ICE: Inline Calibration for Memristor Crossbar-based Computing Engine

Boxun Li[1], Yu Wang[1], Yiran Chen[2], Hai (Helen) Li[2], Huazhong Yang[1]
[1]Dept. of E.E., Tsinghua National Laboratory for Information Science and Technology,
Tsinghua University, Beijing, China
[2]Dept. of E.C.E., University of Pittsburgh, Pittsburgh, USA
[1] Email: yu-wang@mail.tsinghua.edu.cn

***Abstract*—The emerging neuromorphic computation provides a revolutionary solution to the alternative computing architecture and effectively extends Moore's Law. The discovery of the memristor presents a promising hardware realization of neuromorphic systems with incredible power efficiency, allowing efficiently executing the analog matrix-vector multiplication on the memristor crossbar architecture. However, during computations, the memristor will slowly drift from its initial programmed state, leading to a gradual decline of the computation precision of memristor crossbar-based computing engine (MCE). In this paper, we propose an inline calibration mechanism to guarantee the computation quality of the MCE. The inline calibration mechanism collects the MCE's computation error through 'interrupt-and-benchmark (I&B)' operations and predicts the best calibration time through polynomial fitting of the computation error data. We also develop an adaptive technique to adjust the time interval between two neighbor I&B operations and minimize the negative impact of the I&B operation on system performance. The experiment results demonstrate that the proposed inline calibration mechanism achieves a calibration efficiency of 91.18% on average and negligible performance overhead (i.e., 0.439%).**

I. Introduction

Power efficiency has become one of the most crucial considerations in computing system design [1]. However, the imminent barrier to Moore's Law in CMOS technology, so-called "power wall", requires a revolutionary architecture to satisfy the continuously growing performance and power efficiency gains [2]. Fortunately, the latest innovations of emerging device technologies provide a promising solution to this problem [3].

Memristor is one of such promising devices that can advance Moore's Law beyond the present silicon roadmap horizons [3]. First, its ultra-high integration density enables a large number of signal connections within a small circuit size [4]. Moreover, the crossbar structure based on the variable resistance states of the memristor provides an incredible execution efficiency of the matrix-vector multiplication, which is one of the most significant operations of artificial neural networks [5]. And the memristor crossbar-based computing engine **(MCE)** can be used to realize a low power approximate computing system with ≥ 400 GFLOPS/W [6].

In memristor technology, the current through a device in history is reflected by the changing of the state of memristor. Such a property leads to the state drifting in a slow rate when using the memristor for computation. For example, a voltage of 0.1V may cause the memristor described in Ref. [7] to deviate $\sim 2\%$ from its initial state in 1 second. As the function of the MCE severely relies on the states of memristors, the computation precision of the computing engine will decrease gradually with the operation time. Therefore, an inline calibration mechanism is necessary to guarantee the computation quality of the MCE.

However, it's very difficult to achieve the best calibration time: even for a single MCE, the time that the computation precision decays to a certain value varies dramatically from time to time. Furthermore, as the speed of tuning the memristor to the specific state could be $\sim 100\times$ slower than the working speed of MCE [6], [8], a frequent calibration will significantly decrease the performance of the computing engine. Therefore, there urges an efficient mechanism to extend the continuous operation time of the MCE before calibration as much as possible.

978-3-9815370-2-4/DATE14/©2014 EDAA

In this paper, we for the first time propose an inline calibration mechanism which periodically 'interrupt-and-benchmark **(I&B)**' the MCE and then predict the best calibration time through polynomial fitting of the benchmark results. The contributions of this paper include:

1) We propose an inline calibration mechanism which collects the MCE's computation error through I&B operations and predicts the best calibration time through polynomial fitting of the computation error data. We also propose an adaptive technique to adjust the time interval between two neighbor I&B operations in order to minimize the negative impact of the I&B operation on system performance.
2) We test the proposed inline calibration mechanism on 4 different MCEs. Experiment results show that the proposed mechanism achieves a calibration efficiency (the proximity of the practical operation time to its ideal value) of 91.18% on average, improving 21.77% compared to the one with a constant calibration period. In addition, the extra performance overhead of the proposed calibration mechanism is only 0.439%.

II. Preliminaries

A. Memristor

Fig. 1(a) shows the physical model of the HP memristor [7]. The model includes a a two-layer thin film of TiO_2. These two layers demonstrate different conductivity and the overall resistance depends on the sum of the two layers. The boundary between the two layers will move when a current passing through the memristor [7]. Therefore, the memristor may deviate slightly from its initial programmed state when using. The memristor also has other attractive features, such as the 'pinched hysteresis loop'. **In this paper, we mainly take advantage of the variable resistance states of the memristor** and all the memristor model and parameters used are taken from Ref. [7].

B. Memristor Crossbar-based Computing Engine

The memristor crossbar architecture can be used to implement the artificial neural network algorithm, which is able to perform intelligent data processing in many domains [6], [5]. An artificial neural network can be expressed by stacking the following basic operation layer by layer [9]:

$$Y_m^i = sigmoid(W_{m\times n}^i \times Y_n^{i-1} + b_m^i) \quad (1)$$

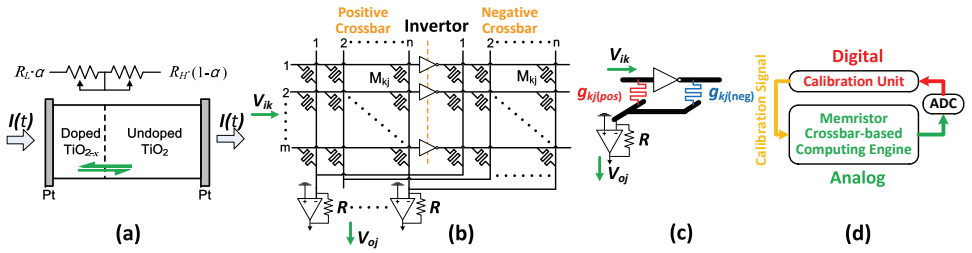

Fig. 1. (a). Physical model of the HP memristor; (b)&(c). Memristor Crossbar-based Computing Engine; (d). Inline Calibration Architecture;

where Y_n^{i-1} is the output of *Layer* $i-1$ and also the input of *Layer* i. $W_{m\times n}^i$ is the weight matrix between *Layer* $i-1$ and *Layer* i. b_m^i is the bias of the layer. The sigmoid function is $f(x)=\frac{1}{1+e^{-x}}$.

It can be observed that the most significant operation is the matrix-vector multiplication, which can be efficiently implemented through the memristor crossbar architecture as shown in Fig. 1(b) [5]. And the sigmoid function can be realized through the circuit described in [10]. Finally, by combining several such architectures layer by layer, we can realize a powerful MCE to accomplish different tasks.

III. Calibration Problem and Motivating Example

As discussed in Section II-A, the memristor will slowly drift from its initial programmed state during computations. As the function of the MCE severely relies on the memristor states, an inline calibration mechanism is necessary to guarantee the computation precision of the MCE. However, there's a calibration problem as follows:

$$T_{total} = n \cdot T_R + m \cdot T_C \quad (T_R \ll T_C) \tag{2}$$

where T_{total} is the total time that the MCE works. T_R represents the time consumption for the computing engine to complete a single regular operation and n represents the number of regular operations that the computing engine completes in total. **The term 'regular operation' is used to represent the operation of processing the customer's input data and distinguish the 'Interrupt-and-Benchmark (I&B)' operation, which interrupts the regular operation, processes the benchmark data, and then works out the computation error of the computing engine.** T_C is the time consumption of calibrating the states of memristors in a computing engine and m is the number of total calibrations.

Because it usually costs much more time to tune the memristor to the specific state ($\sim 5\mu s$ in [8]) than to use the MCE to complete a single regular operation ($\leq 50ns$ in [6]) a frequent calibration will significantly decrease the computing efficiency of the computing engine. The computing efficiency (η) of a MCE can be defined as follows:

$$\eta = \frac{n \cdot T_R}{T_{total}} = \frac{T_R}{T_R + \frac{m}{n} \cdot T_C} \tag{3}$$

The calibration problem can be described as minimizing the extra calibration factor ($\frac{m}{n} \cdot T_C$) to maximizing the computing efficiency η. Assuming that the computing engine completes n_i regular operations between Calibration $i-1$ and Calibration i, and the MCE must be calibrated when its computation error reaches a specific value, there exists a maximum number of regular operations ($\sup n_R$) for each n_R. Therefore, in order to maximize η, we should try to make n_i approach $\sup n_R$ as much as possible. We define a term **'calibration efficiency'** (γ) to evaluate the approximation degree of n_R to $\sup n_R$ as follows:

$$\gamma = \frac{n_R}{\sup n_R} \tag{4}$$

Therefore, the calibration problem can be transformed into making the number of regular operations n_R approach its maximum value $\sup n_R$ between any two neighbor calibrations to maximize the calibration efficiency γ. However, $\sup n_R$ varies a lot, and therefore, it's usually very difficult to predict the best time for calibration. For example, Fig. 2 illustrates the simulation results of a MCE used to complete recognition of handwritten digits [11] in Table I. The operation frequency is 20MHz and the sequence of the input data is randomly selected from the dataset. The noise rate is set to 5%. It can be seen that the maximum calibration time ($\sup n_R$) varies a lot: the time that the recognition accuracy drops to 95% for the first time is 29.2s, 35.5s, 41.2s, and 43.3s, respectively. The minimum result is only 67.43% of the maximum one. **As a result, the calibration period should be dynamic, instead of constant.**

IV. Inline Calibration Mechanism

We propose an inline calibration mechanism to predict the best calibration time of a MCE and achieve the best calibration efficiency. The mechanism is based on an inline calibration unit attached to the original MCE as shown in Fig. 1(d). The calibration unit will periodically perform I&B operation on the computing engine and predict the best calibration time through polynomial fitting of the computation error data.

The algorithm of the proposed inline calibration mechanism is illustrated in Algorithm 1. There are two major techniques of the proposed inline calibration: calibration time prediction technique by polynomial fitting and adaptive technique of adjusting the time interval between two neighbor I&B operations.

A. Calibration Time Prediction Technique by Polynomial Fitting

The calibration mechanism will first collect several practical computation error of the MCE through the I&B operation. All the practical computation error achieved will be stored in $\delta_{history}$ array and all the time points of the I&B operation will be also stored in $T_{history}$ array for polynomial fitting. (Line 11 $\sim$ 24 in Algorithm 1)

After the length of these two arrays reaches S, the required number of data for fitting, the calibration mechanism will begin to fit the collected data to an N^{th}-degree polynomial and work out a function $f(T,\delta)$ to represent the approximate relationship between the computation data and time. Polynomial fitting is one of the most common mechanisms to approximate an unknown

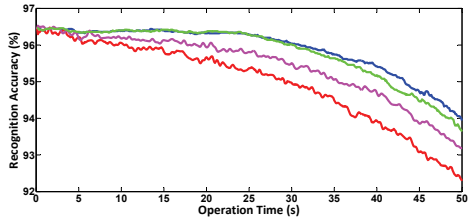

Fig. 2. The drift of the memristor state causes a decline of the MCE's performance (recognition accuracy in this simulation) over time. And the time that the accuracy decreases to a certain value varies a lot in 4 simulations.

Algorithm 1: Calibration Algorithm

Input: M, $\{B_T\}$, T_{start},T_{min}, $\sup\delta$, ϵ, S, D_{max}, N, $Type$
Output: $T_{calibration}$

1 $T_{interrupt} \leftarrow T_{start}$;
2 $T_{history} \leftarrow []$;
3 $\delta_{history} \leftarrow []$;
4 $\delta_{predict} \leftarrow 0$;
5 $i \leftarrow 0$; // Number of Interrupt-and-Benchmark Operations;
6 $t \leftarrow 0$; // Number of Regular Operations;
7 $D \leftarrow 0$; // Times of Doubling T_{min};
8 $T_{calibration} \leftarrow Inf$;
9 **while** $t < T_{calibration}$ **do**
10 **if** $t == T_{interrupt}$ **then**
11 **// Interrupt and Benchmark;**
12 Interrupt the regular operation of M;
13 Use $\{B_T\}$ to compute the practical computation error δ;
14 $i++$;
15 $T_{history}(i) \leftarrow t$;
16 $\delta_{history}(i) \leftarrow \delta$;
17 **if** $\delta > \sup\delta$ **then**
18 $display$('Calibration Method Fail!');
19 break;
20 **end**
21 **// Adaptively Adjust the Time Interval;**
22 **if** $i < S$ **then**
23 $T_{interrupt} \leftarrow T_{interrupt} + T_{min}$;
24 **end**
25 **else**
26 **if** $|\delta - \delta_{predict}| < \epsilon$ **then**
27 **if** $D < D_{max}$ **then**
28 $D++$;
29 **end**
30 **end**
31 **else**
32 $D \leftarrow 0$;
33 **end**
34 $T_{interrupt} \leftarrow T_{interrupt} + T_{min} \cdot 2^D$;
35 **// Polynomial Fitting;**
36 Use an N^{th}-degree polynomial fitting to obtain the function $f(T, \delta)$ of the last S data from $T_{history}$ and $\delta_{history}$;
37 $\delta_{predict} = f(T = T_{interrupt}, \delta)$;
38 **// Predict Best Calibration Time;**
39 **if** $\delta_{predict} \geq \sup\delta$ **then**
40 Solve $\sup\delta = f(T, \delta)$ and the solutions are $T_{roots}[N]$;
41 **// Choose the best solution;**
42 **for** $k = 1 \rightarrow N$ **do**
43 **if** $T_{roots}(k) \in \Re$ && $T_{roots}(k) > t$ && $T_{roots}(k) < T_{calibration}$ **then**
44 $T_{calibration} \leftarrow T_{roots}(k)$;
45 **end**
46 **end**
47 **end**
48 **end**
49 **end**
50 Perform a regular operation;
51 $t++$;
52 **end**
53 Calibrate M;

function and has been widely used in circuit analysis, test, and many other domains [12], [13]. The calibration mechanism will use each computed function $f(T, \delta)$ to predict the the computation error $\delta_{predict}$ at the next I&B operation point $T_{interrupt}$. If the predicted computation error is less than the maximum tolerable computation error ($\delta_{predict} < \sup\delta$), the calibration mechanism will assume that there's no need for calibration. (Line $35 \sim 37$ in Algorithm 1)

However, once the predicted computation error is greater than the maximum tolerable computation error ($\delta_{predict} \geq \sup\delta$), the calibration mechanism will assume that it's necessary to calibrate the MCE before the next time of I&B operation and start to predict the best calibration time by solving the equation $\sup\delta = f(T, \delta)$. Because the polynomial function is continuous and $\delta < \sup\delta < \delta_{predict}$, there must exists at least one real solution in the range from the current $T_{interrupt}$ to the next $T_{interrupt}$. The calibration mechanism will set the final predicted calibration time $T_{calibration}$ to the minimum T_{roots} of the solutions that is greater than the current operation time t. (Line $38 \sim 47$ in Algorithm 1)

One thing need to be mentioned is that the computation error of a MCE demonstrates Markov property, as the new generated deviation only accumulate to the current state of the memristor. Therefore, the proposed calibration algorithm only use the last S data form $\delta_{history}$ and $T_{history}$ for polynomial fitting. This technique can help reduce the disturbance of the past computation error and prevent overfitting.

B. Adaptive Technique of Adjusting the Time Interval between Two neighbor I&B Operations

In order to achieve the relationship between the computation error and time, we must perform the I&B operation to collect enough data for polynomial fitting. However, the I&B operation will reduce the efficient times of regular operations. Therefore, the I&B operation will have a negative impact on the performance of the MCE and the calibration efficiency (γ) should be modified as follows:

$$\gamma = \frac{n_R - k \cdot n_{I\&B}}{\sup n_R} \tag{5}$$

where $n_{I\&B}$ is a constant number representing the regular operation times cost for calculating benchmarks by each I&B operation and k is the total times of I&B operations before calibration. Therefore, we should also minimize k, the number of I&B operations between two neighbor calibrations, to achieve the best calibration efficiency.

As shown in Fig. 2, the MCE is able to work steadily for a constant period of time. Therefore, the calibration mechanism will first make the MCE work continuously for a certain constant period of time (T_{start}) without any I&B operation to reduce the unnecessary I&B operations. (Line 1 in Algorithm 1)

Then the I&B operation will begin. In order to minimize the number of I&B operations, the calibration mechanism will also use the achieved function $f(T, \delta)$ to help adjust the time interval between two neighbor I&B operations: The calibration mechanism will predict the computation error $\delta_{predict}$ at the next point of I&B point ($T_{interrupt}$) through $f(T, \delta)$. Then at the next time of I&B operation, the calibration mechanism will compare the practical computation error δ and the predicted computation error $\delta_{predict}$. If the absolute difference between δ and $\delta_{predict}$ is less than a certain value (ϵ), the calibration mechanism will assume that the function is accurate and there's no need to perform the polynomial fitting frequently. Therefore, the time interval between the two neighbor I&B operations will be doubled. Otherwise, the calibration mechanism will set the time interval back to T_{min}. Moreover, there's a limit of the times of doubling the time interval (D_{max}) in case that the time interval becomes too large to capture a sudden spurt of the computation error. (Line $26 \sim 34$ in Algorithm 1)

Finally, at the ideal situation, the proposed adaptive technique of adjusting the time interval between two neighbor I&B operations can reduce the number of I&B operations from $O(N)$ to $O(\log_2 N)$.

V. Experiment Results

A. Experiment Setup

We use the 4 benchmarks in Table I to test the proposed inline calibration mechanism. The working frequency of the MCE is 20MHz and we test each MCE's accurate performance every 0.01s with 5,000 examples. We choose 50 (1%) from the 5,000 examples to form the small benchmark set in the calibration mechanism. We keep randomly choosing examples and comparing the computation

TABLE I
SELECTED MEMRISTOR CROSSBAR-BASED COMPUTING ENGINES

Name	Target Function	Type	Size	MSE
HMAX	Distance Calculation	Object Detection	$4 \times 32 \times 1$	0.002
KMeans	Centroid Calculation	Clustering	$8 \times 64 \times 1$	0.018
Sobel	Sobel Gradient	Image Processing	$25 \times 100 \times 25$	0.019
MNIST	-	Pattern Recognition	$784 \times 300 \times 10$	0.0063

error of the two dataset until the absolute difference is less than 1% of the MSE of the total examples.

The minimum time interval between two neighbor I&B operations (T_{min}) is set to 0.01s. The maximum tolerable computation error ($\sup \delta$) is set to $10\times$ of the initial MSE in Table I (the MNIST benchmark is set to 0.01). The maximum tolerable absolute difference (ϵ) between $\delta_{predict}$ and δ is set to 5% of the corresponding $\sup \delta$. All the types of computation error in the simulation are mean square error (MSE). The required number of data for polynomial fitting (S) is set to 9 ("Poly2") and 10 ("Poly3"). The maximum times of doubling the time interval (D_{max}) is set to 8. The start time of the I&B operation (T_{start}) is set T_{start} to 70% of the minimum time that δ reaches $\sup \delta$ in 10 different simulations. And the constant calibration period is set to 90% of the minimum time that δ reaches $\sup \delta$. The noise rate is set to 5%. Each computing engine is tested 5 times with the same initial conditions.

B. Experiment Results

The statistical results are illustrated in Table II, where "Poly2" represents the calibration mechanism based on 2^{nd}-degree polynomial fitting and "Poly3" represents the calibration mechanism based on 3^{rd}-degree polynomial fitting. "Constant" stands for the calibration mechanism which uses a constant calibration period.

In order to quantify the negative impact of the I&B operation, we define a parameter Δ to represent the extra performance overhead of I&B operations:

$$\Delta = \frac{k \cdot n_{I\&B}}{n_R} \tag{6}$$

where $k \cdot n_{I\&B}$ is the extra times of I&B operations and n_R represents the total times that the MCE performs regular operations.

It can be seen that the calibration mechanism based on 2^{nd}-degree polynomial fitting performs better than other calibration mechanisms and realizes the a calibration efficiency (γ) of 91.18% on average, improving 21.77% compared to the one with a constant calibration period. In addition, an extra performance overhead (Δ) of 0.439% demonstrates that the proposed calibration mechanism has negligible impact on the performance of the MCE.

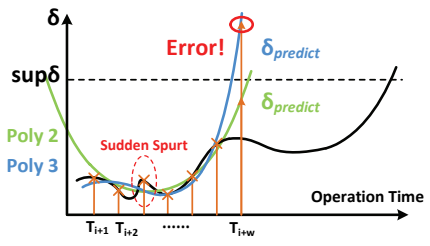

Fig. 3. This figure illustrates the reason that "Poly3" doesn't perform better than "Poly2". When a sudden spurt of the computation appears, Poly3 will work out a 3^{rd}-degree polynomial fitting with an aggressive growth rate and leads to a smaller prediction of $T_{calibration}$.

TABLE II
SIMULATION RESULTS OF THE PROPOSED CALIBRATION MECHANISM

Name	Term	HMAX	KMeans	Sobel	MNIST	Average
Uniform	γ[1]	79.35	77.82	70.67	71.67	74.88
Poly 2	γ[1]	92.84	93.89	89.06	89.42	91.18
	$+$[2]	17.00	20.01	26.04	24.76	21.77
	Δ[3]	0.299	0.147	0.372	0.937	0.439
Poly 3	γ[1]	89.98	69.17	81.17	74.17	78.62
	$+$[2]	13.40	-11.11	14.86	3.49	5.01
	Δ[3]	0.325	0.186	0.396	1.068	0.402

[1] γ: Calibration Efficiency (%);
[2] $+$: Improvement (%);
[3] Δ: Extra Performance Overhead (%);

VI. CONCLUSIONS

In this work, we propose an efficient inline calibration mechanism for MCE to eliminate the negative impact of the memristor state drifting on system performance. We also provide an adaptive technique to reduce the extra performance overhead from $O(n)$ to $O(\log_2 n)$. The experiment results on 4 different benchmarks show that the proposed inline calibration mechanism is able to achieve a calibration efficiency of 91.18% as well as a negligible extra performance overhead. There is an ongoing debate over which spice level model should be universally adopted for the memristor-based design. For future work, we will try to select and integrate proper spice level memristor models to analyze the hardware and power overhead of the proposed approach.

ACKNOWLEDGMENT

This work was supported by National Natural Science Foundation of China (No. 61373026, No. 61261160501, No. 61028006), 973 project 2013CB329000, National Science and Technology Major Project (2013ZX03003013-003), Youth Talent Development Plan of Beijing (YETP0099) and NSF CAREER CNS-1253424.

REFERENCES

[1] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Micro*, 2012.
[2] Sung Mo Steve Kang and Sangho Shin. Energy-efficient memristive analog and digital electronics. In *Advances in Neuromorphic Memristor Science and Applications*, pages 181–209. Springer, 2012.
[3] Beiye Liu, Miao Hu, Hai Li, Zhi-Hong Mao, et al. Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine. In *DAC*, 2013.
[4] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301, 2010.
[5] Miao Hu, Hai Li, Qing Wu, and Garrett S Rose. Hardware realization of bsb recall function using memristor crossbar arrays. In *Proceedings of the 49th Annual Design Automation Conference*, 2012.
[6] Boxun Li, Yi Shan, Miao Hu, Yu Wang, Yiran Chen, and Huazhong Yang. Memristor-based approximated computation. In *Proceedings of ISLPED 2013*.
[7] Robinson E Pino et al. Statistical memristor modeling and case study in neuromorphic computing. In *DAC, 2012*.
[8] Wei Yi, Frederick Perner, et al. Feedback write scheme for memristive switching devices. *Applied Physics A*, 102:973–982, 2011.
[9] Simon S Haykin. *Neural networks and learning machines*, volume 3. Prentice Hall New York, 2009.
[10] G. Khodabandehloo et al. Analog implementation of a novel resistive-type sigmoidal neuron. *TVLSI*, 20(4):750 –754, april 2012.
[11] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 1998.
[12] Sehun Kook, Hyun Woo Choi, and A. Chatterjee. Low-resolution dac-driven linearity testing of higher resolution adcs using polynomial fitting measurements. *TVLSI*, 2013.
[13] Lerong Cheng, Jinjun Xiong, and Lei He. Non-gaussian statistical timing analysis using second-order polynomial fitting. *TCAD*, 2009.