CASCADE: Connecting RRAMs to Extend Analog Dataflow In An End-To-End In-Memory Processing Paradigm

Teyuh Chou, Wei Tang, Jacob Botimer, and Zhengya Zhang
{teyuh,weitang,botimerj,zhengya}@umich.edu
University of Michigan, Ann Arbor

ABSTRACT

Processing in memory (PIM) is a concept to enable massively parallel dot products while keeping one set of operands in memory. PIM is ideal for computationally demanding deep neural networks (DNNs) and recurrent neural networks (RNNs). Processing in resistive RAM (RRAM) is particularly appealing due to RRAM's high density and low energy. A key limitation of PIM is the cost of multi-bit analog-to-digital (A/D) conversions that can defeat the efficiency and performance benefits of PIM. In this work, we demonstrate the CASCADE architecture that connects multiply-accumulate (MAC) RRAM arrays with buffer RRAM arrays to extend the processing in analog and in memory: dot products are followed by partial-sum buffering and accumulation to implement a complete DNN or RNN layer. Design choices are made and the interface is designed to enable a variation-tolerant, robust analog dataflow. A new memory mapping scheme named R-Mapping is devised to enable the in-RRAM accumulation of partial sums; and an analog summation scheme is used to reduce the number of A/D conversions required to obtain the final sum. CASCADE is compared with recent in-RRAM computation architectures using state-of-the-art DNN and RNN benchmarks. The results demonstrate that CASCADE improves the energy efficiency by 3.5× while maintaining a competitive throughput.

CCS CONCEPTS

• **Computer systems organization → Neural networks**; *Single instruction, multiple data.*

KEYWORDS

Process in memory, Resistive RAM, Neural network accelerator

ACM Reference Format:
Teyuh Chou, Wei Tang, Jacob Botimer, and Zhengya Zhang. 2019. CASCADE: Connecting RRAMs to Extend Analog Dataflow In An End-To-End In-Memory Processing Paradigm. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3352460.3358328

MICRO-52, October 12–16, 2019, Columbus, OH, USA

ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358328

1 INTRODUCTION

In recent years, machine learning is becoming the dominant workload for the next-generation computation systems. As one of the most important machine learning kernels, deep neural networks (DNNs) and recurrent neural networks (RNNs) are now being widely deployed in tasks from image analysis to speech recognition.

DNN workloads are typically highly vectorized with well-defined dataflow patterns. A large number of digital DNN accelerators [7, 9, 10, 17, 32] have been designed to achieve high performance and high efficiency. With the continued increase in DNN complexity and the growing demand for faster and lower power machine learning use cases, we see the need of power efficient chips that can be employed in a wide range of applications including autonomous driving and mobile devices. In such edge applications, an advanced nonvolatile memory, such as resistive RAM (RRAM), can play an important role thanks to its high storage density, low leakage power and fast wake-up from sleep [13].

Beyond high-density storage, recent work started looking into in-RRAM computation, a form of processing in memory (PIM), to enable massively parallel dot products in an RRAM crossbar array without moving the stored operands. The core computation of a DNN layer can be easily mapped to an RRAM crossbar array: the input activations of a DNN layer are applied to the wordlines (WL) of an RRAM crossbar as voltage pulses, and the weights are stored as conductances of the RRAM crossbar. A dot product is obtained from the bitline (BL) of each column of the RRAM crossbar by the physics of Ohm's law for multiplication and Kirchhoff's law for accumulation. The simple and elegant in-RRAM dot product has been projected to achieve impressive performance and efficiency [5, 11, 21, 26, 29, 35].

In-RRAM dot product is a form of analog computation. When used as a part of a digital system, the digital inputs to the RRAM crossbar need to be converted to voltage pulses using digital-to-analog converters (DACs), and the outputs of the RRAM crossbar in the form of analog currents need to be integrated and digitized using analog-to-digital converters (ADCs). In-RRAM computation pushes the resolution requirement of analog computation to accommodate tens or hundreds of products of multi-bit WL pulses with multi-bit RRAM conductances that are summed together. High-resolution ADCs are required, adding a significant overhead. As RRAM crossbar size and device resolution continue to increase, the required ADC resolution also increases. It would not be surprising to see that analog-to-digital (A/D) conversion will eventually dominate the area and energy consumption of in-RRAM computation to an extent that renders in-RRAM computation impractical.

A survey of recent work on in-RRAM computation highlights the overhead of A/D conversion as a severe limitation. As a full-fledged DNN acclerator in RRAM, ISAAC [38] employs 8-bit ADCs that

are estimated to cost 58% of the power and 31% of the silicon area. PRIME [12] uses sense amplifiers (SAs) instead of conventional ADCs to reduce area. However, an SA is only capable of resolving one bit at a time. To obtain a 6-bit digital output, the SA uses up to 2^6 cycles in decision time, resulting in a long latency that is exponentially dependent on the resolution. The SA interface limits the throughput for demanding applications.

The second limitation of in-RRAM computation is that even a single layer in a state-of-the-art DNN or RNN can be too large to fit on a practical RRAM crossbar. The reason is twofold. First, despite the rapid progress in RRAM technology development, most RRAM crossbars demonstrated are of sizes from 64×64 to 256×256 [46], allowing up to 64 to 256 partial sums to be accumulated on each BL. In comparison, a single point in the output feature map (fmap) of a fully-connected (FC) layer in AlexNet [27] requires up to 9,216 partial sum accumulations, and a single point in the output fmap of a convolutional layer in GoogLeNet [43] requires up to 1,728 partial sum accumulations, both easily exceeding the number of analog accumulations that can be done in a practical RRAM crossbar. Therefore, one kernel computation needs to be separated and mapped to multiple RRAM crossbars. The resulting partial sums from multiple crossbars need to be digitized and accumulated in the digital domain. Second, it is impractical to assume that a 16-bit or even a 8-bit weight value can be reliably stored in one RRAM cell. Multi-level cell (MLC) requires the use of more complex DACs and ADCs, and can be more easily affected by noise and process variation. It is more practical to map a multi-bit weight value to multiple RRAM cells. Similarly, it is more practical to separate an input to units of 1 or 2 bits and apply them serially to simplify the circuitry and reduce the noise and variation uncertainty. It is also more practical to activate a subset of WLs, instead of all WLs, in a large RRAM crossbar. All of these practical approaches lead to more partial sums that need to be digitized and digitally accumulated.

In essence, in-RRAM computation consists of at least three parts: in-RRAM dot products, A/D conversion, and digital accumulation of partial sums. Currently, only the first part is done in RRAM, while the second and the third part are done by conventional CMOS circuits. Besides the aforementioned overhead of high-resolution A/D conversion, we estimate that the energy and area of digital partial-sum accumulation can surpass in-RRAM dot products to yield the core in-RRAM computation insignificant.

In this work, we present CASCADE, an in-RRAM computation architecture for DNNs and RNNs, to specifically address the problems of high-cost A/D conversion and digital partial sum accumulation associated with the current in-RRAM computation approach. The contributions of this work are as follows:

(1) We choose more practical and robust in-memory dot products by reducing effective BL resolution to ensure noise and variation tolerance. Only low-resolution analog outputs can be reliably cascaded. We analyze the trade-off between resolution and inference accuracy, and show that only by lowering the BL resolution, a good accuracy can be reliably achieved.
(2) We propose R-Mapping scheme to use a buffer RRAM to perform in-RRAM partial sum accumulation, replacing digital partial sum accumulation. The analog summation bypasses the A/D conversions of low-order sums, reducing the number of A/D conversions.
(3) We connect MAC RRAMs to buffer RRAMs by using the transimpedance amplifiers (TIAs) as the interface to convert MAC RRAMs' BL outputs from analog current to analog voltage that can directly feed to buffer RRAMs as inputs. Cascading MAC RRAMs with buffer RRAMs not only enables the "analog" dataflow to meet the computation requirement of a DNN or RNN layer, but also keeps all intermediate values in analog and in memory to obtain the highest possible energy efficiency and performance.

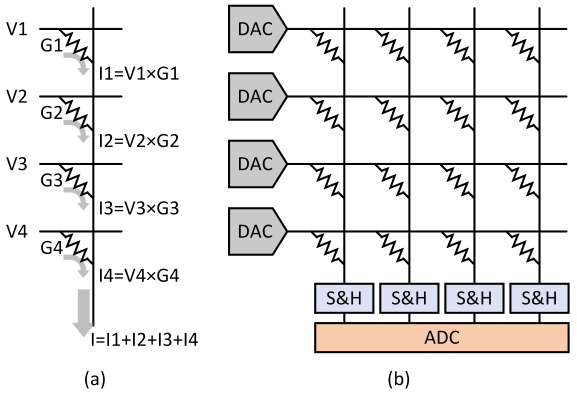

Figure 1: (a) A dot product implemented in an RRAM crossbar. (b) Parallel dot products implemented in an RRAM crossbar array. The input vector is converted to voltage pulses by DACs and the weights are stored in the RRAM crossbar. The BL currents are sampled and held (S&H) and converted to digital values.

2 BACKGROUND

An RRAM cell is a metal-insulator-metal (MIM) device that stores information via its programmable conductance. Typical RRAM devices are constructed in a crossbar array to provide dense storage to fulfill the growing demand of low-power nonvolatile memory. In addition to storage, an RRAM crossbar array can be used to perform parallel dot products.

Figure 1 shows examples of a dot product and parallel dot products using RRAM crossbars. In Figure 1(b), a 4×4 weight matrix is stored on the RRAM crossbar as conductances. A 1×4 input vector is sent to four DACs to be converted to read voltage pulses and applied to the WLs of the crossbar. A read voltage pulse over an RRAM cell's conductance produces a current that represents the product of the voltage input with the conductance. The currents through RRAM cells along a column are aggregated on the BL to complete a dot product.

As illustrated in the above example, a read voltage pulse is applied to a WL that drives a row of RRAM cells. Activating multiple or all of the WLs in the RRAM crossbar enables the second dimension of parallelism. Throughout the dot product operation, one set of operands, e.g., the 4×4 weight matrix in the above example, is kept in memory, saving significant time and energy in data movement.

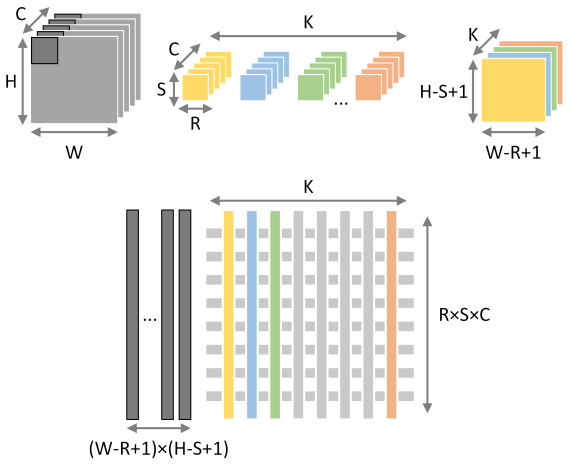

Figure 2: Mapping of convolution operation on an RRAM crossbar array.

In addition to dot products, an RRAM device supports accumulation in a write process. Applying consecutive write pulses, i.e., set or reset pulses, to an RRAM cell increases or decreases its conductance [6, 24]. The benefit of in-RRAM accumulation can be significant, as a typical n-step digital accumulation requires n reads from memory to fetch the temporary sum, and n writes to memory to update the sum, which are all eliminated by in-RRAM accumulation.

2.1 Workloads and Mapping to RRAM

DNNs and RNNs have emerged to be one of the most important machine learning workloads. A DNN consists of layers of convolution (CONV), pooling, normalization, and fully-connected (FC) layers. CONV and the FC layers are the most computation-intense and memory-intense layers.

A CONV layer is shown in Figure 2. An input activation sized $W \times H$ of C channels is convolved with K kernels sized $R \times S$ of C channels to produce an output fmap sized $X \times Y$ of K channels ($X = W - R + 1$ and $Y = H - S + 1$). The kernels (weights) are learned through a training algorithm. A point (x, y, k) of the output fmap, $f^{out}(x, y, k)$, is calculated as follows:

$$f^{out}(x,y,k) = \sigma\Big(\sum_{c=1}^{C}\sum_{r=1}^{R}\sum_{s=1}^{S} f^{in}(x+r, y+s, c) \times w_k(r,s,c)\Big), \quad (1)$$

where $f^{in}(x, y, c)$ is the input activation at (x, y, c), $w_k(r, s, c)$ is the weight value of k^{th} kernel at (r, s, c), and σ is the activation function.

Each elementary step of the computation involves the product of a pair of input activation and weight. To compute each point in the output fmap, (1) describes three layers of loops over R, S, and C, involving accumulation of $R \times S \times C$ partial sums. To increase throughput, the three layers of loops can be unrolled or partially unrolled. To complete one entire output fmap, there are three additional outer loops over X, Y, and K.

An FC layer can be viewed as a special case of CONV layer with $W = R$ and $H = S$, i.e., the dimensions of the input activations and kernels are matched. The $R \times S$ for a FC layer is typically larger than a CONV layer. Since the dimensions of the input activations and weights are matched, an output fmap is sized $1 \times 1 \times K$.

To perform inference, the weights w of a CONV or a FC layer remain static as the input activations f^{in} are streamed in. Therefore, it is advantageous to store the weights in RRAM, reuse the weights as new input activations are applied. One efficient way of storing weights in an RRAM crossbar array is shown in Figure 2, where one $R \times S \times C$ kernel is stored in $R \times S \times C$ cells as conductances in one column, and K kernels are stored across K columns. The illustration in Figure 2 assumes that the RRAM array consists of at least $R \times S \times C$ rows and K columns, and each RRAM device provides a sufficient resolution to store a weight value. Following this mapping, $R \times S \times C$ partial sums are accumulated on one BL to complete the computation of one point in the output fmap.

Practical RRAM arrays may not provide nearly as many rows or columns. The $R \times S \times C$ of a layer in a state-of-the-art DNN or RNN can easily exceed 10,000. Furthermore, a practical RRAM cell may not provide enough distinguishing levels to store a 16-bit or even an 8-bit weight value. The technology limitation requires a weight value to be stored in multiple RRAM cells, and the $R \times S \times C$ accumulations to be separated and performed on multiple BLs of one RRAM array or multiple arrays. Each BL is digitized by an ADC or an SA, and the final accumulation is done in the digital domain.

The underlying computation in an RNN can be mapped in the same manner. For example, consider the long short-term memory (LSTM) [16, 20], a kind of RNN. Given an input sequence $X = (x_1, x_2, ..., x_T)$, where x_t is the input at time step t, $t \in \{1, ..., T\}$, a typical LSTM layer is defined as follow.

$$\text{Input gate} : i_t = \sigma(W_x^i x_t + U_h^i h_{t-1}) \quad (2)$$

$$\text{Forget gate} : f_t = \sigma(W_x^f x_t + U_h^f h_{t-1}) \quad (3)$$

$$\text{Output gate} : o_t = \sigma(W_x^o x_t + U_h^o h_{t-1}) \quad (4)$$

$$\text{Candidate memory} : \tilde{c}_t = \tanh(W_x^c x_t + U_h^c h_{t-1}) \quad (5)$$

$$\text{Memory cell} : c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (6)$$

$$\text{Hidden state} : h_t = o_t \odot \tanh(c_t), \quad (7)$$

where W^j and U^j, $j = \{i, f, o, c\}$, are parameters learned through a training algorithm, $\odot$ denotes element-wise multiplication, σ is the sigmoid function, and tanh is the hyperbolic tangent function. Both σ and tanh are element-wise nonlinear activation functions. The computation intensive part is the matrix-vector product in (2) to (5), which can be rewritten as:

$$\begin{bmatrix} W_x^i & U_x^i \\ W_x^f & U_x^f \\ W_x^o & U_x^o \\ W_x^c & U_x^c \end{bmatrix} \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \quad (8)$$

Using this formulation, the dot products in an LSTM can be implemented in RRAM following the same approach.

2.2 In-RRAM Computation

ISAAC [38], PRIME [12] and PipeLayer [41] are three recently published architectures for implementing DNN and RNN through in-RRAM computation. There are also examples of in-SRAM computation [3, 14, 47] and in-DRAM computation [37] that follow the

	ISAAC [38]	PRIME [12]	PipeLayer [41]	CASCADE
Input bitwidth	16	6 w/ fraction encoding	16	16
Input bits per cycle (b_{WL})	1	3	1	1
Weight bitwidth	16	8 w/ fraction encoding	16	16
Cell resolution (b_{cell})	2	4	4	1
Array size ($N_{rows} \times N_{cols}$)	128×128	256×256	128×128	64×64
BL resolution (b_{BL})	9	15	11	7
Output bitwidth	8 w/ encoding	6 w/ truncation	N/A	6 w/ encoding
Output interface	ADC	SA	Spiking integrate and fire	TIA

Table 1: In-RRAM MAC architecture comparison.

same concept. A comparison of key aspects of ISAAC, PRIME and PipeLayer is shown in Table 1.

Row circuitry. PRIME applies both width- and level-modulation to each WL. A 6-bit input is converted to one of 8 voltage levels over two pulse periods. Providing 8 precise voltage levels is challenging, complicating the DAC circuits. ISAAC streams inputs in a bit-serial fashion: a binary input is sent to the WL driver one bit at a time, and a voltage pulse is produced and passed onto the WL. A 1-bit driver is simpler to design and the read process is also better controlled. PipeLayer also adopts ISAAC's bit-serial input streaming.

BL resolution. In-RRAM computation can produce a high BL resolution. PRIME uses 4 bits per RRAM cell and a 256-row RRAM array, resulting in a BL resolution of 15 bits. The outputs are truncated to 6 bits to be practical. ISAAC uses bit-serial input streaming, stores 2 bits per cell, and uses a 128-row array to reduce the BL resolution to 9 bits. The bit-serial input streaming is also adopted by PipeLayer. However, PipeLayer stores 4 bits per cell, leading to a 11-bit BL resolution.

Column circuitry. PRIME uses an SA for each BL. The SA takes up to 2^n cycles to perform 2^n comparisons to produce a n-bit output, where n is the output bitwidth. ISAAC uses an 8-bit ADC, which can be costly in terms of power and area. The ADC is shared among 128 BLs in an array to amortize the cost. PipeLayer uses an integrate-and-fire component for each BL to generate spikes. The serial integration is slow and the latency scales with 2^n.

Physical implementation challenges. State-of-the-art PIM chips demonstrate the challenges in designing peripheral circuitry to support a high BL resolution. To be realistic, the resolution of inputs and weights are often lowered, and only a subset of WLs are activated. The latest PIM chips, including the ones based on SRAM [3, 14, 47] and the ones based on RRAM [45], chose to digitize only the most significant bits (MSBs) to reduce the cost of A/D conversion. In particular, 1-bit output was used in [47]. Doing so severely limits the application of PIM. In [8], only 9 WLs are activated per column. Since high-resolution PIM can be affected by process, voltage and temperature (PVT) variations [15], it is critical to take variations and noise into account in PIM designs.

2.3 A/D Conversion for In-RRAM Computation

A/D conversion is an integral part of in-RRAM computation, and it contributes about 60% of the power consumption based on the latest work [38]. Common A/D conversion choices are ADC or SA. An ADC's complexity and power consumption depend on its sampling rate and resolution. In-RRAM computation has a relatively relaxed sampling rate, due to the intrinsic RC delay of WL and BL propagation. However, in-RRAM computation can require a high resolution that depends on the the resolution of the analog read pulse b_{WL}, the resolution of RRAM cell b_{cell}, and the number of rows that are activated in parallel N_{rows}. With the growing desire of using multi-level cell (MLC) RRAM and a higher degree of parallelism by activating more rows in parallel, the A/D resolution is constantly raised. Since the area and energy consumption of A/D conversion scale exponentially with the resolution [36, 38], designing the A/D conversion for in-RRAM computation is a main challenge.

An SA is commonly found in the peripheral circuitry of single-level cell (SLC) memories such as SRAM or DRAM. An SA can be viewed as a 1-bit ADC that compares BL voltage with a reference voltage to produce a 1-bit output. An SA can serve as a multi-bit ADC by sweeping the reference voltages, i.e., a reference voltage ramp, and keeping track of when the SA output flips using a counter [12]. The SA circuitry is simple and compact, but using SA for multi-bit A/D conversion can cost a high latency of up to 2^n cycles, where n is the resolution.

3 THE CASCADE ARCHITECTURE

The CASCADE in-RRAM computation architecture targets inference in edge/IoT devices with a stringent energy and area envelope. A CASCADE chip is made of analog processing units (APUs) that each consists of a number of RRAM crossbar arrays, as shown in Figure 3. An RRAM array can be tasked with performing in-memory dot products, buffering or in-memory accumulation. CASCADE executes a DNN or RNN model layer by layer as in [9]. The trained weights in a layer are first loaded to the APUs from main memory. The weights and inputs are assumed to be 16 bits, and the dot products are quantized to 16 bits.

Compared to ISAAC, PRIME or PipeLayer, CASCADE uses an efficient analog dataflow with a TIA interface at the output of a multiply-accumulate (MAC) RRAM to convert the analog current to voltage. The voltage is applied to a buffer RRAM directly to accomplish partial-sum accumulation. The results are sent to summing amplifiers and ADCs to convert to digital values, followed

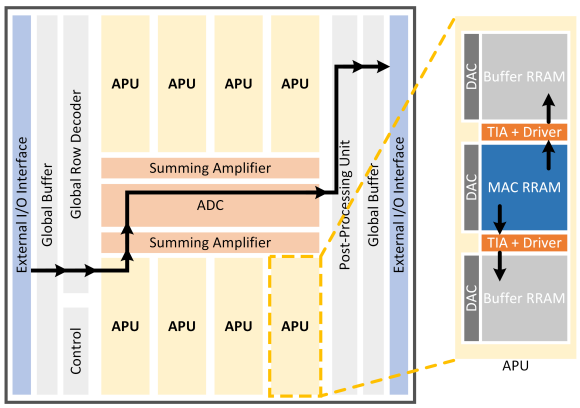

Figure 3: Illustration of the CASCADE architecture. The bold lines indicate the proposed dataflow. The zoom-in view shows an analog processing unit (APU).

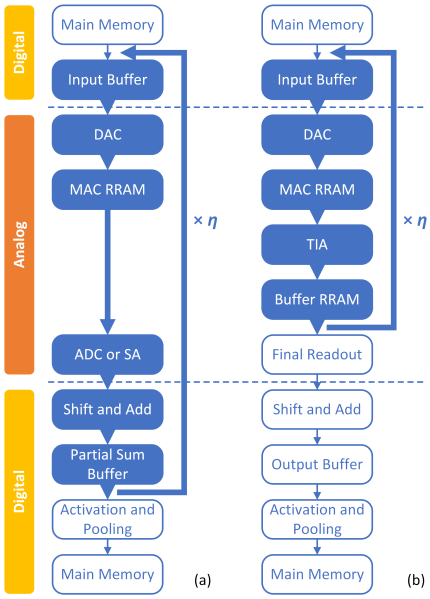

Figure 4: Comparison of (a) in-RRAM MAC and digital accumulation of partial sums, and (b) in-RRAM MAC and in-RRAM buffering and accumulation of partial sums. The dashed lines indicate D/A and A/D boundaries. The inner loop is highlighted in blue.

by activation, normalization and pooling. The outputs are stored in main memory for the next layer of processing. Cascading MAC RRAMs with buffer RRAMs realizes the core computation of a CONV or FC layer in analog and in RRAM. Since A/D conversion occurs in the very end of the computation, redundant conversions of intermediate values are saved.

3.1 Input Streaming and Weight Mapping in MAC RRAM

In designing CASCADE, we adopt the bit-serial streaming of input ($b_{WL} = 1$) to a MAC RRAM. Each 16-bit input ($\eta = 16$) is streamed from LSB to MSB as 16 WL pulses. A 1-bit WL driver is simpler to design, more compact and consumes less power than a multi-bit DAC.

The weight values are stored in a MAC RRAM using b_{cell} bits per cell. A 16-bit weight value ($\omega = 16$) is mapped to ω/b_{cell} RRAM cells. To limit the BL resolution and the impact of variation and noise, we use 1-bit weight mapping ($b_{cell} = 1$) and moderate-sized RRAM array of 64×64 ($N_{rows} = 64, N_{cols} = 64$). In this way, the BL resolution is kept to 7 bits, lower than all the previous work as shown in Table 1. Following the encoding in [38], the BL resolution is further reduced from 7 to 6 bits. With bit-serial input streaming and binary weight mapping in a MAC RRAM, only two voltage references are needed, one for read and one for write, simplifying routing and driver circuitry.

In CASCADE, a 16-bit weight is stored in 16 cells in a row; and a 64×64 MAC RRAM stores 4 16-bit weights per row and 256 weights in total. The 64×64 MAC RRAM can be effectively divided into 4 64×16 subsections, each subsection represents a 64×1 16-bit weight vector. The MAC RRAM performs the dot products of a 1×64 input bit vector with four 64×1 16-bit weight vectors at a time.

3.2 Buffering of Partial Sums in Buffer RRAM

After in-RRAM dot products, the BLs of the MAC RRAM carry the analog partial sums associated with every bit of the weights. Using bit-serial input streaming, every new input bit vector produces a new set of analog partial sums that need to be aligned and accumulated.

Partial sum accumulation is also needed due to the mapping of wide dot products on multiple MAC RRAMs. As discussed previously, in the case of large $R \times S \times C$, a wide dot product needs to be separated and mapped to multiple MAC RRAMs or one MAC RRAM through time-multiplexing. The partial sums need to be accumulated to obtain the final result.

Digital Accumulation. In previous work [12, 38, 41], the accumulation of partial sums is done in the digital domain. The dataflow is illustrated in Figure 4, and it follows the steps below for every input bit vector:

(1) Convert the BL outputs of a MAC RRAM to digital partial sums using ADCs or SAs;
(2) Read out the temporary sums stored in SRAM or registers;
(3) Shift and accumulate the partial sums by S+A;
(4) Truncate the LSBs of the sum to maintain a given bitwidth;
(5) Write back the updated sum to SRAM or registers;

As illustrated in Figure 4, for a 16-bit input, the partial-sum accumulation incurs 16 A/D conversions and data movement in and out of SRAM or registers, which significantly worsens the energy efficiency and performance. Some of the A/D conversions are wasteful due to the LSB truncation in Step (4).

Analog in-RRAM Buffering and Accumulation. The CASCADE architecture employs analog buffering and in-RRAM accumulation by cascading a MAC RRAM with two buffer RRAMs via

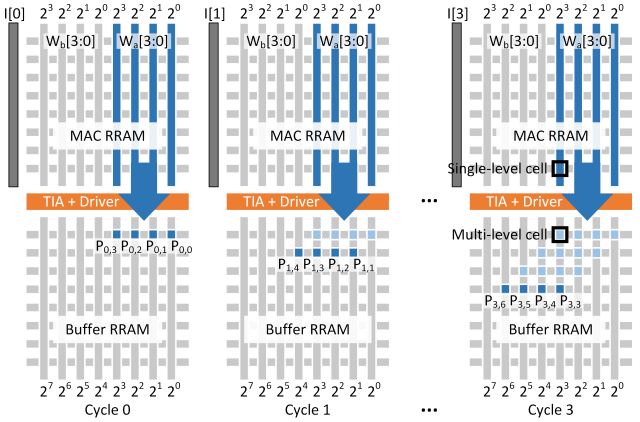

Figure 5: Illustration of R-Mapping for 4-bit weights and 4-bit inputs. The inputs are serially streamed in with LSB first. The MAC RRAM stores two sets of 4-bit weights W_a and W_b. The arrows indicate the align and store of partial sums through TIAs and input drivers. The partial sum $P_{i,j}$ is stored in the ith row and jth column in cycle i of the buffer RRAM.

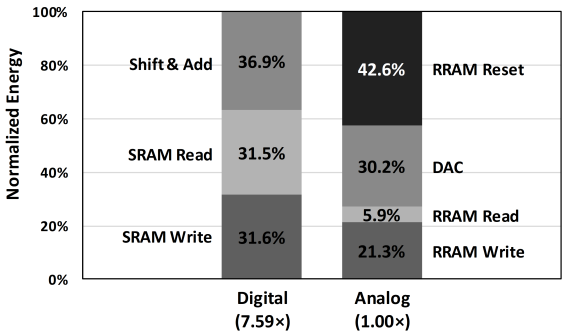

Figure 6: Normalized energy consumption of digital partial-sum accumulation and analog in-RRAM partial-sum buffering and accumulation.

TIA interface, as shown in Figure 3. The dataflow is illustrated in Figure 4, and it follows the steps below for every input bit vector:

(1) Convert the BL outputs of MAC RRAM to analog voltages using TIAs;
(2) Align the voltages as inputs to buffer RRAMs to store the analog partial sums;

Since the MAC RRAM's BL resolution is 6 bits, we propose to use 6-bit MLC RRAM [1] for the buffer RRAMs. After the serial streaming of the 16-bit inputs are complete, the analog partial sums stored in the buffer RRAM are accumulated before the final A/D conversions.

To support in-RRAM accumulation of partial sums, we propose R-Mapping scheme as illustrated in Figure 5. Consider LSB-first bit-serial input streaming and a 64×16 subsection of a MAC-RRAM that stores a 16-bit weight vector. First, the dot products are computed for the input bit vector 0 and the 16-bit weight vector. The outputs are 16 analog partial sums (one per BL). These 16 analog partial sums are written to a buffer RRAM at address i. Next, the dot products are computed for the input bit vector 1 and the 16-bit weight vector. The outputs of 16 analog partial sums are left-shifted by 1 and written to the buffer RRAM at address $i + 1$, as shown in Figure 5. At the completion of the bit-serial input streaming, the partial sums are stored in 16 rows and 31 columns of the buffer RRAM (or 15 rows and 30 columns if the inputs are signed numbers). The R-Mapping scheme allows the final accumulation to be done in one read of the buffer RRAM described in Section 3.4.

In CASCADE, we assume signed inputs, and the partial sums computed by a 64×16 subsection of a MAC RRAM are written to 15 rows and 30 columns of a buffer RRAM. We connect one 64×64 MAC RRAM to two 15×30 buffer RRAMs, as shown in Figure 3, so each buffer RRAM stores the partial sums from two subsections of the MAC RRAM.

Figure 6 shows the energy breakdown of the digital accumulation of partial sums and the analog in-RRAM buffering and accumulation. The analog in-RRAM buffering and accumulation is estimated to use 7.59× less energy than the digital partial-sum accumulation. The significant savings are mainly attributed to the elimination of repeated SRAM read and write accesses required by the digital approach.

With analog in-RRAM buffering and accumulation, the A/D conversion is only exercised after the final partial-sum accumulation, as illustrated in Figure 4. In addition, the number of A/D conversions is limited to the number bits needed for the final output, eliminating redundant conversions of intermediate values.

Buffer RRAM Write Consideration. A standard RRAM write uses a relatively high voltage. It costs high energy and is the primary reason for the limited endurance [42]. In this work, we propose to use a lower voltage to write to buffer RRAMs, and 1T1R RRAM with a transistor to control the write current [39]. These lead to improved endurance and lower energy. Low-voltage write to RRAM can be non-deterministic [23]. In simulation, the errors and variations due to non-deterministic write are incorporated as part of the noise analysis in Section 3.5. We allocate one clock period for write to avoid pipeline stalling.

3.3 TIA Interface between MAC RRAM and Buffer RRAM

A TIA is used to convert an input current to a proportional output voltage. A conventional TIA is constructed using an operational transconductance amplifier (OTA) with a resistor in the feedback connection. The conventional design can be slow in settling, and consume a large on-current. It is also difficult to set the output voltage range appropriate for driving the buffer RRAM.

We design a new TIA circuit as shown in Figure 7(a) to convert MAC RRAM's BL current to voltage, and holds the voltage for driving the buffer RRAM. The TIA circuit operates in two phases: sensing and transfer. In the sensing phase, SW0 and SW1 are closed to enable a feedback loop to convert the BL current to voltage on C_{amp}; SW2 is open and SW3 is closed to detach C_{out} and precharge it to V_{DD}. After the sensing phase, SW0 and SW1 are open to detach the BL from the TIA; and SW3 is open to complete the precharge. In

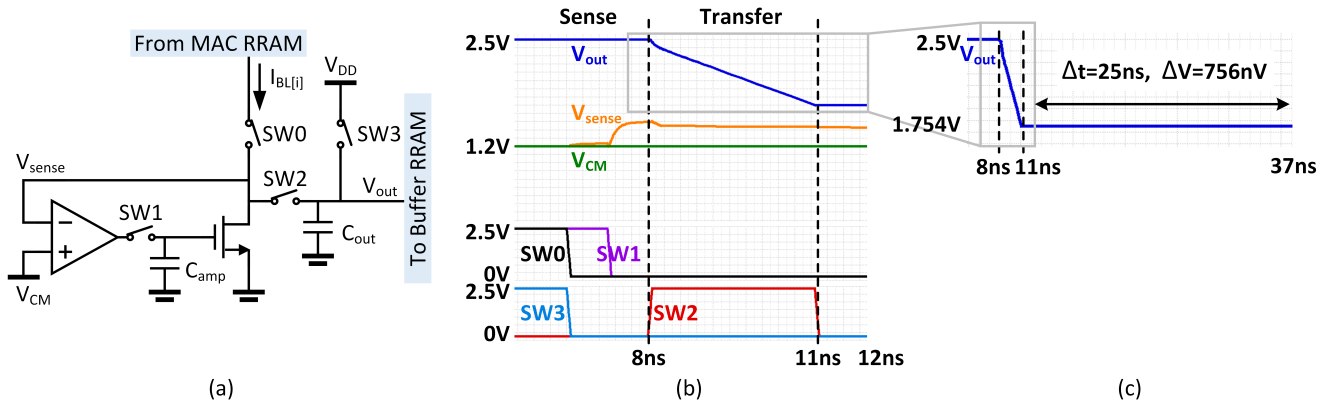

Figure 7: (a) Schematic of TIA for converting an input current to a proportional output voltage; (b) simulation waveforms of the TIA designed in a 65nm CMOS technology and the charge leakage of the output capacitor over 25ns of the buffer RRAM write period; and (c) zoom-out view of the TIA sensing and transfer phase.

the transfer phase, SW2 is closed to allow the sampled voltage on C_{amp} to drive the NMOS to reproduce the BL current to discharge C_{out}. After the transfer is complete, SW2 opens and the voltage is held on C_{out} for driving the buffer RRAM.

Compared to the conventional TIA design, the proposed TIA offers a faster settling time, lower energy per conversion, and flexibility in setting the output voltage range for driving the next stage. The TIA circuit is designed and simulated as shown in Figure 7(b) to obtain realistic parameters for system evaluations. We used metal-oxide-metal capacitor for C_{out}. The simulation shows that the leakage on C_{out} results in a negligible voltage drop of 756nV over the 25ns RRAM write period as shown in Figure 7(c).

3.4 Final Accumulation and A/D Conversion

Following the R-Mapping scheme, analog partial sums from one subsection of a MAC SRAM are stored in 15 rows and 30 columns of a buffer RRAM. After summed together, 10 A/D conversions are needed with resolutions ranging from 6 to 10 bits.

The final accumulation is illustrated in Figure 8. The 30 BLs can be divided into groups to efficiently obtain the output of a required resolution. For example, if a 16-bit output is required, the 9 BLs in the MSB group directly contribute to the required 16-bit output resolution, and they are digitized by suitable 6-bit to 10-bit ADCs. The BLs in the LSB group are connected to analog summing amplifiers as shown in in Figure 8(b), with each BL current appropriately scaled before summing. The analog sum of a low-order group is fed as the input to the next high-order group. In this way, the low-order groups are compressed to one carry-in to the MSB group. The digital values are then added together to produce the final sum.

With the analog accumulation scheme, the number of A/D conversions is reduced, and the number of digital summations is also reduced. The low-order accumulations are done more efficiently in the analog domain. Although analog accumulation can be less precise than digital accumulation, it produces only a carry-in to the MSB group and the imprecision becomes negligible. With fewer A/D conversions, we can choose an ADC of a lower sampling frequency to reduce the energy per conversion step [30]. Fewer number of A/D conversions also makes it possible to share ADCs to reduce area.

3.5 Noise Tolerance

Since the CASCADE architecture connects MAC RRAMs with buffer RRAMs, and relies on the analog dataflow from MAC RRAMs to buffer RRAMs, it is critical to check the variation and noise tolerance of the end-to-end system. Analytically, we can lump the variation, noise and non-idealities of analog circuits as effective noise on the MAC RRAM BL, and measure the signal-to-noise ratio (SNR). The SNR affects the classification accuracy as shown in Figure 9. In this example, we used a 2-layer MLP as the workload. Different system configurations require different levels of SNR. A higher SNR means a lower margin for noise tolerance.

Suppose we aim at a 90% classification accuracy, a 6-bit BL resolution (e.g., 1 input bit/cycle, 1-bit MAC RRAM cell, and 64-row MAC RRAM as in CASCADE) requires a minimum SNR of 25 dB, while a 11-bit BL resolution (e.g., 1 input bit/cycle, 4-bit MAC RRAM cell, and 128-row MAC RRAM as in PipeLayer [41]) requires a minimum SNR of 35 dB. The noise tolerance of the PipeLayer configuration is 10 dB lower than the CASCADE configuration. The CASCADE architecture adopts a 6-bit BL resolution to ensure the robustness of the end-to-end analog and in-memory computation.

4 EVALUATION

We first establish the reference architectures based on ISAAC and PRIME. We then provide an exploration of the CASCADE design space to show the capabilities of the architecture as well as its limitations. Finally, an instance of the CASCADE architecture is evaluated using realistic workloads with comparisons made against the references.

4.1 Methodology

Reference Architectures. We use two reference architectures: 1) an ADC-based architecture adapted from ISAAC [38] and an

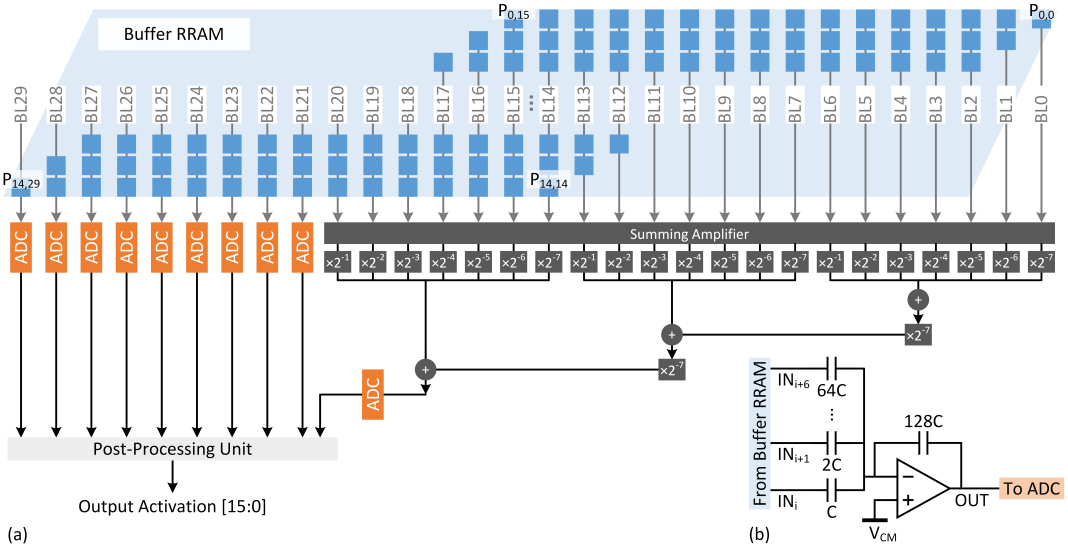

Figure 8: (a) Illustration of the final accumulation; and (b) schematic of a summing amplifier.

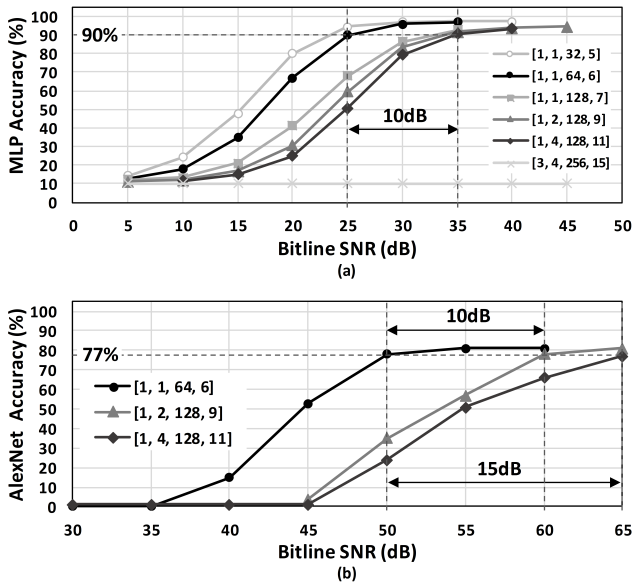

Figure 9: (a) Two-layer MLP classification accuracy for different configurations noted by [b_{WL}, b_{cell}, N_{rows}, b_{BL}], and (b) AlexNet top-5 classification accuracy for CASCADE and configurations based on ISAAC and PipeLayer.

	CASCADE	ADC-based	SA-based
In-RRAM dot product	Input: 16 bits, bit-serial streaming Weight: 16 bits, binary mapping RRAM array size: 64 × 64		
Partial-sum accumulation	Analog in-RRAM	Digital sum SRAM	Digital sum SRAM
A/D resolution	10 bits	6 bits	6 bits
A/D activity	10 times per 16 cycles	256 times per 16 cycles	256 times per 16 cycles
A/D normalized latency	1	1	2^6
Number of ADCs	7 ADCs per 80 arrays	80 ADCs per 80 arrays	80×64 SAs per 80 arrays

Table 2: Configurations of the CASCADE, ADC-based and SA-based reference architectures.

SA-based architecture adapted from PRIME [12]. To make a fair comparison, the architectures all employ RRAM crossbar arrays of the same size, and all utilize bit-serial input streaming and binary weight mapping.

Table 2 summarizes the three architectures for comparison. The key difference is that CASCADE performs in-RRAM buffering and accumulation, while the two reference architectures perform digital accumulation after converting the analog partial sums using ADCs and SAs. With an efficient TIA interface and analog buffering and accumulation, CASCADE reduces the number of A/D conversions from 256 per 16 cycles to 10 per 16 cycles.

Component Models. Our evaluations were done using a 65nm technology and a 65nm RRAM model from [8]. The SRAM model is constructed based on the results obtained from a memory compiler. We adopted most of the circuit component models from ISAAC [38] and scaled them to 65nm. The analog components including ADC, SA, summing amplifier and S&H were obtained from recent literature and scaled to 65nm. In particular, we used the successive approximation (SAR) ADC from [28], same as in ISAAC. The area and energy of a SAR ADC and the resolution scaling follow [36]. The SA model was adapted from [31]. The summing amplifier was from [33]. The TIA was designed in a 65nm CMOS technology and simulated in Cadence Spectre to obtain power, latency and variation.

AlexNet	VGG-A	VGG-B	VGG-C	MSRA-A	MSRA-B	MSRA-C	DeepFace	NeuralTalk
11 × 11, 96/4 (1)	3 × 3, 64 (1)	3 × 3, 64 (2)	3 × 3, 64 (2)	7 × 7, 96/2 (1)	7 × 7, 96/2 (1)	7 × 7, 96/2 (1)	11 × 11, 32 (1)	FC-2400
5 × 5, 256 (1)	3 × 3, 128 (1)	3 × 3, 128 (2)	3 × 3, 128 (2)				9 × 9, 16/2 (1)	FC-8791
3 × 3, 384 (1)	3 × 3, 256 (2)	3 × 3, 256 (3)	3 × 3, 256 (4)	3 × 3, 256 (5)	3 × 3, 256 (6)	3 × 3, 384 (6)	9 × 9, 16 (1)	
3 × 3, 384 (1)	3 × 3, 512 (2)	3 × 3, 512 (3)	3 × 3, 512 (4)	3 × 3, 512 (5)	3 × 3, 512 (6)	3 × 3, 768 (6)	7 × 7, 16/2 (1)	
3 × 3, 256 (1)	3 × 3, 512 (2)	3 × 3, 512 (3)	3 × 3, 512 (4)	3 × 3, 512 (5)	3 × 3, 512 (6)	3 × 3, 896 (6)	5 × 5, 16 (1)	
FC-4096							FC-4096	
FC-4096							FC-4030	
FC-1000								

Table 3: Benchmarks for evaluation. Convolution layers are denoted as $R \times S, K/D$ (L), where R, S and K correspond to the notations used in Figure 2, and L denotes the number of such layers.

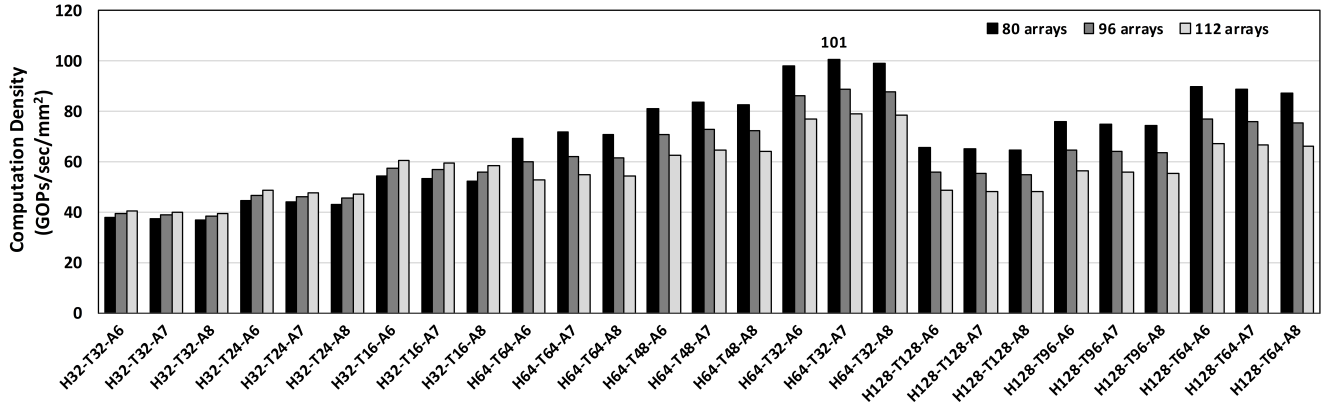

Figure 10: Computation density across the design space. A notation, e.g., H64-T32-A7 R80, represents 80 64×64 RRAM arrays with 32 TIAs per array and 7 ADCs shared by the arrays.

Benchmarks. We used 11 benchmarks including 10 DNNs and 1 RNN to evaluate the CASCADE architecture and compare it with the references. The details of the DNNs and RNN are listed in Table 3. We used ImageNet image classification dataset for AlexNet [27], ResNet [19], 3 types of VGG [40], GoogLeNet [43], and 3 types of MSRA [18], face recognition for DeepFace [44], and image captioning for NeuralTalk [25].

4.2 CASCADE Design Space Exploration

The CASCADE architecture is parameterized by 4 variables: 1) the size of the RRAM array $H \times H$, simply denoted by H; 2) the number of RRAM arrays, denoted by R; 3) the number of TIAs per array, denoted by T; and 4) the number of ADCs, denoted by A. We assume that the total weight storage capacity is 40KB × 80 blocks = 3.2MB and a DDR4 I/O bandwidth of 25.6GB/s between the CASCADE chip and external memory.

The computation density measured in GOPs/s/mm^2 is shown in Figure 10. In general, using larger RRAM arrays provides a higher computation density due to the more dot products and accumulations that can be performed in RRAM arrays at the same time. However, the larger the array size, the higher the I/O bandwidth, the ADC resolution, and the cost of interface circuitry, including S&Hs, TIAs, summing amplifiers, and ADCs. The optimal number of RRAM arrays in one APU and the optimal number of APUs are limited by the I/O bandwidth. The peak performance of 101 GOPs/s/mm^2 can be achieved by 80 64×64 RRAM arrays, 32 TIAs per array and 7 central ADCs (denoted by H64-T32-A7 R80).

4.3 Performance and Energy Consumption

We use 80 APU blocks to evaluate the energy and performance for comparison with the references. Each APU block contains 80 64×64 RRAM arrays with 32 TIAs per array and 7 ADCs shared by the arrays.

Figure 11(a) shows the energy consumption of CASCADE compared to the two reference architectures for the 10 DNN and 1 RNN benchmarks. The CASCADE architecture achieves an average 3.5× lower energy than the ADC-based architecture and 11.0× lower energy than the SA-based architecture across all benchmarks. Figure 12 shows the energy breakdown for the three architectures to shed light on the competitive advantages of CASCADE. The input buffer and in-RRAM dot products consume the same amount of energy across all three architectures. However, CASCADE's TIA interface consumes 77.5× lower energy than the ADC interface and 325.4× lower energy than the SA interface. The latter is due to the long latency of the SA-based A/D conversion.

Figure 11(b) shows the throughput of CASCADE compared to the two reference architectures for the 10 DNN and 1 RNN benchmarks. In average, the CASCADE architecture achieves 1.86× higher throughput than the ADC-based architecture, and 17.83× higher

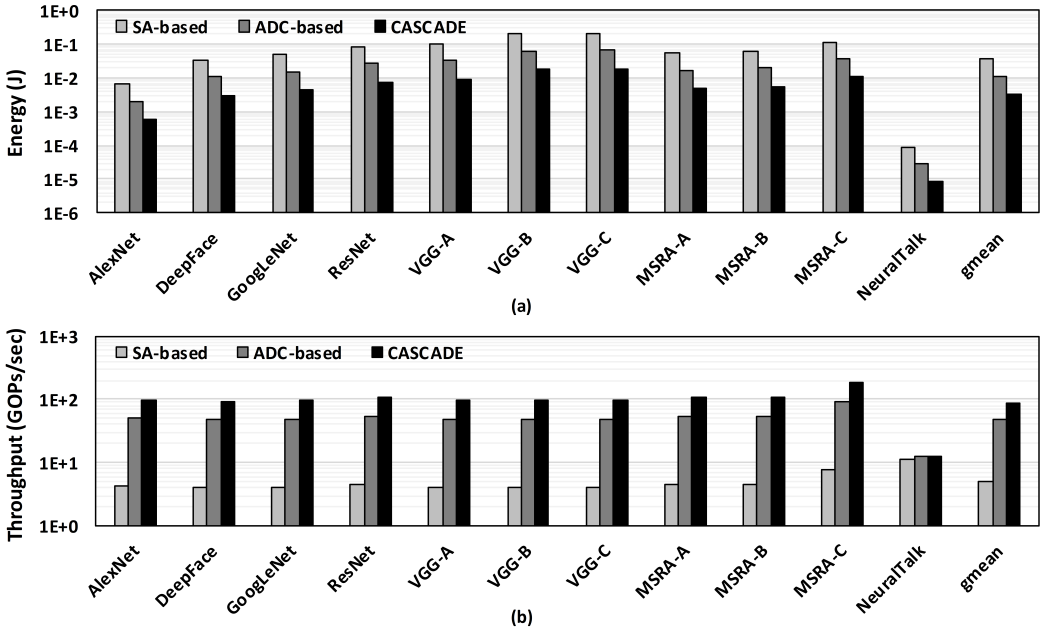

Figure 11: (a) Energy consumption of CASCADE compared to reference architectures running DNN and RNN benchmarks; (b) Throughput of CASCADE compared to reference architectures running DNN and RNN benchmarks.

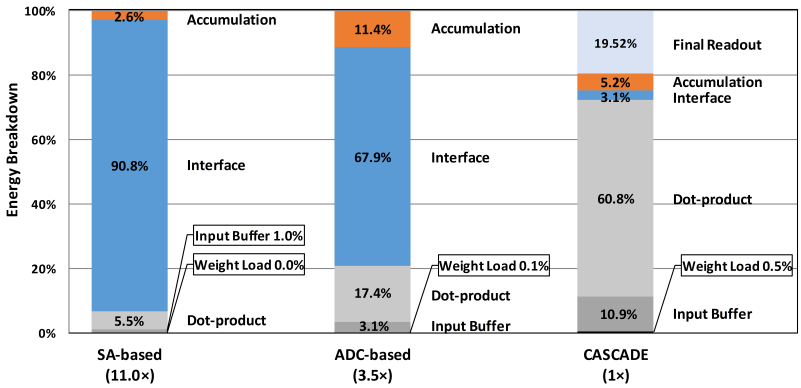

Figure 12: Energy breakdown of CASCADE and two reference architectures.

throughput than the SA-based architecture due to the long latency of A/D conversion.

In summary, CASCADE improves upon the ADC-based architecture in energy. As an example of the ADC-based architecture, ISAAC [38] has already demonstrated improvements of 14.8×, 5.5×, and 7.5× in throughput, energy, and computation density over DaDianNao [9]. The 64-chip DaDianNao has demonstrated 450.65× speedup and 150.31× lower energy than an NVIDIA K20M GPU. Therefore, we expect that the benefits of CASCADE will be on top of the previously demonstrated gains over an ASIC chip or a GPU.

4.4 Extension to Spiking Neural Networks

The CASCADE architecture can be adapted to support spiking neural networks (SNNs). The TIA output capacitor can be used as the integration capacitor. A comparator can be added to generate spikes if the voltage on the integration capacitor exceeds a threshold, following the approach in [2, 4, 22, 34]. To implement a SNN, we only need the MAC RRAMs, TIAs, and additional comparators. The buffer RRAMs can be bypassed.

5 CONCLUSION

This work presents CASCADE, an architecture that connects MAC RRAMs for computing dot products with buffer RRAMs for in-RRAM buffering and accumulating partial sums through an efficient TIA interface. Dot products and partial-sum accumulations are the essential operations for implementing a DNN or RNN. Keeping both parts in RRAM and in analog ensures a high energy efficiency by removing the overhead of A/D conversion and digital accumulation.

We demonstrate a new R-Mapping scheme to efficiently accumulate partial sums, and an analog summation approach to bypass the A/D conversions of low-order bits. As a result, the CASCADE architecture minimizes the number of A/D conversions and keeps the A/D conversions at the very end of the entire computation. The CASCADE architecture is pipelined to achieve a high performance, and it consumes 3.5× lower energy than an ADC-based in-RRAM computation architecture in processing DNN and RNN workloads. Built on realistic RRAM technology constraints, the CASCADE architecture offers a higher SNR margin for variation and noise tolerance while keeping a light-weight CMOS periphery circuitry.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their inputs. The work is supported in part by NSF CCF-1900675.

REFERENCES

[1] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov. 2012. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* 23, 7, 075201.

[2] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, Y. Jaoudi, and G. W. Burr. 2018. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* 558, 7708, 60–67.

[3] A. Biswas and A. P. Chandrakasan. 2018. Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications. In *IEEE International Solid-State Circuits Conference (ISSCC)*. 488–490.

[4] I. Boybat, M. Le Gallo, S. R. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, and E. Eleftheriou. 2018. Neuromorphic computing with multi-memristive synapses. *Nature Communications* 9, 1, 2514.

[5] G. W. Burr, R. M. Shelby, C. di Nolfo, J. W. Jang, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. Kurdi, and H. Hwang. 2014. Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element. In *IEEE International Electron Devices Meeting (IEDM)*.

[6] P.-Y. Chen, D. Kadetotad, Z. Xu, A. Mohanty, B. Lin, J. Ye, S. Vrudhula, J. Seo, Y. Cao, and S. Yu. 2015. Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 854–859.

[7] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*. 269–284.

[8] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang, T.-H. Hsu, Y.-C. King, C.-J. Lin, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang. 2018. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *IEEE International Solid-State Circuits Conference (ISSCC)*. 494–496.

[9] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 609–622.

[10] Y.-H. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture*. 367–379.

[11] Z. Chen, B. Gao, Z. Zhou, P. Huang, H. Li, W. Ma, D. Zhu, L. Liu, X. Liu, J. Kang, and H.-Y. Chen. 2015. Optimized learning scheme for grayscale image recognition in a RRAM based analog neuromorphic system. In *IEEE International Electron Devices Meeting (IEDM)*.

[12] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie. 2016. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *Proceedings of the International Symposium on Computer Architecture*. 27–39.

[13] P.-F. Chiu, M.-F. Chang, C.-W. Wu, C.-H. Chuang, S.-S. Sheu, Y.-S. Chen, and M.-J. Tsai. 2012. Low store energy, low VDDmin, 8T2R nonvolatile latch and SRAM with vertical-stacked resistive memory (memristor) devices for low power mobile applications. *IEEE Journal of Solid-State Circuits* 47, 6, 1483–1496.

[14] S. K. Gonugondla, M. Kang, and N. Shanbhag. 2018. A 42pJ/decision 3.12TOPS/W robust in-memory machine learning classifier with on-chip training. In *IEEE International Solid-State Circuits Conference (ISSCC)*. 490–492.

[15] S. K. Gonugondla, M. Kang, and N. R. Shanbhag. 2018. A variation-tolerant in-memory machine learning classifier via on-chip training. *IEEE Journal of Solid-State Circuits* 53, 11, 3163–3173.

[16] A. Graves, A. Mohamed, and G. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*. 6645–6649.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the International Symposium on Computer Architecture*. 243–254.

[18] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *The IEEE International Conference on Computer Vision (ICCV)*. 1026–1034.

[19] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[20] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8, 1735–1780.

[21] 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. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Proceedings of the Design Automation Conference*.

[22] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis. 2013. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology* 24, 38, 384010.

[23] G. Indiveri, E. Linn, and S. Ambrogio. 2016. ReRAM-based neuromorphic computing. *Resistive Switching: From Fundamentals of Nanoionic Redox Processes to Memristive Device Applications*, 715–735.

[24] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu. 2010. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters* 10, 4, 1297–1301.

[25] A. Karpathy and L. Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3128–3137.

[26] Y. Kim, Y. Zhang, and P. Li. 2015. A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 11, 4.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[28] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Brändli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici. 2013. 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* 48, 12, 3049–3058.

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

[30] B. Murmann. 2018. ADC Performance Survey 1997-2018 (ISSCC & VLSI Symposium). [Online]. Available: http://web.stanford.edu/~murmann/adcsurvey.html.

[31] T. Na, B. Song, J. P. Kim, S. H. Kang, and S.-O. Jung. 2017. Offset-canceling current-sampling sense amplifier for resistive nonvolatile memory in 65 nm CMOS. *IEEE Journal of Solid-State Circuits* 52, 2, 496–504.

[32] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture*. 27–40.

[33] X. Peng, W. Sansen, L. Hou, J. Wang, and W. Wu. 2011. Impedance adapting compensation for low-power multistage amplifiers. *IEEE Journal of Solid-State Circuits* 46, 2, 445–451.

[34] M. D. Pickett, G. Medeiros-Ribeiro, and R. S. Williams. 2013. A scalable neuristor built with Mott memristors. *Nature Materials* 12, 2, 114–117.

[35] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov. 2015. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 7550, 61–64.

[36] M. Saberi, R. Lotfi, K. Mafinezhad, and W. A. Serdijn. 2011. Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs. *IEEE Transactions on Circuits and Systems I: Regular Papers* 58, 8, 1736–1748.

[37] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 273–287.

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

[39] S.-S. Sheu, P.-C. Chiang, W.-P. Lin, H.-Y. Lee, P.-S. Chen, Y.-S. Chen, T.-Y. Wu, F. T. Chen, K.-L. Su, M.-J. Kao, K.-H. Cheng, and M.-J. Tsai. 2009. A 5ns fast write multi-level non-volatile 1 K bits RRAM memory with advance write scheme. In

IEEE Symposium on VLSI Circuits. 82–83.

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

[41] L. Song, X. Qian, H. Li, and Y. Chen. 2017. PipeLayer: A pipelined ReRAM-based accelerator for deep learning. In *IEEE International Symposium on High Performance Computer Architecture*. 541–552.

[42] D. B. Strukov. 2016. Endurance-write-speed tradeoffs in nonvolatile memories. *Applied Physics A* 122, 4, 302.

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–9.

[44] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1701–1708.

[45] C.-X. Xue, W.-H. Chen, J.-S. Liu, J.-F. Li, W.-Y. Lin, W.-E. Lin, J.-H. Wang, W.-C. Wei, T.-W. Chang, T.-C. Chang, T.-Y. Huang, H.-Y. Kao, S.-Y. Wei, Y.-C. Chiu, C.-Y. Lee, C.-C. Lo, Y.-C. King, C.-J. Lin, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang. 2019. A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In *IEEE International Solid-State Circuits Conference (ISSCC)*. 388–390.

[46] P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, H.-S. P. Wong, and H. Qian. 2017. Face classification using electronic synapses. *Nature Communications* 8, 15199.

[47] J. Zhang, Z. Wang, and N. Verma. 2016. A machine-learning classifier implemented in a standard 6T SRAM array. In *IEEE Symposium on VLSI Circuits*. 1–2.