical sigmoid function and its analog implementation reported in [30]. The output of analog implementation is multiplied by 0.5 for normalization. A significant difference can be observed.

### A. Training Phase: Neural Approximator Training Algorithm

The RRAM approximate computing unit is based on an RRAM implementation of neural approximator. The approximator must be trained efficiently for each specific function. The training process can be realized by adjusting the weights in the network layer by layer [24]. The update of each weight ($w_{ji}$) can be expressed as

$$w_{ji} \leftarrow w_{ji} + \eta \cdot \delta_j \cdot x_i \tag{9}$$

where $x_i$ is the value of node $i$. $\eta$ is the learning rate. $\delta_j$ is the error back propagated from node $j$ in the next neighbor layer. $\delta_j$ depends on the derivative of the activation function (e.g., sigmoid function) as described in Section II-B.

In the RRAM-ACU training phase, both calculations of sigmoid function and its derivative should be adjusted according to the analog sigmoid circuit. Fig. 7 illustrates a comparison between the accurate mathematical sigmoid function and its hardware implementation reported in [30]. The I–V relationship is simulated with HSPICE. There is a significant difference between them. Therefore, we replace the mathematical sigmoid activation function by its simulation results in the training scheme of RRAM-ACU.

Finally, it is worth noting that most weights are small (around zero) after a proper training.[2] For example, more than 90% weights of the trained network[3] are within the range of $[-1.5, 1.5]$ for all the benchmarks used in this paper. The limitation of weight amplitude can simplify the design of RRAM state tuning scheme and help improve the tuning speed.

### B. Mapping Phase: Mapping Neural Approximator Weights to RRAM Conductance States

Once the weights of a neural approximator are determined, the parameters need to be mapped to the appropriate states of RRAM devices in the crossbar arrays. Improperly converting the network weights to the RRAM conductance states may result in the following problems.

1) The converted results are beyond the actual range of the RRAM device.
2) The dynamic range of converted results is so small that the RRAM state may easily saturate.
3) The converted results are so high that the summation of output voltages will exceed the working range of Op Amps.

In order to prevent the above problems, we propose a parameter mapping algorithm to convert the weights of neural approximator to appropriate conductance states of RRAM devices.

The mapping process can be abstracted as an optimization problem. The feasible range of the weights of neural approximators can be expressed as a function of RRAM parameters

$$-R_S \cdot (g_{\mathrm{ON}} - g_{\mathrm{OFF}}) \leq w \leq R_S \cdot (g_{\mathrm{ON}} - g_{\mathrm{OFF}}) \tag{10}$$

where $g_{\mathrm{ON}} = (1/R_{\mathrm{ON}})$ and $g_{\mathrm{OFF}} = (1/R_{\mathrm{OFF}})$. $R_{\mathrm{ON}}$ and $R_{\mathrm{OFF}}$ are the lowest and highest resistance states of RRAM devices. All the weights should be scaling within this range.

[2]A neural network will trend to overfit when many weights of the network are large [31]. Overfitting is a problem that the model learns too much, including the noise, from the training data. The trained model will have poor predictive performance on the unknown testing data which are not covered by the training set.

[3]We use $\ell_2$ regularization in the training scheme. Regularization is a technique widely used in the neural network training to limit the amplitude of network weight, avoid overfitting, and improve model generalization [31]. To be specific, for the $\ell_2$ regularization, a penalty of the square of the two-norm of network weights will be proportionally added to the loss function of the network. So the error of the network and the amplitude of weights will be balanced and optimized simultaneously in the training process [31].

In order to extend the dynamic range and reduce the impact of process variation, we adjust $g_{\mathrm{ON}}$ and $g_{\mathrm{OFF}}$ to

$$g'_{\mathrm{ON}} = \frac{1}{\eta \cdot \Delta_{\mathrm{ON}} + R_{\mathrm{ON}}} \tag{11}$$

$$g'_{\mathrm{OFF}} = \frac{1}{R_{\mathrm{OFF}} - \eta \cdot \Delta_{\mathrm{OFF}}} \tag{12}$$

where $\Delta_{\mathrm{ON}}$ and $\Delta_{\mathrm{OFF}}$ represent the maximum deviation of $R_{\mathrm{ON}}$ and $R_{\mathrm{OFF}}$ induced by process variation of the crossbar array, respectively. $\eta$ is a scale coefficient which is set to 1.1∼1.5 in our design to achieve a safety margin.

The risk of improper conversion can be measured by the following risk function:

$$\mathrm{Risk}(g_{\mathrm{pos}}, g_{\mathrm{neg}}) = |g_{\mathrm{pos}} - g'_{\mathrm{mid}}| + |g_{\mathrm{neg}} - g'_{\mathrm{mid}}| \tag{13}$$

where

$$g'_{\mathrm{mid}} = \frac{g'_{\mathrm{ON}} + g'_{\mathrm{OFF}}}{2} \tag{14}$$

and $g_{\mathrm{pos}}$ and $g_{\mathrm{neg}}$ represent the conductance states of each paired RRAM devices in the positive and negative crossbar arrays, respectively, as (8).

Combining the constraints and the risk function, the parameter mapping problem can be described as the optimization problem shown below

$$\left(g^*_{\mathrm{pos}}, g^*_{\mathrm{neg}}\right) = \arg\min \mathrm{Risk} \tag{15}$$

$$\text{s.t.} \begin{cases} R_S \cdot (g^*_{\mathrm{pos}} - g^*_{\mathrm{neg}}) = w \\ g'_{\mathrm{ON}} \le g_{\mathrm{pos}} \le g'_{\mathrm{OFF}} \\ g'_{\mathrm{ON}} \le g_{\mathrm{neg}} \le g'_{\mathrm{OFF}}. \end{cases} \tag{16}$$

The optimal solutions of this optimization problem are

$$\begin{cases} g^*_{\mathrm{pos}} = g'_{\mathrm{mid}} + \frac{w}{2R_S} \\ g^*_{\mathrm{neg}} = g'_{\mathrm{mid}} - \frac{w}{2R_S}. \end{cases} \tag{17}$$

These are the appropriate conductance states of RRAM devices with the minimum risk of improper parameter conversion.

### C. Tuning Phase: Tuning RRAM Devices to Target States

After the weights of neural approximator are converted into RRAM conductance states, a state tuning scheme is required to program RRAM devices in an RRAM-ACU to target states.

Due to the stochastic characteristics of RRAM resistance change, program-and-verify (P&V) method is commonly used in multilevel state tuning [32]. As shown in Fig. 8, the RRAM device will be first initialized to LRS. Then a sequence of write pulses will be applied to tune RRAM devices gradually. Each write pulse is followed by a read pulse to verify the current conductance state. The amplitude of read pulse should be small enough to not change the RRAM conductance state. The P&V operation will keep on performing until the verify step detects that the RRAM device has reached the target range.

The P&V method choose LRS as the initial state because of the following reasons.

1) LRS is much more uniform than HRS. When an RRAM device is switched between HRS and LRS repeatedly, LRS is able to be uniform while HRS usually varies a lot among different cycles [17], [18], [33].
2) As shown in Fig. 1(b), the resistance change process from LRS to HRS is gradual, while the opposite process is abrupt. It is easier to achieve multiple resistance states from LRS than HRS, although HRS may help reduce the power consumption.
3) Finally, the target resistance states are closer to LRS according to (17). As HRS is usually >100× larger than LRS, initializing RRAM devices to LRS will require much less pulses to reach the target resistance range.

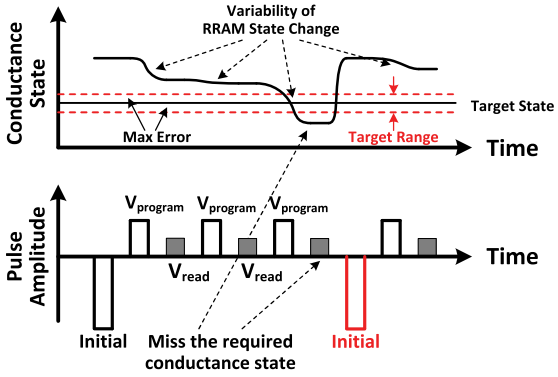



Fig. 8. P&V scheme for multilevel RRAM conductance state tuning.

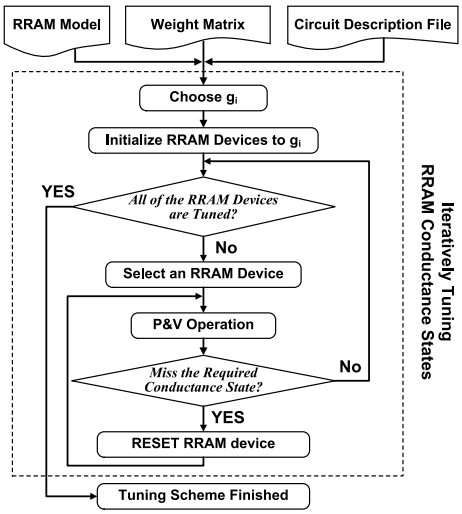



Fig. 9. Proposed state tuning scheme for RRAM-ACU.

However, tuning RRAM devices to accurate $g'_{\mathrm{mid}}$, $g^*_{\mathrm{pos}}$, or $g^*_{\mathrm{neg}}$ as (17) still requires large effort with P&V method. Considering the physical characteristics of RRAM devices and the circuit architecture of RRAM-ACU, we propose a simple but efficient RRAM state tuning scheme as illustrated in Fig. 9. The proposed RRAM state tuning scheme includes the following two steps.

Step 1: Initializing all the RRAM devices in the paired crossbar arrays to the same initial state $g_i$. We hope that only one RRAM device in the pair needs tuning after we initialize all the RRAM devices to $g_i$. The choice of $g_i$ is a major concern in this state tuning scheme. It should be able to approximate

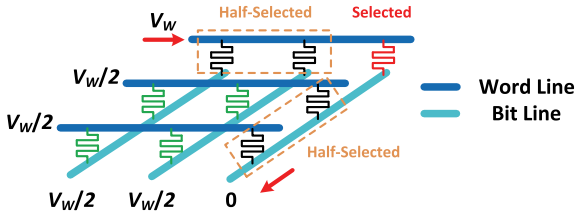



Fig. 10. Tuning RRAM devices with half-select method to mitigate sneak path problem.

most of the optimal states ($g'_{\text{mid}} + (|w|/2R_S)$) in the crossbar array, and should be both uniform and easy to reach for RRAM devices. Therefore, we choose $g_i$ to be close to $g'_{\text{mid}}$ because most $w_{ij}$ are close to zero as discussed in Section IV-A and the optimal states ($g'_{\text{mid}} + (|w|/2R_S)$) will be close to $g'_{\text{mid}}$. On top of that, we choose $g_i$, which should be a uniform LRS that can be achieved easily according to the physical characteristics of RRAM devices. For example, for the $HfO_x$ RRAM devices used in this paper, the lowest resistance state is $R_{\text{ON}} \approx 290\Omega$ [18]. And we set $g_i$ to $\sim(500\Omega)^{-1}$ as it is both close to $g_{\text{ON}}/2$ and can be easily achieved by limiting the compliance current [18].

Step 2: Tuning the positive and negative crossbar arrays to satisfy $R_S \cdot (g_{\text{pos}} - g_{\text{neg}}) = w$. After initializing RRAM devices to $g_i \approx g'_{\text{mid}}$, only one RRAM device in each paired RRAM devices will need to be tuned according to (17). The state tuning scheme will perform P&V operations on the corresponding RRAM device until (16) is satisfied.

Another problem of the state tuning scheme is that the variability of resistance state change may make RRAM devices miss the target conductance range. Considering that the set back process is abrupt and hard to control, and most target states that are close to $g_i$ (e.g., the requirement of resistance change is usually around tens of Ohm), in this paper, the proposed state tuning scheme will reset the RRAM device to the initial state $g_i$. There is no need to prepare a complicated partial setback operation at the cost of increasing the circuit complexity.

The last problem in the state tuning scheme is the sneak path problem. Sneak path usually exits in the memory architecture. As only one cell will be selected in the memory read or write operation, it will be difficult for the architecture to isolate the selected RRAM device from the unselected cells. The unselected cells will form a sneak path, which will disturb the output signals and impact the unselected cells' states [34]. However, when an RRAM crossbar array is used for computation, all the cells will be selected for computation. In other words, no sneak path can be formed in this case. By contrast, each output port can only be used to tune one RRAM device in the corresponding column. We cannot select and tune all the RRAM devices in the crossbar array at the same time. The sneak path still exists in the state tuning scheme.

In order to mitigate the impact of sneak path in the state tuning scheme, the half-select method is adopted [14]. Fig. 10 illustrates the principle of half-select method. The method is aimed to reduce the voltage drop between the selected and unselected cells to reduce the sneak path current and its impact. A half-select voltage ($V_W/2$), instead of connecting to the ground, will be applied on the unselected word line and bit line. The maximum voltage drop between the selected and unselected cells is $V_W/2$ instead of $V_W$. Therefore, the sneak path current is reduced and the unselected cells are protected.

The half-select method mitigate the sneak path problem at the cost of extra power consumption. We further reduce the direct component in the original half-select method to alleviate this problem. To be specific, a ($V_W/2$) and ($-V_W/2$) voltage will be applied on the world line and bit line of the selected cell, respectively. And other unselected cells will be connect to the ground instead of a half-select voltage ($V_W/2$). This technique can reduce around 75% of the power consumption compared with the original method.

Finally, we note that only the RRAM devices in different word lines and bit lines can be tuned in parallel. A parallel state tuning scheme can significantly improve the tuning speed of RRAM-ACU but will require extra copies of peripheral circuits and additional control logic. As the energy consumption (the product of tuning time and power consumption) of tuning the entire RRAM crossbar array remains almost the same, there will be a tradeoff between the tuning speed and the circuit size in the RRAM state tuning scheme. In order to save more area for AD/DAs and Op Amps, each RRAM-ACU is equipped with only one set of tuning circuit in this paper.

## V. System Modeling and Overhead Discussion

In this section, we discuss modeling the performance and energy consumption of the proposed RRAM-based analog approximate computing system at the system level. The model will be used to analyze the system performance, quantize and demonstrate major tradeoffs, and explore the application scenarios of RRAM-based analog approximate computing.

### A. System Level Modeling

Modeling an RRAM-based approximate computing at the system level mainly includes three parts.

1) Modeling the RRAM crossbar array and its peripheral circuits, such as Op Amps, sigmoid circuits, and analog inverters.
2) Modeling the interface: AD/DAs.
3) Evaluating the configuration overhead, especially the time and energy consumption of RRAM state tuning.

For the first part, we use a Verilog-A RRAM device model to build up the simulation program with integrated circuit emphasis (SPICE)-level crossbar array as described in Section II-A. We use a fine-grained SPICE-level simulation because the physical characteristics RRAM devices are different from the ideal linear resistance with continuous variable states, and other nonideal factors, such as the interconnect resistance (IR)-drop in the crossbar, are also difficult to estimate.

For the second part, we extract the parameters like accuracy, speed, power, and area from fabricated chips. Because this aim of this paper is to explore the feasibility and potential

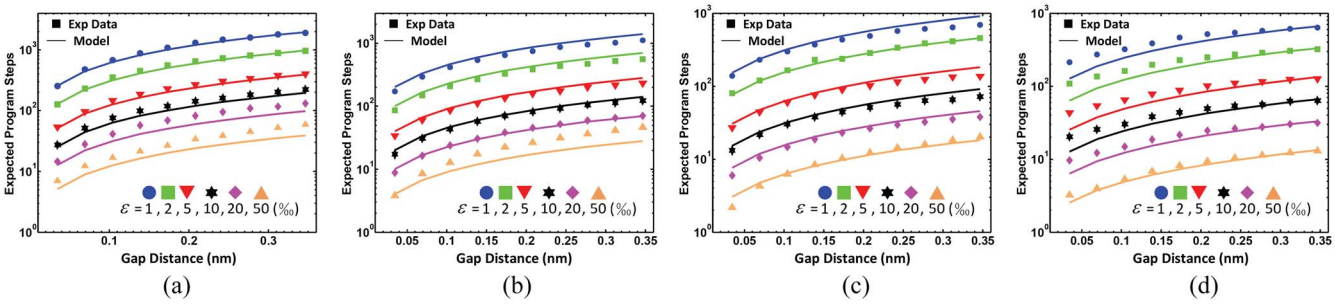



Fig. 11. Performance of the proposed predictive model. The width of write pulses are (a) 1, (b) 2, (c) 5, and (d) 10 ns. The experiment results are achieved by statistically analyzing 5000 independent simulation results for each set of parameters.

of RRAM-based analog approximate computing, we mainly focus on the choice, instead of the design, of AD/DAs. Extracting necessary parameters from fabricated chips is able to satisfy the requirement of modeling the RRAM-based computing system at system level.

Finally, after the RRAM-ACUs are configured properly, the system can perform approximate computing with high-power efficiency. However, tuning RRAM devices to target states is usually time and energy consuming, due to the large number of RRAM devices in RRAM-ACU and the random resistance change of RRAM devices. As a result, the configuration of RRAM-ACU becomes a major overhead of RRAM-based approximate computing. A frequent configuration will drastically decrease the efficiency of RRAM-based computing system. The system should operate continuously without reconfiguration to alleviate this overhead.

At the system level, the energy efficiency (floating-point operations per second/J) of the whole system along with the operating cycles can be calculated through the following equation:

$$\eta_{\text{overall}} = \frac{E_{\text{configure}} + E_{\text{operate}} \times \text{Cycles}}{\text{Cycles} \cdot \text{Insts}} \tag{18}$$

where $\eta_{\text{overall}}$ is the energy efficiency of an RRAM-ACU when the configuration overhead is estimated. $E_{\text{configure}}$ is the total energy consumption cost to program the RRAM-ACU. $E_{\text{operate}}$ is the average[4] energy cost for each approximate computing operation. Cycles is the number of operating cycles after the RRAM-ACU is programmed. Insts represents the number of x86-64 instructions required to complete the same application as an RRAM-ACU.

In this model, the upper limit of system energy efficiency is $E_{\text{operate}}/\text{Insts}$, and $E_{\text{configure}}$ will significantly impact the performance when the system is reconfigured frequently.

### B. Predictive Model of RRAM-ACU Configuration Overhead

The overhead of configuring an RRAM-ACU mainly depends on the efficiency of RRAM state tuning. The estimation of configuration overhead $E_{\text{configure}}$ requires the steps of tuning an RRAM device to the target state and the energy consumption of each step.[5] For the latter, as RRAM devices are tuned around LRS of $g'_{\text{mid}}$ in (17), we can use the energy consumed by a tuning pulse applied on $g'_{\text{mid}}$ to approximate the energy consumption of each step. However, it is usually very hard to predict the tuning steps as the RRAM resistance change is stochastic as described in Section II-A.

To simplify the estimation of tuning steps,[6] we develop a predictive compact model to calculate the expected tuning steps efficiently. The relationship between the change of gap distance ($\Delta_d$) and the expected tuning steps [$E(N)$] can be approximated through the following equation:

$$E(N) = \left[ \frac{e\,\xi}{\varepsilon\, T_w} \cdot \left( \frac{\alpha_w}{\sqrt{T_w}} \Delta_d + \beta_w \right) \right] \tag{19}$$

where the gap distance change $\Delta_d$(nm) represents the difference of RRAM tunneling gap ($d$) between the target and initial conductance state. $\Delta_d$ can be calculated through (1). $V_w$(V) and $T_w$(ns) are the amplitude and width of RRAM write pulses, respectively. $\varepsilon$(‰) is the maximum acceptable deviation of RRAM conductance state. $e$ is the Euler's number. $\alpha_w(\sim 2000)$ and $\beta_w(\sim 25)$ are fitting parameters. $\xi$ is a parameter influenced by the gap change speed and can be represented as

$$\xi \propto \frac{\Delta d}{\Delta t} \tag{20}$$

where $\Delta d/\Delta t$ depends on the device parameters as [18]. $\xi \approx 1$, when $V_w = -1.2$ V and $T_w = 1$ ns.

Fig. 11 verifies the predictive compact model. The reference "Exp Data" are simulation data collected by using a MATLAB-based RRAM device model as [18], [21] to simulate the stochastic tuning process of RRAM devices. We generate 5000 independent simulation results for each set of parameters. The amplitude of the write pulse is set to $-1.2$ V and the read pulse is set to 0.1 V. The initial resistance state is set to 500 Ω. The lines in Fig. 11 represent the results calculated

[4]We can achieve a fine-grained energy consumption with SPICE-based simulation. But an average energy consumption is enough to evaluate the energy efficiency along with operating cycles at system level.

[5]An RRAM device usually requires an initial forming process to get the resistance state changeable. The initial forming process is required only once after the RRAM device is fabricated. In this paper, we assume that all the RRAM devices are already formed before executing approximate computing tasks, and we do NOT include the forming process in the predictive model.

[6]Tuning an RRAM device to the target state can be modeled as a stochastic process. The accurate probability of successfully tuning an RRAM device with $N$ steps can be calculated recursively. The detailed derivation process is provided in the Appendix.

TABLE II
BENCHMARK DESCRIPTION

| Name | Description | Type | x86-64 Insts | Training Set | Testing Set | NN Topology | NN MSE (CPU) | NN MSE (RRAM) | Error Metric | Error |
|---|---|---|---|---|---|---|---|---|---|---|
| FFT | Radix-2 Cooley-Tukey Fast Fourier | Signal Processing | 34 | 32,768 Random Floating Point Numbers | 2,048 Random Floating Point Numbers | $1 \times 8 \times 2$ | 0.0046 | 0.0071 | Average Relative Error | 10.72% |
| Inversek2j | Inverse Kinematics for 2-Joint Arm | Robotics | 100 | 10,000 (x, y) Random Coordinates | 10,000 (x, y) Random Coordinates | $2 \times 8 \times 2$ | 0.0038 | 0.0053 | Average Relative Error | 9.07% |
| Jmeint | Triangle Intersection Detection | 3D Gaming | 1,079 | 10,000 Pairs of 3D Triangle Coordinates | 10,000 Pairs of 3D Triangle Coordinates | $18 \times 48 \times 2$ | 0.0117 | 0.0258 | Miss Rate | 9.50% |
| JPEG | JPEG Encoding | Compression | 1,257 | Three $512 \times 512$ Color Images | One $220 \times 220$ Color Image | $64 \times 16 \times 64$ | 0.0081 | 0.0153 | Image Diff | 11.44% |
| K-Means | K-Means Clustering | Machine Learning | 26 | 50,000 Pairs of (R,G,B) Values | One $220 \times 220$ Color Image | $6 \times 20 \times 1$ | 0.0052 | 0.0081 | Image Diff | 7.59% |
| Sobel | Sobel Edge Detector | Image Processing | 88 | One $512 \times 512$ Color Image | One $220 \times 220$ Color Image | $9 \times 8 \times 1$ | 0.0286 | 0.0026 | Image Diff | 4.00% |

TABLE III
DETAILED PARAMETERS OF PERIPHERAL CIRCUITS IN RRAM-ACU

| | |
|---|---|
| Technology Node | 180nm |
| RRAM Tunneling Gap | 0.2nm - 1.9nm |
| RRAM Resistance Range | 290Ω - 500kΩ |
| $R_S$ | 2kΩ |
| Op Amp | ∼4.8mW |
| ADC | 8bit, ∼3.1mW |
| DAC | 12bit, ∼40mW |
| Frequency | 800MHz |

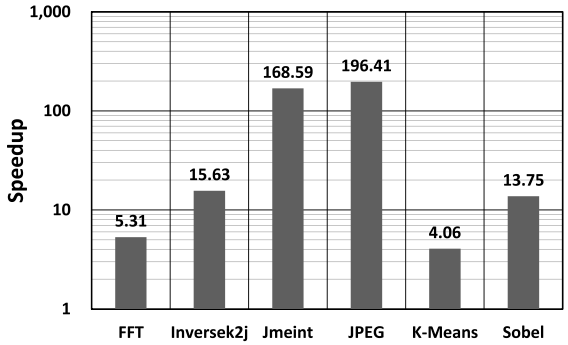



Fig. 12. Speedup of the RRAM-ACU under different benchmarks.

by the predictive compact model. The points are the reference data generated by the simulation of RRAM state tuning process. The predictive compact model fits the Exp Data well.

## VI. EVALUATION

To evaluate the performance and efficiency of the proposed RRAM-based analog approximate computing, we apply our design to several benchmarks, ranging from the signal processing, gaming, compression to the object recognition. A sensitivity analysis is also performed to evaluate the robustness of the RRAM-based computing system.

### A. Experiment Setup

In the experiment, a Verilog-A RRAM device model reported in [18] and [21] is used to build up the SPICE-level crossbar array. We choose the 65 nm technology node to model the interconnection of the crossbar array and reduce the IR-drop. The parameters of the interconnection are calculated with the International Technology Roadmap for Semiconductors 2013 [35]. The sigmoid circuit is the same as reported in [30]. The Op Amps, ADCs, and DACs used for simulation are that reported in [36]–[38], respectively. The working frequency of each RRAM-ACU is set to 800 MHz. Detailed parameters of peripheral circuits are summarized in Table III. Moreover, the maximum amplitude of input voltage is set to 0.5 V to achieve an approximate linear I–V relationship of RRAM devices. The state tuning scheme described in Section IV-C is used to program the RRAM-ACU. The amplitude of the write pulse is set to $-1.2$ V and the read pulse is set to 0.1 V. The pulse width is set to 5 ns. The maximum acceptable deviation ($\varepsilon$) of RRAM conductance state is set to 1% when programming RRAM-ACU. All the simulation results of the RRAM crossbar array are achieved with HSPICE. And the configuration overhead is estimated with the predictive compact model introduced in Section V-B.

### B. Benchmark Evaluation

Table II summarizes the benchmarks used in the evaluation. The benchmarks are the same as that described in [1], which are used to test the performance of a x86-64 CPU at 2 GHz equipped with a CMOS-based digital neural processing unit. The "neural network (NN) Topology" term in the table represents the size of each neural network. For example, "$9 \times 8 \times 1$" represents a neural approximator with nine nodes in the input layer, eight nodes in the hidden layer, and one node in the output layer. The MSE is tested both on CPU and SPICE-based RRAM-ACU after training. The training scheme has been described in Section IV-A, which is modified for RRAM-ACU. The size of the crossbar array in the RRAM-ACU is set to $64 \times 64$ to satisfy all the benchmarks. The unused RRAM devices in the crossbar array are set to the highest resistance states to minimize the sneak path problem. And the unused input and output ports are connected to the ground.

The simulation results are illustrated in Figs. 12 and 13. Compared with the x86-64 CPU at 2 GHz, the RRAM-ACU

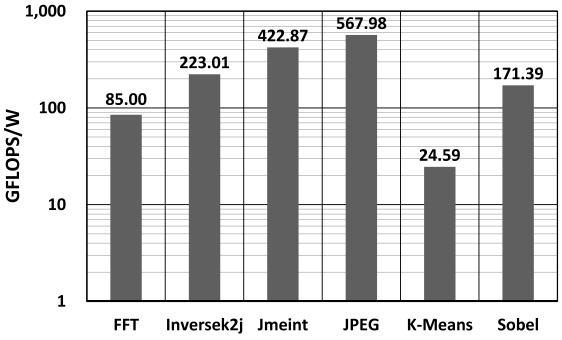



Fig. 13. Power efficiency of the RRAM-ACU under different benchmarks.

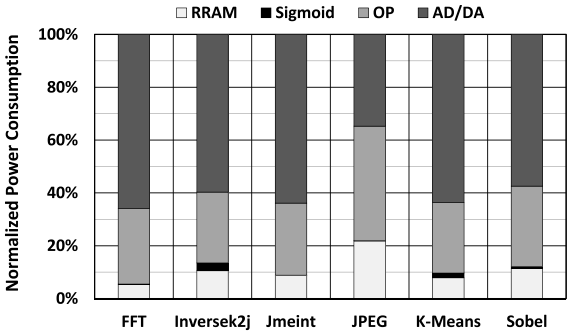



Fig. 14. RRAM-ACU power consumption breakdowns.

achieves 567.98 GFLOPS/W power efficiency and 196.41× speedup at most. And for the whole set of selected diverse benchmarks, the RRAM-ACU provides 249.14 GFLOPS/W and speedup of 67.29× with quality loss of 8.72% on average. The improvement of processing speed mainly depends on the capability of the neural approximator. As the RRAM-ACU is able to transfer a set of instructions into a neural approximator and execute them with only one cycle, the speedup achieved by an RRAM-ACU increases linearly with the number of instructions the neural approximator represents. For example, the "Jmeint" and "Joint Photographic Experts Group (JPEG)" benchmarks achieve >150× speedup as their neural approximators successfully implement the complex tasks that require more than a thousand instructions in traditional x86-64 architectures. In contrast, the "K-Means" and "fast fourier transform (FFT)" benchmarks achieve the least speedup (∼10×) because of the simplicity of tasks. And for the improvement of power efficiency, although the RRAM-ACU for a complex task is able to achieve more speedups, a bigger neural approximator may be also demanded to accomplish more power-consuming tasks. However, as the NN topology increases slower than the instruction number in the experiment, the complex tasks still achieve better power efficiency.

Fig. 14 illustrates the power consumption breakdowns of RRAM-ACUs. The sigmoid circuit is power efficient as there are only six MOSFETs used in the circuit [30]. The power consumption of sigmoid circuit mainly depends on the output voltage. For example, most outputs will be close to zero after the JPEG encoding. And therefore, the sigmoid circuit takes a negligible part of power consumption in the JPEG benchmark. In contrast, the outputs of sigmoid circuits in the "Inversek2j" and K-Means are much larger and the power consumption increases as a result. Compared with the sigmoid circuit, most of the power is consumed by Op Amps and AD/DAs. RRAM devices only take 10%–20% of the total energy consumption in RRAM-ACU, and the ratio increases with the NN topology. Therefore, how to reduce the energy consumed by peripheral circuits may be a challenge to further improve the efficiency of RRAM-based analog approximate computing.

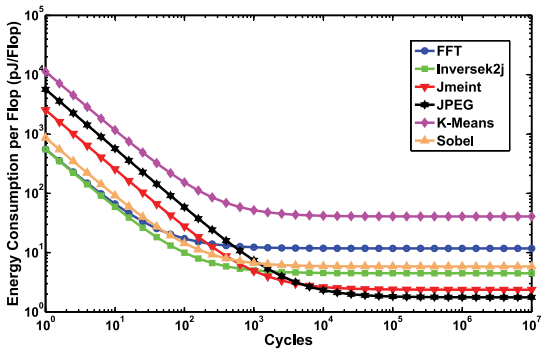



Fig. 15. Energy efficiency of RRAM-ACU along with the operating time when configuration overhead is considered.

Finally, Fig. 15 illustrates the energy efficiency of RRAM-ACU along with the operating time when the configuration overhead is considered. The energy efficiency of the whole system is calculated through the following equation according to (18). It can be seen that the RRAM-ACU should keep operating for a period of time to reduce the impact of configuration overhead and increase the energy efficiency. The configuration overhead increases with the size of neural approximator. For the benchmarks with a small NN topology, e.g., "Sobel" and FFT, the configuration overhead is small. Only $\sim 10^3$ cycles (@800 MHz) are needed to reach a good performance. However, for the complex tasks, more operation cycles ($\sim 10^5$) are required to achieve better energy efficiency.

In Section VII, the simulation results demonstrate the efficiency of RRAM-ACU as well as the feasibility of a dynamic reconfiguration. And there is a tradeoff among the task complexity, power efficiency, and configuration overhead: the more difficult the task, the better power efficiency an RRAM-ACU can achieve, but the more operating cycles are required to hide the larger configuration overhead.

### C. System Level Evaluation: HMAX

In order to evaluate the performance of RRAM-ACU at system level, we conduct a case study on HMAX application. HMAX is a famous bio-inspired model for general object recognition in complex environment [39]. The model consumes more than 95% amount of computation to perform pattern matching in S2 Layer by calculating the distance between the prototypes and units [13], [39]. The amount of computation is too huge to realize real-time video processing on conventional CPUs while the computation accuracy requirement is not strict [40]. In this section evaluation, we apply the proposed RRAM-based approximate computing framework to conduct the distance calculations to promote the data processing efficiency.

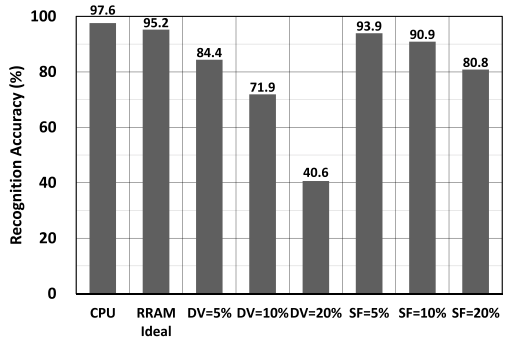



Fig. 16. Performance of RRAM-based HMAX under different noise conditions, where "device variation" represents device variation and "signal fluctuation" represents input signal fluctuation.

We use 1000 images (350 of cars and 650 of the other categories) from PASCAL Challenge 2011 database [41] to evaluate the performance of the HMAX system on the digital and the RRAM-based approximated computation framework. Each image is of $320 \times 240$ pixels with complex background. The HMAX model contains 500 patterns of car images which remain the same on each platform. A correct result indicates both right judgment on the classification of the object and successful detection on the object location.

The RRAM approximate commuting framework illustrated in Fig. 2 is used to support the HMAX approximate. Each RRAM PE consists of four six-input RRAM-ACU for Gaussian calculations and one for four-input multiplication. Therefore, each RRAM PE can realize a 24-input distance calculation per clock cycle [13].

The results of correct rate are shown in Fig. 16. The performance of RRAM-based approximate computing under different noise conditions is also considered. The device variation represents the deviation of the RRAM conductance state and the signal fluctuation represents the deviation of the input signals. As we can observe, the correct rate degradation is only 2.4% on the ideal RRAM-based approximate computing with respect to the CPU platform. This degradation can be easily compensated by increasing the amount of patterns [39].

Moreover, when taking the noise into consideration, the device variation will significantly impact the recognition accuracy. As the performance of neural approximator mainly depends on the RRAM conductance states, the device variation will significantly impact the computation quality and make the recognition accuracy decrease a lot. For example, a 10% device variation can result in a >50% decrease of the recognition accuracy. Therefore, the device variation should be suppressed to satisfy the application requiring high accuracy. Compared with the device variation, the impact of signal fluctuation is much less, which demonstrates that we may use DACs with less precision but less power consumption, in the RRAM-ACU to further improve the power efficiency of the whole system.

The power efficiency evaluation of the RRAM-based HMAX accelerator is given in Table IV. The detailed comparisons with other platforms are given in Table V. The parameters of the HMAX model as well as the evaluation image dataset are different among different platforms. It is hard to compare the recognition accuracy of different implementations. However, we can still compare the efficiency of different platforms through the unified power consumption per frame. The simulation results show that the power efficiency of RRAM-based approximated computation framework is higher than 300 GFLOPS/W. And compared to other platforms like FPGA, GPU, and CPU [40], RRAM-based HMAX achieves a performance up to 6.214 frames/s/W, which is 12.8–270.2× higher than its digital counterparts.

TABLE IV
POWER EFFICIENCY OF THE RRAM-BASED HMAX

| AD/DA (mW) | Analog (mW) | Total (mW) | x86-64 Insts | Frequency (MHz) | Efficiency (GFlops/W) |
|---|---|---|---|---|---|
| 963.1 | 511.96 | 1475.06 | 558 | 800 | 302.64 |

TABLE V
POWER EFFICIENCY COMPARISON WITH DIFFERENT PLATFORMS (FPGA, GPU, CPUS IN [40])

| Parameters | Proposed | FPGA | GPU | CPUs |
|---|---|---|---|---|
| Size of input image | $320 \times 240$ | $256 \times 256$ | | |
| HMAX orientations | 12 | | | |
| HMAX scale | 12 | | | |
| HMAX prototypes | 500 | 5120 | | |
| Average size of prototypes | 8 | | | |
| Cycles for calculation | 32 | - | | |
| Calculation amount/frame | $5455 \times 500$ | - | | |
| Frequency (MHz) | 800 | - | | |
| Power (W) | 1.475 | 40 | 144 | 116 |
| Unified fps/W | 6.214 | 0.483 | 0.091 | 0.023 |
| Speed Up | - | 12.86 | 68.29 | 270.17 |

## VII. CONCLUSION

In this paper, we propose a power efficient approximate computing framework with the emerging RRAM technology. We first introduce an RRAM-based approximate computing framework by integrating our programmable RRAM-ACU. We also introduce a complete configuration flow to program the RRAM-based computing hardware efficiently. Finally, the proposed RRAM-based computing system is modeled at system level, and a predictive compact model is developed to estimate the configuration overhead and explore the application scenarios of RRAM-based analog approximate computing.

Besides exploring the potential of RRAM-based approximate computing, this paper still faces many challenges. For example, the IR-drop caused by the interconnect resistance influences the RRAM computation quality and severely limits the scale of the crossbar system [25]. IR-drop reduction or compensation techniques are demanded to support applications, such as the deep learning, which require a large crossbar size. Besides, many RRAM specific issues, such as the impact of temperature on the resistive switching behavior and I–V relationship, should be also considered to enhance the system reliability in future work [42].

## APPENDIX

The probability of successfully tuning an RRAM device to the target resistance range with $N$ steps can be calculated by

the following expansion:

$$P(\text{Step} = N) = \sum_{i=1}^{N-2} P_r(i) \cdot P_s(N - i - 1|\text{Initial}) + P_s(N|\text{Initial}) \tag{21}$$

where $P_r(n)$ represents that the state tuning scheme detects that the RRAM device misses the required range at the $n$th step and reset it at the $n + 1$th step. $P_s(n|\text{Initial})$ represents that the RRAM device is successfully tuned to the required range with $n$ steps without initialization.

According to [18], the tunneling gap change caused by a voltage pulse follows a Gaussian distribution, whose mean depends on the previous gap ($d$) of the RRAM devices. As the RRAM-ACU mainly takes advantage of the LRS of RRAM devices, $d$ is usually very small (0.2–0.5 nm). We can assume that tunneling gap change caused by each voltage pulse is approximately independent identically distributed and follows the same Gaussian distribution. Because the summation of a series of independent Gaussian distributions is still a Gaussian distribution, $P_s(n|\text{Initial})$ can be represented as follows:

$$P_s(n|\text{Initial}) = \int_{D-\varepsilon}^{D+\varepsilon} N\left(x|n\mu, n\sigma^2\right) \mathrm{d}x \tag{22}$$

where $N(x|\mu, \sigma^2)$ is the Gaussian probability density function that represents the tunneling gap change caused by one voltage pulse. $D$ is the distance between the target and initial tunneling gap. $\varepsilon$ is the maximum absolute deviation of resistance state.

For the other part, $P_r(n)$ can be calculated recursively as

$$P_r(n) = \sum_{i=1}^{n-2} P_r(i) \cdot P_m(n - i - 1|\text{Initial}) + P_m(n) \tag{23}$$

where $P_m(n|\text{Initial})$ represents the probability that the RRAM device *misses* the required resistance with $m$ steps after initialization. $P_m(n|\text{Initial})$ can be expressed as

$$P_m(n|\text{Initial}) = \int_{D+\varepsilon}^{+\infty} N\left(x|n\mu, n\sigma^2\right) \mathrm{d}x. \tag{24}$$

Finally, by combining (21)–(23), the detailed probability of tuning an RRAM device with $N$ steps can be achieved.

## REFERENCES


[1] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. Int. Symp. Microarchit. (MICRO)*, Vancouver, BC, Canada, 2012, pp. 449–460.

[2] DARPA. (Dec. 1, 2012). *Power Efficiency Revolution for Embedded Computing Technologies*. [Online]. Available: http://www.darpa.mil/program/power-efficiency-revolution-for-embedded-computing-technologies

[3] *Tesla® Kepler™ GPU Accelerators*, NVIDIA Corp., Santa Clara, CA, USA, 2012. [Online]. Available: http://www.nvidia.com/content/tesla/pdf/tesla-kseries-overview-lr.pdf

[4] *Intel Microprocessor Export Compliance Metrics*, Intel, Santa Clara, CA, USA, 2015.

[5] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. IEEE 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, San Jose, CA, USA, 2011, pp. 365–376.

[6] S. T. Chakradhar and A. Raghunathan, "Best-effort computing: Re-thinking parallel software and hardware," in *Proc. 47th ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, Jun. 2010, pp. 865–870.

[7] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2013, pp. 48–54.

[8] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit.*, Davis, CA, USA, 2013, pp. 1–12.

[9] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[10] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Proc. 49th Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2012, pp. 796–801.

[11] C. Liu, J. Han, and F. Lomardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proc. Conf. Design Autom. Test Europe*, Dresden, Germany, 2014, Art. ID 95.

[12] V. Narayanan *et al.*, "Video analytics using beyond CMOS devices," in *Proc. Conf. Design, Autom. Test Europe (DATE)*, Dresden, Germany, 2014, pp. 1–5.

[13] B. Li *et al.*, "Memristor-based approximated computation," in *Proc. IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, Beijing, China, Sep. 2013, pp. 242–247.

[14] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie, "Design implications of memristor-based RRAM cross-point structures," in *Proc. IEEE Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Grenoble, France, 2011, pp. 1–6.

[15] S. H. Jo *et al.*, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, 2010.

[16] M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of BSB recall function using memristor crossbar arrays," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2012, pp. 498–503.

[17] H.-S. P. Wong *et al.*, "Metal-oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

[18] S. Yu *et al.*, "A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation," *Adv. Mater.*, vol. 25, no. 12, pp. 1774–1779, Mar. 2013.

[19] Y. Deng *et al.*, "RRAM crossbar array with cell selection device: A device and circuit interaction study," *IEEE Trans. Electron Devices*, vol. 60, no. 2, pp. 719–726, Feb. 2013.

[20] 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, 2012, Art. ID 075201.

[21] X. Guan, S. Yu, and H.-S. P. Wong, "A SPICE compact model of metal oxide resistive switching memory with variations," *IEEE Electron Device Lett.*, vol. 33, no. 10, pp. 1405–1407, Oct. 2012.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[23] Y. Ito, "Approximation capability of layered neural networks with sigmoid units on two layers," *Neural Comput.*, vol. 6, no. 6, pp. 1233–1243, Nov. 1994.

[24] L. Fausett, Ed., *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 1994.

[25] P. Gu *et al.*, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," in *Proc. IEEE 20th Asia South Pac. Design Autom. Conf. (ASPDAC)*, Chiba, Japan, 2015, pp. 106–111.

[26] S. O. Cannizzaro, A. D. Grasso, R. Mita, G. Palumbo, and S. Pennisi, "Design procedures for three-stage CMOS OTAs with nested-miller compensation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 5, pp. 933–940, May 2007.

[27] W. Oh and B. Bakkaloglu, "A CMOS low-dropout regulator with current-mode feedback buffer amplifier," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 10, pp. 922–926, Oct. 2007.

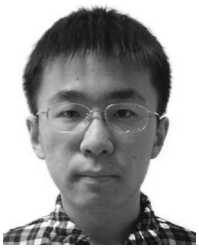
[28] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design*. New York, NY, USA: Oxford Univ. Press, 2002.

[29] B. Li, Y. Wang, Y. Chen, H. H. Li, and H. Yang, "ICE: Inline calibration for memristor crossbar-based computing engine," in *Proc. Conf. Design Autom. Test Europe*, Dresden, Germany, 2014, pp. 1–4.

[30] G. Khodabandehloo, M. Mirhassani, and M. Ahmadi, "Analog implementation of a novel resistive-type sigmoidal neuron," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 4, pp. 750–754, Apr. 2012.

[31] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Comput.*, vol. 7, no. 2, pp. 219–269, 1995.

[32] F. Bedeschi *et al.*, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 217–227, Jan. 2009.
[33] H. Y. Lee *et al.*, "Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO2 based RRAM," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2008, pp. 1–4.
[34] S. Kannan, J. Rajendran, R. Karri, and O. Sinanoglu, "Sneak-path testing of crossbar-based nonvolatile random access memories," *IEEE Trans. Nanotechnol.*, vol. 12, no. 3, pp. 413–426, May 2013.
[35] International Roadmap Committee, *International Technology Roadmap for Semiconductors: 2013 Edition*, Semicond. Ind. Assoc., San Francisco, CA, USA, 2013. [Online]. Available: http://www.itrs.net/ITRS%201999-2014%20Mtgs,%20Presentations%20&%20Links/2013ITRS/Summary2013.htm
[36] K. Gulati and H.-S. Lee, "A high-swing CMOS telescopic operational amplifier," *IEEE J. Solid-State Circuits*, vol. 33, no. 12, pp. 2010–2019, Dec. 1998.
[37] L. Kull *et al.*, "A 3.1mW 8b 1.2GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32nm digital SOI CMOS," in *ISSCC Dig. Tech. Papers*, San Francisco, CA, USA, 2013, pp. 468–469.
[38] W.-T. Lin and T.-H. Kuo, "A 12b 1.6GS/s 40mW DAC in 40nm CMOS with >70dB SFDR over entire Nyquist bandwidth," in *ISSCC Dig. Tech. Papers*, San Francisco, CA, USA, 2013, pp. 474–475.
[39] J. Mutch and D. G. Lowe, "Object class recognition and localization using sparse features with limited receptive fields," *Int. J. Comput. Vis.*, vol. 80, no. 1, pp. 45–57, Oct. 2008.
[40] A. A. Maashri *et al.*, "Accelerating neuromorphic vision algorithms for recognition," in *Proc. 49th Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2012, pp. 579–584.
[41] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comp. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.
[42] C. Walczyk *et al.*, "Impact of temperature on the resistive switching behavior of embedded $HfO_2$-based RRAM devices," *IEEE Trans. Electron Devices*, vol. 58, no. 9, pp. 3124–3131, Sep. 2011.



**Boxun Li** (S'13) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2009, where he is currently pursuing the M.S. degree with the Department of Electronic Engineering.

His current research interest include energy efficient hardware computing system design, and parallel computing based on GPU.

**Peng Gu** (S'14) received the B.S. degree in electronic engineering from the NICS Group, Tsinghua University, Beijing, China, in 2015. He is currently pursuing the Ph.D. degree with the SEALab, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA.

His current research interests include low power system design and hardware acceleration and computing with emerging devices. He has authored and co-authored several papers in DAC, Asia and South Pacific Design Automation Conference (ASPDAC), and Great Lakes Symposium on Very-large-Scale Integration (GLSVLSI).

**Yi Shan** received the B.S. degree and the Ph.D. degree in electronics engineering with the Nanoscale Integrated Circuits and Systems Lab group from Tsinghua University, Beijing, China, in 2008 and 2014, respectively.

He is currently a Senior Research and Development Engineer with the Institute for Deep Learning, Baidu Inc., Beijing. His current research interest include heterogeneous parallel/distributed computing based on GPU cluster for deep learning applications and hardware computing on field-programmable gate array (FPGA) for other applications, such as stereo vision, search engine, and brain network analysis.

**Yu Wang** (S'05–M'07–SM'14) received the B.S. and Ph.D. (Hons.) degrees in electronic engineering from Tsinghua University, Beijing, China, in 2002 and 2007, respectively.

He is currently an Associate Professor with the Department of Electronic Engineering, Tsinghua University. His current research interests include parallel circuit analysis, application specific hardware computing (especially on the brain related problems), and power/reliability aware system design methodology. He has authored and coauthored over 130 papers in refereed journals and conferences.

Dr. Wang was a recipient of the IBM X10 Faculty Award in 2010, the Best Paper Award in IEEE Computer Society Annual Symposium on Very-large-Scale Integration (ISVLSI) 2012, the Best Poster Award in HEART 2012, and six best paper nominations in ASPDAC/International Conference on Hardware/Software Codesign and System Synthesis/International Symposium on Low Power Electronics and Design (ISLPED). He serves as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and the *Journal of Circuits, Systems, and Computers*. He is the TPC Co-Chair of the International Conference on Field Programmable Technology (ICFPT) 2011 and the Finance Chair of the ISLPED 2012–2015. He serves as a TPC member in several important conferences, such as DAC, FPGA, Design, Automation and Test in Europe (DATE), ASPDAC, ISLPED, International Symposium on Quality Electronic Design (ISQED), ICFPT, and ISVLSI.

**Yiran Chen** (M'05) received the B.S. (Hons.) and M.S. (Hons.) degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2005.

After five years in industry, he joined the University of Pittsburgh, Pittsburgh, PA, USA, in 2010, as an Assistant Professor and was promoted to Associate Professor with the Department Electrical Communication Engineering, in 2014. He has published one book, several book chapters, and over 200 technical publications. He holds 86 U.S. and international patents with 15 pending applications.

Dr. Chen was a recipient of three best paper awards from the ISQED 2008, the ISLPED 2010, and the GLSVLSI 2013, and several other nominations from the DAC, the DATE, and the ASPDAC, the National Science Foundation CAREER Award in 2013, and the ACM SIGDA Outstanding Young Faculty Award in 2014. He was an Invitee of 2013 U.S. Frontiers of the Engineering Symposium of National Academy of Engineering. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the *ACM Journal on Emerging Technologies in Computing Systems*, ACM Special Interest Group on Design Automation, and E-News. He has served on the technical and organization committees of about 40 conferences.

**Huazhong Yang** (M'97–SM'00) was born in Ziyang, Sichuan Province, China, in 1967. He received the B.S. degree in microelectronics and the M.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993, and 1998, respectively.

In 1993, he was with the Department of Electronic Engineering, Tsinghua University, where he is currently a Specially Appointed Professor of the Cheung Kong Scholars Program. His current research interests include wireless sensor networks, data converters, parallel circuit simulation algorithms, nonvolatile processors, and energy-harvesting circuits. He has authored and co-authored over 300 technical papers and holds 70 granted patents.