Unlocking Wordline-level Parallelism for Fast Inference on RRAM-based DNN Accelerator

Yeonhong Park, Seung Yul Lee, Hoon Shin, Jun Heo, Tae Jun Ham, and Jae W. Lee

Seoul National University
{ilil96, triomphant1, zmqp1, j.heo, taejunham, jaewlee}@snu.ac.kr

ABSTRACT

In-memory computing is rapidly rising as a viable solution that can effectively accelerate neural networks by overcoming the memory wall. Resistive RAM (RRAM) crossbar array is in the spotlight as a building block for DNN inference accelerators since it can perform a massive amount of dot products in memory in an area- and power-efficient manner. However, its in-memory computation is vulnerable to errors due to the non-ideality of RRAM cells. This error-prone nature of RRAM crossbar limits its wordline-level parallelism as activating a large number of wordlines accumulates non-zero current contributions from RRAM cells in the high-resistance state as well as current deviations from individual cells, leading to a significant accuracy drop. To improve performance by increasing the maximum number of concurrently activated wordlines, we propose two techniques. First, we introduce a lightweight scheme that effectively eliminates the current contributions from high-resistance state cells. Second, based on the observation that not all layers in a neural network model have the same error rates and impact on the inference accuracy, we propose to allow different layers to activate non-uniform numbers of wordlines concurrently. We also introduce a systematic methodology to determine the number of concurrently activated wordlines for each layer with a goal of optimizing performance, while minimizing the accuracy degradation. Our proposed techniques increase the inference throughput by 3-10× with a less than 1% accuracy drop over three datasets. Our evaluation also demonstrates that this benefit comes with a small cost of only 8.2% and 5.3% increase in area and power consumption, respectively.

1 INTRODUCTION

Deep neural networks (DNNs) have demonstrated very promising results in multiple task domains of AI such as computer vision, natural language processing, and robotics. While DNNs effectively perform sophisticated jobs often surpassing human performance, they usually accompany a tremendous amount of computation and high memory demands. For example, ResNet-18 [10] requires 1.83 Giga-operations for processing a single image from ImageNet dataset [8] and has 11.7 millions parameters. In conventional Von Neumann architecture where computation and data storage are separated, not only the amount of computation but also the high data movement of DNN is a performance bottleneck. The limited memory bandwidth, and more importantly the energy wall severely restricts the performance [1].

In-memory computing is a powerful paradigm that can largely cut back the overhead from data movement. Specifically, in the context of in-memory computing for DNN, resistive RAM (RRAM) has received a lot of attention. RRAM features high storage density, fast read speed, and very low energy consumption when organized in a crossbar array. More importantly, RRAM crossbar array inherently supports highly parallel in-memory dot product operations, so can execute the core computations of DNN. The weights are programmed as conductances of an RRAM crossbar and the feature maps are applied to the wordlines of the RRAM crossbar as voltage pulses. The voltage pulses induce currents flowing through the cells, and the currents are accumulated on the bitline to represent the output of a dot product. Exploiting such properties, many RRAM-based DNN accelerators have been proposed [1, 4, 6, 23, 26, 34].

The dot product operation of RRAM crossbar, however, is error-prone. The current flowing through the bitline which corresponds to the dot product output can result in a wrong value because of RRAM cell's non-ideality. Particularly, the incorrect readout of RRAM crossbar's bitline current happens due to the following two factors: (i) the accumulation of high-resistance state cells' currents, and (ii) the current variations resulting from cell resistance variations.

Such an error-prone nature of in-memory dot product of RRAM crossbar imposes a restriction on its parallel execution. Ideally, all the wordlines of RRAM crossbar array (e.g., 256, 512) can be activated at once to perform a bulk of dot products in parallel. However, the more the wordlines are activated, the more the deviations are accrued along the bitline current. For this reason, existing RRAM-based accelerator chips often limit the maximum number of (concurrently) activated wordlines (MAW). For example, recent RRAM prototypes concurrently activate only 9 out of 256 wordlines [2, 31] or 16 out of 512 wordlines [33]. This proves our point that the reliability issue limits the performance of an RRAM-based DNN accelerator by preventing a full exploitation of the potential wordline-level parallelism in RRAM crossbar arrays.

In this paper, we introduce two key techniques that can alleviate the performance degradation stemming from this reliability issue. First, we propose a way that can mitigate the readout errors by dynamically compensating for the amount of bitline current resulting from the high-resistance state cells in a very efficient way. Specifically, our technique turns the difficult task of identifying the current contributions from the high-resistance state cells into the

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415664

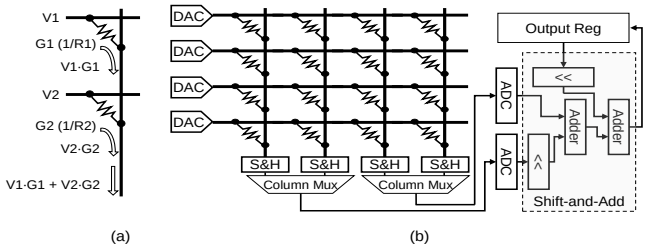

Figure 1: (a) A dot product computation in an RRAM crossbar array. (b) RRAM crossbar array for parallel dot products.

easy task of counting the number of 1s in the input, and presents an efficient hardware implementation for the proposed technique. Second, motivated from the fact that the end-to-end accuracy is less sensitive to an increase in the MAW for some layers than others as they are more resilient to errors and/or have sparser input characteristics (e.g., lots of zeros), we suggest to exploit different MAW for each layer in the neural network model. For this purpose, we also propose a systematic approach to determine MAW for each layer. By profiling the marginal impact of an increase in MAW on the end-to-end accuracy for each layer, we can derive proper MAW for all layers in the model that optimizes performance while satisfying the accuracy goal. Evaluation results show that our techniques enable 3-10× speedup on various datasets while maintaining the accuracy drop within 1% and incurring a limited hardware cost.

In summary, this paper makes the following contributions:

- We analyze the effectiveness of wordline-level parallelism compared to the other performance scaling strategies.
- We propose an effective yet lightweight method to mitigate the impact of high-resistance state cell currents on the bitline readout errors.
- We propose to set MAW differently for each layer to maximize wordline-level parallelism across the entire NN model and introduce a systematic approach that finds an optimized MAW for each layer with a small search overhead.
- We evaluate the proposed techniques using three datasets, CIFAR-10, CIFAR-100, and Tiny ImageNet, to demonstrate substantial throughput gains at a minimal area and power cost.

2 BACKGROUND AND MOTIVATION

2.1 RRAM-based Matrix Multiplication Accelerator

RRAM Crossbar Array for Matrix Multiplication. RRAM is a type of non-volatile memory that stores value by programming cell resistances. Usually, RRAM cells are fabricated in the form of the crossbar, which is highly dense and energy-efficient. This crossbar array of RRAM not only works as data storage but also can work as a very efficient in-memory matrix-vector multiplication accelerator. Figure 1(a) shows how RRAM crossbar can be utilized to execute the dot product. Specifically, when voltage V_1 and V_2 is asserted to the first and the second wordline (row), respectively, the resulting current at the end of the bitline (column) is equal to $G_1 \cdot V_1 + G_2 \cdot V_2$, which is the dot product between vector $(G_1 = 1/R_1, G_2 = 1/R_2)$ and (V_1, V_2). Extending this concept, Figure 1(b) shows how a $m \times n$

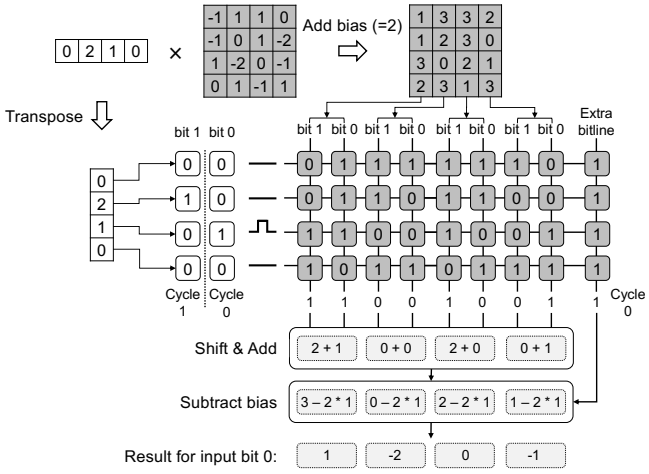

Figure 2: Multi-bit (2-bit) precision matrix-vector multiplication on RRAM crossbar.

RRAM crossbar array (4×4 in this example) executes the matrix-vector multiplication. For this purpose, a wordline driver, a digital-to-analog converter (DAC), drives the value corresponding to a vector-to-be-multiplied, while $m \times n$ RRAM crossbar array stores the value for $m \times n$ binary matrix. Then, the resulting current at each bitline (column) is sampled and latched on S&H (Sample-and-Hold) peripheral. At this point, ADC (Analog-to-Digital Converters), which is usually shared by multiple bitlines, can be utilized to convert the latched current value to a digital number. The need for shift-and-add unit is explained in the following paragraph.

Multi-bit Precision Support. It is technically possible to represent a multi-bit number with a single resistor. However, in practice, representing a multi-bit number with a single resistor often incurs significant programming overhead, peripheral area, and noise uncertainty [6, 13, 23]. As a result, practical designs [2, 27, 31, 33] utilize a single-level RRAM cell (i.e., a single resistor is in one of two states: low-resistance state and high-resistance state). A single multi-bit value is bit-sliced and stored on multiple bitlines. Similarly, it is also common to utilize a binary logic level at the wordline driver (i.e., wordline driver either drives a high or low voltage) and feed inputs in a bit-serial manner [6, 23, 34].

Figure 2 illustrates how RRAM crossbar performs a 2-bit matrix-vector multiplication. As shown in Figure 2, both matrix and vector are transposed. At the first cycle, LSBs of the input vector are converted to the voltage pulses by DAC and sent to the RRAM array. RRAM array performs analog matrix-vector multiplication, and the resulting bitline currents are converted back to digital value by ADC. The ADC outputs are shifted and added to aggregate partial sums corresponding to different bit positions of each weight value. After subtracting bias, which is explained in the next paragraph, the result for input bit 0 is available. In the next cycle, the result for input bit 1 comes out which is shifted and added with that for input bit 0 to produce the final output.

Signed Arithmetic. To represent negative numbers with positive integers, we utilize an efficient weight encoding scheme proposed in [23]. This scheme scales the range of a n-bit fixed-point integer from $[-2^{n-1}, 2^{n-1} - 1]$ to $[0, 2^n - 1]$ by adding a constant, 2^{n-1}.

The scaled positive weight values are mapped onto a crossbar array. If so, the bitline current represents a dot-product result with the biased weights, which should be compensated as much as the bias included in the current to obtain the correct result of the signed arithmetic. This can be done by figuring out the number of 1s in the input and subtracting the product of that number and the bias (2^{n-1}) to the bitline output. For this purpose, an extra bitline is added to each crossbar (in Figure 2). All cells in this bitline are programmed to low-resistance state (LRS) so that the resulting current represents the number of 1s in the inputs. We assume this way of handling signed arithmetic throughout this paper.

2.2 RRAM Accelerator Performance Scaling

Processing Multiple Arrays in Parallel. The performance of RRAM-based accelerator can be improved with various mechanisms. The most straightforward way is to simply utilize multiple numbers of crossbar arrays (along with its peripherals). Such a performance scaling strategy naturally results in throughput gains proportional to the increase in area and power. An alternative and better way to improve performance is to increase the number of wordlines (rows) or bitlines (columns) processed in parallel.

Processing Multiple Bitlines in Parallel. The number of bitlines processed in parallel is closely related to the number of ADCs. Technically, RRAM array computation itself happens in a very short time (often less than five nanoseconds [32, 35]). However, a single ADC can only read a single value every ADC cycle. As a result, if there is only one ADC for RRAM array with 128 bitlines, it takes a total of 128 ADC cycles to retrieve all bitline currents and convert them to digital values. In this case, a single ADC is effectively processing each bitline in series. If multiple ADCs are utilized, multiple bitlines can be processed in parallel, and throughput will improve accordingly. However, this not only leads to an increase in the area and power spent on analog-to-digital conversion but also leads to a proportional increase in the number of S+A (Shift-and-Add) units to avoid them from introducing a bottleneck.

Processing Multiple Wordlines in Parallel. The best way to improve the performance is to increase the number of wordlines processed in parallel. We refer to the maximum number of wordlines processed in parallel as MAW. Increasing the MAW does not change the number of ADCs, and no peripheral hardware needs to be replicated. Instead, ADCs only need to increase its resolution accordingly. For example, if 32 wordlines are processed in parallel instead of 8 wordlines, the resulting value can have a range of (0, 32) instead of (0, 8). As a result, such an extension would require 6-bit ADCs instead of 4-bit ADCs. Increasing the resolution of ADCs leads to an increase in the ADC area and power. However, assuming successive-approximation ADC (SAR-ADC), a widely used ADC type for its cost-efficiency, the area and power cost from increased ADC resolution (2-bit) is much less than that of four ADCs which are required to achieve a similar level of bitline-level parallelism. An increase in the ADC resolution also requires a bitwidth increase in S+A (Shift-and-Add) unit as well as a larger output register for storing wider partial sums. However, their additional cost is also much smaller than that of replicating ADC and the S+A unit.

Quantitative Comparison of Different Scaling Schemes. Figure 3 shows the area and power cost of different performance scaling strategies. We assume a basic unit for matrix multiplication with eight crossbar arrays of single-level RRAM, which we refer to as computing unit (CU). The default CU configuration is shown in Table 1. As a baseline setting, we use 4-bit ADC, assuming the MAW of 8. The area and power models for analog components are derived from ISSAC [23]. We synthesize the digital components with the 40nm TSMC standard cell library and scale their area and power to the same technology as the analog components (32nm). For ADC area scaling across different resolutions, we scale each component area and power from a SAR ADC [16] with the following rules [22]: i) exponential area increase across resolutions for the capacitive DAC, and ii) linear area increase across resolutions for the other components including reference buffer, memory, and clock. We linearly increase the ADC cycle time with the resolution growth. From the figure, we can conclude that exploiting wordline-level parallelism (i.e., increasing the number of wordlines to process in parallel) is a much better way to improve the performance than CU-level parallelism (i.e., increasing the number of CUs) or bitline-level parallelism (i.e., increasing the number of bitlines to process in parallel). However, in practice, this is a challenging task for the reasons outlined in the following section.

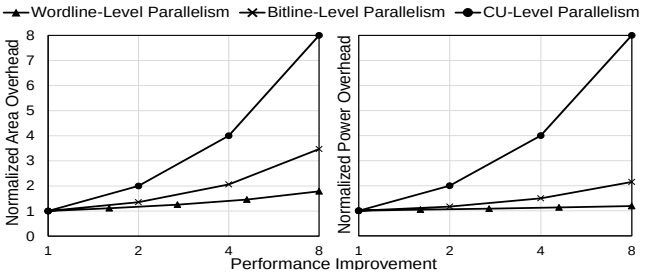

Figure 3: Normalized CU area and power increase for three different performance scaling strategies

Table 1: Baseline CU configuration

CU (1.2GHz, 32nm process)				
Component	**Unit #**	**Specification**	**Power (mW)**	**Area (mm^2)**
RRAM Crossbar	8	128 × 128	2.4	0.0002
DAC	8 × 128	1-bit DAC	4	0.00017
S+H	8 × 128	-	0.01	0.00004
ADC	8	4-bit ADC	1.46	0.0036
S+A	4	-	0.07	0.00035
Input Reg	1	1KB	1.19	0.0049
Output Reg	1	0.28KB	0.18	0.00199
CU total	-	-	**9.31**	**0.0113**

2.3 Challenges in Exploiting Wordline-level Parallelism

The main challenge of exploiting wordline-level parallelism stems from the non-ideality of RRAM cells. As the MAW increases, more errors resulting from each RRAM cell are accumulated and end up inducing errors on the ADC readout output. Specifically, two RRAM characteristics act as the main hindrances to the MAW increase: small RRAM On/Off ratio and cell resistance variation.

Small On/Off ratio. When mapping weight values of neural networks on RRAM crossbar arrays, cells representing logical zero are programmed as high resistance state (HRS), and cells representing

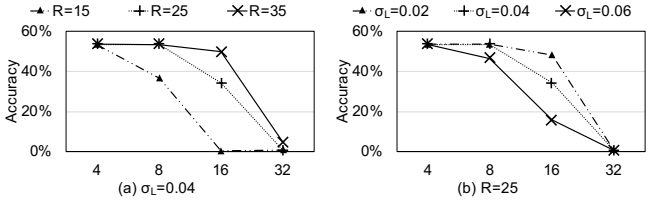

Figure 4: Impact of MAW on Top-1 inference accuracy of ResNet-18 [10] on Tiny Imagenet [18]. RRAM cells with varying characteristics (i.e., On/Off ratio R and LRS Cell resistance deviation σ_L) are used. X-axis represents MAW.

logical ones are programmed as low resistance state (LRS). Cells do not pass any current when the corresponding wordline is not activated (input 0). When the corresponding wordline is activated (input 1), ideally, HRS cells should not pass any current to represent logical zero, and only LRS cells should pass a constant amount of current (I_{LRS}). However, in practice, HRS cells pass non-zero current (I_{HRS}). Such currents can be accumulated, and the accumulated currents may eventually become as much as the current for the LRS cell (I_{LRS}). At that point, the read output is likely to be wrong. This may not be an issue for some RRAM cell types [20, 29] that have a very high ratio (e.g., 10^4) between the resistance of HRS and LRS, called On/Off Ratio. On the other hand, there are cells with much lower On/Off ratio (e.g., 15) with other benefits such as lower leakage current, lower programming cost, higher density, or/and higher retention time [30]. For such cells, this becomes a significant issue. For example, if the On/Off ratio is 15, accumulating 15 currents for HRS state (i.e., 15 I_{HRS}) may result in the ADC output of 1 since it ends up producing the current for LRS state (I_{LRS}).

This is not something one can statically correct by judiciously setting the reference current because different logical values can lead to a single current value. For example, 15 zeros and a single 1 out of 16 total activated wordlines result in a current of $15I_{HRS} + 1I_{LRS}$. Meanwhile, two 1s out of two total activated wordlines also result in a current of $2I_{LRS}$, which is same as the previous one.

Cell Resistance Variation. RRAM cells are typically composed of metal-oxide layers in-between two metal electrodes. By applying appropriate voltage/current to a cell, oxygen vacancies in the metal-oxide layer either form or breakdown conductive filament leading to resistance change. This formation of conductive filament involves randomness, and therefore even with extensive program/read scheme, some variation in the resistance of RRAM cell is inevitable [11]. It has been reported that, for both HRS and LRS cells, the resistance follows the lognormal distribution characterized by the resistance value and resistance deviation for each state [12]. Such statistical randomness in cell resistance leads to a deviation of bitline current. When the MAW is small, this is not really a problem since a single cell's current variation typically falls within the noise margin of ADC. However, when multiple cell variations are accumulated, the resulting value has a higher variance. This is especially problematic since this error follows the lognormal distribution that can exhibit a high deviation.

Effect of the MAW on Accuracy. The non-idealities of RRAM cells explained above are significant problems, and they in fact act as a limiting factor when scaling the MAW. For example, practical RRAM-based DNN accelerators [2, 27, 31–33] does not concurrently process all wordlines of RRAM crossbar. For example, recent RRAM macros can concurrently activate only 9 out of 256 wordlines [2, 31] or 16 out of 512 wordlines [33].

Figure 4 shows the impact of adjusting MAW on neural network model accuracy across different RRAM cells with varying characteristics. For this experiment, we faithfully followed the methodology of DL-RSIM [19] for the accuracy simulation. As the baseline setting, we choose On/Off ratio (R) and resistance deviation of LRS (σ_L) and HRS cell (σ_H) to be 25, 0.04, and 0.4 respectively, assuming an oxide-based (HfO_2) RRAM [9]. For comparison, we also experimented with R of 15 and 35, and σ_L of 0.02 and 0.06. As a baseline setting, we configured reference currents of ADC in a way that maximizes the margin by taking a mean of all possible currents for each logical value [19]. For example, when MAW is 8, we distinguish between logical 1 and logical 2 by judging whether the input signal is smaller or bigger than the halfway point between the mean of I_{LRS}, $I_{LRS} + I_{HRS}$, ..., $I_{LRS} + 7I_{HRS}$ (considering a varying number of activated wordlines from one to eight) and the mean of $2I_{LRS}$, $2I_{LRS} + I_{HRS}$, ..., $2I_{LRS} + 6I_{HRS}$.

As shown in the figure, the model accuracy drops to near-zero once the MAW exceeds R. Even before then, the accuracy drops substantially (e.g., 17% on MAW of 8 for R of 15). Similarly, an increase in the LRS cell resistance variation results in the even more accuracy drop as the MAW increases. This proves our point that the non-ideality of RRAM cells are limiting the potential performance gain from the increased MAW.

3 UNLOCKING WORDLINE-LEVEL PARALLELISM

3.1 Overview

To enable the performance scaling with wordline-level parallelism, non-ideality of RRAM cells need to be properly addressed. Our work presents two techniques to mitigate non-ideality of RRAM cell and enable the RRAM-based neural network accelerators to exploit wordline-level parallelism with much larger MAW without notably degrading accuracy. First, Section 3.2 introduces a technique to efficiently mitigate errors resulting from accumulation of high-resistance state (HRS) cell currents. Second, motivated from the fact that each layer in a neural network model has a different level of tolerance for errors, Section 3.3 presents a mechanism to identify the proper MAW for each layer with minimal impact on model accuracy. With these two techniques, our work presents a new way to scale RRAM-based accelerator performance with a much less hardware cost than the alternative scaling strategies.

3.2 Input-Aware Current Compensation

The most natural way to counter the impact of the unwanted current shift that HRS cells cause is to simply subtract the sum of HRS cell currents from the resulting bitline current. If N wordlines are activated and L among N cells are in LRS while H cells are in HRS, the resulting bitline current I_{BL} is as follows.

$$I_{BL} = L \cdot I_{LRS} + H \cdot I_{HRS}$$

For example, in Figure 5(a), N, L, and H are 3, 1, and 2, respectively, so I_{BL} is $I_{LRS} + 2 \cdot I_{HRS}$. If it is possible to eliminate the second term from the above equation for I_{BL}, identifying L from the remaining

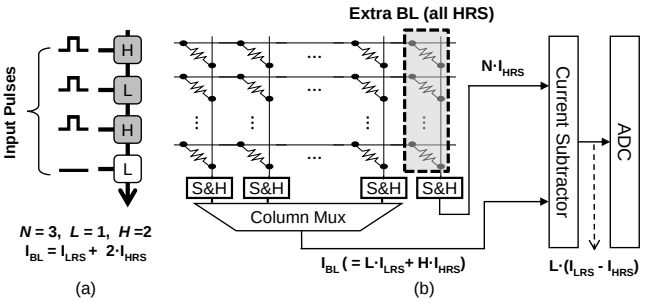

Figure 5: Illustration of the hardware for input-aware current compensation scheme

current (i.e., $L \cdot I_{LRS}$) with ADC is a trivial task assuming no random current variations. However, subtracting the current $H \cdot I_{HRS}$ is a fundamentally infeasible task because H is a value that is only available at the readout. Fortunately, we can rewrite the above expression as follows.

$$\begin{aligned} I_{BL} &= L \cdot I_{LRS} + H \cdot I_{HRS} \\ &= L \cdot I_{LRS} + (N - L) \cdot I_{HRS} \\ &= L(I_{LRS} - I_{HRS}) + N \cdot I_{HRS} \end{aligned}$$

From the equation above, it is obvious that subtracting $N \cdot I_{HRS}$ makes it easy to identify L from remaining currents with a conventional ADC, considering that $(I_{LRS} - I_{HRS})$ is a constant. Subtracting $N \cdot I_{HRS}$ is possible since N is the number of 1s in the input vector rather than the number of 0s in the output vector (i.e., H). Below, we discuss how we efficiently implement the hardware for such current subtraction and identify L with a conventional SAR-ADC.

Step 1: Subtracting $N \cdot I_{HRS}$. To dynamically generate the current $N \cdot I_{HRS}$, our proposal adds an extra bitline to the crossbar array. All the cells in this extra bitline are then programmed to HRS. When the wordline driver activates specific wordlines, the resulting current for this bitline roughly corresponds to $N \cdot I_{HRS}$. Then, we add a current subtractor before each ADC so that the ADC takes the subtraction results (i.e., roughly equals $L(I_{LRS} - I_{HRS})$) as input. Figure 5 illustrates the crossbar structure extended for input-aware current compensation scheme.

One might remember that the original architecture already has a mechanism to count the activated wordlines (N) to handle signed arithmetic (Section 2.1), and wonder if we still need them considering that we can add a new way to count N with HRS cells. The short answer is yes. The newly added HRS-cell-based extra bitline is not well-suited for obtaining N itself. First, the I_{HRS} is small and thus identify N from $N \cdot I_{HRS}$ is challenging. Second, HRS cells tend to have higher resistance variance than LRS cells. For these two reasons, we still need a separate LRS-cell-based extra bitline for counting N, which is used to handle signed arithmetic.

Step 2: Obtaining L with ADC. To maximize a margin between an input signal and a reference current, the reference currents should be a halfway between two adjacent possible input signals. In other words, the optimal reference currents for $L(I_{LRS} - I_{HRS})$ would be $\frac{1}{2}(I_{LRS} - I_{HRS})$, $\frac{3}{2}(I_{LRS} - I_{HRS})$, ... , $\frac{2^{k+1}-3}{2}(I_{LRS} - I_{HRS})$ where k is ADC resolution. A multi-level sense amplifier-based ADC, which is widely used by the recent macros for RRAM-based DNN accelerator [2, 27, 31], usually keeps a separate current generator for each reference current and thus can produce reference currents according to the optimal values. However, a cost-effective type of ADC like a successive approximation register (SAR) ADC uses a binary weighted DAC for reference generation [7, 17]. The binary weighted DAC dynamically produces $2^n - 1$ reference currents by combination of n binary weighted currents (e.g., producing current 1,2, ... , 7 with current 1,2,4). Even for this type of ADC, an optimized set of reference currents can be generated with the addition of a constant current subtractor. Specifically, a current subtractor can be added to subtract the constant current of $\frac{1}{2}(I_{LRS} - I_{HRS})$ from the generated reference current.

Comparison to Prior Work. A recent prior work IA-REF [2, 3] also identified the issue of accumulated HRS cell currents and proposed a way to mitigate this. Specifically, IA-REF first counts the number of activated wordlines (i.e., N = wordlines with 1s) in the digital domain with a digital popcount unit (i.e., adders), and then set the reference currents for the ADC to the halfway points among following currents: $I_{LRS}+(N-1)I_{HRS}$, $2I_{LRS}+(N-2)I_{HRS}$, ... , $(N-1)I_{LRS}+I_{HRS}$, NI_{LRS}[1]. For this purpose, IA-REF prepares a set of N reference currents for each N value, and then pre-programs them to the separate RRAM array. Once the N is determined, IA-REF drives the first N wordlines of this auxiliary RRAM array to retrieve N reference currents. While this is effective for further error reduction on small MAW, the method quickly becomes impractical for larger MAW that it was not designed for. For example, for MAW of 128, this method requires roughly 128×128 RRAM crossbar array as well as a wide (128 elements) digital popcount unit.

3.3 Layer-wise MAW Selection

The input-aware current compensation scheme addresses the potential errors resulting from the accumulation of HRS cell currents. However, random current deviations can still be accumulated and produce an error for the larger MAW. To achieve further performance improvement, we propose to assign different MAW for different layers instead of utilizing the same MAW for the whole neural network model. The intuitions behind the idea is that i) different layers of the same network exhibits different amount of errors for the same MAW, and ii) even the same amount of errors on different layers has a varying impact on the end-to-end model accuracy.

Different layers of the same network induce different amount of errors for the same MAW because their inputs are different. For example, some layers are followed by the popular ReLU activation function which clamps all negative values to zero. For such layers, the number of 0s in the input tends to be larger than the conventional layer. In that case, even for the same MAW, the actual number of activated wordlines (i.e., 1s in the inputs) tend to be small because value zero is represented as eight 0s in its 8-bit representation. We also empirically verified that the same amount of errors on different layers has a varying impact on the end-to-end model accuracy. For example, we find that the very first layer of convolutional neural network models tends to be much more sensitive to errors. This is

[1] Technically, IA-REF adopts a custom sensing scheme named distance-racing sensing scheme. This scheme interprets I_{BL} by comparing the distance between I_{BL} and the higher reference current and the distance between I_{BL} and the lower reference current. While the exact implementation is different, the distance-racing sensing scheme and the conventional sensing scheme are conceptually equivalent, and thus we assumed the latter for concise explanation.

natural considering that the first layer is responsible for extracting key features directly from the input.

Overview. We propose a systematic approach to identify the proper MAW for each layer in the given NN model. Specifically, we enable the user to provide a target accuracy loss and find the MAW for each layer which results in less than the target accuracy loss on the validation set. The process consists of two phases: layer-wise sensitivity profiling and iterative search to find proper MAW.

Profiling. For each layer, we profile the impact on the end-to-end model accuracy as the specific layer's MAW increases. Specifically, we measure change, not drop, as sometimes accuracy accidentally increases from errors resulting from the MAW increase. Figure 6(a) illustrates the profiling process for the first layer of a 4-layer neural network model. For this purpose, we change the MAW for the first layer (e.g., 8, 16, 32) while the other layers run in a pure software mode with no error, and then measure the deviation (labeled "Dev" in Figure 6(a)) from the ideal end-to-end accuracy (i.e., the accuracy for the case where this model runs without any error). Then, the marginal cost for each MAW is calculated by comparing the deviation with the immediately previous level. For example, the cost to increase the MAW of Layer 1 from 16 to 32 is 0.3 (=0.9-0.6). The cost for the minimum supported MAW (8), which we denote as base MAW, is not defined.

Iterative Search. Once the profiling for every layer finishes, the iterative search starts. Starting from the state 0 which is defined as the tuple of base MAW (i.e., (8, 8, 8, 8) in Figure 6(b)), the search defines the following states by increasing the MAW of a layer that has the least marginal cost. For example, Figure 6(b) visualizes how state 1, 2, 3 are defined. At state 0, all the layers are initialized with the base MAW (8). Since the marginal cost of increasing the MAW for the layer 4 is the least among available options (gray-colored cells), state 1 is defined as (8, 8, 8, 16). The next state (state 2) is then defined as (8, 8, 8, 32) since the marginal cost for increasing the MAW for layer 4 is the least among available choices once again. State 3 is then defined as (8, 16, 8, 32) as the layer 2 now has the least marginal cost of increasing the MAW. This process continues and a series of 8 states are defined in this example.

To determine when to stop advancing states, we introduce a user parameter called target accuracy loss. We move forward as far as the simulated validation accuracy does not degrade more than the target accuracy loss from the ideal accuracy (i.e., accuracy without any error). Examining the validation accuracy for every state will require a multiple rounds of validations that is identical to the number of total states. To avoid such a huge computational cost, we perform a binary search to find the best policy that satisfies the validation accuracy constraint. Figure 6(c) illustrates how the binary search reduces the total number of validations.

At the first round, we proceed as many as the half of the total possible states (i.e., [# of layers] × [# of possible MAW values - 1]) and then perform the validation of the corresponding policy. As the accuracy drop is less than the target accuracy loss (0.5%), we move further as much as the half of the remaining states, and reach state 6. However, at this time, the validation accuracy drops from the baseline more than the target accuracy loss. Thus, we move backward to the halfway point between the current state and the last visited state (State 5 in this example). We repeat the process until we cannot move anymore. The resulting policy would be the last one that has satisfied the target accuracy loss. In the example, the search stops at State 5 and outputs its policy as it satisfies the accuracy constraint. The binary search limits the required rounds of validation to be the log of the number of the total states.

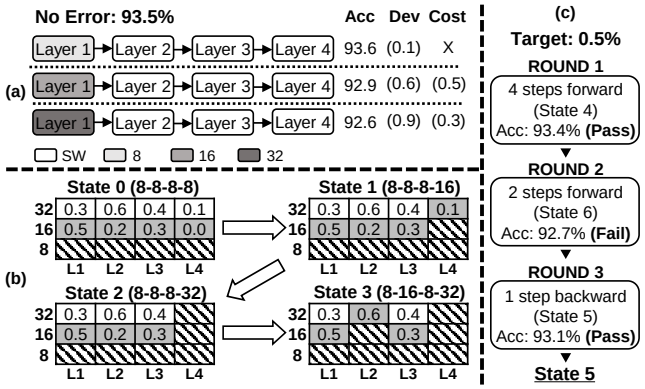

Figure 6: Example for MAW selection process on a 4-layer network. (a) shows the process of the sensitivity profiling of the layer 1. (b) shows how the iterative search is performed utilizing profiling results. (c) shows how we determine the best policy satisfying the validation constraint.

Note that this policy does not guarantee the exact accuracy loss in the real test set. It is well known that NN algorithms' behavior is slightly different on validation set and the test set, and thus there is no real way to guarantee the actual accuracy loss on the test set resulted from our MAW selection policy. Still, if the validation set is sufficiently representative, it is expected that the similar accuracy loss will be observed in the test set as well.

Relation between the MAW and ADC Resolution. As outlined in Section 2.2, support for the increased MAW requires an increase in the ADC resolution. For example, to support the MAW of 128, ADC resolution needs to be 8-bit so that it can properly support 0-128 range. However, observations from real neural network model inputs indicate that it is rare for the resulting bitline currents from the crossbar array to exceed certain value. This is natural in some sense. For example, if the uniform distribution is assumed for the 8-bit input vector (activation of the previous layer), about half of the bits (e.g., 4 out of 8-bit) will be zero. The same may apply for the 8-bit weights. Since the bitline current accumulates 1 if and only if both the currently processed bits in the input and the weight are 1, the chance of the bitline accumulating 1 from a single wordline activation is roughly 1/4. As a result, for a specific MAW, the expected value for the ADC outcome is 1/4 of it. In this case, utilizing a ADC with a single bit less resolution (i.e., the ADC can convert to [0, MAW/2]) and clipping the signal when it exceeds the supported range has an negligible impact on the end-to-end accuracy. Furthermore, we find that it is also a good idea to give up an ability to detect MAW/2 value for the extra bit reduction. In summary, we recommend the usage of ADC with $(\log_2 M - 1)$-bit resolution, not $(log_2 M + 1)$-bit resolution, where M represents the largest among MAWs of all layers. Tightening the ADC resolution affects not only area and power but also the speedup since ADC latency, scaling linearly with the resolution, is in the critical path. Section 4.3 demonstrates the validity of this design choice.

4 EVALUATION

4.1 Methodology

Workloads. To evaluate the effectiveness of the two proposed schemes, we conducted experiments on three datasets with different complexity (CIFAR-10, CIFAR-100 [15], Tiny ImageNet [18]), and used two CNN models (ResNet-18 [10], VGG-11 [25]). By default, we used 8-bit for both weight and activation values of the networks. We have trained the models using floating-point values and fine-tuned them for fixed-point quantization. For weight quantization, we clipped the weight values with a fixed threshold for each layer that minimizes the mean-squared-error between the floating point and the quantized values [24, 28]. For activation quantization, we used parameterized clipping activation [5].

Accuracy Simulation. To estimate the end-to-end inference accuracy of RRAM-based accelerator, we implemented a custom simulator faithfully following the methodology of DL-RSIM [19] and integrated it into the Pytorch [21]. By default, we set the On/Off ratio of RRAM cell to be 25 [9]. We set the resistance variance for HRS and LRS resistance as 0.4 and 0.04, respectively [9].

Baseline Architecture. We assume a generic RRAM-based DNN accelerator which is organized in a hierarchical structure similar to a well-known prior work ISAAC [23]. Section 2.1 explained the structure for the computing unit (CU), a basic building block for the DNN accelerator. In the baseline architecture, 12 CUs comprise a single processing element (PE) along with various peripherals such as weight/activation buffers as well as miscellaneous function units including sigmoid units, pooling units, etc, and there are 336 PEs in total. When measuring the performance, we followed the mapping strategy based on the inter-layer pipeline of ISAAC [23] which minimizes the buffer requirement between PEs.

4.2 Input-Aware Current Compensation

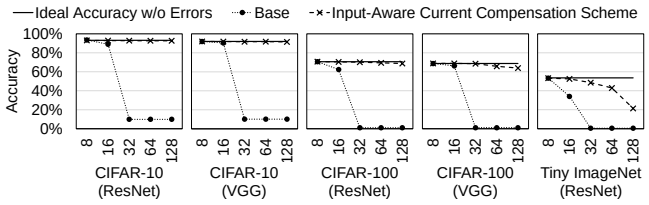

Figure 7: Impact of input-aware current compensation scheme on the accuracy across varying MAW (x-axis).

To measure the effectiveness of input-aware current compensation scheme (in Section 3.2), we change the MAW from 8 to 128 and run the whole model with the specific MAW, and then compare the accuracy degradation from the baseline. We increment the ADC resolution by one every time the MAW gets doubled. Figure 7 shows the result for this experiment. For all datasets, the accuracy of baseline begins to noticeably degrade from the MAW of 16 and becomes unusuable when the MAW is larger than 16. On the other hand, with our proposed input-aware current compensation scheme, the accuracy remains fairly high with the increased MAW. Especially for CIFAR-10, the accuracy is nearly identical to the ideal case even with the MAW of 128. On CIFAR-100 and Tiny ImageNet, however, the accuracy continuously decreases although much more gently

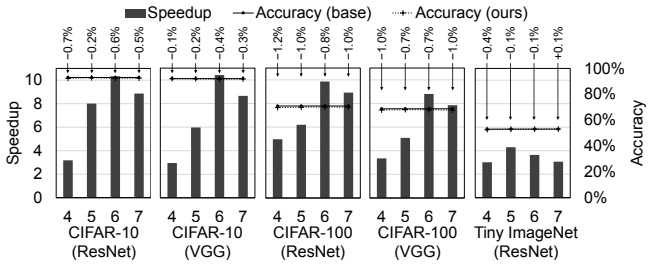

Figure 8: Speedups and accuracy degradation of our proposed scheme (input-aware current compensation + layer-wise MAW selection) across different workloads. X-axis represents the ADC resolution.

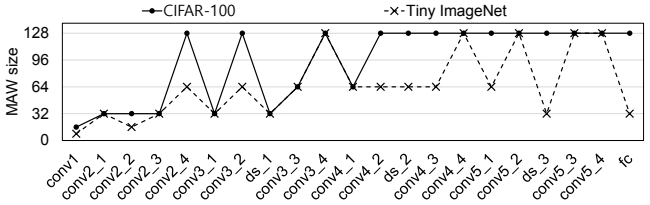

Figure 9: Selected MAW across layers for CIFAR-100 and Tiny ImageNet on ResNet-18 Network. 6-bit ADC is used.

compared to the baseline, which demonstrates that the current compensation alone is not sufficient to achieve significant performance improvements maintaining the accuracy. The MAW can increase only up to 32 and 16 for CIFAR-100 and Tiny ImageNet which translates to 3.3× and 1.6× speedup, respectively.

4.3 Input-Aware Current Compensation with Layer-wise MAW Selection.

Figure 8 shows the performance and accuracy degradation for cases where both input-aware current compensation scheme and the layer-wise MAW selection policy are applied. Specifically, we included the MAW of 8, 16, 32, 64, and 128 to the search space and set the target accuracy loss to 1%. Furthermore, the same experiments were conducted with the different ADC resolutions (4-8 bits) to observe its impact (see discussion at the end of Section 3.3). As shown in the figure, our scheme achieves the substantial speedup across all workloads, especially on the recommended ADC resolution ($log_2 M - 1 = 6$). Such a speedup is all achieved within around 1% accuracy degradation. Note that all bars in the figure achieve the limited accuracy degradation since our layer-wise MAW selection policy always limits the accuracy degradation by forgoing the potential speedup resulting from the larger MAW.

Figure 8 also explores the impact of ADC resolution. The increased ADC resolution reduces the errors from clipping, and can potentially enable the higher MAW for certain layers. However, at the same time, it reduces the throughput of all layers because a larger ADC resolution leads to a longer ADC latency. As discussed in Section 3.3, the ADC resolution of ($log_2 M - 1 = 6$) results in the best performance in many cases. This is because such a ADC resolution minimizes the case of accuracy degradation from clipping while also providing the better ADC latency compared to the full-resolution case (8-bit for MAW = 128).

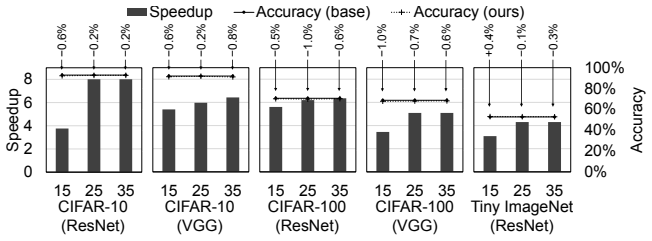

Figure 10: Sensitivity to cell On/Off ratio.

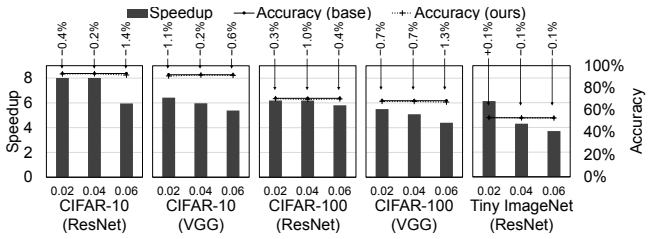

Figure 11: Sensitivity to cell resistance deviation.

For Tiny ImageNet dataset, the result is slightly better for 5-bit ADC. This is because the MAW of 128 is rarely used to limit the accuracy degradation. Figure 9 shows the selected MAW across layers for CIFAR-100 and Tiny ImageNet on ResNet-18. The figure shows that the MAW of 128 is only selected for 5 layers out of 21 layers. In this case, the choice of 5-bit ADC does not degrade the accuracy much while providing slightly better speedup for all layers. On the other hand, the figure shows that the MAW of 128 is used on many layers for CIFAR-100 dataset.

Note that the result on Figure 8 is much better than the ones presented in Figure 7. It is easy to see the effectiveness of the layer-wise MAW selection for CIFAR-100 and Tiny ImageNet as only limited performance gain (3.3×, 1.6×) has been achieved solely with the current compensation while Figure 8 shows around 9×, 4× speedup for the same datasets.

Additional Hardware Cost. The performance gain of the two proposed techniques incurs very small hardware cost. The increase of area and power of CU when using 6-bit ADC is measured to be 25.3% and 8.7% following the way discussed in Section 2.2. This cost translates to 8.2% and 5.3% of the chip-level overhead as CUs account for 37.5% and 62.8% of the total chip area and power respectively, assuming that non-CU components in the chip follow the area and power model reported in [23].

4.4 Sensitivity Studies

Sensitivity to Cell On/Off Ratio. We changed On/Off ratio from the baseline setting (i.e., 25) to 15 and 35. For the On/Off ratio of 15, the baseline MAW was adjusted to 4 from 8, since the MAW 8 resulted in the severe accuracy degradation. The MAW of 16 still resulted in the severe accuracy degradation even with the On/Off-ratio = 35, and the baseline MAW was set to 8 for that case. The Figure 10 shows the experimental result. Generally speaking, our scheme achieves higher speedup on higher On/Off ratio. When the On/Off ratio is extremely low, the MAW is severely limited and the amount of speedup that our scheme can achieve becomes limited as well. On the other hand, a higher On/Off ratio lets our layer-wise MAW selection scheme aggressively utilize a large MAW.

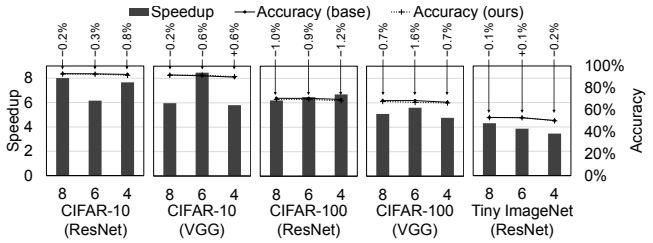

Figure 12: Sensitivity to quantization levels. The x-axis represents the quantized bitwidth for both activations and weights. We keep activation values of the first layer as 8-bit regardless of the quantization level following the practice of many other studies on DNN quantization[5, 14, 36].

Sensitivity to Cell Resistance Deviation. We changed σ from the baseline setting (i.e., 0.04) to 0.02 and 0.06, and then observe the impact of such changes on the speedup. Figure 11 shows that the speedup tends to decrease as σ is increased. This is natural since the larger σ indicates the larger chance of accumulated errors from cell resistance deviation exceeding the ADC sensing margin. Consequently, our layer-wise MAW selection policy ends up choosing the more conservative MAW, which results in the less speedup. Still, our scheme shows over 3× speedup across all evaluated cases.

Sensitivity to Quantization Levels. For the efficient computation, it is common to perform further quantization on neural network models. Figure 12 shows that our technique provides substantial speedup regardless of the quantization levels.

5 CONCLUSION

We propose two key techniques that enables the RRAM-based accelerators to exploit further wordline-level parallelism and improve the performance at a low cost. First, we reduce a large part of the bitline current readout errors with a cost-effective error mitigation mechanism called input-aware current compensation that alleviates the accumulation of the non-zero HRS cell currents. Second we propose a layer-wise MAW selection scheme exploiting the fact that different layer has different resilience to the increase of the MAW. We introduce an algorithm to find a proper MAW for each layer under an accuracy constraint by profiling each layer's sensitivity to the end-to-end accuracy. Our evaluation demonstrates that, when input-aware current compensation and layer-wise MAW selection are both applied, we achieve $3-10\times$ speedup with less than 1% accuracy degradation across three datasets. This benefit comes with a small cost of only 8.2% and 5.3% increase in area and power consumption, respectively, at the chip level.

ACKNOWLEDGMENTS

This research was supported by Samsung Electronics and also by Nano-Material Technology Development Program through National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2016M3A7B4909668). The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Jae W. Lee is the corresponding author.

REFERENCES

[1] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. 2019. PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. Association for Computing Machinery, 715–731.

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

[3] Wei-Hao Chen, Chunmeng Dou, Kai-Xiang Li, Wei-Yu Lin, Pin-Yi Li, Jian-Hao Huang, Jing-Hong Wang, Wei-Chen Wei, Cheng-Xin Xue, Yen-Cheng Chiu, Frederick Chen, Chorng-Jung Lin, Ren-Shuo Liu, Chih-Cheng Hsieh, Kea-Tiong Tang, J. Yang, Mon-Shu Ho, and Meng-Fan Chang. 2019. CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. *Nature Electronics* 2 (08 2019). https://doi.org/10.1038/s41928-019-0288-0

[4] 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 *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. Association for Computing Machinery, 27–39.

[5] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks.

[6] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52)*. Association for Computing Machinery, 114–125.

[7] Chun-Cheng Liu, Yi-Ting Huang, Guan-Ying Huang, Soon-Jyh Chang, Chung-Ming Huang, and Chih-Haur Huang. 2009. A 6-bit 220-MS/s time-interleaving SAR ADC in 0.18-μm digital CMOS process. In *2009 International Symposium on VLSI Design, Automation and Test*. 215–218.

[8] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.

[9] Daniele Garbin, E. Vianello, Olivier Bichler, Quentin Rafhay, Christian Gamrat, Gerard Ghibaudo, Barbara DeSalvo, and Luca Perniola. 2015. HfO₂-Based OxRAM Devices as Synapses for Convolutional Neural Networks. *Electron Devices, IEEE Transactions on* 62 (2015).

[10] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[11] C. Ho, S. Chang, C. Huang, Y. Chuang, S. Lim, M. Hsieh, S. Chang, and H. Liao. 2017. Integrated HfO2-RRAM to achieve highly reliable, greener, faster, cost-effective, and scaled devices. In *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2.6.1–2.6.4.

[12] K.C. Hsu, Feng-Min Lee, Y.Y. Lin, E.K. Lai, J.Y. Wu, D.Y. Lee, Min-Hee Lee, H.-L Lung, K.Y. Hsieh, and C.Y. Lu. 2015. A Study of Array Resistance Distribution and a Novel Operation Algorithm for WOx ReRAM Memory. In *Proceedings of International Conference on Solid State Devices and Materials*.

[13] 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 *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *J. Mach. Learn. Res.* 18, 1 (Jan. 2017), 6869–6898.

[15] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images.

[16] Lukas Kull, Thomas Toifl, Martin L. Schmatz, Pier Andrea Francese, Christian Menolfi, Matthias Braendli, Marcel A. Kossel, Thomas Morf, Toke Meyer Andersen, and Yusuf 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 (2013), 3049–3058.

[17] Lukas Kull, Thomas Toifl, Martin L. Schmatz, Pier Andrea Francese, Christian Menolfi, Matthias Braendli, Marcel A. Kossel, Thomas Morf, Toke Meyer Andersen, and Yusuf 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 (2013), 3049–3058.

[18] Ya Le and Xuan Yang. 2015. Tiny ImageNet Visual Recognition Challenge. https://tiny-imagenet.herokuapp.com/.

[19] M. Lin, H. Cheng, W. Lin, T. Yang, I. Tseng, C. Yang, H. Hu, H. Chang, H. Li, and M. Chang. 2018. DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[20] Dilip Maiti, Sudipto Debnath, Sk Masum Nawaz, Bapi Dey, Enakhi Dinda, Dipanwita Roy, Sudipta Ray, A. Mallik, and Syed Arshad Hussain. 2017. Composition-dependent nanoelectronics of amido-phenazines: non-volatile RRAM and WORM memory devices. *Scientific Reports* 7 (12 2017). https://doi.org/10.1038/s41598-017-13754-w

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

[22] 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 (2011), 1736–1748.

[23] 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 *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. Association for Computing Machinery, 14–26.

[24] S. Shin, K. Hwang, and W. Sung. 2016. Fixed-point performance analysis of recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 976–980.

[25] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[26] L. Song, X. Qian, H. Li, and Y. Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 541–552.

[27] F. Su, W. Chen, L. Xia, C. Lo, T. Tang, Z. Wang, K. Hsu, M. Cheng, J. Li, Y. Xie, Y. Wang, M. Chang, H. Yang, and Y. Liu. 2017. A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory. In *2017 Symposium on VLSI Technology*. T260–T261.

[28] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. 2015. Resiliency of Deep Neural Networks under Quantization. *ArXiv* abs/1511.06488 (2015).

[29] Yi-Hsin Ting, Jui-Yuan Chen, Chun-Wei Huang, Ting-Kai Huang, Cheng-Yu Hsieh, and Wen-Wei Wu. 2017. Observation of Resistive Switching Behavior in Crossbar Core–Shell Ni/NiO Nanowires Memristor. *Small* 14 (12 2017). https://doi.org/10.1002/smll.201703153

[30] Hong Wang and Xiaobing Yan. 2019. Overview of Resistive Random Access Memory (RRAM): Materials, Filament Mechanisms, Performance Optimization, and Prospects. *physica status solidi (RRL) – Rapid Research Letters* 13, 9 (2019), 1900073. https://doi.org/10.1002/pssr.201900073

[31] C. Xue, W. Chen, J. Liu, J. Li, W. Lin, W. Lin, J. Wang, W. Wei, T. Chang, T. Chang, T. Huang, H. Kao, S. Wei, Y. Chiu, C. Lee, C. Lo, Y. King, C. Lin, R. Liu, C. Hsieh, K. Tang, and M. Chang. 2019. 24.1 A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6ns Parallel MAC Computing Time for CNN Based AI Edge Processors. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. 388–390.

[32] C. Xue, W. Chen, J. Liu, J. Li, W. Lin, W. Lin, J. Wang, W. Wei, T. Huang, T. Chang, T. Chang, H. Kao, Y. Chiu, C. Lee, Y. King, C. Lin, R. Liu, C. Hsieh, K. Tang, and M. Chang. 2020. Embedded 1-Mb ReRAM-Based Computing-in- Memory Macro With Multibit Input and Weight for CNN-Based AI Edge Processors. *IEEE Journal of Solid-State Circuits* 55, 1 (2020), 203–215.

[33] C. Xue, T. Huang, J. Liu, T. Chang, H. Kao, J. Wang, T. Liu, S. Wei, S. Huang, W. Wei, Y. Chen, T. Hsu, Y. Chen, Y. Lo, T. Wen, C. Lo, R. Liu, C. Hsieh, K. Tang, and M. Chang. 2020. 15.4 A 22nm 2Mb ReRAM Compute-in-Memory Macro with 121-28TOPS/W for Multibit MAC Computing for Tiny AI Edge Devices. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. 244–246.

[34] Tzu-Hsien Yang, Hsiang-Yun Cheng, Chia-Lin Yang, I-Ching Tseng, Han-Wen Hu, Hung-Sheng Chang, and Hsiang-Pang Li. 2019. Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks. In *Proceedings of the 46th International Symposium on Computer Architecture*. Association for Computing Machinery, 236–249.

[35] Shihui Yin, Xiaoyu Sun, Shimeng Yu, and Jae sun Seo. 2019. High-Throughput In-Memory Computing for Binary Deep Neural Networks with Monolithically Integrated RRAM and 90nm CMOS. *ArXiv* abs/1909.07514 (2019).

[36] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *ArXiv* abs/1606.06160 (2016).