Endurance-Aware Mapping of Spiking Neural Networks to Neuromorphic Hardware

Twisha Titirsha, Shihao Song, Anup Das, Jeffrey Krichmar, Nikil Dutt, Nagarajan Kandasamy, and Francky Catthoor

Abstract—Neuromorphic computing systems are embracing memristors to implement high density and low power synaptic storage as crossbar arrays in hardware. These systems are energy efficient in executing Spiking Neural Networks (SNNs). We observe that long bitlines and wordlines in a memristive crossbar are a major source of parasitic voltage drops, which create current asymmetry. Through circuit simulations, we show the significant endurance variation that results from this asymmetry. Therefore, if the critical memristors (ones with lower endurance) are overutilized, they may lead to a reduction of the crossbar's lifetime. We propose eSpine, a novel technique to improve lifetime by incorporating the endurance variation within each crossbar in mapping machine learning workloads, ensuring that synapses with higher activation are always implemented on memristors with higher endurance, and vice versa. eSpine works in two steps. First, it uses the Kernighan-Lin Graph Partitioning algorithm to partition a workload into clusters of neurons and synapses, where each cluster can fit in a crossbar. Second, it uses an instance of Particle Swarm Optimization (PSO) to map clusters to tiles, where the placement of synapses of a cluster to memristors of a crossbar is performed by analyzing their activation within the workload. We evaluate eSpine for a state-of-the-art neuromorphic hardware model with phase-change memory (PCM)-based memristors. Using 10 SNN workloads, we demonstrate a significant improvement in the effective lifetime.

Index Terms—Neuromorphic Computing, Spiking Neural Networks (SNNs), Non-Volatile Memory (NVM), Memristor, Endurance.

1 Introduction

Spiking Neural Networks (SNNs) are machine learning approaches designed using spike-based computations and bio-inspired learning algorithms [1]. Neurons in an SNN communicate information by sending spikes to other neurons, via synapses. SNN-based applications are typically executed on event-driven neuromorphic hardware such as DYNAP-SE [2], TrueNorth [3], and Loihi [4]. These hardware platforms are designed as tile-based architectures with a shared interconnect for communication [5] (see Fig. 1a). A tile consists of a crossbar for mapping neurons and synapses of an application. Recently, memristors such as Phase-Change Memory (PCM) and Oxide-based Resistive RAM (OxRRAM) are used to implement high-density and low-power synaptic storage in each crossbar [6]–[11].

As the complexity of machine learning models increases, mapping an SNN to a neuromorphic hardware is becoming increasingly challenging. Existing SNN-mapping approaches have mostly focused on improving performance and energy [12]–[18], and reducing circuit aging [19]–[21]. Unfortunately, memristors have limited endurance, ranging from 10^5 (for Flash) to 10^{10} (for OxRRAM), with PCM somewhere in between ($\approx 10^7$). We focus on endurance issues in a memristive crossbar of a neuromorphic hardware and propose an intelligent solution to mitigate them.

We analyze the internal architecture of a memristive crossbar (see Fig. 3) and observe that parasitic components on horizontal and vertical wires of a crossbar are a major source of parasitic voltage drops in the crossbar. Using detailed circuit simulations at different process (P), voltage (V), and temperature (T) corners, we show that these voltage drops create current variations in the crossbar. For the same spike voltage, current on the shortest path is significantly higher than the current on the longest path in the crossbar, where the length of a current path is measured in terms of its number of parasitic components. These current variations create asymmetry in the self-heating temperature of memristive cells during their weight updates, e.g., during model training and continuous online learning [22], which directly influences their endurance.

The endurance variability in a memristive crossbar becomes more pronounced with technology scaling and at elevated temperature. If this is not incorporated when executing a machine learning workload, critical memristors, i.e., those with lower endurance may get overutilized, leading to a reduction in the memristor lifetime.

In this work, we formulate the *effective lifetime*, a joint metric incorporating the endurance of a memristor, and its utilization within a workload (see Sec. 5). Our **goal** is to maximize the minimum effective lifetime. We achieve this goal by first exploiting technology and circuit-specific characteristics of memristors, and then proposing an endurance-aware *intelligent mapping* of neurons and synapses of a machine learning workload to crossbars of a hardware, ensuring that synapses with higher activation are implemented on memristors with higher endurance, and vice versa.

Endurance balancing (also called *wear leveling*) is previously proposed for classical computing systems with Flash storage, where a virtual address is translated to differ-

- *T. Titirsha, S. Song, A. Das, and N. Kandasamy are with the Department of Electrical and Computer Engineering, Drexel University, PA, 19147.*

  E-mail: {tt624,shihao.song,anup.das,nk78}@drexel.edu
- *N. Dutt and J. Krichmar are with the Department of Computer Science, University of California, Irvine, CA, USA.*
- *F. Catthoor is with Imec, Belgium and KU Leuven, Belgium.*

Manuscript received Month DD, Year; revised Month DD, Year.

ent physical addresses to balance the wear-out of Flash cells [23]–[27]. Such techniques cannot be used for neuromorphic hardware because once synapses are placed to crossbars they access the same memristors for the entire execution duration. Therefore, it is necessary to limit the utilization of critical memristors of a neuromorphic hardware during the initial mapping of neurons and synapses.

To the best of our knowledge, no prior work has studied the endurance variability problem in neuromorphic hardware with memristive crossbars. To this end, we make the following novel **contributions** in this paper.

- We study the parasitic voltage drops at different P, V, & T corners through detailed circuit simulations with different crossbar configurations.
- We use these circuit simulation parameters within a compact endurance model to estimate the endurance of different memristors in a crossbar.
- We integrate this endurance model within a design-space exploration framework, which uses an instance of Particle Swarm Optimization (PSO) to map SNN-based workloads to crossbars of a neuromorphic hardware, maximizing the effective lifetime of memristors.

The proposed endurance-aware technique, which we call eSpine, operates in two steps. First, eSpine partitions a machine learning workload into clusters of neurons and synapses using the Kernighan-Lin Graph Partitioning algorithm such that, each cluster can be mapped to an individual crossbar of a hardware. The objective is to reduce inter-cluster communication, which lowers the energy consumption. Second, eSpine uses PSO to map clusters to tiles, placing synapses of a cluster to memristors of a crossbar in each PSO iteration by analyzing their utilization within the workload. The objective is to maximize the effective lifetime of the memristors in the hardware. We evaluate eSpine using 10 SNN-based machine learning workloads on a state-of-the-art neuromorphic hardware model using PCM memristors. Our results demonstrate an average 3.5x improvement of the effective lifetime with 7.5% higher energy consumption, compared to a state-of-the-art SNN mapping technique that minimizes the energy consumption.

2 Background

Figure 1a illustrates a tile-based neuromorphic hardware such as DYNAP-SE [2], where each tile consists of a crossbar to map neurons and synapses of an SNN. A crossbar, shown in Figure 1b, is an organization of row wires called wordlines and column wires called bitlines. A synaptic cell is connected at a crosspoint, i.e., at the intersection of a row and a column. Pre-synaptic neurons are mapped along rows and post-synaptic neurons along columns. A $n \times n$ crossbar has n pre-synaptic neurons, n post-synaptic neurons, and n^2 synaptic cells at their intersections. Memristive devices such as Phase-Change Memory (PCM) [7], Oxide-based Resistive RAM (OxRRAM) [6], Ferroelectric RAM (FeRAM) [28], Flash [29], and Spin-Transfer Torque Magnetic or Spin-Orbit-Torque RAM (STT- and SoT-MRAM) [30] can be used to implement a synaptic cell. [1] This is illustrated in Figure 1c, where a memristor is represented as a resistance.

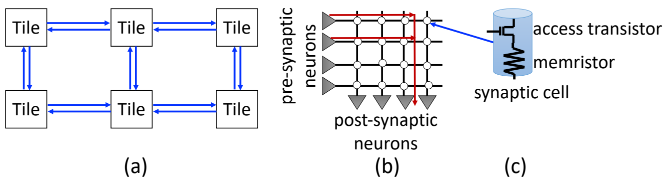


Fig. 1. Neuron and synapse mapping to a tile-based neuromorphic hardware such as DYNAP-SE [2].

We demonstrate eSpine for PCM-based memristive crossbars. We start by reviewing the internals of a PCM device. The proposed approach can be generalized to other memristors such as OxRRAM and SOT-/STT-MRAM by exploiting their specific structures (see Section 6.1).

Figure 2(a) illustrates how a chalcogenide semiconductor alloy is used to build a PCM cell. The amorphous phase (logic '0') in this alloy has higher resistance than its crystalline phase (logic '1'). When using only these two states, each PCM cell can implement a binary synapse. However, with precise control of the crystallization process, a PCM cell can be placed in a partially-crystallized state, in which case, it can implement a multi-bit synapse. Phase changes in a PCM cell are induced by injecting current into resistor-chalcogenide junction and heating the chalcogenide alloy. Figure 2 (b) shows the different current profiles needed to program and read in a PCM device.

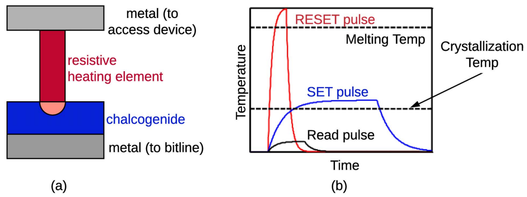


Fig. 2. (a) A phase change memory (PCM) cell and (b) current needed to SET, RESET, and read a PCM cell.

3 Analyzing Technology-Specific Current Asymmetry in Memristive Crossbars

Long bitlines and wordlines in a crossbar are a major source of parasitic voltage drops, introducing asymmetry in current propagating through its different memristors. Figure 3 shows these parasitic components for a 2x2 crossbar. We simulate this circuit using LTspice [35], [36] with technology-specific data from predictive technology model (PTM) [37]. We make the following three key observations.

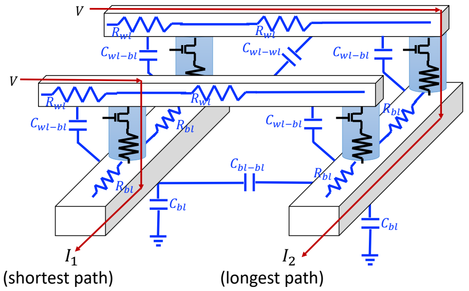


Fig. 3. Parasitcs of bitlines and wordlines in a memristive crossbar.

Observation 1: *The current on the longest path from a pre- to a post-synaptic neuron in a crossbar is lower than the current on its shortest path for the same input spike voltage and the same memristive cell conductance programmed along both these paths.*

[1] Beside neuromorphic computing, some of these memristor technologies are also used as main memory in conventional computers to improve performance and energy efficiency [31]–[34].

Figure 4 shows the difference between currents on the shortest and longest paths for 32x32, 64x64, 128x128, and 256x256 memristive crossbars at 65nm process node. The input spike voltage of the pre-synaptic neurons is set to generate $200\mu A$ on ther longest paths. This current value corresponds to the current needed to amorphize the crystalline state of a PCM-based memristor.

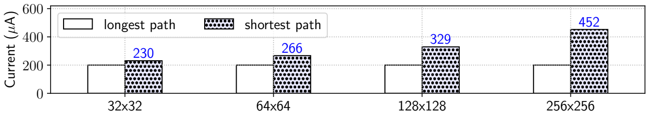


Fig. 4. Difference between current on the shortest and the longest path for different crossbar sizes.

We observe that the current injected into the post-synaptic neuron on the longest path is lower than the current on the shortest path by 13.3% for 32x32, 25.1% for 64x64, 39.2% for 128x128, and 55.8% for 256x256 crossbar. This current difference is because of the higher voltage drop on the longest path, which reduces the current on this path compared to the shortest path for the same amount of spike voltage applied on both these paths. The current difference increases with crossbar size because of the increase in the number of parasitic resistances on the longest current path, which results in larger voltage drops, lowering the current injected into its post-synaptic neuron. Therefore, to achieve the minimum $200\mu A$ current on this path, the input spike voltage must be increased, which increases the current on the shortest path. This observation can be generalized to all current paths in a memristive crossbar. Current variation in a crossbar may lead to difference in synaptic plasticity behavior and access speed of memristors [16], [38]–[41]. A circuit-level solution to address the current differences is to add proportional series resistances to the current paths in a crossbar. However, this circuit-level technique can significantly increase the area of a crossbar (n^2 series resistances are needed for a nxn crossbar). Additionally, adding series resistances can increase the power consumption of the crossbar. Although current balancing in a crossbar can be achieved by adjusting the biasing of the crossbar's cells, a critical limitation is that this and other circuit-level solutions do not incorporate the activation of the synaptic cells, which is dependent on the workload being executed on the crossbar. Therefore, some of its cells may get utilized more than others, leading to endurance issues. We propose a system-level solution to exploiting the current and activation differences via intelligent neuron and synapse mapping.

Current imbalance may not be a critical consideration for smaller crossbar sizes (e.g., for 32x32 or smaller) due to comparable currents along different paths. However, a neuron is several orders of magnitude larger than a memristor-based synaptic cell [42]. To amortize this large neuron size, neuromorphic engineers implement larger crossbars, subject to a maximum allowable energy consumption. The usual trade-off point is 128x128 crossbars for DYNAP-SE [2] and 256x256 crossbars for TrueNorth [3].

Observation 2: *Current variation in a crossbar becomes significant with technology scaling and at elevated temperatures.*

Figure 5 plots the current on the shortest path in a 128x128 memristive crossbar for four process corners (65nm, 45nm, 32nm, and 16nm) and four temperature corners (25°C, 50°C, 75°C, and 100°C) with all memristors configured in their crystalline state with a resistance of $10K\Omega$. The input spike voltage of the crossbar is set to a value that generates $200\mu A$ on the longest path at each process and temperature corners. We make two key conclusions.

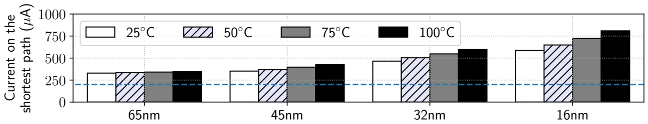


Fig. 5. Current obtained on the shortest path in a 128x128 memristive crossbar at 65nm, 45nm, 32nm, and 16nm technology nodes for 4 ambient temperatures (25°C, 50°C, 75°C, and 100°C). The input spike voltage is adjusted to obtain $200\mu A$ on the longest path.

First, current on the shortest path is higher for smaller process nodes. This is because, with technology scaling, the value of parasitic resistances along the bitline and wordline of a current path increases [38], [43], [44]. The unit wordline (bitline) parasitic resistance ranges from approximately 2.5Ω (1Ω) at 65nm node to 10Ω (3.8Ω) at 16nm node. The value of these unit parasitic resistances are expected to scale further reaching $\approx 25\Omega$ at 5nm node [38]. This increase in the value of unit parasitic resistance increases the voltage drop on the longest path, reducing the current injected into its post-synaptic neuron. Therefore, to obtain a current of $200\mu A$ on the longest path, the input spike voltage must be increased, which increases the current on the shortest path.

Second, current reduces at higher temperature. This is because, the leakage current via the access transistor of each memristor in a crossbar increases at higher temperature, reducing the current injected into the post-synaptic neurons. To increase the current to $200\mu A$, the spike voltage is increased, which increases the current on the shortest path.

Based on the two observations and the endurance formulation in Section 4, we show that higher current through memristors on shorter paths in a memristive crossbar leads to their higher self-heating temperature and correspondingly lower cell endurance, compared to those on the longer current paths in a crossbar. Existing SNN mapping approaches such as SpiNeMap [13], PyCARL [45], DFSynthesizer [12], and SNN Compiler [46] do not take endurance variation into account when mapping neurons and synapses to a crossbar. Therefore, synapses that are activated frequently may get mapped on memristors with lower cell endurance, lowering their lifetime.

Observation 3: *Synapse activation in a crossbar is specific to the machine learning workload as well as to mapping of neurons and synapses of the workload to the crossbars.*

Figure 6 plots the number of synaptic activation, i.e., spikes propagating through the longest and the shortest current paths in a crossbar as fractions of the total synaptic activation. Results are reported for 10 machine learning workloads (see Sec. 7) using SpiNeMap [13]. We observe that the number of activation on the longest and shortest current paths are on average 3% and 5% of the total number of activation, respectively. Higher synaptic activation on shorter current paths in a crossbar can lead to lowering of the lifetime of memristors on those paths due to their lower cell endurance (see observations 1 and 2, and the endurance and lifetime formulations in Section 4).

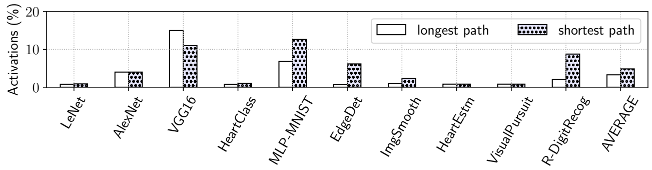


Fig. 6. Fraction of activation of memristor on the longest and shortest current paths in a crossbar using SpiNeMap [13].

4 Endurance Modeling

We use the phenomenological endurance model [47], which computes endurance of a PCM cell as a function of its self-heating temperature obtained during amorphization of its crystalline state. Figure 7 shows the iterative approach to compute this self-heating temperature (T_{SH}) [48], [49].

At start of the amorphization process, the temperature of a PCM cell is equal to the ambient temperature T_{amb}. Subsequently, the PCM temperature is computed iteratively as follows. For a given crystalline fraction V_C of the GST material within the cell, the thermal conductivity k is computed using the `TC Module`, and PCM resistance R_{PCM} using the `PCMR Module`. The thermal conductivity is used to compute the heat dissipation W_d using the `HD Module`, while the PCM resistance is used to compute the Joule heating in the GST W_j for the programming current I_{prog} using the `JH Module`. The self-heating temperature T_{SH} is computed inside the `SH Module` using the Joule heating and the heat dissipation. Finally, the self-heating temperature is used to compute the crystallization fraction V_c using the `CF Module`. The iterative process terminates when the GST is amorphized, i.e., $V_c = 0$. We now describe these steps.

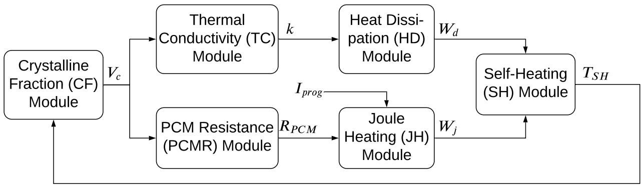

Thermal Conductivity (TC) Module
k
Heat Dissipation (HD) Module
W_d
Crystalline Fraction (CF) Module
V_c
Self-Heating (SH) Module
T_{SH}
I_{prog}
PCM Resistance (PCMR) Module
R_{PCM}
Joule Heating (JH) Module
W_j

Fig. 7. Iterative approach to calculating the self-heating temperature of a PCM cell during amorphization.

- **Crystallization Fraction (CF) Module:** CF represents the fraction of solid in a GST during the application of a reset current. V_c is computed using the Johnson-Mehl-Avrami (JMA) equation as

$$V_c = \exp\left[-\alpha \times \frac{(T_{SH} - T_{amb})}{T_m} \times t\right], \tag{1}$$

where t is the time, $T_m = 810K$ is the melting temperature of the GST material [48], [49], T_{amb} is the ambient temperature computed using [15], [50], and $\alpha = 2.25$ is a fitting constant [48], [49].

- **Thermal Conductivity (TC) Module:** TC of the GST is computed as [51]

$$k = (k_a - k_c) \times V_c + k_a, \tag{2}$$

where $k_a = 0.002WK^{-1}cm^{-1}$ for amorphous GST, $k_c = 0.005WK^{-1}cm^{-1}$ for crystalline GST [48], [49].

- **PCM Resistance (PCMR) Module:** The effective resistance of the PCM cell is given by

$$R_{PCM} = R_{set} + (1 - V_c) \times (R_{reset} - R_{set}), \tag{3}$$

where $R_{set} = 10K\Omega$ in the crystalline state of the GST and $R_{reset} = 200K\Omega$ in the amorphous state.

- **Heat Dissipation (HD) Module:** Assuming heat is dispersed to the surrounding along the thickness of the PCM cell, HD is computed as [52]

$$W_d = \frac{kV}{l^2}(T_{SH} - T_{amb}), \tag{4}$$

where $l = 120\ nm$ is the thickness and $V = 4 \times 10^{-14}cm^3$ is the volume of GST [48], [49].

- **Joule Heating (JH) Module:** The heat generation in a PCM cell due to the programming current I_{prog} is

$$W_j = I_{prog}^2 \times R_{PCM}. \tag{5}$$

- **Self-Heating (SH) Module:** The SH temperature of a PCM cell is computed by solving an ordinary differential equation as [48]

$$T_{SH} = \frac{I_{prog}^2 R_{PCM} l^2}{kV} - \left[1 - \exp\left(-\frac{kt}{l^2 C}\right)\right] + T_{amb}, \tag{6}$$

where $C = 1.25JK^{-1}cm^{-3}$ is the heat capacity of the GST [48], [49].

The endurance of a PCM cell is computed as [47]

$$\text{Endurance} \approx \frac{t_f}{t_s}, \tag{7}$$

where t_f and t_s are respectively, the failure time and the switching time. In this model, to switch memory state of a PCM cell, an ion (electron) must travel a distance d across insulating matrix (the gate oxide) upon application of the programming current I_{prog}, which results in the write voltage V across the cell. Assuming thermally activated motion of an with activation energy U_s and local self-heating thermal temperature T_{SH}, the switching speed can be approximated as

$$t_s = \frac{d}{v_s} \approx \frac{2d}{fa} exp\left(\frac{U_s}{k_B T_{SH}}\right) exp\left(-\frac{qV}{2k_B T_{SH}}\frac{a}{d}\right), \tag{8}$$

where $d = 10nm$, $a = 0.2nm$, $f = 10^{13}Hz$, and $U_s = 2eV$ [47].

The failure time is computed considering that the endurance failure mechanism is due to thermally activated motion of ions (electrons) across the same distance d but with higher activation energy U_F, so that the average time to failure is

$$t_f = \frac{d}{v_f} \approx \frac{2d}{fa} exp\left(\frac{U_f}{k_B T_{SH}}\right) exp\left(-\frac{qV}{2k_B T_{SH}}\frac{a}{d}\right) \tag{9}$$

where $U_f = 3ev$ [47].

The endurance, which is the ratio of average failure time and switching time, is given by

$$\text{Endurance} \approx \frac{t_f}{t_s} \approx \exp\left(\frac{\gamma}{T_{SH}}\right), \tag{10}$$

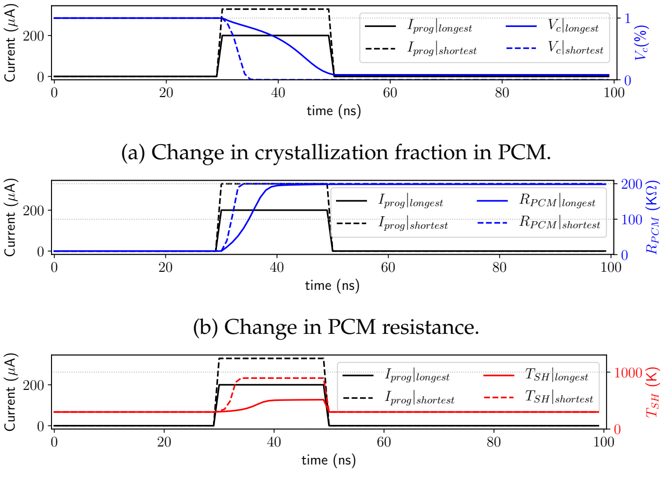
where $\gamma = 1000$ is a fitting parameter [47].

The thermal and endurance models are used in our SNN mapping framework to improve endurance of neuromorphic hardware platforms (see Section 8). Although we have demonstrated our proposed SNN mapping approach using these models (see Section 5), the mapping approach can be trivially extended to incorporate other published models.

4.1 Model Prediction

The thermal and endurance models in Equations 6 and 10, respectively are integrated as follows. The self-heating temperature of Equation 6 is first computed using the PCM's programming current. This self-heating temperature is then used to compute the endurance using Equation 10.

Figure 8 shows the simulation of the proposed model with programming currents of $200\mu A$ and $329\mu A$, which correspond to the longest and shortest current paths in a 65nm 128x128 PCM crossbar at 298K. Figures 8a, 8b, and 8c plot respectively, the crystallization fraction, the PCM resistance, and the temperature for these two current values. We make the following two key observations.



(a) Change in crystallization fraction in PCM.

(b) Change in PCM resistance.

(c) Change in PCM temperature.

Fig. 8. Validation of the proposed model.

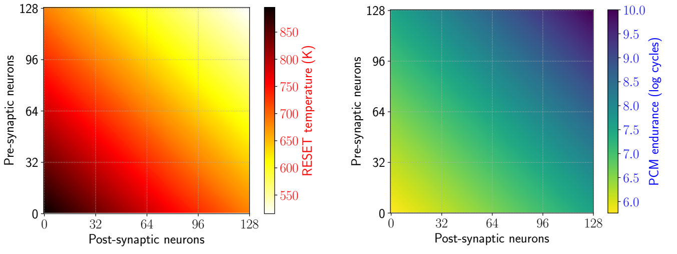
First, the speed of amorphization depends on the current, i.e., with higher programming current, the GST material amorphizes faster. This means that the PCM cells on shorter current paths are faster to program. Second, the self-heating temperature is higher for higher programming current. This means that PCM cells on shorter current paths have lower endurance.

Figure 8 is consistent with the change in crystallization volume, resistance, and self-heating temperature in PCM cells as reported in [48], [49]. Figure 9 plots the temperature and endurance maps of a 128x128 crossbar at 65nm process node with $T_{amb} = 298K$. The PCM cells at the bottom-left corner have higher self-heating temperature than at the top-right corner. This asymmetry in the self-heating temperature creates a wide distribution of endurance, ranging from 10^6 cycles for PCM cells at the bottom-left corner to 10^{10} cycles at the top-right corner. These endurance values are consistent with the values reported for recent PCM chips from IBM [53].

Our goal is to assign synapses with higher activation towards the top-right corner using an intelligent SNN mapping technique, which we describe next.

5 Endurance-Aware Intelligent Neuron and Synapse Mapping

We present eSpine, our novel endurance-aware technique to map SNNs to neuromorphic hardware. To this end, we first



(a) Thermal map for PCM RESET operations in a 128x128 crossbar.

(b) Endurance map of the PCM cells in a 128x128 crossbar.

Fig. 9. Temperature and endurance map of a 128x128 crossbar at 65nm process node with $T_{amb} = 298K$.

formulate a joint metric *effective lifetime* ($\mathcal{L}_{i,j}$), defined for the memristor connecting the i^{th} pre-synaptic neuron with j^{th} post-synaptic neuron in a memristive crossbar as

$$\mathcal{L}_{i,j} = \mathcal{E}_{i,j}/a_{i,j}, \tag{11}$$

where $a_{i,j}$ is the number of synaptic activations of the memristor in a given SNN workload and $\mathcal{E}_{i,j}$ is its endurance. Equation 11 combines the effect of software (SNN mapping) on hardware (endurance and temperature) in neuromorphic computing. eSpine aims to maximize the minimum normalized lifetime, i.e.,

$$F_{\text{opt}} = \text{maximize}\{\min_{i,j} \mathcal{L}_{i,j}\} \tag{12}$$

In most earlier works on wear-leveling in the context of non-volatile main memory (e.g., Flash), lifetime is computed in terms of utilization of NVM cells, ignoring the variability of endurance within the device. Instead, we formulate the effective lifetime by considering a memristor's endurance and its utilization in a workload. This is to allow cells with higher endurance to have higher utilization in a workload.

5.1 High-level Overview

Figure 10 shows a high-level overview of eSpine, consisting of three abstraction layers – the application layer, system software layer, and hardware layer. A machine learning application is first simulated using PyCARL [45], which uses CARLsim [54] for training and testing of SNNs. PyCARL estimates spike times and synaptic strength on every connection in an SNN. This constitutes the workload of the machine learning application. eSpine maps and places neurons and synapses of a workload to crossbars of a neuromorphic hardware, improving the effective lifetime. To this end, a machine learning workload is first analyzed to generate clusters of neurons and synapses, where each cluster can fit on a crossbar. eSpine uses the Kernighan-Lin Graph Partitioning algorithm of SpiNeMap [13] to partition an SNN workload, minimizing the inter-cluster spike communication (see Table 1 for comparison of eSpine with SpiNeMap). By reducing the inter-cluster communication, eSpine reduces the energy consumption and latency on the shared interconnect (see Sec. 8.2). Next, eSpine uses an instance of the Particle Swarm Optimization (PSO) [55] to map the clusters to the tiles of a hardware, maximizing the minimum effective lifetime of memristors (Equation 11) in each tile's crossbar. Synapses of a cluster are implemented on memristors using the synapse-to-memristor mapping, ensuring that those with higher activation are mapped to memristors with higher endurance, and vice versa.

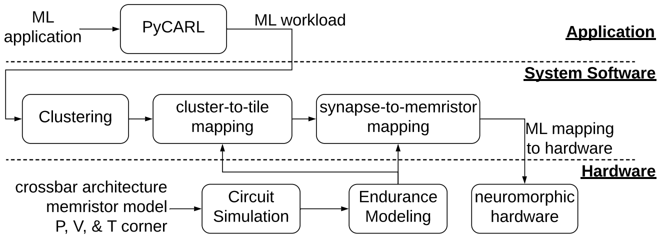


Fig. 10. High-level overview of eSpine.

To perform the optimization using PSO, eSpine uses crossbar specification, including its dimensions, architecture, and memristor technology, and performs circuit simulations at a target P, V, and T corner. Extracted currents in the crossbar are used in the endurance model (see Sec. 4) to generate the endurance map, which is then used in the cluster-to-tile and synapse-to-memristor mapping, optimizing the effective lifetime.

Table 1 reports the differences between the objective function of SpiNeMap and eSpine. In addition to the comparison between SpiNeMap and eSpine, we also show the performance of a hybrid approach SpiNeMap++ (see Fig. 14), which uses the synapse-to-memristor mapping of eSpine with SpiNeMap. See our results in Section 8.

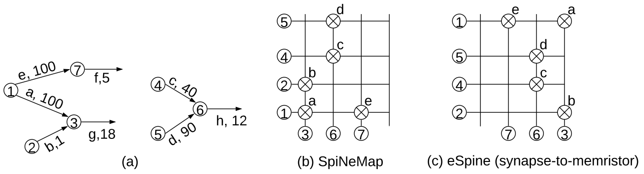
TABLE 1
eSpine vs. SpiNeMap [13].

		SpiNeMap [13]	eSpine (proposed)
Clustering	Algorithm	Kernighan-Lin Graph Partitioning [56]	Kernighan-Lin Graph Partitioning [56]
	Objective	Energy	Energy
Cluster-to-Tile	Algorithm	PSO	PSO
	Objective	Energy	Effective Lifetime
Synapse-to-Memristor	Algorithm	—	Sorting heuristic
	Objective		Effective Lifetime

Although PSO is previously proposed in SpiNeMap, our novelty is in the use of the proposed synapse-to-memristor mapping step, which is integrated inside each PSO iteration to find the minimum effective lifetime.

5.2 Heuristic-based Synapse-to-Memristor Mapping

Figure 11 illustrates the synapse-to-memristor mapping of eSpine and how it differs from SpiNeMap. Figure 11a illustrates the implementation of four pre-synaptic and three post-synaptic neurons on a 4x4 crossbar. The letter and number on a connection indicate the synaptic weight and number of activation, respectively. Existing technique such as SpiNeMap maps synapses arbitrarily on memristors. As a result, a synapse with higher activation may get placed at the bottom-left corner of a crossbar where memristors have lower endurance (see Fig. 11b). eSpine, on the other hand, incorporates the endurance variability in its synapse-to-memristor mapping process. It first sorts pre-synaptic neurons based on their activation, and then allocates them such that those with higher activation are placed at the top-right corners, where memristors have higher endurance (see Fig. 11c). Once the pre-synaptic neurons are placed along the rows, the post-synpatic neurons are placed along the columns, considering their connection to the pre-synaptic neurons, and their activation. In other words, post-synaptic neurons with higher activation are placed towards the right corner of a crossbar. This is shown in Fig. 11c, where the post-synaptic neuron 7 (with 5 activation) is mapped to the left of the post-synaptic neuron 3 (with 18 activation), both of which receives input from the same pre-synaptic neuron 1. This is done to incorporate the online weight update mechanism in SNNs, which depend on both the pre- and post-synaptic activation (see Section 7.1). This synapse-to-memristor mapping is part of Alg. 1 (lines 9-10).



(a) (b) SpiNeMap (c) eSpine (synapse-to-memristor)

Fig. 11. Synapse-to-memristor mapping of eSpine.

5.3 PSO-based Cluster-to-Tile Mapping

To formulate the PSO-based optimization problem, let $G(C, S)$ be a machine learning workload with a set C of clusters and a set S of connections between the clusters. The workload is to be executed on a hardware $H(T, L)$ with a set T of tiles (each tile has one crossbar) and a set L of links between the tiles. Mapping of the application G to the hardware H, $\mathcal{M} = \{m_{x,y}\}$ is defined as

$$m_{x,y} = \begin{cases} 1 & \text{if cluster } c_x \in C \text{ is mapped to tile } t_y \in T \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

Algorithm 1 computes the minimum effective lifetime of all memristors in the hardware for a given mapping $\mathcal{M}$.

Algorithm 1: `MinEffLife()`: Compute minimum effective lifetime of crossbars for mapping $\mathcal{M}$.

```
   Input: $\mathcal{M}$
   Output: $\mathcal{L}$
1  for $t_y \in T$ /* iterate for each tile in the hardware */
2  do
3      $S_y = \{c_x\} \ni m_{x,y} = 1$/* clusters mapped to $t_y$ */
4      $\mathcal{L}^y_{i,j} = 0 \; \forall \; \{i,j\} \in 1, 2, \cdots, M$/* Initialize the effective lifetime on tile $t_y$. */
5      for $c_k \in S_y$ /* iterate for each cluster */
6      do
7          $N_k = \{n\}$/* pre-synaptic neurons of $c_k$ */
8          $A_k = \{a\}$/* number of activations of $n$ */
9          sort $A_k$/* sort the pre-synaptic neurons in descending order of their activations. */
10         map $N_k$ to the crossbar using sorted $A_k$/* place the pre-synaptic neurons sorted by their activations starting from the farthest input in the crossbar. */
11         repeat lines 7-10 for post-synaptic neurons;
12         $\mathcal{L}^y_{i,j} = \mathcal{L}^y_{i,j} + \mathcal{E}_{i,j}/a_{i,j}$ /* using Equation 11 */
13     end
14     $\mathcal{L}_y = \min\{\mathcal{L}^y_{i,j}\}$/* minimum effective lifetime */
15 end
16 return $\min\{\mathcal{L}_y\}$/* return minimum effective lifetime of all crossbars */
```

For each tile, the algorithm first records all clusters mapped to the tile in the set S_y (line 3), and initializes the effective lifetime of the crossbar on the tile (line 4). For each cluster mapped to the tile, the algorithm records all its pre-synaptic neurons in the set N_k (line 7) and their activation, i.e., the number of spikes in the set A_k (line 8). The two sets are sorted in descending order of A_k (line 9). Next, the cluster (i.e, pre-synaptic neurons, post-synaptic neurons, and their synaptic connections) is placed on the crossbar (line 10-11). To do so, pre-synaptic neurons with higher activation are mapped farther from the origin (see Fig. 11) to ensure they are on longer current paths. This is to incorporate the endurance variability within each crossbar. The post-synaptic neurons are mapped along the columns by sorting their activation. With this mapping, the effective lifetime is computed (line 12). The minimum effective lifetime is retained (line 14). The algorithm is repeated for all tiles of the hardware. Finally, the minimum effective lifetime of all crossbars in the hardware is returned (line 16).

The **fitness function** of eSpine is

$$F = \texttt{MinEffLife}(\mathcal{M}) \tag{14}$$

The **optimization objective** of eSpine is

$$\mathcal{L}_{\text{min}} = \mathcal{L}_{\text{a}}, \text{ where } a = \arg\min\{\text{MinEffLife}(\mathcal{M}_i) | i \in 1, 2, \cdots\}, \tag{15}$$

The constraint to this optimization problem is that a cluster can map to exactly 1 tile, i.e.,

$$\sum_y m_{x,y} = 1 \;\forall\; x \tag{16}$$

To solve Equation 15 using PSO, we instantiate n_p swarm particles. The position of these particles are solutions to the fitness functions, and they represent cluster mappings, i.e., $\mathcal{M}$'s in Equation 15. Each particle also has a velocity with which it moves in the search space to find the optimum solution. During the movement, a particle updates its position and velocity according to its own experience (closeness to the optimum) and also experience of its neighbors. We introduce the following notations.

$$\begin{aligned} D &= |\mathcal{C}| \times |\mathcal{V}| = \text{dimensions of the search space} \\ \mathbf{\Theta} &= \{\theta_l \in \mathbb{R}^D\}_{l=0}^{n_p-1} = \text{positions of particles in the swarm} \\ \mathbf{V} &= \{\mathbf{v}_l \in \mathbb{R}^D\}_{l=0}^{n_p-1} = \text{velocity of particles in the swarm} \end{aligned} \tag{17}$$

Position and velocity of swarm particles are updated, and the fitness function is computed as

$$\begin{aligned} \mathbf{\Theta}(t+1) &= \mathbf{\Theta}(t) + \mathbf{V}(t+1) \\ \mathbf{V}(t+1) &= \mathbf{V}(t) + \varphi_1 \cdot \Big(P_{\text{best}} - \mathbf{\Theta}(t)\Big) + \varphi_2 \cdot \Big(G_{\text{best}} - \mathbf{\Theta}(t)\Big) \\ F(\theta_l) &= \mathcal{L}_l = \text{MinEffLife}(M_l) \end{aligned} \tag{18}$$

where t is the iteration number, φ_1, φ_2 are constants and P_{best} (and G_{best}) is the particle's own (and neighbors) experience. Finally, local and global bests are updated as

$$\begin{aligned} P^l_{\text{best}} &= F(\theta_l) \text{ if } F(\theta_l) < F(P^l_{\text{best}}) \\ G_{\text{best}} &= \underset{l=0,\ldots n_p-1}{\arg\min}\; P^l_{\text{best}} \end{aligned} \tag{19}$$

Due to the binary formulation of the mapping problem (see Equation 13), we need to binarize the velocity and position of Equation 17, which we illustrate below.

$$\begin{aligned} \hat{\mathbf{V}} &= \texttt{sigmoid}(\mathbf{V}) = \frac{1}{1+\text{e}^{-\mathbf{V}}} \\ \hat{\Theta} &= \begin{cases} 0 & \text{if } \texttt{rand()} < \hat{\mathbf{V}} \\ 1 & \text{otherwise} \end{cases} \end{aligned} \tag{20}$$

Figure 12 illustrates the PSO algorithm. The algorithm first initializes positions of the PSO particles (13). Next, the algorithm runs for N_{PSO} iterations. At each iteration, the PSO algorithm evaluates the fitness function (F) and updates its position based on the local and global best positions (Equation 18), binarizing these updates using Equation 20.

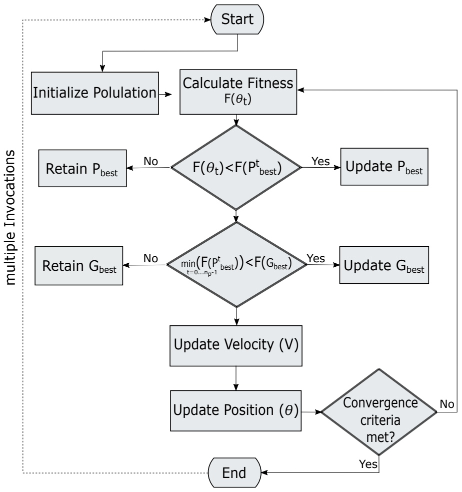

Start
Initialize Polulation
Calculate Fitness $F(\theta_t)$
Retain P_{best}
No
$F(\theta_t)<F(P^t_{best})$
Yes
Update P_{best}
Retain G_{best}
No
$\min_{t=0\ldots n_p-1}(F(P^t_{best}))<F(G_{best})$
Yes
Update G_{best}
Update Velocity (V)
Update Position (θ)
Convergence criteria met?
No
Yes
End
multiple Invocations

Fig. 12. Flow chart of our PSO algorithm.

The PSO algorithm of eSpine can be used to explore the energy and lifetime landscape of different neuron mapping solutions to the hardware. Section 8.3 illustrates such exploration for a representative application. eSpine gives designers the flexibility to combine energy and lifetime metrics beyond simply obtaining the minimum energy and maximum lifetime mappings (for instance, minimizing energy for a given lifetime target, and vice versa).

6 Extended Scope of eSpine

6.1 Other Memristor Technologies

Temperature-related endurance issues are also critical for other memristor technologies such as FeRAM and STT-/SOT-MRAM. A thermal model for Magnetic Tunnel Junction (MTJ), the basic storage element in STT-MRAM based memoristor, is proposed in [57]. According to this model, the self-heating temperature is due to the spin polarization percentages of the free layer and the pinned layer in the MTJ structure, which are dependent on the programming current. Similarly, a thermal model for FeRAM-based memristor is proposed in [58]. These models can be incorporated directly into our SPICE-level crossbar model to generate the thermal and endurance maps, similar to those presented in Figure 9 for PCM. The proposed cluster-to-tile mapping and the synapse-to-crossbar mapping (see Section 5) can then use these maps to optimize the placement of synapses for a target memristor technology, improving its endurance. Although the exact numerical benefit may differ, eSpine can improve endurance for different memristor technologies.

6.2 Other Reliability Issues

There are other thermal-related reliability issues in memristors, for instance retention-time [59]–[61] and transistor circuit aging [62]. Retention time is defined as the time for which a memristor can retain its programmed state. Recent studies show that retention time reduces significantly with increase in temperature [59]. Retention time issues are relevant for supervised machine learning, where the synaptic weights are programmed on memristors once, during inference. For online learning (which is the focus of this work), synaptic weight update frequency is usually much smaller than the retention time. Therefore, a reduction in retention time is less of a concern. Nevertheless, by lowering the average temperature of crossbars, eSpine also addresses the retention time-related reliability concerns in memristors.

7 Evaluation Methodology

7.1 Use-Case of eSpine

Figure 13 illustrates the use-case of eSpine applied for online machine learning. We use Spike-Timing Dependent Plasticity (STDP) [63], which is an unsupervised learning algorithm for SNNs, where the synaptic weight between a pre- and a post-synaptic neuron is updated based on the timing of pre-synaptic spikes relative to the post-synaptic spikes.[2] STDP is typically used in online settings to improve accuracy of machine learning tasks.

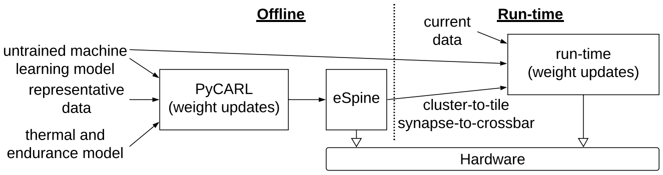

Offline
Run-time
current data
untrained machine learning model
representative data
thermal and endurance model
PyCARL (weight updates)
eSpine
run-time (weight updates)
cluster-to-tile synapse-to-crossbar
Hardware

Fig. 13. Use-Case of eSpine.

A machine learning model is first analyzed offline using PyCARL with representative workload and data set. This is to estimate the relative activation frequency of the neurons in the model when it is trained at run-time using current data. Although neuron activation can deviate at run-time, our more detailed analysis shows that using representative workload and data set, such deviations can be limited to only a few neurons in the model.[3] We have validated this observation for the evaluated applications that use ECG and image data (see Section 7).

The activation information obtained offline is processed using eSpine (see Figure 10 for the details of eSpine) to generate cluster-to-tile and synapse-to-crossbar mappings. The offline trained weight updates are discarded to facilitate relearning of the model from current (in-field) data. The untrained machine learning model is placed onto the hardware using the mappings generated from eSpine.

Although online learning is the main focus, eSpine is also relevant for supervised machine learning, where no weight updates happen at run-time. By mapping the most active neurons to the farthest corner of a crossbar (i.e., on longest current paths), eSpine minimizes crossbar temperature, which reduces 1) leakage current and 2) circuit aging.

7.2 Evaluated Applications

We evaluate 10 SNN-based machine learning applications that are representative of three most commonly-used neural network classes — convolutional neural network (CNN), multi-layer perceptron (MLP), and recurrent neural network (RNN). These applications are 1) LeNet based handwritten digit recognition with 28×28 images of handwritten digits from the MNIST dataset; 2) AlexNet for ImageNet classification; 3) VGG16, also for ImageNet classification; 4) ECG-based heart-beat classification (HeartClass) [64], [65] using electrocardiogram (ECG) data; 5) multi-layer perceptron (MLP)-based handwritten digit recognition (MLP-MNIST) [66] using the MNIST database; 6) edge detection (EdgeDet) [54] on 64×64 images using difference-of-Gaussian; 7) image smoothing (ImgSmooth) [54] on 64×64 images; 8) heart-rate estimation (HeartEstm) [67] using ECG data; 9) RNN-based predictive visual pursuit (VisualPursuit) [68]; and 10) recurrent digit recognition (R-DigitRecog) [66]. Table 2 summarizes the topology, the number of neurons and synapses of these applications, and their baseline accuracy on DYNAP-SE using SpiNeMap [13].

TABLE 2
Applications used to evaluate eSpine.

Class	Applications	Synapses	Neurons	Topology	Accuracy
CNN	LeNet	282,936	20,602	CNN	85.1%
	AlexNet	38,730,222	230,443	CNN	90.7%
	VGG16	99,080,704	554,059	CNN	69.8 %
	HeartClass [64]	1,049,249	153,730	CNN	63.7%
MLP	DigitRecogMLP	79,400	884	FeedForward (784, 100, 10)	91.6%
	EdgeDet [54]	114,057	6,120	FeedForward (4096, 1024, 1024, 1024)	100%
	ImgSmooth [54]	9,025	4,096	FeedForward (4096, 1024)	100%
RNN	HeartEstm [67]	66,406	166	Recurrent Reservoir	100%
	VisualPursuit [68]	163,880	205	Recurrent Reservoir	47.3%
	R-DigitRecog [66]	11,442	567	Recurrent Reservoir	83.6%

7.3 Hardware Models

We model the DYNAP-SE neuromorphic hardware [2] with the following configurations.

- A tiled array of 4 tiles, each with a 128x128 crossbar. There are 65,536 memristors per crossbar.
- Spikes are digitized and communicated between cores through a mesh routing network using the Address Event Representation (AER) protocol.
- Each synaptic element is a PCM-based memristor.

To test the scalability of eSpine, we also evaluate DYNAP-SE with 16 and 32 tiles.

Table 3 reports the hardware parameters of DYNAP-SE.

TABLE 3
Major simulation parameters extracted from [2].

Neuron technology	65nm CMOS
Synapse technology	PCM
Supply voltage	1.2V
Energy per spike	50pJ at 30Hz spike frequency
Energy per routing	147pJ
Switch bandwidth	1.8G. Events/s

7.4 Evaluated Techniques

We evaluate the following techniques (see Fig. 14).

- **SpiNeMap:** This is the baseline technique to map SNNs to crossbars of a hardware. SpiNeMap generates clusters from an SNN workload, minimizing the inter-cluster communication. Clusters are mapped to

[2] Apart from STDP, many other online learning algorithms depend on the activation of both the pre- and post-synaptic neurons.

[3] In the worst-case, the lifetime obtained using eSpine for these few neurons will be similar to SpiNeMap. However, for most neurons in the model, eSpine significantly outperforms SpiNeMap. Therefore, the lifetime obtained using eSpine is higher (see Section 8.1).

tiles minimizing the energy consumption. Synapses of a cluster are implemented on memristors arbitrarily, without incorporating their endurance.

- **SpiNeMap++:** This is an extension of SpiNeMap, where the cluster-to-tile mapping is performed using SpiNeMap, minimizing energy consumption, and the synapse-to-memristor mapping is performed using eSpine, maximizing effective lifetime.
- **eSpine:** This is another extension of SpiNeMap. eSpine uses only the clustering technique of SpiNeMap, thereby minimizing the inter-cluster communication, which also improves energy consumption and latency. The cluster-to-tile and synapse-to-memristor mappings are performed using PSO, maximizing the effective lifetime. Furthermore, eSpine allows to explore the entire Pareto space of energy and lifetime.

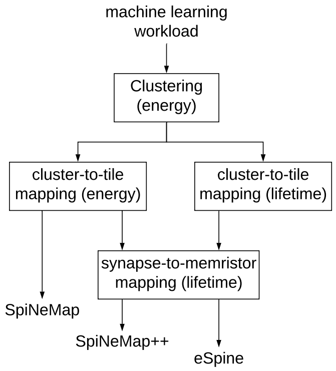

machine learning workload
Clustering (energy)
cluster-to-tile mapping (energy)
cluster-to-tile mapping (lifetime)
synapse-to-memristor mapping (lifetime)
SpiNeMap
SpiNeMap++
eSpine

Fig. 14. Evaluated techniques.

7.5 Evaluated Metric

We evaluate the following metrics.

- **Effective lifetime:** This is the minimum effective lifetime of all memristors in the hardware.
- **Energy consumption:** This is the total energy consumed on the hardware. We also evaluate the static and dynamic energy consumption.
- **Compilation time:** This is the time it takes for the PSO to find a solution.

8 Results and Discussions

8.1 Normalized Lifetime

Figure 15 compares the effective lifetime obtained using each technique for each evaluated application on DYNAP-SE. We make the following two key observations.

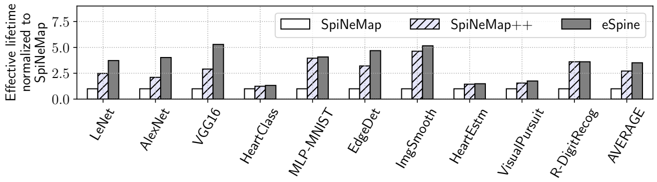


Fig. 15. Effective lifetime for the evaluated applications.

First, between SpiNeMap and SpiNeMap++, SpiNeMap++ has an average 2.7x higher effective lifetime than SpiNeMap. Although both SpiNeMap and SpiNeMap++ have the same cluster-to-tile mapping, SpiNeMap++ maps synapses of a cluster intelligently on memristors of a crossbar, incorporating 1) the endurance variability of memristors in a crossbar and 2) the activation of synapses in a workload. Therefore, SpiNeMap++ has higher effective lifetime than SpiNeMap, which maps synapses arbitrarily to memristors of a crossbar. Second, eSpine has the highest effective lifetime than all evaluated techniques. The effective lifetime of eSpine is higher than SpiNeMap and SpiNeMap++ by average 3.5x and 1.30x, respectively. Although both eSpine and SpiNeMap++ uses the same synapse-to-memristor mapping strategy, i.e., they both implement synapses with higher activation using memristors with higher endurance, the improvement of eSpine is due to the PSO-based cluster-to-tile mapping, which maximizes the effective lifetime. Third, for some applications such as MLP-MNIST and R-DigitRecog, the effective lifetime using eSpine is comparable to SpiNeMap++. For these applications, the cluster-to-tile mapping of SpiNeMap is already optimal in terms of the effective lifetime. For other applications, eSpine is able to find a better mapping, which improves the effective lifetime (by average 38% compared to SpiNeMap++).

8.2 Energy Consumption

Figure 16 reports the energy consumption of SpiNeMap and eSpine on DYNAP-SE, distributed into 1) dynamic energy, which is consumed in crossbars to generate spikes (`dynamic`), 2) communication energy, which is consumed on the shared interconnect to communicate spikes between crossbars (`comm`), and 3) static energy, which is consumed in crossbars due to the leakage current through the access transistor of each memristor cell (`static`). We make the following four key observations.

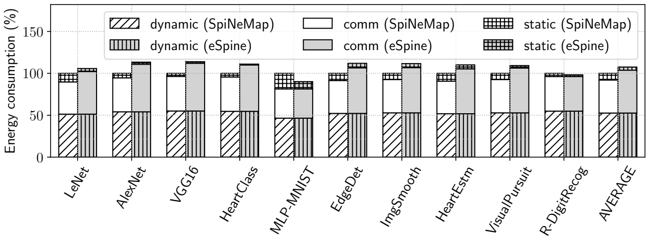


Fig. 16. Energy distribution for the evaluated applications.

First, the dynamic energy, communication energy, and static energy constitute respectively, 52.6%, 39.4%, and 8% of the total energy consumption. Second, eSpine does not alter spike generation, and therefore, the dynamic energy consumption of eSpine is similar to SpiNeMap. Third, eSpine's cluster-to-tile mapping strategy is to optimize the effective lifetime, while SpiNeMap allocates clusters to tiles minimizing the energy consumption on the shared interconnect. Therefore, the communication energy of SpiNeMap is lower than eSpine by an average of 21.4%. Finally, eSpine reduces the average temperature of each crossbar by implementing synapses with higher activation on longer current paths where memristors have lower self-heating temperature. Therefore, the leakage power consumption of eSpine is on average 52% lower than SpiNeMap.

8.3 Energy Tradeoffs

Figure 17 shows the normalized effective lifetime and the normalized energy of the mappings explored using the PSO algorithm for LeNet. The figure shows the mappings that are Pareto optimal with respect to lifetime and energy.

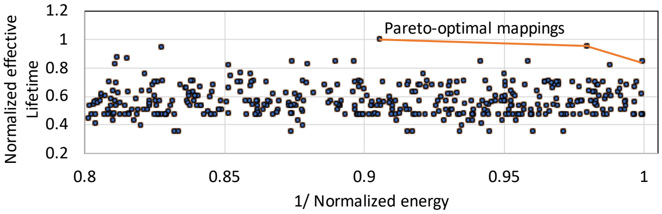


Fig. 17. Mapping explorations for LeNet.

Figure 18 reports the energy consumption of SpiNeMap, SpiNeMap++, and eSpine on DYNAP-SE for each evaluated application. We make the following two key observations.

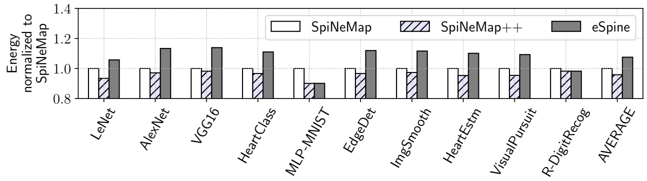


Fig. 18. Energy consumption for the evaluated applications.

First, the energy consumption of SpiNeMap++ is lower than SpiNeMap by an average of 4%. This reduction is due to the reduction of leakage current, which is achieved by using memristors with lower self-heating temperature. The energy consumption of eSpine is higher than both SpiNeMap and SpiNeMap++ by an average of 7.5% and 11.6%, respectively. Although eSpine, like SpiNeMap++, lowers the static energy consumption by its intelligent synapse-to-memristor mapping, the higher energy consumption of eSpine is due to the increase in the energy consumption on the shared interconnect of the hardware. However, by using an energy-aware clustering technique to begin with, eSpine ensures that the energy consumption is not excessively higher. From the results of Sections 8.1 & 8.3, we make the following two key conclusions. First, SpiNeMap++, which is SpiNeMap combined with the proposed synapse-to-memristor mapping, is best in terms of energy, achieving 2.7x higher lifetime than SpiNeMap. Second, eSpine, which is our proposed cluster-to-tile and synapse-to-memristor mappings combined, is best in terms of lifetime, achieving 3.5x higher lifetime than SpiNeMap.

8.4 Performance

Table 4 reports the performance of the evaluated applications using eSpine (Column 3). Results are compared against Baseline, which uses PyCARL [45] to estimate the accuracy of these applications on hardware assuming that the current injected in each memristor is what is needed for its synaptic weight update (Column 2). The table also reports the accuracy using eSpine, where the synaptic weights are scaled as proposed in [40] to compensate for the accuracy loss due to the current imbalance in a crossbar (Column 4). We make the following two key observations.

First, the Baseline has the highest accuracy of all. This is because, the PyCARL framework of Baseline assumes that the current through all memristors in a crossbar are the same. Second, current imbalance can lead to a difference between the expected and actual synaptic plasticity based on the specific memristor being accessed. Therefore, we see an average 3% reduction in accuracy using eSpine. However, the current imbalance-aware synapse update strategy, when combined with eSpine can solve this problem. In fact, we estimate that the accuracy of machine learning applications using this synaptic update strategy is on average 2% higher than eSpine and only 1% lower than the Baseline.

TABLE 4
Accuracy of Baseline (PyCARL [45]), eSpine, and eSpine combined with [40] for the evaluated applications.

Application	Accuracy (%)			Application	Accuracy (%)		
	Baseline	eSpine	eSpine + [40]		Baseline	eSpine	eSpine + [40]
LeNet	85.1	84.2	85.0	AlexNet	90.7	88.7	89.8
VGG16	69.8	64.4	67.8	HeartClass	63.7	59.2	62.4
MLP-MNIST	91.6	91.3	91.6	EdgeDet	100	86	96.8
ImgSmooth	100	100	100.0	HeartEstm	67.9	67.9	67.9
VisualPursuit	47.3	47.3	47.3	R-DigitRecog	83.6	81.5	83.6

8.5 Average Temperature

Figure 19 plots the average self-heating temperature of the PCM cells in four crossbars in DYNAP-SE executing LeNet workload using SpiNeMap and eSpine. We make the following two observations.

First, eSpine maps active memristive synapses towards the top right corner of a crossbar. However, such mapping does not lead to a significant change in the ambient temperature. This is because of the the chalcogenide alloy (e.g., $Ge_2Sb_2Te_5$ [69]) used to build a PCM cell, which keeps the self-heating temperature of the cell concentrated at the interface between the heating element and the amorphous dome (see Figure 2), with only a negligible spatial heat flow to the surrounding [70].

Second, the average self-heating temperature of eSpine is lower than SpiNeMap. This is because of the synapse-to-memristor mapping technique of eSpine, which places synapses with higher activation on longer current paths, where the self-heating temperature of a memristor is lower. By reducing the average temperature, eSpine lowers the leakage current through the access transistor of a memristor, which we discussed in Section 8.2.

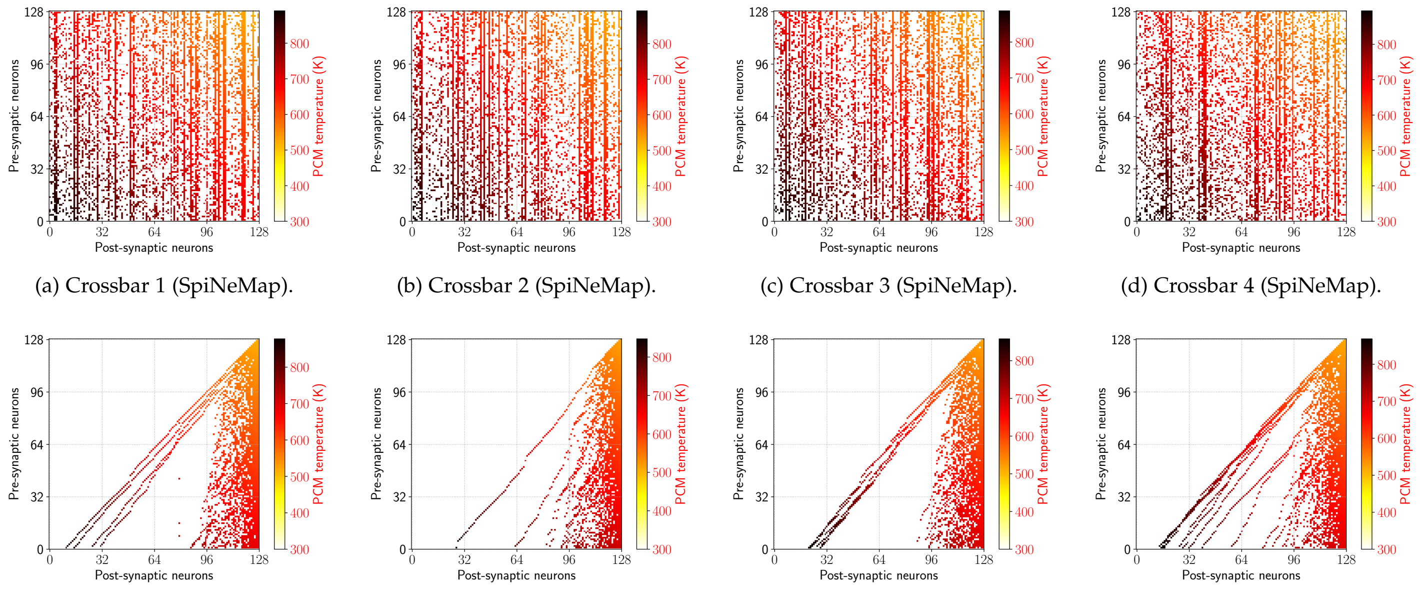
8.6 Resource Scaling

Figure 20 compares the lifetime normalized to SpiNeMap for each evaluated application on DYNAP-SE with 4-tile (4 crossbars), 16-tile (16 crossbars), and 32-tile (32 crossbars).

We observe that with 4, 16, and 32 tiles in the system, eSpine provides an average 3.5x, 5.3x, and 6.4x lifetime improvement, respectively for the evaluated applications compared to SpiNeMap. This is because with more tiles in the system, the workload gets distributed across the available crossbars of the hardware, resulting in lower average utilization of memristors, improving their lifetime.

8.7 Compilation Time

Table 5 reports eSpine's compilation time and the effective lifetime normalized to SpiNeMap for three different settings of PSO iterations. We observe that as the number of PSO iterations is increased, the effective lifetime increases for all applications. This is because with increase in the number of iterations, the PSO is able to find a better solution.



(a) Crossbar 1 (SpiNeMap). (b) Crossbar 2 (SpiNeMap). (c) Crossbar 3 (SpiNeMap). (d) Crossbar 4 (SpiNeMap).

(e) Crossbar 1 (eSpine). (f) Crossbar 2 (eSpine). (g) Crossbar 3 (eSpine). (h) Crossbar 4 (eSpine).

Fig. 19. Average temperature of the four crossbars in DYNAP-SE executing LeNet workload using SpiNeMap and eSpine.

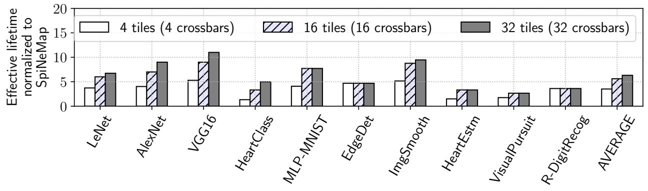


Fig. 20. Lifetime normalized to SpiNeMap for the evaluated applications on DYNAP-SE with 4, 16, and 32 tiles.

However, the compilation time also increases. We observe that the compilation time is significantly large for larger applications like VGG16 with 100 PSO iterations. However, we note that the PSO-based optimization is performed once at design-time. Furthermore, the PSO-iterations is a user-defined parameter, and therefore, it can be set to a lower value to generate a faster mapping solution, albeit a lower lifetime improvement. Finally, we observe that increasing the PSO iterations beyond 100 leads to a significant increase in the compilation time for all applications with minimal improvement of their effective lifetime.

TABLE 5
Compilation time and solution quality tradeoff.

Application	PSO Iterations = 1		PSO Iterations = 10		PSO Iterations = 100	
	Compilation Time (sec)	Norm. Effective Lifetime	Compilation Time (sec)	Norm. Effective Lifetime	Compilation Time (sec)	Norm. Effective Lifetime
LeNet	232.8	2.5	1,650.6	3.4	23,311.4	3.7
AlexNet	331.7	2.1	2,431.8	3.1	45,617.4	4.0
VGG16	886.8	2.9	8,156.0	4.2	110,123.6	5.3
HeartClass	731.5	1.2	7,796.9	1.2	79,557.9	1.3
MLP-MNIST	3.4	4.0	17.2	4.1	327.3	4.1
EdgeDet	37.7	3.2	225.5	3.8	3,909.2	4.7
ImgSmooth	26.2	4.6	91.1	4.6	1,327.4	5.2
HeartEstm	109.0	1.4	595.1	1.4	7,303.6	1.5
VisualPursuit	112.8	1.6	1,139.7	1.8	17,183.7	1.8
R-DigitRecog	28.5	3.6	127.7	3.6	2,155.6	3.6

9 Conclusion

In this work, we present eSpine, a simple, yet powerful technique to improve the effective lifetime of memristor-based neuromorphic hardware in executing SNN-based machine learning workloads. eSpine is based on detailed circuit simulations at different process, voltage, and temperature corners to estimate parasitic voltage drops on different current paths in a memristive crossbar. The circuit parameters are used in a compact endurance model to estimate the endurance variability in a crossbar. This endurance variability is then used within a design-space exploration framework for mapping neurons and synapses of a workload to crossbars of a hardware, ensuring that synapses with higher activation are implemented on memristors with higher endurance, and vice versa. The mapping is explored using an instance of the Particle Swarm Optimization (PSO). We evaluate eSpine using 10 SNN workloads representing commonly-used machine learning approaches. Our results for DYNAP-SE, a state-of-the-art neuromorphic hardware demonstrate the significant improvement of effective lifetime of memristors in a neuromorphic hardware.

Acknowledgments

This work is supported by the National Science Foundation Faculty Early Career Development Award CCF-1942697 (CAREER: Facilitating Dependable Neuromorphic Computing: Vision, Architecture, and Impact on Programmability).

References

[1] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, 1997.

[2] S. Moradi, N. Qiao, F. Stefanini *et al.*, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *TBCAS*, 2017.

[3] M. V. Debole, B. Taba, A. Amir *et al.*, "TrueNorth: Accelerating from zero to 64 million neurons in 10 years," *Computer*, 2019.

[4] M. Davies, N. Srinivasa, T. H. Lin *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, 2018.

[5] A. Balaji, Y. Wu, A. Das, F. Catthoor, and S. Schaafsma, "Exploration of segmented bus as scalable global interconnect for neuromorphic computing," in *GLSVLSI*, 2019.

[6] A. Mallik, D. Garbin, A. Fantini, D. Rodopoulos, R. Degraeve *et al.*, "Design-technology co-optimization for OxRRAM-based synaptic processing unit," in *VLSIT*, 2017.

[7] G. W. Burr, R. M. Shelby *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, 2017.

[8] P. Wijesinghe, A. Ankit, A. Sengupta, and K. Roy, "An all-memristor deep spiking neural computing system: A step toward realizing the low-power stochastic brain," *TETCI*, 2018.

[9] M. Hu, H. Li, Y. Chen, Q. Wu *et al.*, "Memristor crossbar-based neuromorphic computing system: A case study," *TNNLS*, 2014.

[10] W. Wan, R. Kubendran, S. B. Eryilmaz, W. Zhang, Y. Liao *et al.*, "33.1 A 74 TMACS/W CMOS-RRAM Neurosynaptic Core with Dynamically Reconfigurable Dataflow and In-situ Transposable Weights for Probabilistic Graphical Models," in *ISSCC*, 2020.

[11] W. Wan *et al.*, "A Voltage-Mode Sensing Scheme with Differential-Row Weight Mapping for Energy-Efficient RRAM-Based In-Memory Computing," in *VLSIT*, 2020.

[12] S. Song, A. Balaji, A. Das, N. Kandasamy *et al.*, "Compiling spiking neural networks to neuromorphic hardware," in *LCTES*, 2020.

[13] A. Balaji, A. Das, Y. Wu, K. Huynh *et al.*, "Mapping spiking neural networks to neuromorphic hardware," *TVLSI*, 2020.

[14] A. Das, Y. Wu, K. Huynh *et al.*, "Mapping of local and global synapses on spiking neuromorphic hardware," in *DATE*, 2018.

[15] T. Titirsha and A. Das, "Thermal-aware compilation of spiking neural networks to neuromorphic hardware," in *LCPC*, 2020.

[16] T. Titirsha and A. Das, "Reliability-performance trade-offs in neuromorphic computing," in *CUT*, 2020.

[17] A. Balaji, S. Song, A. Das, J. Krichmar, N. Dutt *et al.*, "Enabling resource-aware mapping of spiking neural networks via spatial decomposition," *ESL*, 2020.

[18] A. Balaji, T. Marty, A. Das, and F. Catthoor, "Run-time mapping of spiking neural networks to neuromorphic hardware," *JSPS*, 2020.

[19] S. Song, A. Das *et al.*, "Improving dependability of neuromorphic computing with non-volatile memory," in *EDCC*, 2020.

[20] A. Balaji, S. Song, A. Das, N. Dutt, J. Krichmar *et al.*, "A framework to explore workload-specific performance and lifetime trade-offs in neuromorphic computing," *CAL*, 2019.

[21] S. Song and A. Das, "A case for lifetime reliability-aware neuromorphic computing," in *MWSCAS*, 2020.

[22] D. L. Silver, Q. Yang, and L. Li, "Lifelong machine learning systems: Beyond learning algorithms," in *AAAI*, 2013.

[23] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras *et al.*, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *MICRO*, 2009.

[24] L.-P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *SAC*, 2007.

[25] L.-P. Chang, T.-W. Kuo, and S.-W. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *TECS*, 2004.

[26] J. Liao, F. Zhang, L. Li, and G. Xiao, "Adaptive wear-leveling in flash-based memory," *CAL*, 2014.

[27] W. Li, Z. Shuai, C. J. Xue, M. Yuan, and Q. Li, "A wear leveling aware memory allocator for both stack and heap management in pcm-based main memory systems," in *DATE*, 2019.

[28] M. Jerry, P.-Y. Chen *et al.*, "Ferroelectric fet analog synapse for acceleration of deep neural network training," in *IEDM*, 2017.

[29] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, 2003.

[30] A. F. Vincent, J. Larroque, N. Locatelli, N. B. Romdhane, O. Bichler *et al.*, "Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," *TBCAS*, 2015.

[31] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Enabling and exploiting partition-level parallelism (PALP) in phase change memories," *TECS*, 2019.

[32] S. Song, A. Das *et al.*, "Exploiting inter-and intra-memory asymmetries for data mapping in hybrid tiered-memories," in *International Symposium on Memory Management (ISMM)*, 2020.

[33] S. Song, A. Das, O. Mutlu *et al.*, "Improving phase change memory performance with data content aware access," in *ISMM*, 2020.

[34] S. Song, A. Das, O. Mutlu *et al.*, "Aging aware request scheduling for non-volatile main memory," in *ASP-DAC*, 2021.

[35] J.-Y. Kweon, Y.-H. Song, and T. T.-H. Kim, "Modelling of phase change memory (PCM) cell for circuit simulation," in *ISOCC*, 2019.

[36] P. Junsangsri, F. Lombardi, and J. Han, "Macromodeling a phase change memory (PCM) cell by HSPICE," in *NANOARCH*, 2012.

[37] W. Zhao and Y. Cao, "Predictive technology model for nano-cmos design exploration," *JETC*, 2007.

[38] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "Modeling and analysis of passive switching crossbar arrays," *TCAS I*, 2017.

[39] J. Woo and S. Yu, "Resistive memory-based analog synapse: The pursuit for linear and symmetric weight update," *Nanotechnology Magazine*, 2018.

[40] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects," in *AICAS*, 2020.

[41] W. Wen, Y. Zhang, and J. Yang, "Renew: Enhancing lifetime for reram crossbar based neural network accelerators," in *ICCD*, 2019.

[42] G. Indiveri, "A low-power adaptive integrate-and-fire neuron circuit," in *ISCAS*, 2003.

[43] A. Ciprut and E. G. Friedman, "Modeling size limitations of resistive crossbar array with cell selectors," *TVLSI*, 2016.

[44] K. Son, K. Cho, S. Kim, S. Park, D. H. Jung *et al.*, "Signal integrity design and analysis of 3-d x-point memory considering crosstalk and ir drop for higher performance computing," *TCPMT*, 2020.

[45] A. Balaji *et al.*, "PyCARL: A PyNN interface for hardware-software co-simulation of spiking neural network," in *IJCNN*, 2020.

[46] Y. Ji, Y. Zhang, W. Chen, and Y. Xie, "Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler," in *ASPLOS*, 2018.

[47] D. B. Strukov, "Endurance-write-speed tradeoffs in nonvolatile memories," *Applied Physics A: Materials Science and Processing*, 2016.

[48] L. Xi, S. Zhitang, C. Daolin *et al.*, "An spice model for phase-change memory simulations," *Journal of Semiconductors*, 2011.

[49] G. Marcolini, F. Giovanardi, M. Rudan, F. Buscemi, E. Piccinini *et al.*, "Modeling the dynamic self-heating of PCM," in *ESSDERC*, 2013.

[50] A. Das, A. Kumar, and B. Veeravalli, "Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems," *TPDS*, 2015.

[51] Y. B. Liao, J. T. Lin *et al.*, "Temperature-based phase change memory model for pulsing scheme assessment," *ICICDT*, 2008.

[52] K. C. Kwong, L. Li, J. He, and M. Chan, "Verilog-A model for phase change memory simulation," *ICSICT*, 2008.

[53] G. W. Burr, M. J. Brightsky, A. Sebastian, H.-Y. Cheng, J.-Y. Wu *et al.*, "Recent progress in phase-change memory technology," *JETCAS*, 2016.

[54] T. Chou, H. Kashyap, J. Xing *et al.*, "CARLsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters," in *IJCNN*, 2018.

[55] J. Kennedy, "Particle swarm optimization," *Encyclopedia of machine learning*, 2010.

[56] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, 1970.

[57] L. Zhang, Y. Cheng, W. Kang, L. Torres, Y. Zhang *et al.*, "Addressing the thermal issues of STT-MRAM from compact modeling to design techniques," *TNANO*, 2018.

[58] A. Gupta, K. Ni, O. Prakash *et al.*, "Temperature dependence and temperature-aware sensing in Ferroelectric FET," in *IRPS*, 2020.

[59] M. Stanisavljevic, A. Athmanathan, N. Papandreou, H. Pozidis *et al.*, "Phase-change memory: Feasibility of reliable multilevel-cell storage and retention at elevated temperatures," in *IRPS*, 2015.

[60] A. Bhattacharjee and P. Panda, "Rethinking non-idealities in memristive crossbars for adversarial robustness in neural networks," *arXiv preprint arXiv:2008.11298*, 2020.

[61] A. M. Zyarah and D. Kudithipudi, "Semi-trained memristive crossbar computing engine with in situ learning accelerator," *JETC*, 2018.

[62] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *DATE*, 2019.

[63] Y. Dan and M.-m. Poo, "Spike timing-dependent plasticity of neural circuits," *Neuron*, 2004.

[64] A. Balaji, F. Corradi *et al.*, "Power-accuracy trade-offs for heartbeat classification on neural networks hardware," *JOLPE*, 2018.

[65] A. Das, F. Catthoor, and S. Schaafsma, "Heartbeat classification in wearables using multi-layer perceptron and time-frequency joint distribution of ECG," in *CHASE*, 2018.

[66] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, 2015.

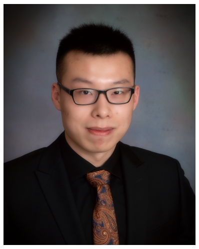
[67] A. Das, P. Pradhapan, W. Groenendaal, P. Adiraju, R. Rajan *et al.*, "Unsupervised heart-rate estimation in wearables with Liquid states and a probabilistic readout," *Neural Networks*, 2018.

[68] H. J. Kashyap, G. Detorakis, N. Dutt, J. L. Krichmar, and E. Neftci, "A recurrent neural network based model of predictive smooth pursuit eye movement in primates," in *IJCNN*, 2018.

[69] S. Ovshinsky, "Reversible electrical switching phenomena in disordered structures," *Physical Review Letters*, 1968.

[70] C. Pigot, M. Bocquet, F. Gilibert, M. Reyboz, O. Cueto *et al.*, "Comprehensive phase-change memory compact model for circuit simulation," *TED*, 2018.

Twisha Titirsha Twisha Titirsha is currently pursuing a Ph.D. degree from the Department of Electrical and Computer Engineering, Drexel University, Philadelphia. She received a Bachelor's degree from Military Institute of Science and Technology, Bangladesh in 2015. Her research interests include computer architecture, non-volatile memory and analog and/or mixed-signal circuit design.



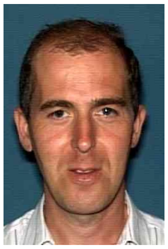
Shihao Song Shihao Song is currently pursuing a Ph.D. degree from Drexel University under the supervision of Dr. Anup Das. He received a Bachelor's degree from Drexel University in 2017. His research interests include computer architecture, non-volatile memory, and compiler design for neuromorphic hardware and accelerators.

Anup Das Dr. Anup Das is an Assistant Professor at Drexel University. He received a Ph.D. in Embedded Systems from National University of Singapore in 2014. Following his Ph.D., he was a post-doctoral fellow at the University of Southampton and a researcher at IMEC. His research focuses on neuromorphic computing and architectural exploration. He is a senior member of the IEEE.

Jeffrey L. Krichmar Jeffrey L. Krichmar received a B.S. in Computer Science in 1983 from the University of Massachusetts at Amherst, a M.S. in Computer Science from The George Washington University in 1991, and a Ph.D. in Computational Sciences and Informatics from George Mason University in 1997. He spent 15 years as a software engineer on projects ranging from the PATRIOT Missile System at the Raytheon Corporation to Air Traffic Control for the Federal Systems Division of IBM. From 1999 to 2007, he was a Senior Fellow in Theoretical Neurobiology at The Neurosciences Institute. He currently is a professor in the Department of Cognitive Sciences and the Department of Computer Science at the University of California, Irvine. He is a Senior Member of IEEE and the Society for Neuroscience.

Nikil D. Dutt Nikil D. Dutt (F) received a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1989, and is currently a Distinguished Professor of Computer Science, Cognitive Sciences, and EECS at the University of California, Irvine. He is also a Distinguished Visiting Professor in the CSE department at IIT Bombay, India. Dutt's research interests are in embedded systems, electronic design automation (EDA), computer systems architecture and software, healthcare IoT, and brain-inspired architectures and computing. He is a Fellow of the ACM, Fellow of the IEEE, and recipient of the IFIP Silver Core Award.

Nagarajan Kandasamy Nagarajan Kandasamy is a Professor in the Department of Electrical and Computer Engineering at Drexel University. His current research interests are in the areas of computer architecture, parallel processing, and embedded and real-time systems. His research has been funded by the National Science Foundation, the U.S. Army Research Office, and the Office of Naval Research, among others. He is a recipient of the NSF CAREER award; best student paper awards in the IEEE Conference on Autonomic Computing, and the Pacific Rim Dependable Computing conference, for work related to performance management and fault detection in computing systems; and a featured article in the Physics in Medicine and Biology Journal for work on GPU-accelerated algorithms for medical imaging. Dr. Kandasamy received his Ph.D. in Computer Science and Engineering from the University of Michigan. He is a senior member of the IEEE.



Francky Catthoor Dr. Francky Catthoor received a Ph.D. in EE from the Katholieke Univ. Leuven, Belgium in 1987. Between 1987 and 2000, he has headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000 he is strongly involved in other activities at IMEC including deep submicron technology aspects, IoT and biomedical platforms, and smart photovoltaic modules, all at IMEC Leuven, Belgium. Currently he is an IMEC fellow. He is also part-time full professor at the EE department of the KULeuven. He has been associate editor for several IEEE and ACM journals. He was elected IEEE fellow in 2005.