ReRAM based Processing-in-memory Architecture for Recurrent Neural Network Acceleration

Yun Long, *Student member, IEEE*, Taesik Na, *Student member, IEEE*, and Saibal Mukhopadhyay, *Senior member, IEEE*

***Abstract*—We present a recurrent neural network (RNN) accelerator design with resistive random-access memory (ReRAM) based processing-in-memory (PIM) architecture. We measure the system throughput and energy efficiency with detailed circuit and device characterization. Re-programmability is enabled with our design and a RNN friendly pipeline is employed to increase the system throughput. We observe that on average the proposed system achieves 79x improvement of computing efficiency compared with GPU baseline. Our simulation also indicates that to maintain high accuracy and computing efficiency, the read noise standard deviation should be less than 0.2, the device resistance should be at least 1 MΩ, and the device write latency should be minimized.**

***Index Terms*—Recurrent neural network (RNN), ReRAM, processing in memory (PIM), long short-term memory (LSTM), gated recurrent unit (GRU), human activity recognition**

I. Introduction

Machine learning has achieved the state-of-the-art performance across a wide range of applications, such as image classification [1], video recognition [2], natural language processing (NLP) [3], and gaming strategies [4], to name a few. Further, deep neural networks (DNNs) can even outperform human-level performance in a few tasks, for example, ImageNet classification [5], and board game Go [4]. Meanwhile, the neural networks complexity and parameter size have skyrocketed over the past few years. Despite the fast advance in general-purpose graphics processing unit (GPGPU), its energy efficiency is still far less than the ultimate '*Intelligence*', human brain, which contains 10^{10} neurons and 10^{14} synapses but only consumes ~20 Watts [6]. One of the bottlenecks comes from the fact that the von Neumann architecture separates the memory and processing unit, introducing significant data movement energy as well as data access latency [7].

Manuscript received September 7th, 2017; revised November 29th, 2017; accepted February 28th, 2018. This material is based on work supported in part by National Science Foundation (#1740197) and Semiconductor Research Corporation.

Yun Long, Taesik Na, and Saibal Mukhopadhyay are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mails: yunlong@gatech.edu; taesik.na@gatech.edu; saibal@ece.gatech.edu).

To address this, emerging non-volatile memory, especially resistive random-access memory (ReRAM) based processing-in-memory (PIM) architecture has been studied to accelerate DNN computation [8-10]. Thanks to the PIM architecture and massive data-level parallelism, significant improvements of computing speed and energy efficiency have been demonstrated [8-11].

Most of the prior ReRAM accelerators focus on accelerating convolutional neural networks (CNNs) but lack supports for recurrent neural networks (RNNs). The core concept of RNN is to utilize the sequential information with a feedback loop, i.e. the output of RNN depends not only on current input but also the previous computation results. In practice, long short-term memory (LSTM) [12] and gated recurrent unit (GRU) [13] are the most popular RNN configurations and have demonstrated unprecedented performance for tasks including Human activity recognition (HAR) [14], video recognition [2], and NLP [3, 15], to name a few.

There are two reasons making existing ReRAM CNN accelerators not suitable for RNN computing. First, in CNN, the inputs are independent from each other and the output is computed based on current input. However, this is not true for RNN since the computation of RNN requires both current input and results from previous steps. The time-dependence makes the pipeline in prior ReRAM CNN accelerator designs not feasible for RNN computing. Second, the operations of RNN is different with CNN especially for LSTM and GRU which contain not only matrix-vector multiplication and non-linear functions but also element-wise multiplication [12, 13, 16].

The acceleration of RNN computation using GPU [15], field-programmable gate array (FPGA) [16, 17], and CMOS application-specific integrated circuit (ASIC) [18] has been explored in prior works. This work presents a ReRAM based RNN accelerator design utilizing PIM architecture. The contributions of this paper are summarized as follows:

- We present a ReRAM based system architecture to accelerate RNN computation. The processing engine (PE) is divided into three subarrays: ReRAM crossbar subarrays for matrix-vector multiplication; special function unit (SFU) subarrays for non-linear function; and multiplier subarrays for element-wise operations.
- A RNN friendly pipeline is employed to increase the system throughput. Dataflow control is discussed for both inference and programming phase. The proposed design

supports various RNN configurations including the basic RNN, LSTM, and GRU.

- We characterize the system throughput, area, and power consumption with detailed circuit and device modeling based on 28nm CMOS technology. We observe that our design achieves 79x computing efficiency improvements compared with GPU baseline on average. We also compare our design with existing machine learning accelerators implemented on various platforms, including ReRAM, FPGA, and ASIC.
- We investigate the computing accuracy considering device variation. The accuracy drop is insignificant when the device noise deviation is less than 0.2. Simulation also indicates that the system performance drops significantly when considering re-programming, indicating the device writing latency must be minimized to achieve high performance.
- We propose that, to maintain high accuracy and computing efficiency, the read noise standard deviation should be less than 0.2, the device resistance should be at least 1 MΩ.

The paper is organized into the following sections. Section II describes the background. Section III presents the system architecture and dataflow control. Section IV discusses the details of implementing the system. Section V presents the results, and section VI concludes the paper.

II. Background

A. Resistive random-access memory (ReRAM)

A ReRAM device consists of three layers. A resistive switching layer (e.g. HfO_x, NiO, TiO_2, Al_2O_3 or their combination) sandwiched between top and bottom electrodes (e.g. Pt, TiN) [19]. Device resistance can be modulated via applying *set* or *reset* voltage. A lot of works have been done to explore physical resistive switching mechanism of ReRAM [20-22]. In this work, we employ the Ion transportation recombination (ITR) model [20]. In ITR model, the complex ion transportation and recombination process is simplified into the growth and rupture of Conductive filament (CF) in the resistive layer. Resistance of device is controlled by the size of the gap (X in Fig. 1(a)) between the electrode and the tip of the CF.

A simplified physical mechanism model, based on the ITR theory for bipolar ReRAM, is adapted in this paper from prior works [20, 23, 24]. We fit this model with existing experimental data [19, 23], shown in Fig. 1(b-d). As this model captures the dynamic behavior of the ReRAM switching, it helps to analyze the write latency and energy. Should note that we can use the same model (with different parameters) to fit experiment data from different devices. As illustrated in Fig. 1(b-d), device in [23] ($TiN/TiO_x/HfO_x/TiO_x/HfO_x/Pt$ structure) shows very different on/off resistance compared with [19] ($TiN/HfO_x/Pt$ structure). With parameter tuning, our model can fit the experiment data from both works very well.

B. ReRAM PIM architecture and the accelerator design

It has been demonstrated that some emerging technology based non-volatile memory such as ReRAM, Phase change

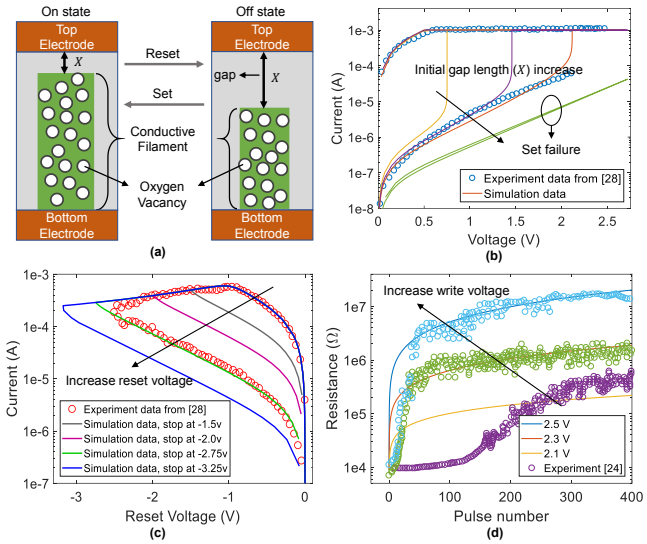


Fig. 1. (a) Simplified ITR resistive switching model. (b) Device SET operations with different initial gap length. (c) Device RESET operations with applied voltage. (d) Resistance modulation with RESET pulses. The pulse width is 50ns. Experiment data are from [24, 28].

memory (PCM), and spin-transfer torque magnetic RAM (STT-MRAM) can perform logic operation beyond storage [8, 9, 25, 26]. This unique feature allows them to serve for both computation and memory, enabling the PIM architecture. A commonly used PIM architecture is shown in Fig. 2(a), a memory bank is divided into three segments: *memory subarrays* which serves as conventional memory; *buffer subarrays* which serves as data buffer; and *processing subarrays* which are utilized to process the data. Private data ports are introduced between buffer subarrays and processing subarrays for high data bandwidth. Further, this private data bus is isolated from main memory bus, hence, it doesn't consume the bandwidth of main memory [8]. As an energy-efficient technique, PIM best exploits the data locality by reconfiguring part of the memory blocks as processing units. Moreover, data

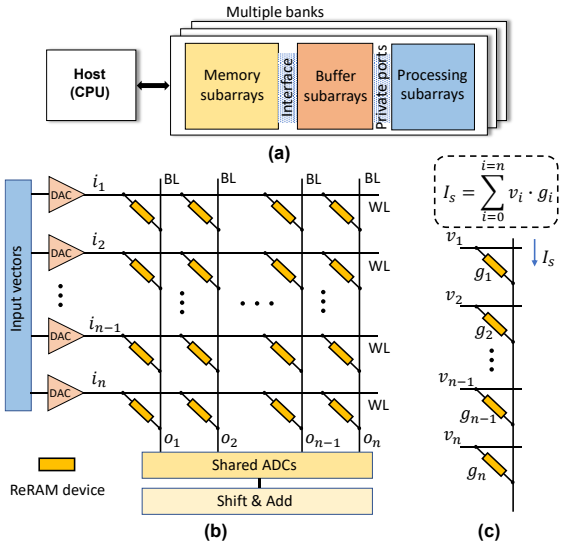


Fig 2. (a) PIM architecture. (b) ReRAM crossbar for matrix-vector multiplication. (c) Current is summed at bitline.

can be directly accessed and transmitted internally without any external memory interfaces. PIM provides large memory bandwidth and fast data access, also reduces the data moving energy, which are very critical to data demanding applications such as machine learning. Among all the candidates, ReRAM based PIM architecture attracts lots of research attentions since that ReRAM has fast writing speed, CMOS compatibility, high on-off ratio and analog feature, i.e. multi-level cell (MLC) [19, 27]. Recent works have demonstrated that ReRAM based PIM architectures achieve orders of magnitude speed and energy efficiency improvements compared with the GPUs and ASIC designs on CNN acceleration [8, 9, 11].

As shown in Fig. 2(b), the core component of ReRAM based accelerator is a crossbar-based processing engine (PE) for matrix-vector multiplication. During computation, neural network's parameters are first programmed into the ReRAM devices. Then input vectors are fed into the crossbar array as wordline (WL) supply voltage. As illustrated in Fig. 2(c), based on Kirchhoff's law, the current summed at each bitline (BL) results the matrix-vector multiplication. Since the computing inside the crossbar array is in analog fashion, DAC and ADC are required for the WL/BL peripherals. Further, due to limited storage capacity of single ReRAM device, multiple cells connecting to the same WL are used to represent one parameter value. For example, 8 ReRAM devices are utilized to represent one 16-bit number with each cell stores 2 bits. Shift & add blocks are used to sum up the computation result from different columns. Further, it is not practical to implement a 16-bit DAC for each row of ReRAM crossbar. Therefore, the input number is divided into several segments and sequentially fed into the network. For instance, 16 clock cycles are needed for a 16-bit input value with 1-bit (the read voltage is either 0 or 1) per cycle.

Note that one-resistor-one-transistor (1T1R) structure is now widely used for ReRAM crossbar design to increase the selectivity and reduce the leakage current [10, 28]. Fig 2 only shows the ReRAM cell for simplicity.

C. *Prior works on Neural Network using ReRAM*

A lot of works have been done to explore ReRAM based machine learning accelerator design. PRIME [8] proposes a PIM architecture to accelerate CNN computation with a detailed analysis of the data representation scheme. ISAAC [11] presents a full-fledged ReRAM CNN accelerator design. System throughput and power consumption are modeled with comprehensive circuit analyses. PipeLayer [9] further optimizes the data flow and pipeline, adds the feature for on-line training. Besides accelerating CNN computing, ReRAM has been widely explored for neuromorphic computing. Recently, Presioso et al. fabricated a 12x12 ReRAM crossbar prototype with a fully operational neuromorphic network which can successfully classify 3x3 pixel black/white images into 3 categories [29].

D. *Recurrent Neural Networks*

Fig. 3(a) shows the basic RNN structure. RNN takes both the current sample (x_t) and the previous calculated network state

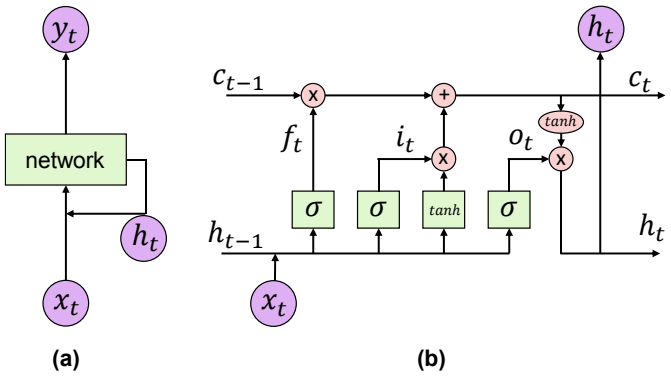


Fig 3. (a) Basic RNN structure. (b) LSTM structure.

(h_t) as the network input. The feedback loop gives RNN the ability to remember and make decision based on previous information. Equations 1-2 shows the computation for the basic RNN cell:

$$h_t = \tanh(x_t U + h_{t-1} W + bias) \quad (1)$$

$$y_t = softmax(h_t V) \quad (2)$$

where x_t, h_t, y_t are the current input, hidden state, and output for current time step, respectively; U, W, V are the matrices which contain the trainable parameters. In theory, RNN is capable of handling long-term dependencies with the feedback loop. However, in practice, it is difficult for RNN to have long-term memory due to the vanishing gradient problem [30].

LSTM are explicitly designed to combat vanishing gradients through a gating mechanism [12]. As shown in Fig. 3(b), a LSTM cell contains three gates: input gate i_t, forget gate f_t, and output gate o_t. They are called gates because the sigmoid function squashes the values of these vectors between 0 and 1, and by multiplying them element-wisely with another vector, we can then define how much information can pass through (i.e. behave like a gate). To be more specific, the forget gate defines how much of the previous state you want to let through; the input gate defines how much of the newly computed state for the current input you want to let through; and the output gate defines how much of the internal state you want to pass through. All the gates have the same dimensions which equal to the size of hidden state. The computation of LSTM is defined in the following equations.

$$i = \sigma(x_t U^i + h_{t-1} W^i + b_i) \quad (3)$$

$$f = \sigma\left(x_t U^f + h_{t-1} W^f + b_f\right) \quad (4)$$

$$o = \sigma(x_t U^o + h_{t-1} W^o + b_o) \quad (5)$$

$$g = tanh\left(x_t U^g + h_{t-1} W^g + b_g\right) \quad (6)$$

$$c_t = c_{t-1} \odot f + g \odot i \quad (7)$$

$$h_t = tanh(c_t) \odot o \quad (8)$$

We observe that there are three main operation types: *matrix-vector multiplication* (e.g. $x_t U^i$ and $h_{t-1} W^i$), *non-linear function* (sigmoid σ and hyperbolic tangent $tanh$), and *element-wise multiplication* (e.g. $g \odot i$).

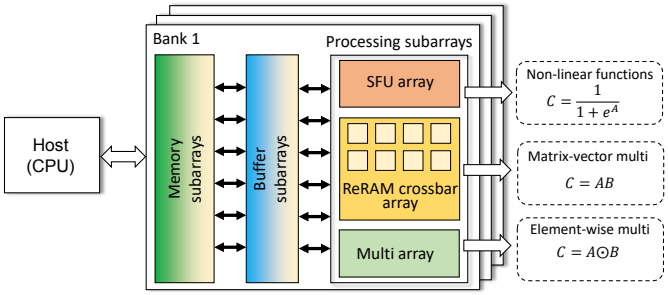


Fig 4. System architecture overview.

A variation of LSTM is called Gated recurrent unit (GRU) [13]. It combines the forget gate and input gate into a single update gate and demonstrates good performance for various applications.

III. System Design

In this section, we first present the system architecture overview. Then we discuss the function of each sub-block in the processing engine. The dataflow and pipeline are then presented.

A. System architecture overview

Fig. 4 presents an overview of the proposed system architecture. Similar with prior PIM architectures, the memory bank is divided into three partitions, including the memory subarrays, buffer subarrays, and processing subarrays. Even though the processing arrays can also be used for data storage [8], we observe that the redesigned array peripherals impose extra data access latency, therefore, the processing subarrays are dedicated for computing in our design.

As mentioned earlier, existing ReRAM PIM architectures are not suitable for accelerating RNN computation. To make it RNN friendly, we further divide the processing subarrays into three segments: ReRAM crossbar array for the matrix-vector multiplication; special function unit (SFU) array for the non-linear functions; and multiplier array handling the element-wise multiplication.

B. Details of system design

Fig. 5 shows a detailed system architecture and the WL/BL peripherals design.

The wordline peripherals support both reading and programming. The switching between reading mode and programming mode is realized by an analog multiplexer (MUX) designed with two transmission gates. The behavior of the analog MUX is controlled by the read enable signal (C_{re}) from dataflow controller. When C_{re} is enabled, the MUX is transparent to read voltage from the read voltage buffer (design with an Op-amp configured in voltage follower mode) and blocks the write voltage. The read voltage is generated from the DAC which transfers the input digital data into analogy voltage. On the other hand, when the C_{re} is low, the MUX is transparent to write input signal and blocks the read voltage input.

Bitline peripherals. The first stage of BL peripherals computation is to sample the current from each column, convert it into voltage and hold it for later analog to digital conversion. This can be accomplished by an Op-amp based sample and hold circuit. Note that the same Op-am in wordline peripherals can be reused here with a capacitor inserted in the feedback loop (Op-amp configured in current integrator mode rather than voltage follower mode [31]). In next stage, ADC is required to convert the analog voltage signal to digital value. One ADC is shared for all the columns in a crossbar because ADC consumes a lot of power and area [11, 32]. A shift & add unit (SAU) is integrated after the ADC to accumulate data from different BLs

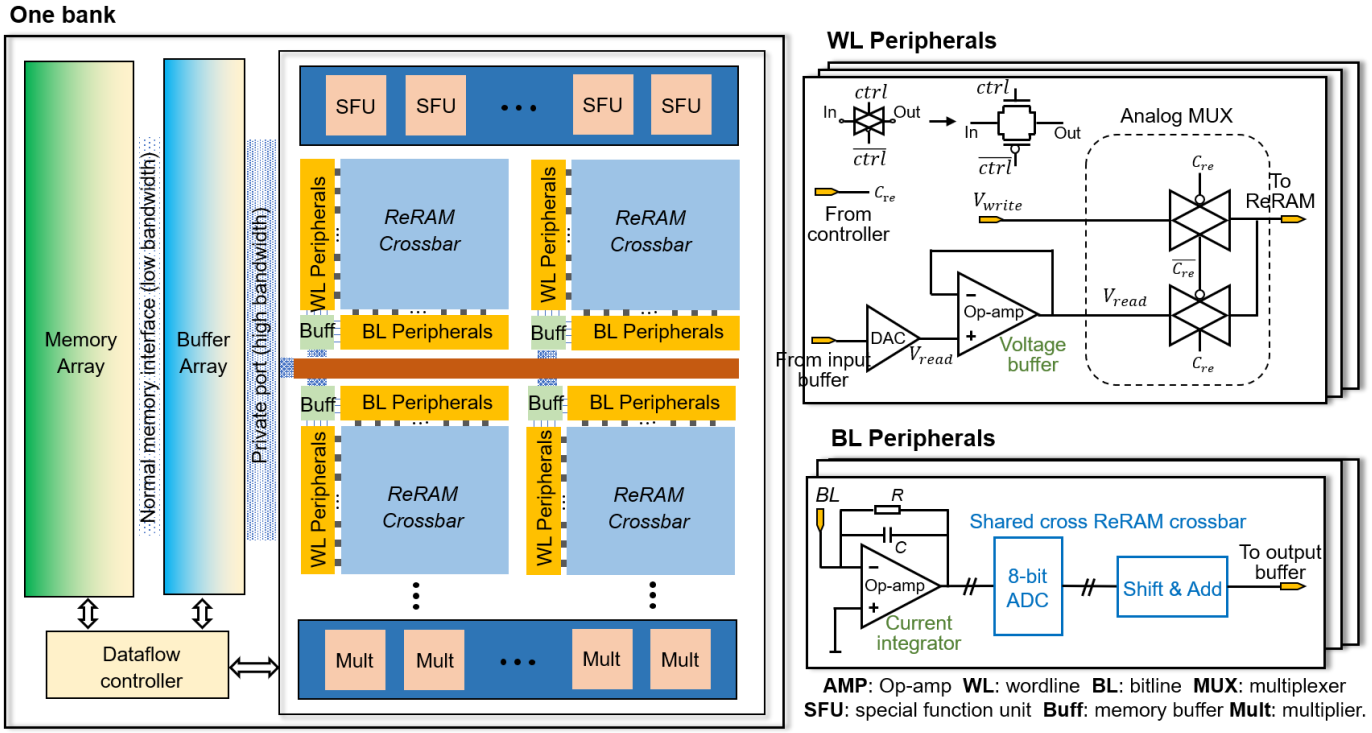


Fig 5. System architecture and the WL/BL peripherals design.

and different cycles. Finally, the digital output will be temporarily stored at the local buffer and then collected back to the buffer subarray.

Special function unit (SFU) and Multiplier. Even though ReRAM crossbar array can efficiently perform computation for matrix-vector multiplication, it is not feasible to solve complex mathematical equations such as sigmoid function (σ), hyperbolic tangent function ($tanh$), and rectified linear unit (ReLU) which are widely used in neural networks serving as activation function. Thus, we implement SFU to handle the computation for complex mathematical functions. Element-wise multiplication is another important operation of LSTM and GRU. We implement an additional multiplier array to handle the element-wise multiplication operation.

Local buffers. During inference, local buffers receive data from buffer subarrays and send the data to WL peripherals. They also collect computing results from BL peripherals and send them back to buffer subarrays. Local buffers are distributed and placed close to crossbar arrays. The addresses for read/write are controlled by dataflow controller and are identical for all local buffers.

Dataflow controller. The dataflow of the proposed system is controlled by the dataflow controller. On the top level, it coordinates the dataflow between memory subarray and buffer subarray, it also manages the data transmission between the buffer subarray and local buffers. During inference, the dataflow controller enables the read enable signal C_{re} for all crossbars. Therefore, the inference input voltages can drive the corresponding WLs while the programming voltage is blocked. On the other hand, during writing, the C_{re} signal is disabled to let the writing voltage get through.

C. *Dataflow and pipeline*

One of the unique features of RNN computation is the data dependency, i.e. the computing in current step relies on the results of previous step. Therefore, the dataflow and pipeline of ReRAM based CNN accelerator is not suitable to accelerate RNN computing. In this section, we first illustrate how to map RNN computing to our design and then propose a RNN friendly pipeline execution framework. We use LSTM as an example. Analysis for the basic RNN and GRU should follow the same procedure.

Before mapping metrices-vector multiplication to ReRAM crossbar array, we need to re-organize the input and parameter matrices. Take equation (3) as an example, two pairs of matrix-vector multiplication are concatenated into one:

$$x_t U^i + h_{t-1} W^i + b_i = [x_t \quad h_{t-1}] \begin{bmatrix} U^i \\ W^i \end{bmatrix} + b_i \qquad (9)$$

Note that adding the bias b_i to the matrix-vector multiplication is trivial since we can use a row of ReRAM cells to store the bias value and force the read voltage to be 1. Fig. 6 (a) shows the matrix-vector operation after concatenation. Here we assume the input vector size is 2 and hidden state vector size is 3. After concatenation, parameter matrix (concatenation of U and W) are programmed into ReRAM cells and input matrix (concatenation of x_t and h_{t-1}) are used as WL input, shown in

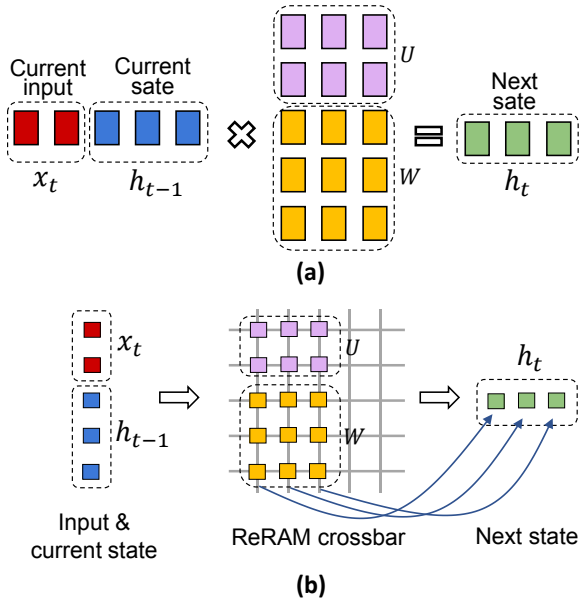


Fig 6. (a) Matrix-vector multiplication after concatenation. (b) Mapping the computing to ReRAM crossbar array.

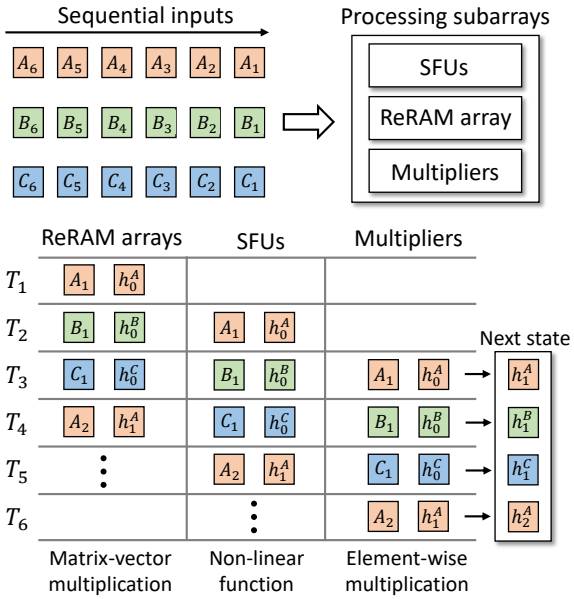


Fig 7. Three stages pipeline for RNN computation.

Fig. 6 (b). For simplicity, in Fig. 6 we assume that each cell can store one parameter value. In practice, due to limited device precision, multiple adjacent cells in the same row are used to store one element and the input signals are also partitioned into several segments and fed into the network sequentially. After computation, results will be first temporary stored in the local buffer and then transmitted back to the buffer subarrays.

The computation for element-wise multiplication and non-linear functions are relatively straightforward. The dataflow controller fetches the data from buffer subarrays to SFU/multiplier array and then collects the results back to buffer arrays.

We develop a RNN friendly pipeline execution flow to increase the system throughput, shown in Fig. 7. Assume we

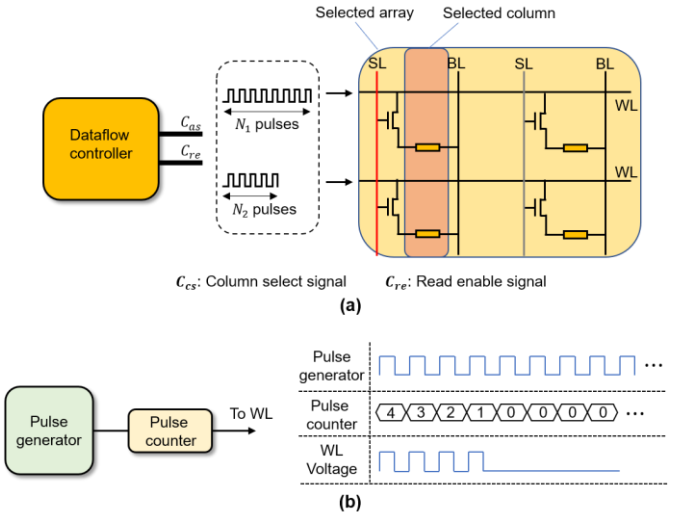


Fig 8. The programming schemes. (a) One column of a crossbar is programmed simultaneously. (b) Pulse series based programming. The conductance change is determined by how many pulses the WL received.

have three input sequences (A, B, and C) which will be processed by the LSTM network. For example, A_1 to A_6 are the first 6 words in a sentence and we want to predict the 7th word based on them. At the first clock cycle T_1, input vector A_1 and the hidden state vector h_0^A calculated from previous stage are fed into the ReRAM crossbar for matrix-vector operations. In next cycle T_2, input vector from another sequence B and the corresponding hidden state h_0^B are fed into the crossbar array; meanwhile, the result of matrix-vector multiplication of A_1 and h_0^A are sent to SFU arrays to perform non-linear function operations. Then, in the third cycle T_3, temporary results for sequences A and B are moved forward to the next step, and sequence C starts the matrix-vector operation. At the end of T_3, the new hidden state h_1^A for sequence A is generated. At the next clock cycle T_4, the second input vector A_2 from sequence A can be fetched to the processing engine since the hidden state h_1^A is ready. With the proposed pipeline, all the processing units in the system are busy, therefore, the maximum throughput can be achieved.

IV. Implementation

In this section, we first present how to enable re-programmability (i.e. device programming), then we discuss the circuit level implementation of the sub-blocks. To get an accurate estimation for power and area, we synthesis the digital design (including dataflow controller, multiplier, and SFU) with Design Compiler and PrimeTime based on 28nm CMOS technology from Synopsys [33]. We also run SPICE simulation for the analog sub-blocks (including Op-amp and analog multiplier) with 32nm PTM model (28nm model is not available in the PTM library) [34].

A. Programmability

Efficient programming is always challenging for ReRAM crossbar array as the write operation normally requires much longer time and consumes more energy than reading [3, 35]. Further, during reading/inference, all the ReRAM devices are involved. Devices connecting to the same WL receive the same input voltage. Therefore, reading can be performed with 1 read clock cycle. However, during writing, each cell needs to be programmed differently and might have diverse values compared with its neighborhood. Thus, multiple clock cycles are required to program an array. Array-level parallel programming scheme has been proposed in prior work [36]. However, the complexity of peripheral circuit design and the ultra-high power density make it prohibitive for large-scale implementation.

In our design, the programming is performed in a column-wise fashion. As shown in Fig. 8, the column selecting signal C_{cs} is used to determine which column to be programmed by turning on the selector transistors of the corresponding column via the select-line (SL). As shown in Fig. 8(a), only the cells in the first column can be programmed while all the other cells are disconnected from the network.

Moreover, ReRAM cells can be programmed into different states, controlled by the number of pulses it receives, as illustrated in Fig. 8(b). The shared pulse generator consistently output identical pulses (same amplitude and length). The pulse counter, initialized with the total number of pulses, counts down once when receiving one pulse. When it counts down to zero, it will block the pulse to be sent to the WL.

With the proposed writing scheme, it takes N writing cycles to program a $N \times N$ ReRAM crossbar array. Note that typically the writing is much slower than reading, therefore, it is highly desired that no re-programming occurs during computing. In section V, we show that the performance will significantly drop when the neural network size is larger than the system capacity (i.e. re-programming is required).

B. Wordline driving ability

A major circuit design challenge of ReRAM accelerator is the WL driving ability issue. It is unrealistic to directly use the output of DAC or other digital circuits to drive the wordline of ReRAM crossbar since they can't provide enough current to drive resistive load. A common approach to increase the driving ability is to insert a buffer stage between the DAC and crossbar array [8, 9, 31]. The buffer can be designed with an operational amplifier (Op-amp) configured at voltage follower mode, as shown in Fig. 9(a). The driving ability is determined by the Op-amp output stage bias current (i.e. transistor size). Increasing the output stage transistors size can reduce the output impedance therefore alleviate the read voltage drop effect. Fig. 9(b) shows the Op-amps power to drive different ReRAM crossbar array size with less than 10% read voltage drop considering various device resistance. We observe that the power consumption of Op-amp increases proportionally with array size and device conductance (i.e. the reverse of device resistance). Therefore, to reduce the power and area overhead of WL buffer, we constrain the ReRAM array size to be 128 x 128. And, in the rest of this paper, we assume the on-state resistance is 1 $M\Omega$ (i.e. LRS is 1 $M\Omega$). We should note that device with high resistance has been reported in prior works with a tradeoff of other device characterizations such as low

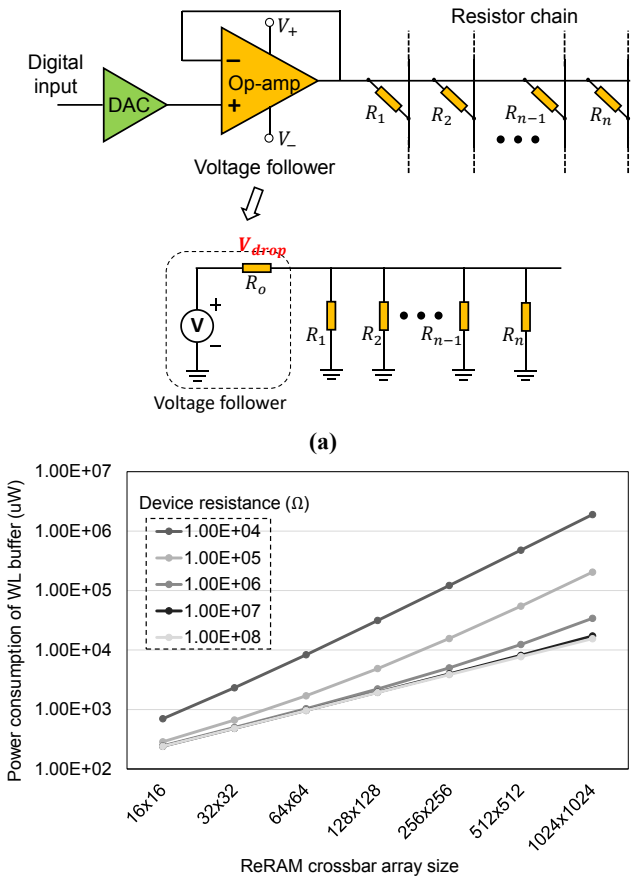


Fig 9. Wordline output voltage drop due to limited driving ability: (a) circuit schematic, and (b) Op-map power scaling for different ReRAM crossbar size.

on/off ratio and worse uniformity [27, 36].

C. *System implementation*

Op-amp design: Op-amp is the most important component because it is the key component of the WL driver (Op-amp configured in voltage follower mode) as well as the sample and hold circuit (Op-amp configured in current integrator mode). To reduce the power consumption and area, we design a simple two-stage Op-amp with detailed SPICE simulations in 32nm CMOS technology.

DAC: The design of DAC is trivial since 1-bit DAC is just a simple voltage buffer. In this paper, we divide the 16-bit fixed point number into 16 1-bit numbers and sequentially feed them to the ReRAM crossbar arrays. This is similar with prior works and has been proved as a sweet spot for ReRAM based accelerator [9, 11].

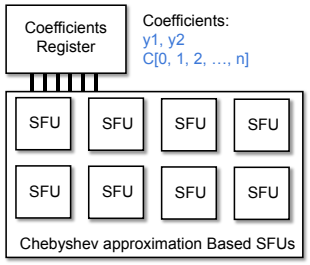
ADC: ADC is non-trivial and has been reported as the main bottleneck for computing speed and introduces great power and chip area overhead [11, 37]. We employ an 8-bit 1.2 GS/s (Giga samples per second) successive approximation register (SAR) ADC design which is optimized for energy efficiency and area [32]. As mentioned earlier, 1 ADC is shared for an entire ReRAM crossbar array to reduce the energy/area overhead. Our simulation indicates that ADC is also the major bottleneck of the system performance in our deisgn, introducing significant

	Taylor expansion	Chebyshev	PWL-1	PWL-2
Accuracy*	2.5e-3	6.4e-4	1.1e-2	4.8e-4
# of intervals	10	10	10	50
# of coefficients	30	50	20	100
Power (uW)	1273	1688	1068	2904
Area (um²)	4009	5212	3028	9357
Clock frequency	500 MHz			

(a)

```
d1=0.0; // temporary variable
d2=0.0; // temporary variable
// approximation range [a, b]
y1=(2.0*x-a-b)/(b-a);  // y1 is parameter
y2=2.0*y1; // y2 is parameter
// n is the number of coefficients
for (j=n-1; j>0; j--)
   sv=d1; // temporary result
   // c[0, 1, 2…] are coefficients
   d1=y2*d1-d2+c[j+1];
   d2=sv;
end
result=y1*d1-d2+0.5*c[1]; // final result
```

(b)



(c)

Fig 10. (a)Comparison of different SFU designs. (b) Pseudo code for Chebyshev approximation. (c) SFU array and coefficients register.

power (43% of system power) and area (49% of the chip area) overhead.

Digital sub-blocks: There are several digital circuits in our design, including the multiplier, SFU, shift & add unit and the dataflow controller. The multiplier in our design supports 16-bit fixed point number operation since it has been demonstrated that 16-bit is sufficient for most machine learning applications [38].

There are multiple different approaches for the SFU design including Taylor expansion based approximation, Chebyshev approximation [39], and piece-wise linear function (PWL). As illustrated in Fig. 10(a), we compare the accuracy and synthesized power/area number of these approaches. The accuracy is measured using sigmoid function (a popular non-linear function used in neural network). And the synthesis results are from 28 nm CMOS technology. With same number of intervals (10 interpolations in the input range [-5, 5]), the Chebyshev exhibits the best accuracy but consumes more power and area since it has more coefficients and computation. PWL approach (PWL-1) have lowest power and smallest area but the accuracy is not good enough. With more coefficients (i.e. interpolations) (PWL-2), we observe comparable accuracy with Chebyshev approach, but the power/area are significantly higher since it has much more coefficients.

In this work, we employ the Chebyshev approximation since it provides the best accuracy with moderate power and chip area overhead. The procedure of Chebyshev approximation is shown in Fig. 10 (b). The coefficients are first calculated with CPU and pre-loaded into the local register file. During computing, SFU will access the register file and calculate the non-linear function, as shown in Fig. 10 (c).

In prior ReRAM accelerator works, circuits are designed to solve a specific function [8, 11, 14]. Compared with them, the SFU based approach is advantageous since it has more flexibility, can be easily reconfigured (i.e. load different coefficients) to solve different functions.

TABLE I: Power consumption and area for sub-blocks in the proposed design.

Component	Power (mW)	Area (um²)	Number
WL peripherals	1.9	40.9	128
BL peripherals	5.0	1550	
Local buffer	0.16	648.0	
ReRAM array	0.01	147.5	
Dataflow controller	0.3	300	1
SFU	1.68	5212	16
Multiplier	0.18	1155	4
Bank total	932.86	0.39 mm²	N/A

Memory array and local buffer: We design our own simulator to model the read/write power and latency based on the experimental data calibrated ITR model [20, 23, 24]. We assume the size of 1T1R structure is $10F^2$ where F is the minimum lithography length in 28nm technology. Similar with prior works [11, 40], eDRAM are employed as local buffer. The power and area of eDRAM is modeled based on [11].

The power consumption and area for each sub-block are listed in Table I.

V. RESULTS

A. Experiments setup

Benchmark: We evaluate the system performance with two RNN based applications. One is NLP where we want to predict the next word in the sentence based on previous input vocabularies. The dataset is available at [41]. The Second application is human activity recognition (HAR) and we use the dataset from [42]. For both applications, we evaluate the performance with three RNN configurations: basic RNN, LSTM, and GRU. For all networks, the number of hidden layer features are 128 (i.e. the length for hidden state vector is 128). Should aware that our system can also be utilized to accelerate the computing for feedforward neural networks such as Multi-layer perceptron (MLP) and CNN. Hence, we also include MLP/CNN into our benchmark. The benchmark MLP has one hidden layer with 256 hidden neurons. The CNN contains 2 Convolutional layers, and 1 fully connected layer. The benchmark information is summarized in Table II.

TABLE II. Benchmarks

Datasets	Description	Networks	Hidden state	*Ops/frame
NLP	Predict next word from previous input words	*bRNN	128	$5.6x10^5$
		LSTM		$2.2x10^6$
		GRU		$1.7x10^6$
HAR	Classify human activity from 6 categories.	*bRNN	128	$2.2x10^6$
		LSTM		$9.0x10^6$
		GRU		$6.7x10^6$
MNIST	Hand written digits classification	MLP	256	$3.9x10^5$
		CNN	/	$9.8x10^6$

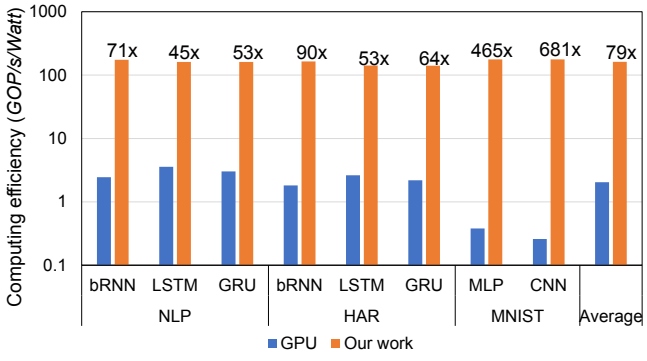
***bRNN:** basic RNN **Ops/frame**: number of operations per input sample

GPU baseline: We perform the experiments with Tensorflow deep learning framework [43] running on a NVIDIA GeForce GTX 1080Ti GPU.

Source code. All the source code (based on Tensorflow) is available for download at https://github.gatech.edu/ylong32.

B. System performance and comparison with other platforms

We first evaluate the computing efficiency and compare with our GPU baseline. The computing efficiency is represented in term of *GOP/s/W* (Giga operations per second per Watt). The GPU power is measured by nvidia-smi provided by NVIDIA CUDA toolkit. Fig. 11(a) shows the computing efficiency comparison considering different datasets and networks. We observe that the average improvement across all the RNN benchmarks are 79x. Interestingly, experiment results indicate that the GPU computing efficiency for MLP/CNN is lower than RNN. This is mainly caused by input data access latency. To be more specific, the input data for feedforward networks all comes from external memory. Differently, input data of RNN are the concatenation of input data and hidden state calculated from previous stage. Since the hidden state are temporary data which only exist in GPU's L1/L2 cache, it can be directly fetched without the latency for accessing external memory. In our PIM architecture, both the input data and temporary hidden state are stored in the buffer subarrays, therefore, the performance for computation of RNN and MLP/CNN is similar. The second reason impacting the GPU performance for CNN computing is that the data structure of convolutional operation should be re-organized for matrix-vector multiplication. This



(a)

Frame rate (Fps) for benchmarks

Frame rate		GPU	Our work
NLP	bRNN	877.2 K	289.6 K
	LSTM	319.5 K	68.0 K
	GRU	359.7 K	88.0 K
HAR	bRNN	161.0 K	69.4 K
	LSTM	58.2 K	14.6 K
	GRU	64.6 K	19.8 K
MNIST	MLP	196.2 K	65.4 K
	CNN	57.4 K	25.4 K

(b)

Fig 11. (a) Computing efficiency in terms of *GOP/s/Watt*. (b) System throughput in terms of frame rate (Fps).

TABLE III. Comparison with other hardware accelerators.

Accelerator Name	ESE	ISAAC	PipeLayer	DaDianNao	EIE	Our work
Target networks	RNN	CNN	CNN	CNN	CNN RNN	CNN RNN
Technology	22 nm	32 nm	/	28 nm	45 nm	28 nm
Approach	FPGA	ReRAM	ReRAM	ASIC	ASIC	ReRAM
Training support	No	No	Yes	No	No	No
Parameter storage	DRAM	eDRAM	ReRAM (In situ)	eDRAM	SRAM	ReRAM (In situ)
Computing efficiency (GOP/s/W)	6.88	380.7	142.9	286.4	174.1	116.3

overhead no longer exists in our system since the parameters are pre-loaded into the ReRAM crossbar arrays.

Should aware that the throughput (i.e. frames per second) of our system is less than the GPU, as shown in Fig 11(b). This is because our system only consumes 0.6 Watt while the average GPU power is around 200 Watt, enabling more than 300x improvements for the energy consumption.

We also compare the computing efficiency of our system with prior works, including FPGA based LSTM accelerator, ESE [16]; ReRAM based CNN accelerators, ISAAC [11] and PipeLayer [9], and ASIC based CNN/RNN accelerator, DaDianNao [40] and EIE [18]. The results are summarized in Table III. The FPGA based approach demonstrates the lowest performance because it stores the parameters in the external DRAM. Also, the maximum clock frequency for FPGA is much lower than the ASIC, which further constrains the performance. Compared with ReRAM based CNN accelerator, we demonstrate similar performance with PipeLayer but less than ISAAC. The reason is that prior works do not consider the driving ability issue and ignore the power consumption of the WL buffer which is one of the major energy hungry component (52.7% of the total power in our design). If ignore the power consumption on the WL buffer, our design can achieve performance with 341 *GOP/s/W*, which is similar with ISAAC. We show that the overheads associated with the peripheral circuit can significantly degrade the computing efficiency.

We observe that the ASIC approach achieves the state-of-the-art computing efficiency. The primary reason is that ASIC designs employ large size on chip memory to store the parameters; therefore, the data movement energy is reduced. However, in ASIC designs, both eDRAM and SRAM are volatile memory, the stored data will vanish after power off, making it not suitable for platforms with limited energy budget such as mobile devices. On the contrary, ReRAM based design is advantageous since the parameters are stored in non-volatile memory. Therefore, we conclude that *if the computing is performed in datacenter where the power supply is sufficient and stable, ASIC based approach is preferred; if the computing is performed in a distributed low power platform, the ReRAM approach provides more benefits.*

C. *Enhance performance with lower bit-precision*

Previous analyses assume that all the parameters and input data are represented with 16-bit fixed point number. To further enhance the computing efficiency, we explore using lower bit precision for computing. Fig. 12 shows the trade-off between computing efficiency and classification accuracy. The data in Fig. 12 is based on HAR dateset. Similar results can be observed for other datasets. Simulation indicates that 8-bit precision demonstrate satisfactory results while lower bit precision (less than 6-bit) show significant accuracy drop. This is consistent with the result in prior work [38]. With 8-bit precision, the performance (*GOP/s/W*) is 4 times higher than 16-bit.

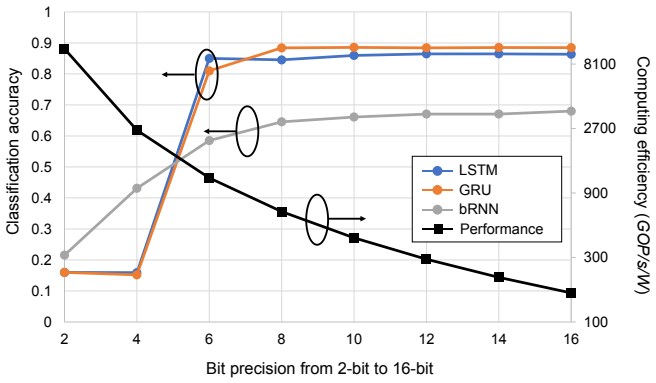


Fig 12. System performance with lower bit-precision.

D. *Impact of device variations*

The device variation of ReRAM can significantly deteriorate system performance. Device variation comes from the stochastic formation and rupture of conductive filament in the resistive layer of ReRAM (i.e., generation and recombination of oxygen vacancy is stochastic) [20, 24]. Variations exist in cycle-to-cycle operation and from one device to another device. Variations can be caused by read or write operation, properties of resistive materials, and various fabrication factors. In this work, we consider using Gaussian noise to represent the device variation. Other forms of device noises such as bit-flip error or random telegraph noise (RTN) can be analyzed in a similar way. We use the following equation to represent the Gaussian noise of device conductance:

$$g_{noise} = g_{ideal} \cdot \left(1 + N(0, \sigma^2)\right) \qquad (10)$$

where g_{ideal} is the expected device conductance without variation; $N(0, \sigma^2)$ is the normal distribution with mean equals to 0 and standard variation σ. It has been measured that the variation is normally less than 0.2 [19].

We evaluate the computing accuracy in terms of the classification accuracy for the benchmarks. As shown in Fig. 13, we observe that the accuracy drop is insignificant when the standard deviation of the added Gaussian is less than 0.2. However, the accuracy drops a lot for all the benchmark tests when the noise level is large. Data in Fig. 13 also indicates that the performance of LSTM and GRU are better than the basic RNN especially for a more complex dataset. For example, with HAR dataset, the performance of the basic RNN is not satisfactory (only 69%) while both LSTM and GRU achieve around 90% accuracy.

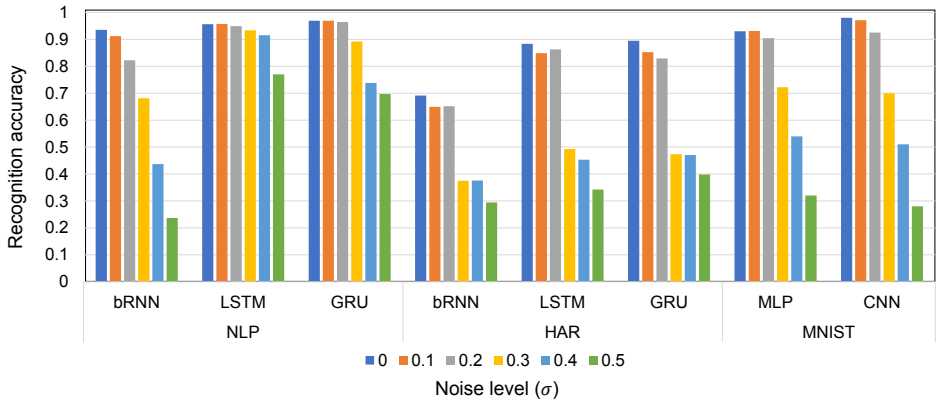


Fig 13. Computing accuracy with different levels of device variation.

E. *Handling large scale networks*

Prior ReRAM based works do not consider the need for re-programming during inference. They assume that the network parameters are programmed into the processing arrays and never change [8, 9, 11, 14]. Even though PipeLayer characterizes the device writing, it only considers the weight update during training [9]. However, re-programming can be necessary, especially when the energy budget is constrained and the system capacity is not enough to hold all parameters simultaneously. It is critical to get a more comprehensive understanding about the system performance with re-programming considered.

Using LSTM as an example, we gradually increase the size of hidden states and evaluate the system throughput and running time with the system capacity unchanged (512 KB), shown in Fig. 14. When the hidden state number is small (< 500), the parameters of the network can be mapped to the system simultaneously. Moreover, multiple small LSTMs can be mapped to the system together, allowing processing several input sequences at the same time. On the other hand, if the hidden state size reaches a threshold when the parameter size is larger than the system capacity, re-programming occurs. We observe a drastic performance and speed drop due to re-programming. Further increasing the number of hidden state introduces more programming cycles along with more throughput and speed drop. Should aware that neural networks with different size of hidden state may have different performance even though they have same number of re-programming. For example, we consider LSTM with 600 and 900 hidden state (the parameter size is 628 KB and 928 KB, respectively). Since the system capacity is 512 KB, requiring 1 re-programming for both networks. Remember that the reading time is a constant, the inference time are same for these two networks. However, networks with more hidden states have more computation. Therefore, the performance (*GOP/s*) is different.

As mentioned earlier, we can increase the number of ReRAM processing arrays in a bank, or use multiple banks tiled together to increase the system capacity, and thus, avoid the re-programming issue. The energy consumption is proportional to the system scale. Another solution is storing more bit per

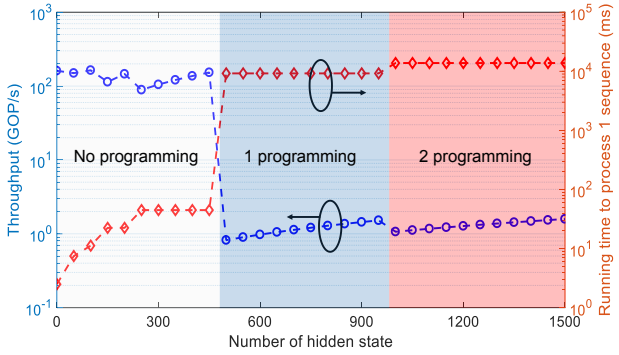


Fig 14. System performance with different number of hidden state.

cells. For example, if one device can store 4-bit, then the maximum capacity becomes 2 MB, 4 times larger than our original design (2-bit/cell). But this will introduce more computation error. The third approach is to increase the crossbar size. However, to drive larger array, the WL buffer size must be scaled up proportionally (more power consumption). Moreover, ADC with more bit precision is required for large crossbar array.

VI. Conclusion

We present a RNN accelerator design based on ReRAM PIM architecture. The proposed architecture is suitable for various RNN computation including the basic RNN, LSTM, and GRU. We measure the system throughput and energy efficiency with detailed circuits/devices characterization. We observe that the computing efficiency of the proposed system achieves 79x improvements compared with GPU baseline on average. Further, the computing accuracy drop is insignificant when the read noise standard deviation is less than 0.2. Lower bit-precision such as 8-bit can enhance the performance with insignificant accuracy loss. We observe that re-programming during inference can significantly deteriorate the performance and should be minimized.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[2] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725-1732.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, 2013, pp. 6645-6649: IEEE.

[4] D. Hassabis, "Artificial Intelligence: Chess match of the century," *Nature,* vol. 544, no. 7651, pp. 413-414, 2017.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 248-255: IEEE.

[6] S. B. Eryilmaz, D. Kuzum, S. Yu, and H.-S. P. Wong, "Device and system level design considerations for analog-non-volatile-memory based neuromorphic architectures," in *Electron Devices Meeting (IEDM), 2015 IEEE International*, 2015, pp. 4.1. 1-4.1. 4: IEEE.

[7] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, 2015, pp. 283-295: IEEE.

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

[9] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, 2017, pp. 541-552: IEEE.

[10] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, 2016, pp. 16.2. 1-16.2. 4: IEEE.

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

[12] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems,* 2016.

[13] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555,* 2014.

[14] Y. Long, E. M. Jung, J. Kung, and S. Mukhopadhyay, "ReRAM Crossbar based Recurrent Neural Network for human activity detection," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, 2016, pp. 939-946: IEEE.

[15] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128,* 2014.

[16] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, and Y. Wang, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *FPGA*, 2017, pp. 75-84.

[17] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: comparison of FPGA, CPU, GPU, and ASIC," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, 2016, pp. 1-4: IEEE.

[18] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, 2016, pp. 243-254: IEEE Press.

[19] B. Gao, Y. Bi, H.-Y. Chen, R. Liu, P. Huang, B. Chen, L. Liu, X. Liu, S. Yu, and H.-S. P. Wong, "Ultra-low-energy three-dimensional oxide-based electronic synapses for implementation of robust high-accuracy neuromorphic computation systems," *ACS nano,* vol. 8, no. 7, pp. 6998-7004, 2014.

[20] B. Gao, S. Yu, N. Xu, L. Liu, B. Sun, X. Liu, R. Han, J. Kang, B. Yu, and Y. Wang, "Oxide-based RRAM switching mechanism: A new ion-transport-recombination model," in *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*, 2008, pp. 1-4: IEEE.

[21] U. Russo, D. Ielmini, C. Cagli, A. L. Lacaita, S. Spiga, C. Wiemer, M. Perego, and M. Fanciulli, "Conductive-filament switching analysis and self-accelerated thermal dissolution model for reset in NiO-based RRAM," in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, 2007, pp. 775-778: IEEE.

[22] S. Park, H. Kim, M. Choo, J. Noh, A. Sheri, S. Jung, K. Seo, J. Park, S. Kim, and W. Lee, "RRAM-based synapse for neuromorphic system with pattern recognition function," in *Electron Devices Meeting (IEDM), 2012 IEEE International*, 2012, pp. 10.2. 1-10.2. 4: IEEE.

[23] P. Huang, X. Liu, W. Li, Y. Deng, B. Chen, Y. Lu, B. Gao, L. Zeng, K. Wei, and G. Du, "A physical based analytic model of RRAM operation for circuit simulation," in *Electron Devices Meeting (IEDM), 2012 IEEE International*, 2012, pp. 26.6. 1-26.6. 4: IEEE.

[24] X. Guan, S. Yu, and H.-S. P. Wong, "A SPICE compact model of metal oxide resistive switching memory with variations," *IEEE electron device letters,* vol. 33, no. 10, pp. 1405-1407, 2012.

[25] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, and E. U. Giacometti, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices,* vol. 62, no. 11, pp. 3498-3507, 2015.

[26] A. F. Vincent, J. Larroque, N. Locatelli, N. B. Romdhane, O. Bichler, C. Gamrat, W. S. Zhao, J.-O. Klein, S. Galdin-Retailleau, and D. Querlioz, "Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," *IEEE transactions on biomedical circuits and systems,* vol. 9, no. 2, pp. 166-174, 2015.

[27] S. Park, A. Sheri, J. Kim, J. Noh, J. Jang, M. Jeon, B. Lee, B. Lee, B. Lee, and H. Hwang, "Neuromorphic speech systems using advanced ReRAM-based synapse," in *Electron Devices Meeting (IEDM), 2013 IEEE International*, 2013, pp. 25.6. 1-25.6. 4: IEEE.

[28] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, 2015, pp. 476-488: IEEE.

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

[30] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks,* vol. 5, no. 2, pp. 157-166, 1994.

[31] X. Wu, V. Saxena, and K. Zhu, "A CMOS spiking neuron for dense memristor-synapse connectivity for brain-inspired computing," in *Neural Networks (IJCNN), 2015 International Joint Conference on*, 2015, pp. 1-6: IEEE.

[32] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Brandli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS," *IEEE Journal of Solid-State Circuits,* vol. 48, no. 12, pp. 3049-3058, 2013.

[33] R. Goldman, K. Bartleson, T. Wood, K. Kranen, V. Melikyan, and E. Babayan, "32/28nm Educational Design Kit: Capabilities, deployment and future," in *Microelectronics and Electronics (PrimeAsia), 2013 IEEE Asia Pacific Conference on Postgraduate Research in*, 2013, pp. 284-288: IEEE.

[34] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on Electron Devices,* vol. 53, no. 11, pp. 2816-2823, 2006.

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

[36] L. Gao, I.-T. Wang, P.-Y. Chen, S. Vrudhula, J.-s. Seo, Y. Cao, T.-H. Hou, and S. Yu, "Fully parallel write/read in resistive synaptic array for accelerating on-chip learning," *Nanotechnology,* vol. 26, no. 45, p. 455204, 2015.

[37] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, 2016, pp. 1-6: IEEE.

[38] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, and A. Borchers, "In-datacenter performance analysis of a tensor processing unit," *arXiv preprint arXiv:1704.04760,* 2017.

[39] M. Price, J. Glass, and A. P. Chandrakasan, "14.4 A scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, 2017, pp. 244-245: IEEE.
[40] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, and N. Sun, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609-622: IEEE Computer Society.
[41] R. Atienza. (2017). *LSTM by Example using Tensorflow*. Available: https://github.com/roatienza/Deep-Learning-Experiments
[42] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A Public Domain Dataset for Human Activity Recognition using Smartphones," in *ESANN*, 2013.
[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, and M. Devin, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467,* 2016.

Yun Long (S'15) received the B.S. degree in microelectronics from Peking University, Beijing, China in 2014. He is currently pursuing the Ph.D degree in Electrical and Computer Engineering with the Georgia Institute of Technology, Atlanta, GA, USA. His current research insterests include emerging technology based machine learning accelerator design and FPGA based high performance computing for dynamical system modeling. He received the Wusi Fellowship and National Fellowship in 2008 and 2009 when he was an undergraduate student in Peking University.

Taesik Na (S'14) received the B.S. and M.S. degrees in electrical engineering and computer science from Seoul National University, Seoul, Korea, in 2006 and 2008, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering at the Georgia Institute of Technology, Atlanta, GA, USA. From 2008 to 2013, he was a circuit design engineer with Samsung Electronics Semiconductor Memory Division, Korea, involved in the design of high-speed DRAMs. His current research interests include low power and secure neuromorphic hardware design. Mr. Na was a recipient of the Korea Foundation for Advanced Studies (KFAS) Scholarship and the Kwanjeong Educational Foundation Doctoral Fellowship during 2006–2008 and 2013–2017, respectively.

Saibal Mukhopadhyay (S'99–M'07–SM'11) received the B.E. degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, India, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 2000 and 2006, respectively. He is currently a Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta. His research interests include design of energy-efficient, intelligent, and secure systems in nanometer technologies.

Dr. Mukhopadhyay was a recipient of the Office of Naval Research Young Investigator Award in 2012, the National Science Foundation CAREER Award in 2011, the IBM Faculty Partnership Award in 2009 and 2010, the SRC Inventor Recognition Award in 2008, the SRC Technical Excellence Award in 2005, the IBM PhD Fellowship Award for years 2004 to 2005. He has authored or co-authored over 200 papers in refereed journals and conferences, and holds five U.S. patents. Dr. Mukhopadhyay is a *Senior Member* of IEEE.