A System-Level Simulator for RRAM-Based Neuromorphic Computing Chips

MATTHEW KAY FEI LEE, YINGNAN CUI, THANNIRMALAI SOMU, and TAO LUO,

Institute of High Performance Computing, Singapore

JUN ZHOU, National University of Singapore

WAI TENG TANG, Institute of High Performance Computing, Singapore

WENG-FAI WONG, National University of Singapore

RICK SIOW MONG GOH, Institute of High Performance Computing, Singapore

Advances in non-volatile resistive switching random access memory (RRAM) have made it a promising memory technology with potential applications in low-power and embedded in-memory computing devices owing to a number of advantages such as low-energy consumption, low area cost and good scaling. There have been proposals to employ RRAM in architecting chips for neuromorphic computing and artificial neural networks where matrix-vector multiplication can be computed in the analog domain in a single timestep. However, it is challenging to employ RRAM devices in neuromorphic chips owing to the non-ideal behavior of RRAM. In this article, we propose a cycle-accurate and scalable system-level simulator that can be used to study the effects of using RRAM devices in neuromorphic computing chips. The simulator models a spatial neuromorphic chip architecture containing many neural cores with RRAM crossbars connected via a Network-on-Chip (NoC). We focus on system-level simulation and demonstrate the effectiveness of our simulator in understanding how non-linear RRAM effects such as stuck-at-faults (SAFs), write variability, and random telegraph noise (RTN) can impact an application's behavior. By using our simulator, we show that RTN and write variability can have adverse effects on an application. Nevertheless, we show that these effects can be mitigated through proper design choices and the implementation of a write-verify scheme.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Computer systems organization** → **Neural networks**; • **Hardware** → **Non-volatile memory**; **Simulation and emulation**;

Additional Key Words and Phrases: Neuromorphic computing, simulator, RRAM

This research is supported by programmatic grant no. A1687b0033 from the Singapore Government Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain).

Authors' addresses: M. K. F. Lee and Y. Cui, Institute of High Performance Computing, 1 Fusionopolis Way, Connexis North 16-16, 138632, Singapore; emails: {matthew_lee, cuiyn}@ihpc.a-star.edu.sg; T. Somu, Institute of High Performance Computing; email: malai_somu@ihpc.a-star.edu.sg; T. Luo, W. T. Tang, and R. S. M. Goh, Institute of High Performance Computing, Singapore; emails: luo_tao@ihpc.a-star.edu.sg, tangwt@ihpc.astar.edu.sg, gohsm@ihpc.a-star.edu.sg; J. Zhou, National University of Singapore, School of Computing, 13 Computing Drive, 117417; email: zhoujun@comp.nus.edu.sg; W.-F. Wong, National University of Singapore, School of Computing; email: wongwfw@nus.edu.sg.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1544-3566/2019/01-ART64

<https://doi.org/10.1145/3291054>

ACM Reference format:

Matthew Kay Fei Lee, Yingnan Cui, Thannirmalai Somu, Tao Luo, Jun Zhou, Wai Teng Tang, Weng-Fai Wong, and Rick Siow Mong Goh. 2019. A System-Level Simulator for RRAM-Based Neuromorphic Computing Chips. *ACM Trans. Archit. Code Optim.* 15, 4, Article 64 (January 2019), 24 pages.

<https://doi.org/10.1145/3291054>

1 INTRODUCTION

Neuromorphic computing is a domain-specific computing approach that uses analog, digital, or mixed-mode integrated circuits to mimic biological architectures of the neural system, including neurons, axons, synapses, and dendrites [40]. Neurons whose inputs and outputs are spikes are used in neuromorphic computing; the resulting spike-based or spiking neural networks (SNNs) are often regarded as third-generation neural networks [39]. Special-purpose built hardware for neuromorphic computing includes the HiCANN chip [12], NeuroGrid [7], SpiNNaker [9], and IBM's TrueNorth chip [17]. SpiNNaker and TrueNorth are fully digital; HiCANN and NeuroGrid are analog or partially analog in design. In TrueNorth, 4096 neurosynaptic cores of size 256×256 are interconnected by an intra-chip network. Using TrueNorth to implement SNNs, Esser et al. demonstrated good accuracies in real-world application benchmarks [22].

Concurrent with the developments in neuromorphic computing, advances in non-volatile resistive switching random access memory (RRAM) have made it a suitable memory technology for realizing neuromorphic computing architectures [11]. For instance, RRAM-based neuromorphic computing hardware has been proposed in [19, 23, 25]. Apart from advantages such as low operating power, high speed and density, memristive and RRAM-based crossbars have been proposed as energy-efficient dot-product engines. These can be used to perform matrix-vector multiplication operations efficiently in the analog domain through current sums [4, 6, 15]. Such approaches are suitable for low-power embedded devices targeting neuromorphic or neural network applications.

However, there remains significant challenges with using RRAM devices as memory cells and as dot-product engines in the design of neuromorphic processors. Non-ideal characteristics of the RRAM, read and write variability, and read-out noise will affect the overall performance of an application. These effects are very difficult to quantify without a system-level simulator.

In this article, we describe a system-level simulator for an RRAM-based neuromorphic computing chip. Many other previous works have focused on simulating circuit-level or architectural-level features, or do not simulate many of the RRAM characteristics [4, 6, 14, 26]. Many of these are also not scalable and are unable to shed light on how large-scale real-world applications would perform on their hardware.

Our focus is on providing a simulator framework for neuromorphic computing chips that integrated a model for neurosynaptic cores with models of Networks-on-Chip (NoCs). Our simulator takes into account various non-ideal RRAM characteristics, including stuck-at-faults (SAFs), write variability, and random telegraph noise (RTN). While it is not our aim in this article to discuss the various design trade-offs for an actual neuromorphic chip, we will show that the simulator is a useful tool for shedding light on how different design choices can impact the final product.

Our contributions are as follows:

- We describe a system-level simulator for an RRAM-based neuromorphic chip that supports full-scale simulation of large real-world applications. The simulator can be used as a tool to evaluate and optimize design choices for an RRAM neuromorphic chip.
- We model the many non-ideal characteristics of RRAM, such as SAFs, write variability, and RTN. To the best of our knowledge, ours is the first end-to-end simulator that supports many non-ideal characteristics for full application simulation.

Fig. 1. Structure of a spike-based neural network (SNN).

- By using our simulator, we discovered that RTN, which is present in RRAM read-outs, is an important factor that could contribute to degradation in application accuracy. To our knowledge, the impact of RTN on applications running on a many-core RRAM neuromorphic chip has not been investigated in previous works.
- We also use the simulator to study the accuracy-rewrite trade-offs when using a write-verify scheme as a mitigation strategy for write variability.
- Our simulator is highly configurable and scalable. Various settings — such as core size, noise model, and network topology — can be configured. Large applications using more than 3000 cores can be simulated and are limited only by CPU and memory resources.

The rest of the article is organized as follows. Section 2 motivates the need for a system-level simulator and provides background information on SNNs, RRAM and the neuromorphic computing architecture. Section 3 details the design of our simulator, with a focus on its configurability and scalability. It also discusses which non-ideal effects of the RRAM are simulated and introduces the NoC model. Section 4 demonstrates and discusses the experimental results on the simulator. Section 5 reviews previous related work and Section 6 presents our conclusions.

2 BACKGROUND AND MOTIVATION

2.1 Spiking Neural Networks

SNNs are generally considered to be the third-generation neural network model. They differ from traditional second-generation artificial neural networks (ANNs) in terms of closeness to biological realism [39]. One of most distinguishing features of SNNs is that the inputs and outputs of the neurons in SNNs are a series of spikes with unit amplitude. Figure 1 shows the structure of an SNN. In Figure 1(a), an example SNN that is fully connected is shown. Its high-level structure does not differ from a fully connected ANN. However, the computation performed is different, as depicted in Figure 1(b), which shows the computational model of an SNN neuron. In an SNN, when a neuron receives spikes from its dendritic inputs, the membrane voltage of the neuron changes according to their combined synaptic weights. When the neuron membrane voltage exceeds a given threshold, it will fire a spike to the given axon output. Various neuron models have been proposed to describe the changing of membrane voltage and the firing of spikes in SNN [32]. In our simulator, we adopt the *leaky-integrate-and-fire* (LIF) neuron model because it achieves a good balance between accuracy and complexity and is easy to implement in hardware. Nevertheless, the simulator is flexible enough to be extended with other neuron models as required by users. The LIF model is also commonly used in previous works on SNNs [4, 22].

Fig. 2. Low- and high-resistance states of an RRAM device.

Equation (1) describes how the membrane voltage in an LIF neuron is updated in our simulator. A single neuron begins a timestep with membrane voltage v_t . The change in membrane voltage due to input synapse activity is Δv_t , and a constant leak v_{leak} is subtracted. The voltage after the *leak and integrate* phase is denoted by v'_t . For a given LIF neuron, the change in membrane voltage is defined in Equation (2), where $x_{i,t}$ is the i th input channel to the neuron at timestep t , and w_i is the weight of the input channel to the neuron. Note that $x_{i,t}$ is a binary variable, that is, $x_{i,t} = 1$, when there is a spike received on the i th input channel at timestep t and $x_{i,t} = 0$ otherwise.

$$v'_t = v_t + \Delta v_t - v_{leak} \quad (1)$$

$$\Delta v_t = \sum_{i=0}^n x_{i,t} w_i \quad (2)$$

Equation (3) is used to determine when an LIF neuron will fire. Essentially, a spike is fired during timestep t if $v'_t \geq v_{th}$, where v_{th} is the given threshold voltage of the neuron. A spike event will set the indicator variable s_t to 1 and the post-firing voltage v_{t+1} to $v'_t - v_{th}$. Otherwise, no spike is emitted ($s_t = 0$) and $v_{t+1} = v'_t$.

$$(s_t, v_{t+1}) = \begin{cases} (0, v'_t), & \text{if } v'_t < v_{th} \\ (1, v'_t - v_{th}), & \text{otherwise.} \end{cases} \quad (3)$$

2.2 Non-Volatile Resistive RAM

The RRAM device is a metal-insulator-metal (MIM) configuration that exhibits hysteretic resistive switching behavior. In this section, we describe the operating principle of the metal-oxide RRAM, which is a structure where a metal-oxide layer is sandwiched between two electrodes [11]. This structure exhibits a resistive switching phenomenon between a low-resistance state (LRS) and a high-resistance state (HRS), which is caused by oxygen migration processes. Different resistance values can be written to the device to represent different weight values. As shown in Figure 2, conductive filaments appear during the *Set* process due to oxygen drift to the metal electrode-oxide interfacial reservoir, giving rise to an LRS. When a *Reset* voltage is applied, oxygen ions drift back and partially fill up the vacancies, causing a gap in the conductive filament and resulting in an HRS.

There are several advantages that non-volatile RRAM technology offers. These are high speed and density, energy efficiency, and compatibility with CMOS technology. In this article, we consider the use of RRAM memory technology for storing synaptic weights as well as for

Fig. 3. RRAM-based neurosynaptic core architecture.

performing weight integration in the analog domain using the sum of currents through the RRAM devices.

2.3 RRAM-Based Neuromorphic Computing Architecture

Figure 3 shows the classic architecture of a neurosynaptic core employing an RRAM crossbar array, which is also used in previous work [4, 6, 15, 19]. The RRAM cells are aligned in a crossbar array, where the *word* lines (horizontal lines) represent axons and the *bit* lines (vertical lines) are the dendrites. Each RRAM cell represents a synapse and its conductance value represents the weight of the synapse. When a spike arrives at an axon, the voltage of the corresponding word line is set to high. The current flowing through the bit lines is determined by the conductance of the RRAM cells. The current of each bit line, which represents an aggregation of the input currents to the destination neuron, is measured and converted to a digital signal by an analog-digital (A/D) converter. The digital value is then forwarded to the CMOS neuron, which realizes the LIF neuron model discussed in Section 2.1.

Since the conductance values of RRAM cells are unidirectional, it is impossible to store positive and negative weights in one bit line. Therefore, a common solution is to assign two distinct bit lines, called a *differential pair*, to each neuron [16]. In a differential pair, one line is for the storage of positive weights and the other is for negative weights. As shown in Figure 3, $W_{i,j}^+$ represents the positive weight and $W_{i,j}^-$ represents the negative weight. In this work, we also adopt the neurosynaptic core architecture shown in Figure 3. Many such cores are connected with an NoC.

2.4 Motivation

Figure 4 demonstrates the need for a system-level simulator for a neuromorphic chip based on the RRAM crossbar array described earlier. It shows the influence of core size, noise model, and network topology on the performance of an application. In this example, a 3-layer fully connected SNN with 512, 256, and 10 neurons in each respective layer is trained on the MNIST benchmark. The trained SNN is executed on our simulator using three different chip configurations, as shown

Fig. 4. MNIST SNN under different simulation conditions.

in Figure 4. In Figure 4(a), noiseless cores of size 1024×1024 were used. The large size allowed the SNN to fit and be mapped into three separate cores, with each hidden layer occupying one core. However, the drawback of the large core size is that hardware resources in the neurosynaptic cores are poorly utilized. Core resource utilization, defined as the fraction of RRAM crossbar cells being used, is 19% in this configuration.

The configuration in Figure 4(b) uses a smaller core size of 256×256 . Under this smaller fan-in constraint, the network cannot be directly mapped into the cores. Instead, the original neural network is split in order to fit into the smaller neurosynaptic cores, with additional cores needed to merge the results from the split cores (network structure is shown in Figure 8). As a result, the accuracy suffers a drop of 3.8%. However, core utilization is improved to $\sim 40\%$. As compared to Figure 4(a), the ring topology used in Figure 4(b) improved the latency of the network slightly due to better balancing of the network traffic.

Finally, Figure 4(c) shows the effect of adding RTN to the SNN shown in Figure 4(b). In addition, a 2D-mesh network topology was used. Accounting for noise in the RRAM devices resulted in a further accuracy drop of 2%. However, the mesh network used in this configuration reduced the latency by half and achieved the lowest latency of the three configurations.

These examples demonstrate that a system-level simulator is a useful tool for evaluating different design choices and device characteristics for an application.

3 SIMULATOR DESIGN

To assist in both the hardware and software designs of RRAM-based neuromorphic computing chips, it is useful to have a cycle-accurate system-level simulator that can simulate the behavior of the RRAM-based neurosynaptic cores, including the non-ideal effects of the RRAM memory crossbars and the NoC interconnect. Without a system-level simulator, it would be challenging to assess the impact of hardware architectural designs on end-user applications.

3.1 Overview

Figure 5 shows a high-level architectural view of our simulator framework. The cycle-accurate simulator models a neuromorphic chip comprising many neurosynaptic cores connected with an

Fig. 5. Architecture of RRAM-based neuromorphic chip simulator.

interconnect as shown in Figure 5(a). The neuromorphic Chip class is the highest-level container that is used to hold N number of Cores. These cores are connected by a Network interface that can be configured with various topologies. An input spike Source can be interfaced to the Chip to drive the computations, and a Sink can be connected to the output neurons to collect simulation statistics.

Figure 5(b) shows the detailed structure of each core. Within each Core is a Crossbar memory interface to store the synaptic weights of a population of neurons. The Neuron interface is responsible for integrating the incoming signals with the respective synaptic weights to produce an outgoing spike if the integrated signal exceeds the membrane potential of the neuron. In the simulator, we also include an optional module that models the non-ideal effects of the RRAM crossbar.

The Network interface implements the NoC that connects all the cores. Figure 5(c) shows the internal structure of each router inside the NoC. Within the router, there are 5 input/output channels ($C0$ to $C4$). The first channel, $C0$, connects to the local core and the others connect to neighboring routers. Incoming packets are received and sent to the decoder. The decoder analyzes the source and destination of each packet based on the header data; the arbiter controls the switch accordingly. Outgoing packets traverse the switch and are stored in the FIFO buffers of the output channels, awaiting to be sent to next hop.

The simulator is time driven; each class or interface shares the same global clock. In addition, power consumption can also be simulated. For each component of the simulator, a trace-based power model is implemented to estimate the power of the system. The RRAM crossbar power model and the A/D converter power model are based on [37] and [38], respectively. The NoC power model is based on Orion 2 [35].

3.2 Non-Ideal Effects of RRAM

In this section, we consider many of the non-ideal effects of RRAM devices in the Crossbar array. A study of the non-ideal effects is important because the membrane voltage of each neuron is determined by the analog current sum of the RRAM devices connected to that neuron (i.e., in a given column). Any effects that affect the resistance value of the RRAM may have an impact on the overall resultant SNN output.

The non-ideal effects of RRAM can be classified into static defects and variability as well as dynamic variability. Static defects and variability are mainly caused by process variations when the RRAM devices are fabricated, and do not change during runtime. An example of this is the SAF defects. Unlike static defects and variability, dynamic variability changes from cycle to cycle

Fig. 6. Empirical data points from [21] and fitted curves for dependence of $\frac{\Delta R}{R}$ on R & for different RRAM materials.

during runtime and is caused by the driving circuits or is the result of electronic noise. In our simulator, we consider the following defects and variability.

3.2.1 Stuck-at-Faults. This is a static defect that can occur in an RRAM device. Due to limitations in the fabrication process and poor control of the forming process, the resistance value of an RRAM device could be stuck to an HRS or LRS, regardless of the set or reset operations. SAF defects not only lead to degradation in the yield of the fabricated chips but also cause severe failures if they are not detected at the test stage, which we will show later. Two SAF effects are possible: either stuck-at-open, where the RRAM device is stuck at the HRS, resulting in abnormally low current flow; or stuck-at-short, where it is stuck at an LRS, resulting in abnormally high current flow. According to studies on RRAM devices and in our RRAM model, SAFs are assumed to be uniformly distributed among the devices in the crossbar array.

3.2.2 Random Telegraph Noise. Due to the random trapping and release of charge carriers along the conductive filaments owing to the presence of lattice defects, the current flow through an RRAM device randomly fluctuates between two stable states. This manifests as a source of noise known as *random telegraph noise* (RTN). This dynamic variation is stochastic in nature and can be modeled as a bistable fluctuating behavior between two resistance levels. In our simulator, the Monte Carlo approach described in [21] is used to model the random switching between the two resistance levels. The amplitude of fluctuation depends on the RRAM material type as well as the resistance range and is approximated with curves fitted to data from [21], as shown in Figure 6.

Compared with static defects and variability, dynamic variability is much more computationally intensive to model because they are cycle-to-cycle variations that occur in every computation. Among the dynamic variations, RTN modeling takes up the most computational time because, for each crossbar computation, every read operation needs to be simulated.

3.2.3 Write Variability. Set and reset variability of the RRAM is another important dynamic variation. During the set and reset process, the migration of ionized defects changes the shape and size of the conductive filament in an RRAM device. Because of the discrete nature of the ionized defects, the resistance values of set and reset states or, equivalently, LRS and HRS, show stochastic variability. For an RRAM-based crossbar array, the set and reset variability is generally exhibited as write variability, which affects the resistance values written to the crossbar array [20]. In order to mitigate write variability, a write-verify scheme may be used to reduce the variability

at the expense of increased write latency. In our simulator, write variability is modeled using log-normal statistics in the quasi-classical limit for both on and off states owing to the exponential dependencies of the current distribution on a large number of random fluctuations in the gap domains of the non-crystalline oxide structure [36]. Data from [20] is used in our modeling of resistance variations. It is important to note that write variability increases with higher target resistance values.

3.3 RRAM Models

In this section, we will describe the details of the non-linear RRAM models currently built into the simulator. Note that the framework is flexible and allows users to extend or replace the models with their own.

Stuck-at-open and stuck-at-short faults refer to faults where a RRAM device in the crossbar is stuck in HRS or LRS, respectively. A probability distribution decides whether each RRAM device in the crossbar is randomly marked as being stuck-at-open, stuck-at-short, or is operating normally. For cases where empirical data is not available, stuck-at-faults are usually assumed to be independent and uniformly distributed. Otherwise, a user-defined probability distribution can be used. The procedure for simulating stuck-at-faults is shown by the following equations. Equations (5) and (6) describe the probability of an RRAM device stuck at HRS or LRS conditioned on the stuck-at-fault occurring in the device (Equation (4)). The probability of a device operating normally is given as $1 - p_{saf}$.

$$\text{Prob}(\text{stuck-at-fault}) = p_{saf} \quad (4)$$

$$\text{Prob}(\text{stuck-at-open} \mid \text{stuck-at-fault}) = p_{open} \quad (5)$$

$$\text{Prob}(\text{stuck-at-short} \mid \text{stuck-at-fault}) = p_{short} \quad (6)$$

RTN in a device's reading process is simulated using a Monte Carlo approach. The RTN fluctuates between two stable states: R (lower resistance level) and $R + \Delta R$ (higher resistance level). We used a ΔR that is determined from curves fitted to the empirical data shown in Figure 6. The listing in Algorithm 1 shows the Monte Carlo procedure used to simulate RTN behavior. Initially, each device is set to one of the resistance levels randomly with equal probability. Subsequently, during each timestep when a device is being read, a random number is drawn from the uniform distribution to determine if the device remains in the current resistance level or switches to the other resistance level. The switching behavior is described in Equations (7) and (8),

$$\text{Prob}(r_t = R + \Delta R \mid r_{t-1} = R) = t_{sample}/t_{off}, \quad (7)$$

$$\text{Prob}(r_t = R \mid r_{t-1} = R + \Delta R) = t_{sample}/t_{on}, \quad (8)$$

where r_t is the resistance in the current timestep, r_{t-1} is the resistance in the previous timestep, t_{sample} is the duration of current flow through the RRAM device in between consecutive reads, and t_{off} and t_{on} is the average time that the device stays in higher or lower resistance levels, respectively.

To simulate write variability, the simulator implements stochastic write behavior based on empirical data [20] that provides the standard deviation (σ_R) of the targeted resistance value (μ_R). Applying a log-normal distribution on write variability [36], the model currently implemented is described by Equations (9) to (11),

$$\tilde{R} = \exp^{m+sZ} \quad (9)$$

$$m = \log_e \mu_R - \frac{1}{2}s^2 \quad (10)$$

ALGORITHM 1: Monte Carlo Simulation of Random Telegraph Noise (RTN) for RRAM device**Input:** r_{t-1} , t_{sample} , t_{on} , t_{off} , R , $R + \Delta R$ **Output:** r_t **Initialize:** $r_0 = R$ or $R + \Delta R$ with equal probability**begin** **if** $r_{t-1} == R$ **then** $n = \text{draw random number from Uniform}(0, 1)$ **if** $n < t_{\text{sample}}/t_{\text{off}}$ **then** $r_t = R + \Delta R$ **end** **else** $r_t = R$ **end** **end** **else if** $r_{t-1} == R + \Delta R$ **then** $n = \text{draw random number from Uniform}(0, 1)$ **if** $n < t_{\text{sample}}/t_{\text{on}}$ **then** $r_t = R$ **end** **else** $r_t = R + \Delta R$ **end** **end****end**

$$s = \left\{ \log_e \left[\left(\frac{\sigma_R}{\mu_R} \right)^2 + 1 \right] \right\}^{1/2}, \quad (11)$$

where $\tilde{R}$ is the actual resistance value written to the device and Z is a random number drawn from the standard normal distribution. In short, when an RRAM device is written with the target value, a random number is generated from a normal distribution with mean m and standard deviation s . The actual resistance value written is the exponential of the generated random number.

3.4 Configurability and Scalability Design

To allow for flexibility in evaluating the influence of different factors on application performance on a neuromorphic chip, our simulator is built with configurability and scalability in mind. We implemented the simulator in a modularized manner; it is easy to plug in or unplug the optional features from the simulator, for example, the non-ideal RRAM effects module.

3.4.1 Interfaces. Key interfaces defined for the simulator are:

Chip: The Chip interface is a high-level abstraction of a hardware chip. The neuromorphic Chip instance comprises a collection of symmetric cores that are connected by an interconnect. It also provides communication interfaces for sending data into the chip and receiving data out from the chip.

Core: This interface represents a functional computational block for a set of neurons. The number of input axons into the Core is defined as the *fan-in* degree; the number of neuronal output axons is defined as the *fan-out* degree. Each Core is driven by the global clock that is fed to the Neuron interface.

Fig. 7. Configurations of key interfaces in the simulator.

Crossbar: This interface is responsible for storing the synaptic weights of each neuron in a memory crossbar array. To support a RRAM-based crossbar array, an RRAM Crossbar instance is constructed using this interface and RRAM effects, such as write variability are modeled.

Neuron: The Neuron interface encapsulates the neuron model and the type of computation that is performed at each clock cycle. For an LIF Neuron instance, the computations in Equations (1) and (3) are performed. Output spikes that are generated are forwarded to the Network interface to be sent to their destination axons.

Network: The Network interface is responsible for providing the on-chip interconnect that supports communication between different Cores. Different topologies can be configured, including parameters such as link bandwidth and buffer size. Section 3.4.2 provides more detail.

Source: The spike Source interface provides the spike data input into the neuromorphic Chip. For instance, it can be used to convert an input image into spike trains to be delivered to the input neurons.

Sink: The Sink interface is used to monitor and interpret spikes from output neurons. For example, a classifier is a sink that translates output spikes into their respective class labels.

Figure 7 shows the different configurations that are supported by these interfaces. In addition, many of the interfaces of the simulator are parameterized, as shown in Table 1. It is convenient to change simulator settings by changing the parameters.

3.4.2 Network-On-Chip. Apart from the RRAM neurosynaptic cores, our simulator explicitly integrates an NoC model through the Network interface. As increasing core counts are needed to support larger SNNs, the NoC becomes critical in determining the performance of the system. Factors that may affect the delivery of spikes between neurons include (a) bandwidth of the physical links, (b) network topology and routing algorithm, (c) buffer resources, and (d) router clock frequency. The building block of our NoC model is the router. Each neurosynaptic core is connected to a router. The router internals include multiple buffers to hold spikes, route compute logic, interfaces to receive spikes from the connected core or other routers, and a switch for spikes to traverse to destination buffers. We model a classic pipelined router model as shown in Figure 5(c). The router has 5 read/write channels, one in each North, South, East, West direction, and a local channel connected to the core. Each channel has associated FIFO buffers. When a core sends out a spike from one of its neurons through a router, the destination information is obtained from the *routing table* and a packet is constructed. The arbiter and switch components are responsible for selecting packets and forwarding to the appropriate channels to be sent out through the connected links. At the destination router, the packet is then stripped to remove routing information and the spike is sent to the corresponding axon in the core. In our model, the buffer sizes

Table 1. Changeable Parameters in the Simulator

Interface	Parameter	Description
Chip	Cores	A list of cores within the chip. Allows for heterogenous chips as each core is independently parameterized.
	NoC	NoC model.
Core	Neurons	A list of neurons within the core.
	Crossbar	The crossbar used for matrix-vector computation within the core.
Crossbar	F_{in}, F_{out}	The dimensions of the crossbar in terms of synapses.
	R_{low}, R_{high}	Range of resistances used to represent values in the range [0,1].
	defect_model	Stuck-at-fault model.
	noise_model	Write variability, relaxation and read noise models.
	Weights	Weight array to be represented in crossbar.
Neuron	len, frac_bits	Fixed-point format and length of neuron accumulator.
	v_{th}	Threshold voltage.
	v_{leak}	Leak voltage.
	dest_core, dest_row	Destination core and row of the neuron's output spikes.
Network	F	NoC operation frequency.
	Top	Topology of NoC.
	Route	Routing strategy of NoC.
	W_{buff}	Buffer size of routers.

can be configured by users. Other configurable parameters — including frequency, topology, and routing algorithm — are shown in Table 1.

Frequency: In our simulator, the core and NoC can run at different relative clock frequencies. The clock frequency is configured as a router-to-core clock ratio. One of the important metrics that determines the performance of the system is the minimum clock ratio required to deliver all the spikes in the system. The lower the clock ratio, the higher the performance of the NoC.

Topology: Our NoC model supports the following topologies: 2D mesh, 2D torus, ring, and double ring. Any number of cores can be simulated for each topology.

Buffer Size: The buffers within the router are all configurable. The size of the buffers may have an impact on the performance of the NoC.

Routing Algorithm: We implement a routing table-based approach, as in [42]. Currently, we support only dimension-based routing for mesh and torus topology. For ring and double ring, we use shortest path-based routing.

Flow Control: We implement the credit-based flow control in our simulator.

4 CASE STUDIES

Our simulator can be used to quantitatively measure how various parameters affect how well an SNN application executes on a given hardware configuration. In this section, we present case

studies of using the simulator to explore how various settings — such as RRAM material type, noise model, and NoC topology — affect application behavior. The experiments demonstrate the utility of the simulator in exploring issues that have not been well studied in previous works, in particular:

- the extent to which RRAM material and non-ideal effects of RRAM can affect the classification accuracy of an SNN,
- the traffic pattern and impact of the NoC in an RRAM-based neuromorphic chip, and
- the scalability and performance of executing large-scale real-world, neural-network-based applications on the simulator.

4.1 Experimental Setup

We first describe the experimental setup, the SNN applications that are used, how the weights of the SNN are mapped to the neurosynaptic cores, and the NoC configuration, followed by a presentation of the simulation results. The experiments were performed in single-threaded mode on an 18-core 2.3GHz Intel Xeon machine with 128GB of memory.

4.1.1 SNN-Based Applications. Two SNN applications are used in the following experiments. The first application is a fully connected SNN (MNIST-MLP) for digit recognition trained and tested using the MNIST dataset. This is used to evaluate the influence of non-ideal RRAM effects and the impact of the NoC. The second application (CIFAR10-CNN) is a large-scale SNN converted from a deep convolutional neural network (CNN) for image recognition, which is trained and tested using the CIFAR-10 dataset. This is used to demonstrate the scalability of our simulator for large-scale SNN applications. Note that these SNN applications were not tuned to achieve state-of-the-art accuracies. Their purpose is to illustrate the usefulness of the simulator in evaluating design choices as well as its configurability and scalability.

MNIST-MLP: This SNN application is converted from a three-layer fully connected neural network that is pretrained on the MNIST dataset. The three-layer network contains 512, 256, and 10 neurons, respectively. In the experiments, because the size of each neurosynaptic core is set to 256×256 , this network is modified to meet the fan-in constraints. This is achieved in the following manner, similar to [14]. Assuming that the fan-in degree of a core is F_{in} , each neuron with an in-degree $d_{in} > F_{in}$ in the original graph is replaced by a neuron that receives input from $\lceil \frac{d_{in}}{F_{in}} \rceil$ intermediate neurons. The intermediate neurons collectively then receive input from the input domain of the original neuron that was replaced. The final network is shown in Figure 8. The input image into the SNN is converted into spike trains via a Poisson process (see Section 4.1.2). The converted SNN application uses 21 neurosynaptic cores.

CIFAR10-CNN: This SNN application is implemented based on the CNN network architecture in [22]. It contains a total of 15 layers, including both convolutional layers and pooling layers. The original CNN network is trained on the CIFAR-10 dataset and converted to SNN using the model-based normalization [18]. As the network architecture already takes into account the fan-in constraints, it can be directly mapped onto the neuromorphic chip that we simulate. The input to the SNN is also converted into spike trains via a Poisson process (see Section 4.1.2). For the classification output, a majority voting scheme with votes accumulated across timesteps is used. This SNN application uses 3680 neurosynaptic cores.

4.1.2 Rate-Based Conversion. Each input image is converted into a rate-based representation using Poisson statistics. For every pixel of the image, a spike train is generated using the gray level of that pixel as the mean value of a Poisson distribution. In other words, the spike train for each

Fig. 8. Network structure of the MNIST-MLP example.

pixel is generated as follows:

$$x_i(t) = \begin{cases} 1, & \text{if } r < \text{pixel}_i/255 \\ 0, & \text{otherwise,} \end{cases} \quad (12)$$

where pixel_i is the i th pixel with gray value between 0 and 255, and r is a random number drawn from a uniform distribution.

4.1.3 Weight Mapping. The weights of the SNN in each application are mapped into resistances – or, equivalently, conductances – in the following manner. Let the gain of the A/D converter (ADC) be α and let the voltage applied to the row when there is a spike be β (the units of α and β are A^{-1} and V). Assuming that the ADC is operating in its linear regime so that no clipping occurs and ignoring quantization effects, then the result of the weighted accumulation can be simplified as

$$\begin{aligned} \Delta y &= f_{ADC} \left(\sum_{i \in F_{in}} g_i^+ v_i \right) - f_{ADC} \left(\sum_{i \in F_{in}} g_i^- v_i \right) \\ &= \alpha \sum_{i \in F_{in}} g_i^+ \beta x_i - \alpha \sum_{i \in F_{in}} g_i^- \beta x_i \\ &= \sum_{i \in F_{in}} \alpha \beta (g_i^+ - g_i^-) x_i, \end{aligned} \quad (13)$$

where g_i^+ and g_i^- are a pair conductances, and x_i is an indicator function denoting the occurrence of a spike. In order to compute $\Delta y = \sum_{i \in F_{in}} w_i x_i$, we have the mapping, $w_i = \alpha \beta (g_i^+ - g_i^-)$. When w_i , α , and β are known, there are many choices of (g_i^+, g_i^-) pairs that can be chosen to represent w_i , subject to the range of resistances that can be written to the RRAM. Being conductances, g_i^+ and g_i^- must be positive. Furthermore, $\frac{1}{g_i^+}$ and $\frac{1}{g_i^-}$ must both fall within the range $[R_{low}, R_{high}]$. In the experiments, when writing a weight w_i , $g_i^{-\text{sign}(w_i)}$ is set to the lowest possible conductance, $\frac{1}{R_{high}}$.

4.1.4 NoC Configuration. We configured the NoC as a 5×5 2D-mesh to simulate a 25-core neuromorphic chip for the MNIST-MLP SNN application. Static XY routing with priority-based

Fig. 9. Performance of MNIST-MLP under RTN with different material and resistance range.

arbitration was used for routing. The FIFO buffer depth is set to 2, and the NoC clock frequency is set to 1MHz, which is 1000× the global clock frequency of 1kHz.

4.2 Evaluation of Non-Ideal RRAM Effects

4.2.1 RTN. Figure 9 shows the influence of RTN on the accuracy of MNIST-MLP. The intensity of RTN is affected by the resistance range and the RRAM material (see Section 3.2.2). In the experiment, we maintain the ratio of R_{high} to R_{low} to 10 and evaluate MNIST-MLP across different resistance ranges. We also evaluate different RRAM materials in the simulation. Average accuracies are shown by the main bars and the standard deviation is denoted by the error bars. In general, we can observe that the RTN degrades application performance as the resistance range increases. When the resistance range is above $10^5\Omega$, the accuracy of NiO- and HfO₂-based RRAM is unacceptably poor. As shown in the figure, among different RRAM materials, the Cu-based RRAM (specifically Cu-doped Ge_{0.3}Se_{0.7} [21]) shows the highest robustness under RTN. When the resistance range is above $10^6\Omega$, the Cu-based RRAM still achieves over 85% accuracy whereas the accuracy of other materials is below 30%. There are two implications to this result. First, for hardware designers of RRAM-based neuromorphic chips, this experiment shows that in order to minimize the influence of RTN, either the operating resistance range should be decreased or other materials such as Cu-based RRAM should be used. Second, for algorithm developers, it is important to design SNN algorithms that are robust under RTN in order to use such hardware.

4.2.2 Write Variability. In this experiment, we explore the effects of write variability. Figure 10(a) shows the accuracy of MNIST-MLP under write variability for different resistance ranges. Like RTN, adverse effects of write variability become more prominent when the resistance range increases. This result indicates that write verification is critical for an RRAM-based neuromorphic chip. For higher resistance ranges, the accuracy is consistently below 10%. Even at the lowest resistance range, the highest accuracy achieved is only 61.4%.

Spurred by this result, we implemented and tested a write-verification scheme to explore the cost of mitigating write variability. In the write-verification scheme, we read back the RRAM cell value immediately after writing into that cell. If the difference between the read value and the designated write value exceeds a predefined tolerance range, we rewrite the value to the cell until the difference is within the tolerance range. Figure 10(b) shows the change in accuracy under different resistance ranges and with the write verification scheme applied. The dots in each line represent the accuracy and average number of writes to an RRAM cell when write verification is

(a) Accuracy loss due to write variability. Even at low RRAM resistance ranges, where variability is low, accuracy degrades from 93.9% (baseline without write variability) to 61.4%.

(b) Accuracy with write-verification scheme. Each curve represents the trade-off between accuracy vs average number of rewrites, for a particular RRAM resistance range. At the extreme, nearly all the accuracy lost to write variability can be recovered, except for the highest RRAM resistance range.

Fig. 10. Performance of MNIST-MLP with write variability.

enabled with a decreasing tolerance range. As the tolerance range decreases, the number of writes also increases to mitigate the effects of write variability. For the resistance range under $10^4\Omega$, the accuracy is significantly improved to about 90% and costs an average number of 0.47 rewrites. In the resistance range from $10^4\Omega$ to $10^5\Omega$, to achieve over 90% accuracy, the average number of rewrites required is 5.31. This number increases to over 24 for the $10^4\Omega$ to $10^5\Omega$ resistance range. The above experiments showed the influence of write variability along with the trade-offs between accuracy and the cost of the write-verify-write scheme. These results suggest that keeping to a lower resistance range can help minimize the effects of write variability as well as help reduce write latency even when write verification is enabled.

Although write-verify-write is a simple scheme, it has been widely adopted in state-of-the-art non-volatile memory implementations [41]. While other schemes have been proposed to reduce the number of rewrites to reduce power consumption and improve performance, they are circuit-level optimizations [29]. Since our simulator focuses on system-level simulation instead of circuit-level simulation, the effects of such circuit-level optimization techniques simply show up as different numbers of rewrites per write operation. Once the high-level characteristics of the circuits are empirically obtained, the simulator can be extended by users to simulate their proposed write-verify schemes more accurately.

(a) Decrease in accuracy due to stuck-at-open defects (b) Decrease in accuracy due to stuck-at-short defects

Fig. 11. Impact of stuck-at-faults on MNIST-MLP accuracy. Stuck-at-short defects have a greater impact than stuck-at-open defects.

4.2.3 Stuck-at-Faults (SAFs). There are two forms of SAF, stuck-at-open when an RRAM cell has abnormally high resistance, and stuck-at-short, when the cell is stuck at a low-resistance state. In the experiments, SAF occurrence is uniformly distributed among all the RRAM cells. Figure 11 shows the accuracy rate of MNIST-MLP under different SAF rates. In the figures, average accuracies are denoted by solid lines; the shaded region shows the variations across different simulation instances. Results show that SAF can severely affect accuracies even at low rates. Compared to stuck-at-open, stuck-at-short has a more adverse effect. Accuracy of the SNN drops over 20% when only 0.02% of the RRAM cells are stuck-at-short. In comparison, accuracy remains higher than 70% when 1% of the RRAM cells are stuck-at-open. The reason for this observation is because stuck-at-short drastically changes the spiking rate of a neuron much more than stuck-at-open. A short cell in the crossbar produces high current in the bit line and causes the neuron to fire more frequently. On the other hand, an open cell tends to reduce the firing rate of the neuron. If the weight of the synapse is low or the synapse does not receive spikes frequently, then the influence of an open cell would be small. These results indicate that it is critical to minimize SAF defects during fabrication, especially for stuck-at-short. Alternatively, axons containing defective cells could be bypassed during mapping of a network to the hardware.

4.3 Evaluation of NoC Impact

Figure 12(a) shows the impact of different topologies on MNIST-MLP. For all of the topologies, we use *Sequential Mapping*, that is, 21 nodes of MNIST-MLP are mapped onto the first 21 hardware cores in increasing order. The y-axis shows performance normalized to the 2D torus topology. The *performance metric* is defined as the minimum number of router clock cycles required to deliver all of the spikes across all cores in the chip. We use an exhaustive search scheme to measure the *performance metric* for a given topology. Figure 12(a) shows that the ring topology is the worst performing owing to its lowest bisection bandwidth. It is interesting to note that 2D torus outperforms 2D mesh by 7.6% for a 25% increase in number of links.

Figure 12(b) shows the impact of changing the mapping of MNIST-MLP nodes onto a 2D mesh network on the *performance metric* in increasing order. For 25 hardware cores and 21 MNIST-MLP nodes, the number of possible mappings is $^{25}P_{21}$. We simulate a subset of randomly chosen 10,000 different mappings and measure the *performance metric* for each given mapping. As shown in Figure 12(b), the mapping has a significant impact on the performance of the system. The metric ranges from 174 router cycles to 1487 router cycles. As evidenced by the simulation results, this

Fig. 12. Evaluating the impact of NoC configurations.

clearly shows that mapping an SNN onto the chip has a significant impact on the resulting performance. Previous works, such as [1, 33, 34], have proposed different mapping algorithms based on variations of the Kernighan-Lin algorithm. As our focus is on developing a system-level simulator for RRAM chips, optimization of mapping algorithms is not addressed in this work.

4.4 Power Simulation

To further illustrate the utility of our simulator, we used it to estimate the power consumption of each component in the neuromorphic chip when executing the MNIST-MLP SNN. For the subsequent evaluations, the resistance range of the RRAM crossbars is assumed to be $10\text{k}\Omega$ to $100\text{k}\Omega$. The signal frequency of the A/D converter is 100MHz , and the technology node of the CMOS components is assumed to be 45nm . Note that we used this technology node to render our results comparable to many reported in the literature. It is fairly easy to extend the simulator for other technology nodes. The results for power consumption are shown in Figure 13. Figure 13(a) shows that the total power consumption of the chip is around $401\mu\text{W}$, and that the NoC dominates the power consumption ($\sim 90\%$) over the cores. Figure 13(b) shows the power consumption of a neurosynaptic core, which comprises an RRAM crossbar array and an A/D converter. The power consumed by the A/D converter is about 58.7% higher than the RRAM crossbar. Finally, Figure 13(c) shows the breakdown for the NoC, from which we can see that the switch and buffer consumes over 80% of the power consumed by the NoC. All of these results indicate that in an RRAM-based

Fig. 13. Power consumption of a neuromorphic chip for running MNIST-MLP (numbers in μW).

Fig. 14. Performance of CIFAR10-CNN on RRAM-based neuromorphic chip without non-ideal RRAM effects.

neuromorphic chip, the CMOS components will be the key bottleneck in terms of power consumption, especially the NoC. Overall, the power efficiency is about 2.67×10^3 GOps/W, which, although lower, is comparable ($\sim 0.65\times$ lower) to other state-of-the-art RRAM crossbar-based computing architectures, such as [24]. In comparison to [15], which does not include an NoC, power efficiency will be more than 10 times lower. The high power consumption of the NoC reported by our simulator shows that optimizing the design of the NoC will be crucial for any low-power neuromorphic chip if the advantages of SNNs and RRAM are not to be marginalized.

4.5 Simulation of Large-Scale SNNs

In this section, we study the scalability of our simulator using the CIFAR10-CNN SNN application. The 15-layer SNN is mapped onto 3680 cores connected by a 2D mesh NoC. Results are shown in Figure 14. Non-ideal RRAM effects are not considered in this experiment. Figure 14(a) shows the change in accuracy with an increase in the timesteps of the simulation. As mentioned in Section 4.1, the SNN is converted from a deep CNN [22], whose original accuracy of 75% is shown by the dotted line. The classification output of CIFAR10-CNN is a majority voting scheme and the votes accumulate across timesteps. If all classes have zero votes, no prediction is made, denoted by “?”. The figure shows that the accuracy gradually grows as the timestep advances and after 100 steps achieves $\sim 60\%$ accuracy. The confusion matrix in Figure 14(b) shows that the network tends to mis-classify categories 3 and 5 for the first 100 timesteps. Notwithstanding the fact that the SNN was not explicitly tuned to achieve the original CNN accuracy, this example adequately

Fig. 15. Network traffic changes with time and saturates after 30 steps to a level of 7.67% of the worst-case traffic.

Table 2. Scalability Analysis of the Simulator for Varying Core Numbers in 2D Mesh Network

2D Mesh	Number of Cores	Simulation time (s)
5×5	25	26.5
8×8	64	65.6
16×16	256	250.5
32×32	1024	1009.4
64×64	4096	4008.4

demonstrates that the simulator can be used to collect runtime statistics and is useful for determining the performance of large-scale SNN applications.

Figure 15 shows the network traffic generated by CIFAR10-CNN, where the solid line denotes the average spike count and the shaded region shows the variations under different runs. In the beginning, the number of spikes in the NoC is low because the input spike trains have just entered the chip. As time advances, more spikes are generated as neurons in the later layers begin to spike. The network traffic, after 30 timesteps, stabilizes at a level that is, on average, 7.67% of the traffic if all neurons of the CIFAR10-CNN were to spike. This result shows that the simulator can be used to shed light on the NoC bandwidth requirement for a large-scale SNN and can help in the hardware design of NoCs for the neuromorphic chip.

Finally, for the purpose of scalability analysis, we simulate various chip sizes from 25 cores up to 4096 cores, as shown in Table 2. We assume a 2D mesh NoC with the configuration shown in the table. In this experiment, we simulate all cores generating spikes for all neurons every core clock cycle (which is invoked every 1000 router cycles). For example, in a 5×5 mesh, each of the 25 cores generates 256 spikes per core cycle. These spikes are transmitted to randomly generated destination cores in 1000 router cycles before the next core cycle. All experiments were run for 35 core cycles (equivalent to 35,000 router cycles). The table shows that the simulation time increases linearly with the number of cores, demonstrating the scalability of our simulator.

4.6 Discussion

In our experiments, we first found that RTN degrades accuracy at high operating resistance ranges. Through simulating with empirical data, our results indicate that (1) RRAM-based neuromorphic chips should employ materials such as Cu-based RRAM or use lower operating resistance ranges, and (2) it is important for algorithm developers to develop SNN training and weight mapping

methods that are robust under RTN in order to use RRAM-based neuromorphic hardware. Second, we showed that the effect of write variability is pronounced and adversely affects application accuracy, even at low-resistance ranges. However, a write-verify scheme can be effective in restoring overall SNN accuracies at the expense of higher rewrite latencies, a trade-off that can be explored with our simulator. Third, we found that NoC topology and the mapping of the SNN to cores both have a significant impact on the router cycles needed to deliver all spikes, a major performance metric. This highlights the need for a good mapping algorithm and the importance of choosing a good network topology for neuromorphic chip designers.

5 RELATED WORK

Neuromorphic Hardware. Research into neuromorphic hardware initially focused on designing circuits that model a single neuron or a small set of neurons accurately. Studies such as [2, 45] proposed analog circuits to emulate the biological structures in neurons. These neuromorphic chips contain a limited number of neurons and are capable of running only trivial neural networks. Large-scale neuromorphic chips such as HiCANN [12] and NeuroGrid [7] with greater power and area efficiency were developed later. The SpiNNaker project developed ARM-based multi-core systems that can be used to simulate the human brain using SNN. IBM has released TrueNorth, a large-scale neuromorphic computing chip that integrated 4096 digital 256×256 neurosynaptic cores [17] capable of running state-of-the-art CNN applications [22].

Advances in RRAM offered potentially higher density and lower energy consumption for neuromorphic chips. Jo et al. showed that memristor-based crossbar memory architecture combined with CMOS neurons can support important synaptic functions [23]. Indiveri et al. demonstrated a design that integrated memristor-based neurosynaptic architectures into large-scale neural networks [10]. Park et al. proposed a neuromorphic computing circuit design based on TiN RRAM and applied it in speech processing [25]. Prezioso et al. proposed the first transistor free design of a metal-oxide-based memristor crossbar which supported fully operative neural networks [16]. Hu et al. proposed the *dot-product engine* (DPE) based on an RRAM crossbar as an efficient way to perform matrix-vector multiplication [15]. They also developed techniques to ensure compatibility of the DPE with software-level accuracy [31]. Chakrabarti et al. later proposed a 3D DPE based on hybrid CMOS and memristor circuits that led to increased density and bandwidth [3].

Inspired by TrueNorth, Yao et al. developed a fully functional large-scale neuromorphic chip based on RRAM [19]. Shafiee et al. proposed an accelerator based on RRAM crossbars [6]. A recent architecture proposed for SNNs, INXS, also uses memristor crossbar arrays, which is shown to be more computational and energy efficient than TrueNorth [24]. However, the impact of various non-ideal RRAM characteristics was not considered in these works.

Simulators for Neuromorphic Computing. Many simulators for neuromorphic computing were initially designed for neuroscience research [5, 28, 30]. Other simulators, including [8, 27], focused on using GPUs to accelerate the performance of SNN simulators. They were primarily used for computational modeling of neural networks and did not consider any hardware.

Other simulators for neuromorphic hardware have been developed to assist in hardware development. For example, COMPASS is a hardware simulator built by IBM for the digital TrueNorth chip [43]. HRLSim was released with more generic assumptions on the hardware architecture and can support more neuron models [13]. Both COMPASS and HRLSim are used for fully digital CMOS design, whereas our simulator can cater to memristor- or RRAM-based designs. A memristor-based SNN simulator was proposed by Querlioz et al. [44]. However, they do not consider the RRAM effects that we have studied. Moreover, their simulator supports simulation of only a single neurosynaptic core. Tang et al. proposed a circuit-level simulator of RRAM-based SNN [26]. In their study, detailed circuit-level features of memristor devices are considered, but they support

the simulation of only synthetic neural networks and it does not have an integrated NoC model. Xia et al. proposed a simulator called MNSIM that simulates memristor-based neuromorphic chips. [14]. Unlike our simulator, MNSIM does not simulate RTN and does not have an NoC model.

6 CONCLUSION

In this article, we introduced a cycle-accurate, system-level simulator for RRAM-based neuromorphic computing chips comprised of many neurosynaptic cores interconnected through an NoC. The neurosynaptic core architecture contains an RRAM crossbar array for storing synaptic weights and is able to perform weight accumulation through a current sum in the analog domain. Our full-scale neuromorphic chip simulator integrates both an RRAM model and an NoC model. In addition, many non-ideal characteristics of the RRAM, such as SAFs, write variability, and RTN, are also integrated into the simulator to allow for the accurate simulation of applications at scale. We demonstrated the use of the simulator in optimizing design choices for a neuromorphic chip. We discovered that write variability and RTN can be important factors that degrade application performance, and that Cu-based RRAM devices have lesser RTN impact compared to other materials. We also used the simulator to study the accuracy-rewrite trade-offs when using a write-verify scheme as a mitigation strategy for write variability. In addition, using the integrated NoC model, we demonstrated the impact that different application mappings and network topologies can have on the performance of an application.

REFERENCES

- [1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. 2015. TrueNorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 10 (2015), 1537–1557.
- [2] J. Arthur and K. Boahen. 2004. Recurrently connected silicon neurons with active dendrites for one-shot learning. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'04)*, Vol. 3. IEEE, 1699–1704.
- [3] Bhaswar Chakrabarti, Miguel Angel Lastras-Montano, Gina Adam, Mirko Prezioso, Brian Hoskins, M. Payvand, A. Madhavan, A. Ghofrani, L. Theogarajan, K.-T. Cheng, et al. 2017. A multiply-add engine with monolithically integrated 3D memristor crossbar/CMOS hybrid circuit. *Scientific Reports* 7 (2017), 42429.
- [4] A. Ankit et al. 2017. RESPARC: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 27:1–27:6.
- [5] A. Davison et al. 2009. PyNN: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics* 2 (2009), 11.
- [6] A. Shafiee et al. 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)*. 14–26.
- [7] B. Benjamin et al. 2014. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE* 102, 5 (2014), 699–716.
- [8] D. Yudanov et al. 2010. GPU-based simulation of spiking neural networks with real-time performance and high accuracy. In *International Joint Conference on Neural Networks (IJCNN'10)*. IEEE, 1–8.
- [9] E. Painkras et al. 2013. SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits* 48, 8 (Aug. 2013), 1943–1953.
- [10] G. Indiveri et al. 2013. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology* 24, 38 (2013), 384010.
- [11] H. Wong et al. 2012. Metal-oxide RRAM. *Proceedings of the IEEE* 100, 6 (June 2012), 1951–1970.
- [12] J. Schemmel et al. 2010. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS'10)*. IEEE, 1947–1950.
- [13] K. Minkovich et al. 2014. HRLSim: A high performance spiking neural network simulator for GPGPU clusters. *IEEE Transactions on Neural Networks and Learning Systems* 25, 2 (2014), 316–331.
- [14] L. Xia et al. 2018. MNSIM: Simulation platform for memristor-based neuromorphic computing system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 5 (May 2018), 1009–1022.

- [15] M. Hu et al. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. 1–6.
- [16] M. Prezioso et al. 2015. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 7550 (2015), 61–64.
- [17] P. Merolla et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.
- [18] P. U. Diehl et al. 2015. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks (IJCNN'15)*. 1–8.
- [19] P. Yao et al. 2017. Face classification using electronic synapses. *Nature Communications* 8 (2017), 15199.
- [20] S. Ambrogio et al. 2014. Statistical fluctuations in HfOx resistive-switching memory: Part I - Set/reset variability. *IEEE Transactions on Electron Devices* 61, 8 (Aug 2014), 2912–2919.
- [21] S. Ambrogio et al. 2014. Statistical fluctuations in HfOx resistive-switching memory: Part II - Random telegraph noise. *IEEE Transactions on Electron Devices* 61, 8 (Aug 2014), 2920–2927.
- [22] S. Esser et al. 2016. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences* 113, 41 (2016), 11441–11446.
- [23] S. H. Jo et al. 2010. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters* 10, 4 (2010), 1297–1301.
- [24] S. Narayanan et al. 2017. INXS: Bridging the throughput and energy gap for spiking neural networks. In *International Joint Conference on Neural Networks (IJCNN'17)*. IEEE, 2451–2459.
- [25] S. Park et al. 2013. Neuromorphic speech systems using advanced ReRAM-based synapse. In *IEEE International Electron Devices Meeting (IEDM'13)*. IEEE, 25–6.
- [26] T. Tang et al. 2015. Spiking neural network with RRAM: Can we use it for real-world application? In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 860–865.
- [27] A. Fidjeland and M. Shanahan. 2010. Accelerated simulation of spiking neural networks using GPUs. In *International Joint Conference on Neural Networks (IJCNN'10)*. IEEE, 1–8.
- [28] M. Gewaltig and M. Diesmann. 2007. NEST (NEural simulation tool). *Scholarpedia* 2, 4 (2007), 1430.
- [29] Amirali Ghofrani, Siddharth Gaba, Melika Payvand, Wei Lu, Luke Theogarajan, Kwang-Ting Cheng, et al. 2015. A low-power variation-aware adaptive write scheme for access-transistor-free memristive memory. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 12, 1 (2015), 3.
- [30] D. Goodman and R. Brette. 2009. The brain simulator. *Frontiers in Neuroscience* 3 (2009), 26.
- [31] Miao Hu, John Paul Strachan, Zhiyong Li, R. Stanley, et al. 2016. Dot-product engine as computing memory to accelerate machine learning algorithms. In *17th International Symposium on Quality Electronic Design (ISQED'16)*. IEEE, 374–379.
- [32] E. M. Izhikevich. 2003. Simple model of spiking neurons. *IEEE Transactions on Neural Networks* 14, 6 (Nov 2003), 1569–1572.
- [33] Yu Ji, Youhui Zhang, Wenguang Chen, and Yuan Xie. 2018. Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 448–460.
- [34] Yu Ji, Youhui Zhang, He Liu, and Weimin Zheng. 2016. Optimized mapping spiking neural networks onto network-on-chip. In *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 38–52.
- [35] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. 2009. ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 423–428.
- [36] V. G. Karpov and D. Niraula. 2017. Log-normal statistics in filamentary RRAM devices and related systems. *IEEE Electron Device Letters* 38, 9 (2017), 1240–1243.
- [37] Boxun Li, Lixue Xia, Peng Gu, Yu Wang, and Huazhong Yang. 2015. Merging the interface: Power, area and accuracy co-optimization for RRAM crossbar-based mixed-signal computing system. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 13.
- [38] Ye Li, Bertan Bakkaloglu, and Chaitali Chakrabarti. 2007. A system level energy model and energy-quality evaluation for integrated transceiver front-ends. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 15, 1 (2007), 90–103.
- [39] W. Maass. 1997. Networks of spiking neurons: The third generation of neural network models. *Neural Networks* 10, 9 (1997), 1659–1671.
- [40] Don Monroe. 2014. Neuromorphic computing gets ready for the (really) big time. *Communications of the ACM* 57, 6 (2014), 13–15.
- [41] Hiroki Noguchi, Kazutaka Ikegami, Satoshi Takaya, Eishi Arima, Keiichi Kushida, Atsushi Kawasumi, Hiroyuki Hara, Keiko Abe, Naoharu Shimomura, Junichi Ito, et al. 2016. 7.2 4MB STT-MRAM-based cache with

- memory-access-aware power optimization and write-verify-write/read-modify-write scheme. In *IEEE International Solid-State Circuits Conference (ISSCC'16)*. IEEE, 132–133.
- [42] M. Papamichael and J. Hoe. 2012. CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 37–46.
- [43] R. Preisslet et al. 2012. COMPASS: A scalable simulator for an architecture for cognitive computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 54.
- [44] D. Querlioz, O. Bichler, and C. Gamrat. 2011. Simulation of a memristor-based spiking neural network immune to device variations. In *International Joint Conference on Neural Networks (IJCNN'11)*. IEEE, 1775–1781.
- [45] K. Zaghloul and K. Boahen. 2004. Optic nerve signals in a neuromorphic chip I: Outer and inner retina models. *IEEE Transactions on Biomedical Engineering* 51, 4 (2004), 657–666.

Received May 2018; revised September 2018; accepted October 2018