Design-Technology Co-Optimization for NVM-based Neuromorphic Processing Elements

SHIHAO SONG, ADARSHA BALAJI, ANUP DAS, and NAGARAJAN KANDASAMY, Drexel University, USA

An emerging use-case of machine learning (ML) is to train a model on a high-performance system and deploy the trained model on energy-constrained embedded systems. Neuromorphic hardware platforms, which operate on principles of the biological brain, can significantly lower the energy overhead of a machine learning inference task, making these platforms an attractive solution for embedded ML systems. We present a design-technology tradeoff analysis to implement such inference tasks on the processing elements (PEs) of a Non-Volatile Memory (NVM)-based neuromorphic hardware. Through detailed circuit-level simulations at scaled process technology nodes, we show the negative impact of technology scaling on the information-processing latency, which impacts the quality-of-service (QoS) of an embedded ML system. At a finer granularity, the latency inside a PE depends on 1) the delay introduced by parasitic components on its current paths, and 2) the varying delay to sense different resistance states of its NVM cells. Based on these two observations, we make the following three contributions. First, on the technology front, we propose an optimization scheme where the NVM resistance state that takes the longest time to sense is set on current paths having the least delay, and vice versa, reducing the average PE latency, which improves the QoS. Second, on the architecture front, we introduce isolation transistors within each PE to partition it into regions that can be individually power-gated, reducing both latency and energy. Finally, on the system-software front, we propose a mechanism to leverage the proposed technological and architectural enhancements when implementing a machine-learning inference task on neuromorphic PEs of the hardware. Evaluations with a recent neuromorphic hardware architecture show that our proposed design-technology co-optimization approach improves both performance and energy efficiency of machine-learning inference tasks without incurring high cost-per-bit.

CCS Concepts: • **Hardware → Neural systems**; **Emerging languages and compilers**; **Emerging tools and methodologies**; • **Computer systems organization → Data flow architectures**; **Neural networks**.

Additional Key Words and Phrases: neuromorphic computing, design-technology co-optimization (dtco), non-volatile memory (NVM), oxide-based resistive random access memory (OxRRAM)

1 INTRODUCTION

Neuromorphic computing systems are integrated circuits that implement the architecture of central nervous system in primates [15, 23, 66]. These systems facilitate energy-efficient computations using Spiking Neural Networks (SNN) [64] for power-constrained embedded devices. To this end, the design workflow is to train a machine learning model (commonly on a backend server) and subsequently, convert the trained model to spike-based computations and deploy it on the neuromorphic hardware of an embedded system. The quality of inference (e.g., accuracy) is assessed in terms of the inter-spike interval (ISI) (see Section B). Therefore, any deviation from its expected value will lead to a degradation of the inference quality.

Authors' address: Shihao Song; Adarsha Balaji; Anup Das; Nagarajan Kandasamy, anup.das@drexel.edu, Drexel University, 3141 Chestnut Street, Philadelphia, PA, USA, 19104.

1539-9087/2022/1-ART1 $15.00
https://doi.org/10.1145/3524068

A typical neuromorphic system such as Loihi [29], DYNAPs [67] and μBrain [94] consists of processing elements (PEs) that communicate spikes using a shared interconnect. Each PE implements neuron and synapse circuitries. A common technique to implement a neuromorphic PE is using an analog crossbar where bitlines and wordlines are organized in a grid with memory cells connected at their crosspoints to store synaptic weights [3, 33, 41, 46, 47, 52, 62, 70, 101]. Neuron circuitries are implemented along bitlines and wordlines. Figure 1 (left) shows the architecture of an $N \times N$ analog crossbar with N bitlines and N wordlines.

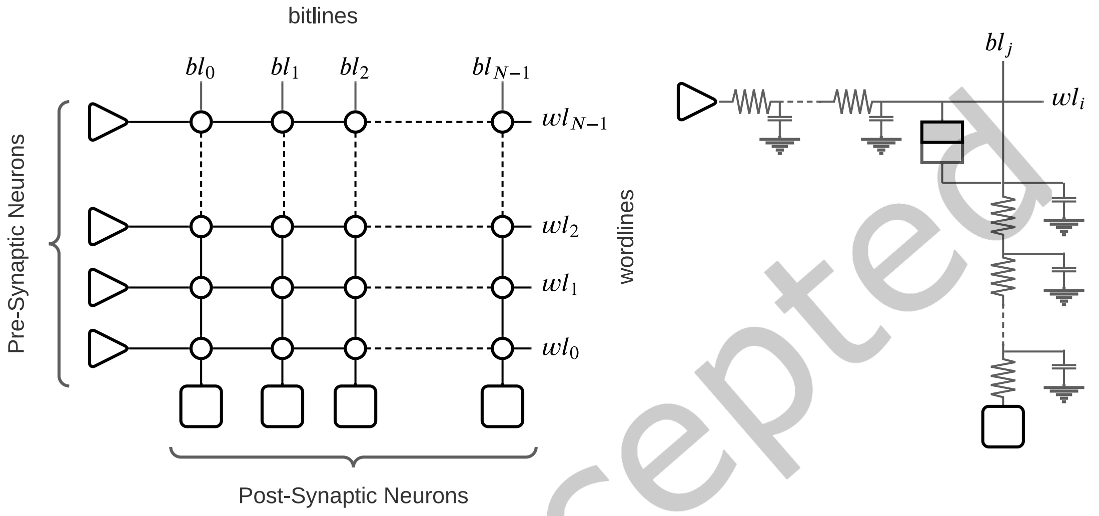


Fig. 1. An $N \times N$ crossbar showing the parasitic components within.

We investigate the internal architecture of a crossbar and find that parasitic components introduce delay in propagating current from a pre-synaptic neuron to a post-synaptic neuron as illustrated in Figure 1 (right). This delay depends on the specific current path activated in a crossbar. Higher the number of parasitic components on a current path, larger is its propagation delay [71, 87, 89, 90, 92]. Parasitic components on bitlines and wordlines are a major source of latency at scaled process technology nodes and they create significant **latency variation** in a crossbar [34, 35, 50, 54, 57, 68, 80, 84, 88]. Such variation can introduce ISI distortion (Section B), which may impact the quality of an inference task [8, 28].

To increase the energy efficiency of a neuromorphic system, Non-Volatile Memory (NVM) such as oxide-based random access memory (OxRRAM), phase-change memory (PCM), ferroelectric RAM (FeRAM), and spin-based magnetic RAM (MRAM) is used to implement the memory cells in a crossbar [16, 65, 95, 96, 99]. An NVM cell can be programmed to a high-resistance state (HRS) or one of many low-resistance states (LRS), implementing multi-bit synaptic weights [60, 65, 98]. To implement a synaptic weight on a memory cell of a crossbar, the synaptic weight is programmed as the conductance of the cell.

A crossbar can accommodate only a fixed number of pre-synaptic connections per post-synaptic neuron. To give an an example, the crossbar in Figure 1 (left) has N pre-synaptic neurons, N post-synaptic neurons, and N^2 memory cells. Each post-synaptic neuron can have a maximum of N pre-synaptic connections. To mitigate the negative impact of technology scaling, e.g., increase in the value of parasitic components on current paths and increase in the power density, N is constrained to a lower value, typically between 128 and 256 (see our tradeoff analysis in Section 2). To map a large SNN model on a multi-PE hardware, a system software framework such as NEUTRAMS [51], NeuroXplorer [12], SentryOS [94], LAVA [61] and DFSynthsizer [77] is commonly used. These frameworks first partition a model into clusters, where a cluster is a subset of neurons and synapses of the model that can be implemented on the architecture of a crossbar. Subsequently, the partitioned clusters are implemented on different crossbars of a neuromorphic hardware.

We make the following two key **observations** related to a neuromorphic PE.

Observation 1: *The latency within a crossbar is a function of the length (i.e., the number of parasitic components) of current paths and the delay to sense the NVM cell activated on a current path.*

Observation 2: *Due to how memory cells are organized in a crossbar, a significant fraction of these memory cells remains unutilized when implementing machine learning inference tasks.* Based on these two observations (which we elaborate in Sections 2-4), we present a design-technology tradeoff analysis to implement machine learning inference tasks on different PEs of an NVM-based neuromorphic system. We make the following four key **contributions**.

- Through detailed circuit-level simulations at scaled process technology nodes, we show that bitline and wordline parasitics are the primary sources of long latency in a crossbar and they create asymmetry in inference latency. With technology scaling, the absolute latency increases and the latency asymmetry becomes increasingly more significant. In addition, different resistance states of a multi-level NVM cell take varying latencies to sense during an inference operation (see Section 2).
- We propose to optimize the implementation of synaptic weights on NVM cells such that the resistance state that takes the longest time to sense is programmed on the NVM cell that has the least parasitic delay in a crossbar. This lowers the latency of a crossbar. (see Section 3)
- We propose an architectural change of introducing isolation transistors in a crossbar to partition it into regions that can be individually power-gated based on their utilization. In this way, we improve energy efficiency. In addition, by isolating the unutilized region of a crossbar from the active region, parasitics of only the active region contribute to latency, rather than both as in a baseline non-partitioned crossbar architecture. This reduces the latency of a crossbar (see Section 4).
- We show that our technological and architectural optimizations can only deliver on its latency and energy improvement promises if they are exploited efficiently by the system software. Therefore, we propose a mechanism to expose our proposed design changes to the system-level, allowing the system software to improve both latency and energy when implementing machine-learning inference tasks on hardware (see Section 5).

We evaluate our design-technology co-optimization approach for a recent neuromorphic hardware using 10 machine learning inference tasks. Results show 12% reduction in average PE latency and 22% lower application energy compared to current state-of-the-art.

To the best of our knowledge, this is the first work that demonstrates the energy and latency improvement of power gating crossbar-based neuromorphic hardware designs.

2 DESIGN-TECHNOLOGY TRADEOFF ANALYSIS

Without loss of generality, we demonstrate the design-technology tradeoff for an OxRRAM-based neuromorphic PE, where each NVM cell can be programmed to the following four resistance levels (i.e., 2-bit per synapse) – 1.5 KΩ, 5.78 KΩ, 13.6 KΩ, and 73 KΩ [20, 21, 32, 65, 75]. Furthermore, we show our analysis for four process technology nodes – 16nm, 22nm, 32nm, and 45nm, which are obtained from our technology provider. The analysis can be easily extended to other NVM types and also to other process technology nodes.

2.1 Cost-per-Bit Analysis for a Neuromorphic PE

The computer memory industry has thus far been primarily driven by the **cost-per-bit metric**, which provides the maximum capacity for a given manufacturing cost. As shown in recent works [57, 68, 69, 80, 82–84], manufacturing cost can be estimated from the area overhead. To estimate the cost-per-bit of a neuromorphic PE, we investigate the internal architecture of a crossbar and find that a neuron circuit can be designed using 20 transistors and a capacitor [49], while an NVM cell is a 1T-1R arrangement with a transistor used as an access device for the cell. Within an $N \times N$ crossbar, there are N pre-synaptic neurons, N post-synaptic neurons, and N^2 synaptic cells. The total area of all the neurons and synapses of a crossbar is

$$\begin{aligned} \text{neuron area} &= 2N(20T + 1C) \\ \text{synapse area} &= N^2(1T + 1R) \end{aligned} \tag{1}$$

where T stands for transistor, C for capacitor, and R for NVM cell. The total synaptic cell capacity is N^2, with each NVM cell implementing 2-bit per synapse. The total number of bits (i.e., synaptic capacity) in the crossbar is

$$\text{total bits} = 2N^2 \tag{2}$$

Therefore, the cost-per-bit of an NxN crossbar is

$$\text{cost-per-bit} = \frac{2N(20T + 1C) + N^2(1T + 1R)}{2N^2} \approx \frac{F^2(27 + 2N)}{N}, \tag{3}$$

where the cost-per-bit is represented in terms of the crossbar dimension N and the feature size F. Equation 3 provides a **back-of-the-envelope** calculation of cost-per-bit. Figure 2 plots the normalized cost-per-bit for four different process technology nodes, with the crossbar dimension ranging from 16 to 256. We make the following two observations.

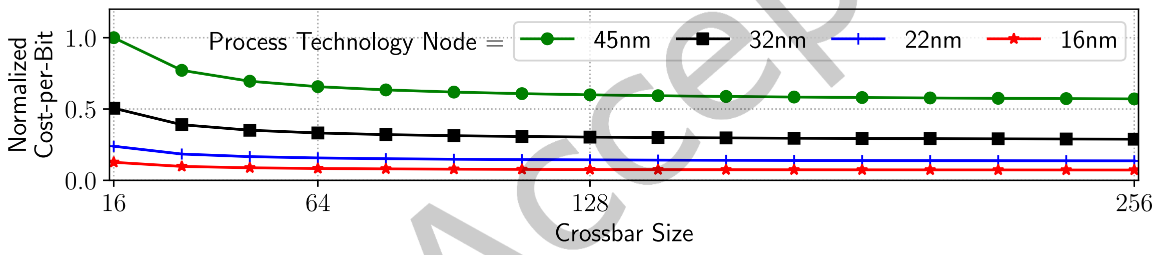


Fig. 2. Cost-per-bit analysis of a crossbar.

First, the cost-per-bit reduces with increase in the dimension of a crossbar, i.e., larger-sized crossbars can accommodate more bits for a given cost. However, both the absolute latency and latency variation increases significantly for larger-sized crossbars, which increases inference latency and reduces the quality of machine learning inference due to an increase in the ISI distortion (see our analysis in Section 2.2). Second, the cost-per-bit reduces considerably with technology scaling. This is due to higher integration density at smaller process technology nodes.

The formulation for the cost-per-bit (Equation 3) depends on the specific neuron architecture of [49] and the one transistor (1T)-based OxRRAM design of [65]. This formulation can be easily extended to other neuron and synapse designs. Furthermore, system designers can use our formulation to configure their neuromorphic hardware, without having to access and plug-in technology-related data.

2.2 Latency Variation in a Neuromorphic PE

Figure 3 shows the difference between the best-case and worst-case latency in a crossbar (expressed as a fraction of $1\mu s$ spike duration) for five different crossbar configurations at 45nm, 32nm, 22nm, and 16nm process technology nodes. See our experimental setup using NeuroXplorer [12] in Section 6.1, which incorporates software, architecture, circuit, and technology.. All NVM cells are programmed to the HRS state, i.e., 73 KΩ (see Section 2.3 for the dependency on resistance states).

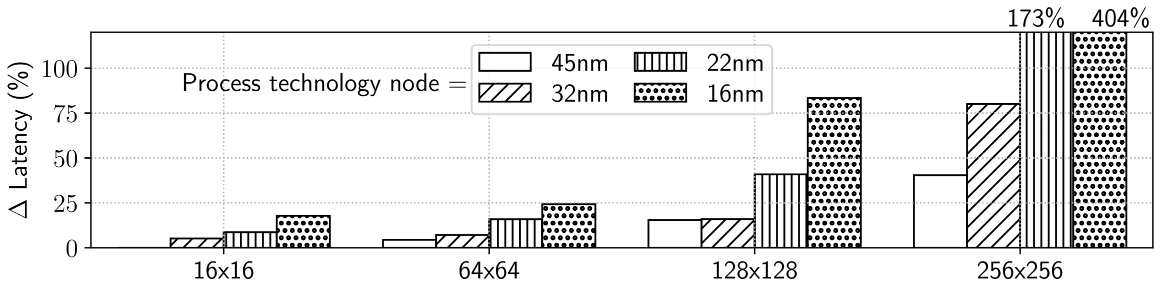


Fig. 3. Variance in latency within a crossbar, expressed as a fraction of a single spike duration.

We make two key observations. First, the latency difference increases with crossbar size due to an increase in the number of parasitic components on current paths. The average latency difference for 256 × 256 crossbar is higher than 16 × 16, 64 × 64, and 128 × 128 crossbar by 16.5x, 13.4x, and 4.5x, respectively. This average is computed across the four process technology nodes. Therefore, smaller-sized crossbars lead to a smaller variation in latency, which is good for performance. However, smaller-sized crossbars also lead to higher cost-per-bit, which we have analyzed in Section 2.1. For most neuromorphic PE designs, 128 × 128 crossbars achieve the best tradeoff in terms of latency variation and cost-per-bit [43, 58, 65, 67, 71, 103].

Second, the latency difference increases significantly for scaled process technology nodes due to an increase in the value of the parasitic component. The average latency difference for 32nm, 22nm, and 16nm process technology nodes is higher than 45nm by 1.3x, 3x, and 6.6x, respectively. The unit wordline (bitline) parasitic resistance ranges from approximately 2.5Ω (1Ω) at 45nm node to 10Ω (3.8Ω) at 16nm node. The value of these unit parasitic resistances is expected to scale further reaching $\approx 25\Omega$ at 5nm node [34, 36, 37, 74, 93]. The unit wordline and bitline capacitance values also scale proportionately with technology. Latency variation increases ISI distortion, which degrades the quality of machine learning inference.

2.3 Varying Latency to Sense NVM Resistance States

The latency (i.e., the delay) on a current path from a pre-synaptic neuron to a post-synaptic neuron within a crossbar is proportion to $R_{eff} \cdot C_{eff}$, where R_{eff} (C_{eff}) is the effective resistance (capacitance) on the path. This delay increases the time it takes for the membrane potential of a post-synaptic neuron to rise above the threshold voltage causing a delay in spike generation.

The effective resistance on a current path depends on the value of parasitic resistances and the resistance of the NVM cell. We analyze the latency impact due to different resistance states. Figure 4 plots the increase in latency (expressed as a fraction of $1\mu s$ spike duration) to sense three NVM resistance states (LRS2, LRS3, and HRS) with respect to LRS1 at 45nm, 32nm, 22nm, and 16nm process technology nodes. These results are for a neuromorphic PE with a 128 × 128 crossbar.

We observe that the latency to sense the HRS state is considerably higher than all three LRS states at all process technology nodes (consistent with [19, 65, 100]). The latency difference increases with technology scaling due to an increase in the size of parasitic components on bitlines and wordlines of a crossbar, which we have analyzed in Section 2.2.

3 PROPOSED TECHNOLOGICAL IMPROVEMENTS

Based on the design-technology tradeoff analysis of Section 2, we now present our technology-related optimization. Without loss of generality, we present our optimization for a 128×128 crossbar-based neuromorphic hardware

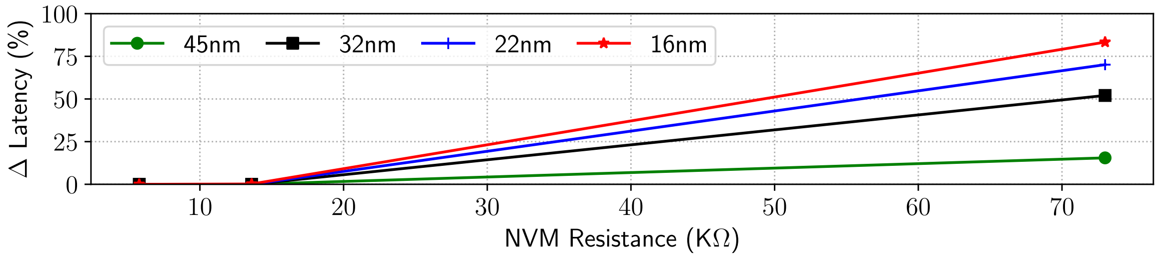


Fig. 4. Latency to sense various NVM resistance states, expressed as a fraction of a single spike duration.

designed at 16nm node. We exploit the following two observations from Section 2: 1) HRS resistance state in an NVM cell takes higher latency to sense than LRS states, and 2) spike propagation latency in a crossbar depends on the number of parasitic components on its current path. The left sub-figure of Figure 5 shows the proposed technological changes. A crossbar is partitioned into three regions. The number of parasitic components on current paths in region A is considerably lower than in the rest of the crossbar. Therefore, all NVM cells in this region (4 in this example) implement only the HRS resistance state, which takes the longest time to sense. Conversely, NVM cells in region B have longer propagation delay due to higher number of parasitic components. Therefore, all NVM cells in this region (9 in this example) implement only the LRS resistance state, which takes the shortest time to sense. Finally, all other NVM cells (i.e., those in region C) are programmable, i.e., these cells can implement all four resistance states. The overall objective is to balance the latency on different current paths within a crossbar. This minimizes the latency variation in a crossbar, which reduces ISI distortion and improves the quality of machine learning inference tasks.

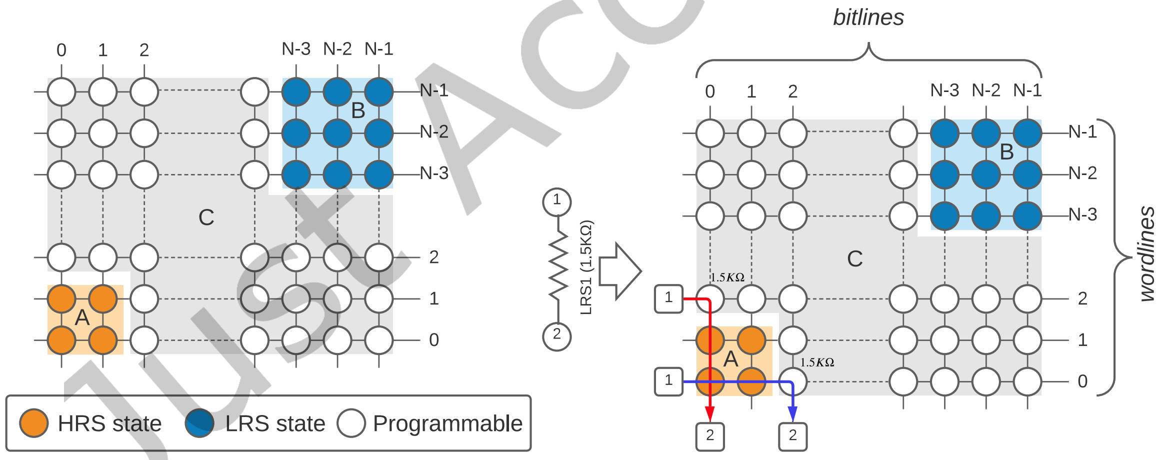


Fig. 5. Our proposed technological change.

The right sub-figure shows a pre- and a post-synaptic neuron connected via a synapse that is programmed to the LRS state. The synaptic connection can be implemented on NVM cells in region B (with only LRS1 state) and region C (with programmable states). The figure illustrates two alternative implementations of these neurons. If the pre-synaptic neuron is implemented on wordline 0, then the post-synaptic neuron *cannot* be implemented on bitlines 0 and 1. This is because NVM cells in region A are all in HRS. In this example, we show the implementation on bitline 2 (see the blue implementation). Conversely, if the post-synaptic neuron is implemented on bitline 0, then the pre-synaptic neuron *cannot* be implemented on wordline 0 and 1 (to avoid using region A). We show the implementation on wordline 2 (see the red implementation).

Formally, the proposed neuromorphic PE is represented by a tuple $\underline{\langle N, N_h, N_l \rangle}$, where N is the dimension of its crossbar. All NVM cells at crosspoints of wordlines $0, 1, \cdots, N_h - 1$ and bitlines $0, 1, \cdots, N_h - 1$ (i.e., region A) can implement only HRS. All NVM cells at crosspoints of wordlines $N - N_l, N - N_l + 1, \cdots, N - 1$ and bitlines $N - N_l, N - N_l + 1, \cdots, N - 1$ (i.e., region B) can implement only LRS. All other NVM cells in the PE's crossbar can implement all four resistance states.

Figure 6 plots the variation of latency in the proposed 128×128 crossbar, normalized to a baseline architecture [8], where any NVM cell can be programmed to any of the four resistance states. See Section 6.4 for a description of this baseline architecture and Section 6.1 for the simulation setup. The variation in latency is measured as ratio of the best-case and worst-case latency in the crossbar. The figure reports latency variation for N_h ranging from 2 to 64 with N_l set to 16, 32, and 64.

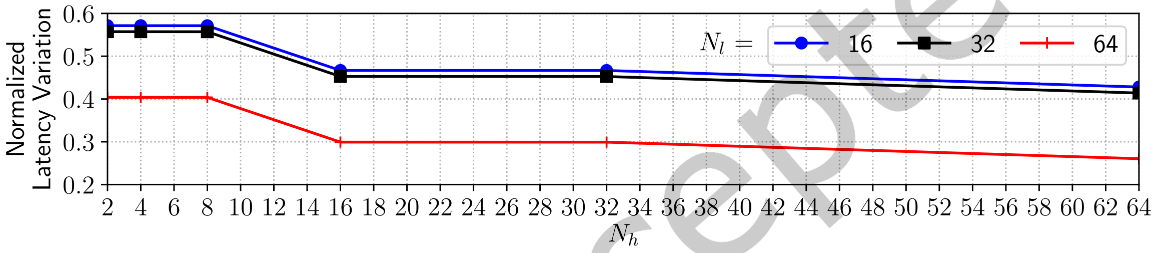


Fig. 6. Latency variation in the proposed crossbar architecture for different settings of N_h and N_l.

We observe that latency variation decreases with an increase in N_h. This is due to an increase in the size of region A, which increases the (worst-case) latency due to an increase in the number of parasitic components on current paths via the HRS state. However, the (best-case) latency of current paths via the LRS state remains the same. Therefore, the latency variation reduces which improves inference quality by lowering the ISI distortion. To illustrate this concept, Figure 7 provides an example where two synapses are mapped to a 4×4 crossbar. In Figure 7a, the red synapse (in HRS state) is mapped to the bottom left corner of the crossbar, while the blue synapse (in LRS state) to the top right corner. The figure shows the timing of two spikes. The input spike on the red and blue synapses are t_1 and t_2, respectively. Without loss of generality let $t_2 > t_1$. The ISI of these two spikes is $t_2 - t_1$. Due to the delay in current propagation through bitlines and wordlines, these two spikes arrive at the output terminal at different times – red synapse with a delay of x and blue synapse with a delay of y. Here, $y > x$. Therefore, ISI of the output spikes is $(t_2 + y) - (t_1 + x)$. The ISI distortion (difference of ISI between input and output) is

$$\text{ISI distortion} = \Big((t_2 + y) - (t_1 + x)\Big) - (t_2 - t_1) = y - x \tag{4}$$

Figure 7b illustrates a scenario where region A is increased to include more cells that are programmed to the HRS state. The mapping process will map the red synapse using the farthest cell of region A. The delay on this synapse is $x + \Delta$, where Δ is the additional delay due to routing spikes on the red synapse via a longer route compared to that in Figure 7a. Therefore, ISI of the output spikes is $(t_2 + y) - (t_1 + x + \Delta)$. The ISI distortion is

$$\text{ISI distortion} = \Big((t_2 + y) - (t_1 + x + \Delta)\Big) - (t_2 - t_1) = y - x - \Delta \tag{5}$$

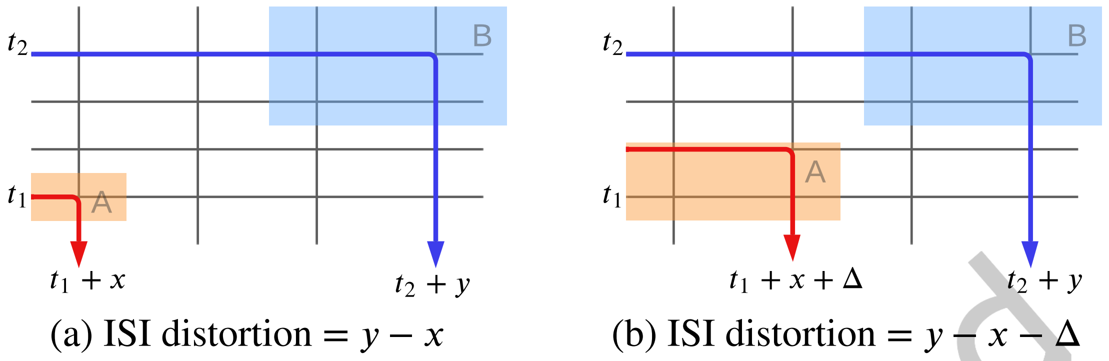


Fig. 7. ISI improvement due to increase in the size of region A.

Comparing Equations 4 & 5, we observe that the ISI distortion reduces due to an increase in the size of region A. ISI distortion also reduces with an increase in N_l due to a reduction in the worst-case latency. We also note that, large N_h may lead to higher average crossbar latency, which impacts real-time performance. Finally, we see that going from N_l = 16 to 32, there is no significant reduction in the latency variation. Although the size of region B increases with an increase in N_l, we observe only marginal reduction of the best-case latency. Overall, with $N_h = N_l = 64$, the latency variation is 74% lower than baseline. This is chosen based on the tradeoff between latency variation and average latency for 128×128 crossbar at 16nm. The tradeoff point can change for other technology nodes and for other crossbar configurations.

3.1 Reduction in Latency Variation

To understand the reduction of latency variation within a crossbar as a result of our technological changes, we provide a simple example. Consider there are only two current paths in a crossbar. The parasitic delay on the shortest and longest current paths are D and $(D + \Delta)$, respectively. The time to sense LRS and HRS NVM states are S and $(S + \delta)$, respectively. Without any optimization, the worst-case condition is triggered when the HRS state is programmed on the longest path and the LRS state on the shortest path. The minimum and maximum latencies are $(D + S)$ and $(D + S + \Delta + \delta)$, respectively. The latency variation is $(\Delta + \delta)$. Using our technology optimization, HRS state is programmed on the shortest path and LRS state on the longest path. The two latencies are $(D + S + \Delta)$ and $(D + S + \delta)$. The latency variation reduces to $(|\Delta - \delta|)$.

Within a crossbar, there are many current paths (N^2 current paths in a $N \times N$ crossbar). The precise reduction in latency variation depends on the specific current paths activated for a synaptic connection, which is controlled during the mapping of a machine learning application to the crossbars of the hardware. In Figure 6, we show a 74% reduction comparing only the shortest and the longest paths in a 128×128 crossbar. In Section 7.2, we evaluate the general case considering the mapping process. We report an average 22% reduction of latency variation.

Reducing the latency variation helps reduce the ISI distortion, which improves the inference quality. In Section 7.4, we report an average 4% increase of inference quality.

3.2 Impact on Latency

While latency variation impacts inference quality, the average crossbar latency impacts the real-time performance. To understand the impact of our technological optimization on the average crossbar latency, we consider the same example of two current paths. Consider there are m synapses with LRS states and n synapses with HRS state. The average latency in the worst-case condition is $\frac{m\cdot(D+S)+n\cdot(D+S+\Delta+\delta)}{m+n}$. Using the technological improvement, the average latency is $\frac{m\cdot(D+S+\Delta)+n\cdot(D+S+\delta)}{m+n}$. Therefore, the change in latency is $\left(\frac{n-m}{n+m}\right)\Delta$. This change in latency depends on 1) current paths activated in a crossbar and 2) the value of n and m, i.e., the number of synaptic connections with HRS and LRS states, respectively. In Section 7.3, we show an average 3% reduction of the average crossbar latency for all the evaluated applications.

4 ARCHITECTURAL ENHANCEMENTS TO NEUROMORPHIC PE

To understand the motivation of the proposed architectural changes, Figure 8 reports the average synapse utilization of 128×128 crossbars in neuromorphic PEs for 10 machine learning models implemented using the spatial decomposition technique of [11], which is a **best-effort approach** to improve the utilization of crossbars in a neuromorphic hardware.

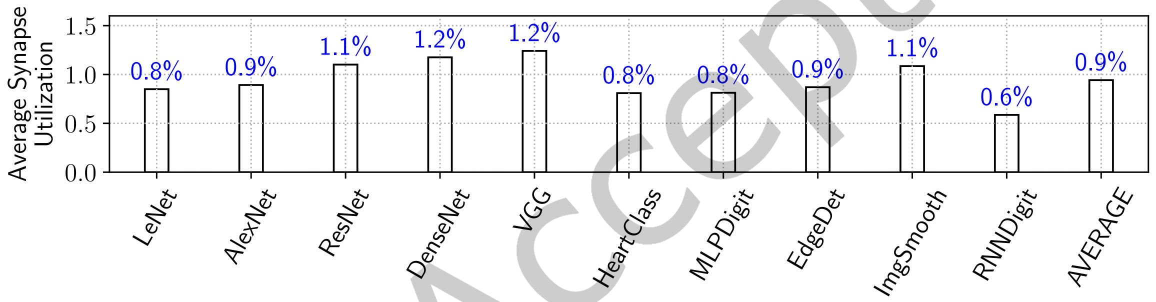


Fig. 8. Average synapse utilization of neuromorphic PEs.

We observe that the average synapse utilization is only 0.9%. This is because a crossbar can accommodate only a limited number of pre-synaptic connections per post-synaptic neuron. To illustrate this, Figure 9 shows three examples of implementing neurons on a 4×4 crossbar. The synapse utilization of the three example scenarios are (a) 25% (4 out of 16), (b) 18.75% (3 out of 16), and (c) 25% (4 out of 16). As the crossbar dimension increases, the utilization drops significantly. For instance, if a 128×128 crossbar is used to implement a single 128-input neuron, i.e., generalization of Fig. 9a, the utilization is only 0.78% (128 utilized synapses out of a total of $128^2 = 16,384$ synapses). Lower synapse utilization leads to lower energy efficiency.

To improve energy efficiency, we propose to partition a neuromorphic PE into regions that can be dynamically power-gated based on its utilization for a given machine-learning inference task. Figure 10 shows the use of isolation transistors in a neuromorphic PE to partition a 4×4 crossbar into active and unutilized regions. Figure 10a illustrates the implementation of only a single neuron function y_1 in the crossbar. To improve energy efficiency, isolation transistors are needed on every bitline (between wordlines 3 and 4) and on every wordline (between bitlines 1 and 2). Figure 10b illustrates the implementation of two neuron functions y_1 and y_2 in the crossbar. In this scenario, isolation transistors are only needed on every wordline (between bitlines 2 and 3). To implement inference on a neuromorphic system, each crossbar may have different utilization of its memory cells. Therefore, to improve energy efficiency in every crossbar, isolation transistors are needed on every bitline

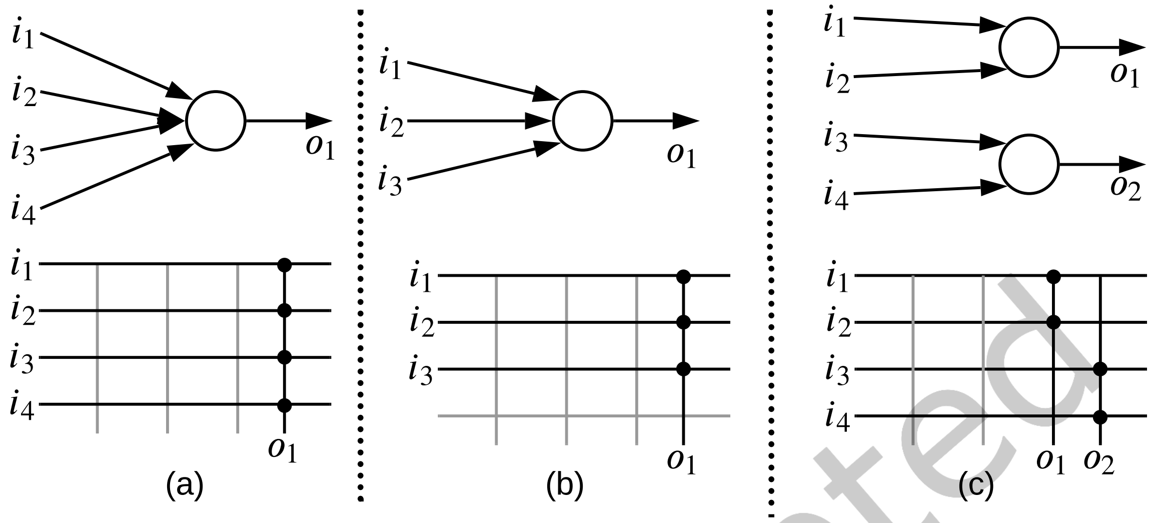
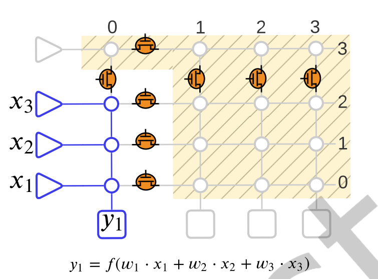
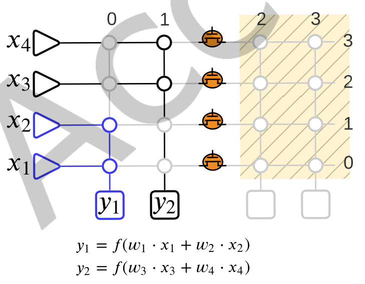


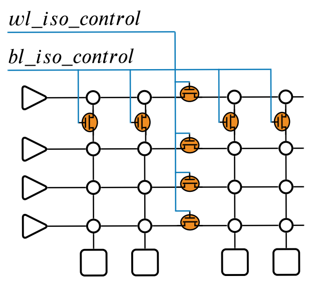
Fig. 9. Implementation of a) one 4-input, b) one 3-input, and c) two 2-input neurons to a 4×4 crossbar.



(a) Implementing a single neuron function in a partitioned crossbar.


x_4
x_3
x_2
x_1
y_1
y_2
$y_1 = f(w_1 \cdot x_1 + w_2 \cdot x_2)$
$y_2 = f(w_3 \cdot x_3 + w_4 \cdot x_4)$

(b) Implementing two neuron functions in a partitioned crossbar .



(c) Proposed crossbar architecture.

Fig. 10. Proposed neuromorphic PE architecture partitioned using isolation transistors.

(and between every pair of wordlines) and on every wordline (and between every pair of bitlines) – a total of 24 isolation transistors for this example 4×4 crossbar (in general, $2N(N-1)$ for an $N \times N$ crossbar). This fine-grained partitioned PE architecture offers flexibility in energy management incorporating crossbar utilization, but leads to a significant increase in the area, latency, and system overhead to control isolation transistors.

To overcome these limitations while improving energy efficiency, we enable a coarse-grained partitioning in a crossbar as illustrated in Figure 10c. In this example, isolation transistors are inserted selectively on every bitline (between wordlines 3 and 4) and on every wordline (between bitlines 2 and 3). This coarse-grained partitioned PE architecture requires a total of 8 isolation transistors (in general, $2N$ for an $N \times N$ crossbar). To reduce the control overhead, isolation transistors on wordlines of a crossbar are controlled using a single control signal `wl_iso_ctrl` and those on bitlines using the signal `bl_iso_ctrl`. Through these two control signals, we enable four distinct configurations of the crossbar, which are summarized in Table 1.

Table 1. Different PE configurations enabled using the two new crossbar control signals.

Crossbar Control		Key Parameters		
wl_iso_ctrl	**bl_iso_ctrl**	**Dimension**	**Energy**	**Latency**
Baseline PE Architecture				
–	–	4×4	$\propto 4*4$	Best-case: $t_{1,1}$ Worst-case: $t_{4,4}$
Proposed Partitioned PE Architecture				
0	0	3×2	$\propto 3*2$	Best-case: $t_{1,1}$ Worst-case: $t_{3,2} - \Delta$
0	1	4×2	$\propto 4*2$	Best-case: $t_{1,1}$ Worst-case: $t_{4,2} - \Delta + t_{ON}$
1	0	3×4	$\propto 3*4$	Best-case: $t_{1,1}$ Worst-case: $t_{3,4} - \Delta + t_{ON}$
1	1	4×4	$\propto 4*4$	Best-case: $t_{1,1}$ Worst-case: $t_{4,4} + 2 \cdot t_{ON}$

In a baseline PE architecture, a crossbar dimension is fixed to 4x4. Its static energy is proportional to the number of memory cells, which is 4*4 = 16 in this example. Latency in the crossbar varies from $t_{1,1}$ (nearest cell or best-case) to $t_{4,4}$ (farthest cell or worst-case).

In the proposed partitioned PE architecture, there are four configurations.

In configuration '00', the crossbar is configured as a 3x2 array with its static energy proportional to 3x2 = 6 memory cells. This is when the unutilized region is power-gated. The best-case latency is $t_{1,1}$ and the worst-case latency is $t_{3,2} - \Delta$, where Δ is the reduction in parasitic delay due to shorter bitlines and wordlines.

In configuration '01', the crossbar is configured as a 4x2 array with its static energy proportional to 4x2 = 8 memory cells. The best-case latency is $t_{1,1}$ and worst-case latency is $t_{4,2} - \Delta + t_{ON}$, where t_{ON} is the delay of the isolation transistor on current paths.

In configuration '10', the crossbar is configured as a 3x4 array with its static energy proportional to 3x4 = 12 memory cells. The best-case latency is $t_{1,1}$ and the worst-case latency is $t_{3,4} - \Delta + t_{ON}$.

In configuration '11', the crossbar is configured as the baseline 4x4 array with its static energy proportional to 4x4 = 16 memory cells. The best-case latency is $t_{1,1}$ while the worst-case latency is $t_{4,4} + 2 \cdot t_{ON}$. Observe that on the longest current path there are now two isolation transistors, resulting in higher worst-case latency than in the baseline design.

Our proposed system software (which we discuss in Section 5) minimizes the use of configuration '11', improving both performance and energy efficiency.

Single Control: The proposed partitioned PE architecture also supports using a single control signal for all the isolation transistors in a crossbar. When using a single control, only the configurations '00' and '11' are used, implementing a 3×2 and a 4×4 array, respectively.

To generalize the discussion for an $N \times N$ crossbar, assume that isolation transistors are inserted on every bitline (between wordlines P and $P + 1$) and on every wordline (between bitlines Q and $Q + 1$). Then, the four configurations are: 00': a PxQ array; '01': a NxQ array; '10': a PxN array; and '11': a NxN array. Formally, $\langle N, N_h, N_l, P, Q \rangle$ represents the proposed partitioned PE architecture. Equation 6 summarizes the notations.

$$\begin{aligned}\langle N \rangle &= \text{a baseline } N \times N \text{ crossbar}\\ \langle N, N_h, N_l \rangle &= N \times N \text{ crossbar with tech. enhancement (Sec. 3)}\\ \langle N, N_h, N_l, P, Q \rangle &= N \times N \text{ crossbar with tech. \& arch. enhancements}\\ & \text{(see Sec. 3 \& 4)} \end{aligned} \tag{6}$$

We introduce the following four terminologies: 1) **expanded mode:** in this mode, a crossbar is operated in configuration '11', 2) **collapsed mode:** in this mode, a crossbar is operated in configurations '00', '01', and '10', 3) **collapsed region**, this is the reduced dimension of the crossbar when operating in configurations '00', '01', and '10', and 4) **far region**, this is the region of the crossbar excluding the collapsed region.

In our design methodology, the far region of a crossbar is power-gated using the two control signals at design-time considering the crossbar's utilization. This is achieved during mapping of neurons and synapses to the hardware. Since neuron and synapse mapping does not change during inference, there is no dynamic power management needed. Consequently, there is also no latency and energy overhead involved in switching the far region on/off at run-time.

4.1 Placing Isolation Transistors in a Crossbar

To illustrate the design space exploration involved in placing isolation transistors in a crossbar, Figure 11(a) illustrates a baseline crossbar with four current path that are activated during mapping of neurons and synapses. Figures 11(b)-(d) show three alternative placements of isolation transistors in the crossbar. In Figure 11(b), P and Q values are kept small. The size of the far region is large. In this figure, only two of the current paths (1 & 2) stays within the collapsed region of the crossbar, while the other two current paths (3 & 4) traverse via the far region. This means that the latency of paths 3 & 4 increases due to the delay of the isolation transistors on current paths. Additionally, the far region cannot be power gated, so there is a limited scope for energy reduction using power gating. Increasing P and Q values further (Figure 11(c)), the far region reduces in size as illustrated in the figure. Although three of the four current paths stay in the collapsed region, the far region still cannot be power-gated due to the presence of path 4 in this region. Finally, Figure 11(d) illustrates a possibility where all current paths stay in the collapsed region. The far region can therefore be power-gated. However, because of the small size of the far region, the energy benefits may not be significant. We explore this latency and energy tradeoffs.

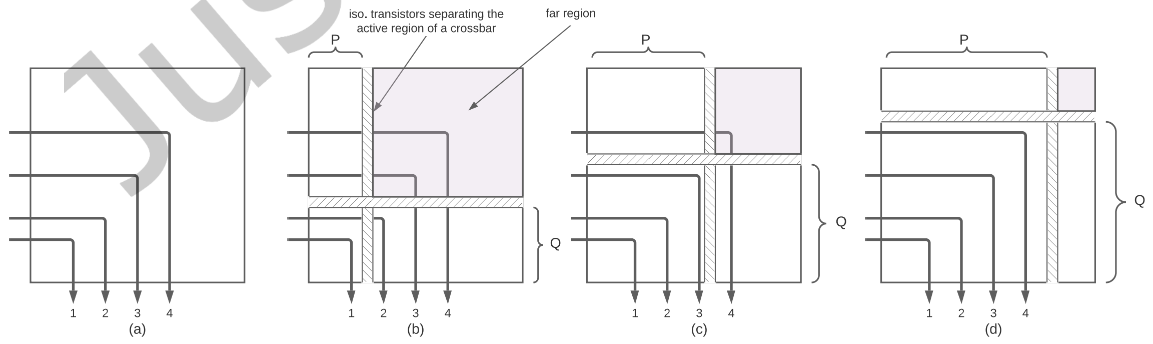


Fig. 11. Placing isolation transistors in a crossbar.

Figure 12 shows the latency and energy tradeoffs in selecting the values of P and Q for the ResNet inference workload implemented on 128×128 crossbars in a neuromorphic hardware. Latency and energy numbers are normalized to baseline. We make the following two key observations.

First, energy is lower for smaller P and Q values. This is because by reducing P and Q, the size of the collapsed region of a crossbar reduces. Therefore, there are more memory cells in the far region that can be power-gated to lower energy.

Second, latency also reduces with a reduction in P and Q values (until P = Q = 80). This is due to shorter bitlines and wordlines of the collapsed region. However, with P = Q = 64 or 72, more clusters of ResNet need crossbars in the expanded mode of operation. This is because synapses in these clusters can no longer fit onto the reduced dimension of a collapsed crossbar. This increases latency due to isolation transistors on current paths. For ResNet, P = Q = 80 is the tradeoff point. The tradeoff point is different for different applications. To select a single crossbar configuration that gives good results for all applications, we perform similar analysis for all evaluated applications (see Section 6.3). Based on such analysis, P = Q = 96 is the selected configuration for the 128×128 crossbar at 16nm technology node.

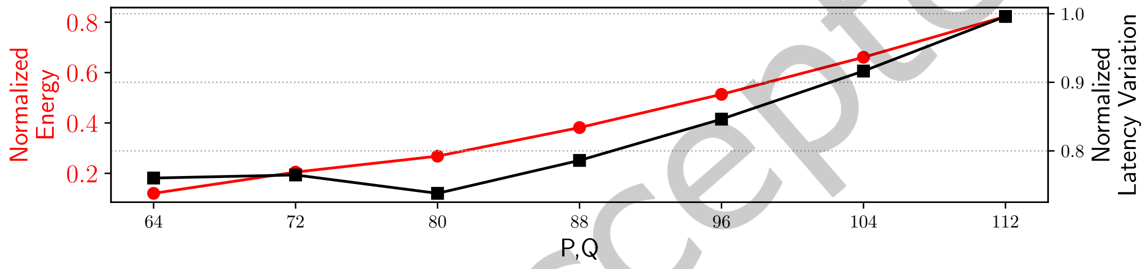


Fig. 12. Selecting P and Q values for the ResNet application.

5 EXPLOITING TECHNOLOGICAL AND ARCHITECTURAL IMPROVEMENTS VIA THE SYSTEM SOFTWARE

To describe the system software, the left subfigure of Figure 13 shows the final crossbar design with isolation transistors that allow each neuromorphic PE to operate in a collapsed or expanded mode. The right subfigure shows control signals for these transistors generated from a centralized controller implemented inside the system software.

Without loss of generality, Figure 14 shows modifications to the baseline system software [61] to exploit the proposed design changes. A trained machine learning model is first partitioned to generate clusters, where each cluster can fit onto a crossbar. These clusters are stored in a cluster queue (`clQ`). In the baseline design, each cluster from the clQ is mapped to an $N \times N$ array (exactly replicating the crossbar dimension of the hardware). The mapping is programmed to the hardware using the cluster placement block. In the proposed design, each cluster of clQ is mapped on four separate arrays – a $P \times Q$ array, a $N \times Q$ array, a $P \times N$ array, and a $N \times N$ array. These mappings go to a configuration selection block, which selects the final mapping for the cluster and the configuration of the corresponding PE based on energy-latency tradeoffs. The configuration is programmed to the hardware by configuring the two control signals `wl_iso_ctrl` and `bl_iso_ctrl`. This allows to power-gate the far region of the crossbar. It is important to note that since we power-gate unused resources of a crossbar only at design-time when admitting an application, we minimize the switching overhead. In the future, we will extend this work to also consider dynamic power management by dynamically controlling the isolation transistors.

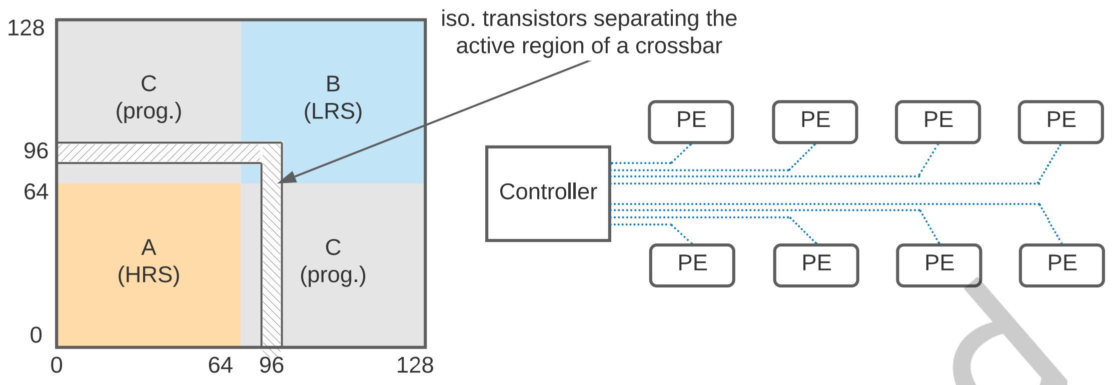


Fig. 13. Final crossbar design using the isolation transistors. The right subfigure shows the control signals generated from the controller when using the proposed partitioned PE architecture in a neuromorphic system.

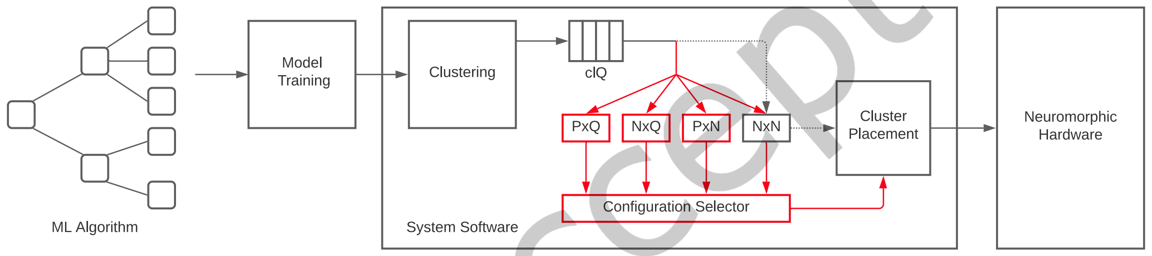


Fig. 14. Proposed system software. All changes are indicated in red.

In selecting the final mapping, the configuration selector first checks to see if a cluster can be mapped to a $P \times Q$ array. If this is possible, then the mapping to the $P \times Q$ array is selected as the final mapping for the cluster, and the corresponding PE is set to operate in configuration '00' (collapsed mode). Otherwise, the configuration selector checks to see if the cluster can be mapped to $N \times Q$ or $P \times N$ array. If so, the corresponding mapping is selected, and the PE is set to operate in configurations '01' or '10', respectively. If the cluster cannot be mapped to either $N \times Q$ or $P \times N$ arrays, the mapping to $N \times N$ array is selected as the final mapping of the cluster with the PE set to operate in configuration '11' (expanded mode). In this way, the proposed system software uses expanded mode only when it is absolutely necessary to do so. Otherwise, it selects the collapsed region to map synapses, improving both latency and energy.

6 EVALUATION METHODOLOGY

6.1 Simulation Framework

We evaluate the proposed design-technology co-optimization approach for OxRRAM-based neuromorphic PEs. Our simulation framework includes NeuroXplorer [12], a cycle-level in-house neuromorphic simulator [12] with programmable crossbar parameters. We configure this framework to simulate crossbars with parameters listed in Table 2.

Circuit-level simulations are performed with technology parameters from the predictive technology model (PTM) [102] and OxRRAM-specific parameters from [19]. We note that, comparing different chip technologies or recommending one technology node over another is not the focus of this work. Instead, we show that for a

Table 2. Major simulation parameters extracted from [29].

Neuron technology	16nm CMOS (original design is at 14nm FinFET)
Synapse technology	HfO_2-based OxRRAM [65]
Supply voltage	1.0V
Energy per spike	23.6pJ at 30Hz spike frequency
Energy per routing	3pJ
Switch bandwidth	3.44 G. Events/s

given process technology node, design optimizations can reduce energy and latency variations. Furthermore, the proposed design-technology co-optimization methodology can be used by system designers to choose the best technology node for their neuromorphic designs by exploring the energy-performance tradeoffs.

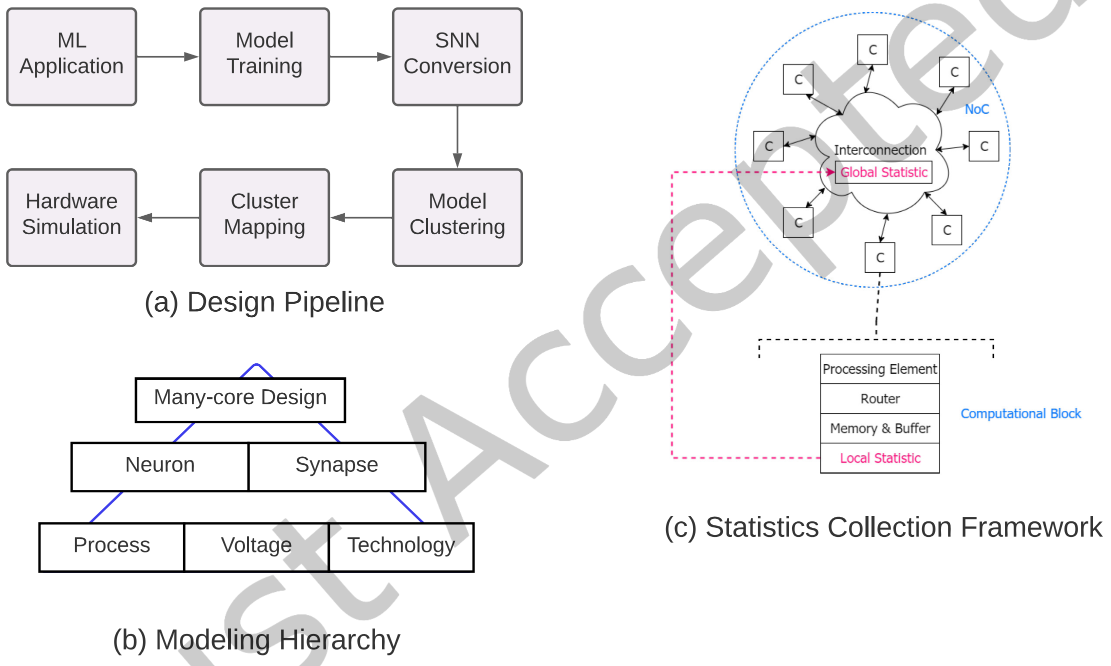


Fig. 15. Design pipeline using NeuroXplorer.

Neuromorphic simulations are performed on a Lambda workstation, which has AMD Threadripper 3960X with 24 cores, 128 MB cache, 128 GB RAM, and 2 RTX3090 GPUs. Figure 15(a) shows the design pipeline implemented using NeuroXplorer. A machine learning model is first trained using frameworks such as Keras and PyTorch. Subsequently, the trained model is converted into SNN using [5, 77]. The trained model is also simulated using an SNN simulator such as CARLsim [22]. NeuroXplorer integrates PyCARL [4], which allows the SNN model to be simulated using other SNN simulators such as Nengo [14], Neuron [44], and Brian [40]. Keras [42] and CARLsim [22] both uses the two GPUs to accelerate model training and SNN functional simulation, respectively.

The SNN simulated model is clustered using the best-effort technique of [11], which maximizes cluster utilization. Clusters of the SNN are mapped to the hardware using the SpineMap technique [8]. Finally, we perform cycle-accurate simulation of the clusters using NeuroXplorer [12].

Figure 15(b) shows the modeling hierarchy of the simulator. At the highest level is the many-core design, which is a tile-based architecture, similar to Loihi [29]. Each PE consists of a crossbar, which is an organization of neurons and synapses. A neurons is modeled using [49] and a synaptic circuit using [65]. At the lowest level are the technology models (see Table 2).

Finally, Figure 15(c) shows the statistics collection framework in NeuroXplorer. It facilitates global statistics collection, where spike arrival times are recorded for each PE (shown as C in the figure). These spike times are then used to compute the ISI distortion (see Appendix B).

6.2 Power Consideration for Isolation Transistors

The additional power required to control the isolation transistors when accessing the RRAM cells in the far region is approximately 3x that of raising a wordline, since raising a wordline requires driving one access transistor per bitline, while accessing the RRAM cells in the far region requires driving two isolation and one access transistor per bitline. The power overhead for accessing RRAM cells in the collapsed mode '01' and '10' is approximately 2x (one isolation and one access transistor) [57, 80, 84]. The energy numbers reported in Section 7.1 incorporates these overheads.

6.3 Evaluated Workloads

We select 10 machine learning inference programs that are representative of three most commonly-used neural network classes: convolutional neural network (CNN), multi-layer perceptron (MLP), and recurrent neural network (RNN). Table 3 summarizes the topology, number of neurons and synapses, number of spikes per image, and baseline quality of these applications on hardware.

Table 3. Applications used to evaluate the proposed approach.

Class	Applications	Dataset	Neurons	Synapses	Avg. Spikes/Frame	Baseline Quality	Obtained Quality
CNN	LeNet	CIFAR-10	80,271	275,110	724,565	86.3%	87.1%
	AlexNet	CIFAR-10	127,894	3,873,222	7,055,109	66.4%	66.9%
	ResNet	CIFAR-10	266,799	5,391,616	7,339,322	57.4%	58.0%
	DenseNet	CIFAR-10	365,200	11,198,470	1,250,976	46.3%	46.5%
	VGG	CIFAR-10	448,484	22,215,209	12,826,673	81.4 %	81.6%
	HeartClass [25]	Physionet	170,292	1,049,249	2,771,634	63.7%	63.9%
MLP	MLPDigit	MNIST	894	79,400	26,563	91.6%	96.4%
	EdgeDet [22]	CARLsim	7,268	114,057	248,603	SSIM = 0.89	0.99
	ImgSmooth [22]	CARLsim	5,120	9,025	174,872	PSNR = 19	22.2
RNN	RNNDigit [31]	MNIST	1,191	11,442	30,508	83.6%	83.7%

6.4 Evaluated Approaches

We evaluate the following techniques.

- *Baseline* [8]. The Baseline approach first clusters a machine-learning inference model to minimize the inter-cluster spike communication. Clusters are then mapped to neuromorphic PEs of the hardware with synapses of each cluster implemented on memory cells of a crossbar without incorporating latency variation. Neuromorphic PEs are not optimized to reduce latency variation, i.e., any resistance state (LRS or HRS) can be programmed on any current path (long or short). Unused crossbars are power-gated to reduce energy consumption. This is the coarse-grained power management technique implemented in many state-of-the-art many-core neuromorphic designs such as Loihi [29], DYNAPs [67], and μBrain [94].

- *Baseline + Design Changes*. This is the Baseline mapping approach implemented on the proposed latency-optimized partitioned neuromorphic PE design. In the proposed design, HRS state, which takes long time to sense, is used only on shorter current paths, ones that have lower parasitic delays. Similarly, LRS state is used only on loner current paths. In addition to coarse-grained power management, we facilitate power gating at a finer granularity in the proposed design. Specifically, by controlling the isolation transistors, we power-gate unused resources within each crossbar.
- *Proposed*. This is the proposed solution where the system software is optimized to exploit the design changes.

7 RESULTS AND DISCUSSIONS

7.1 Energy Efficiency

Figure 16 plots the energy efficiency of the evaluated techniques normalized to Baseline. We make the following two key observations.

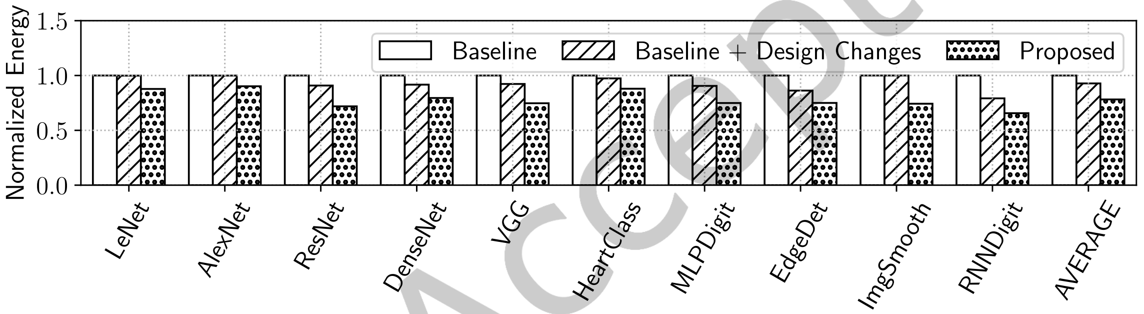


Fig. 16. Energy consumption normalized to Baseline.

First, with the proposed design changes, energy reduces by only 7% compared to Baseline. This is because, both in Baseline and Baseline with the proposed design changes, synapses of a cluster are implemented randomly on NVM cells of a crossbar causing them to be distributed across the crossbar dimension. Therefore, there remains a limited scope to collapse the crossbar and use power-gating to save energy. Second, the proposed design-technology co-optimization approach has the lowest energy (22% lower than Baseline and 16% lower than Baseline with the proposed design changes). This improvement is due to the proposed system software, which exploits the design changes in implementing machine learning inference on neuromorphic PEs. In particular, synapses are implemented to maximize the utilization of the collapsed region in each crossbar of the hardware. If all of a cluster's synapses fit into the collapsed region, then the far region can be isolated from the collapsed region using isolation transistors and power-gated to save energy.

7.2 Latency Variation

Figure 17 plots the latency variation normalized to Baseline. We make the following three key observations.

First, with the proposed design changes, latency variation increases compared to Baseline by average 1%. This is because of the increase in latency associated with the delay of isolation transistors on current paths. Second, the latency variation using the proposed approach is 30% lower than Baseline and 32% lower than Baseline with

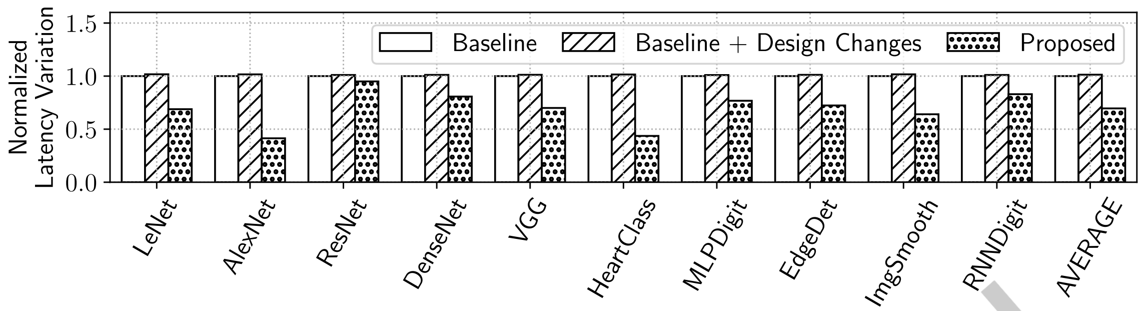


Fig. 17. Latency variation normalized to Baseline.

the proposed design changes. The reason for these improvements is three fold – 1) optimizing NVM resistance states in a crossbar such that the state that takes the longest time to sense is programmed on current paths that have the least propagation delay, 2) isolating the collapsed region of a crossbar from the far region, to reduce current propagation delay, and 3) exploiting these changes during the implementation of a machine learning inference using the proposed system software, which uses the far region of a crossbar only when it is absolutely necessary to do so. Otherwise, it improves both latency and energy by operating the crossbar in the collapsed mode.

Finally, the latency variation using the proposed approach varies across different applications. This is because the proposed approach exploits the latency and energy tradeoffs differently for different applications. The latency variation is similar to the Baseline for ResNet, while it is significantly lower than the Baseline for HeartClass.

Using the results from Sections 7.1 and 7.2, we conclude that the proposed approach introduces maximum gain for applications where the latency and energy tradeoffs can be better exploited. For all other applications, it either minimizes energy or minimizes latency variation.

7.3 Real-time Performance

One of the key hardware performance metrics for neuromorphic computing is real-time performance, which is a function of the crossbar latency. To evaluate real-time performance, Figure 18 plots the crossbar latency of the proposed approach and the Baseline for the evaluated applications. Results are normalized to the Baseline.

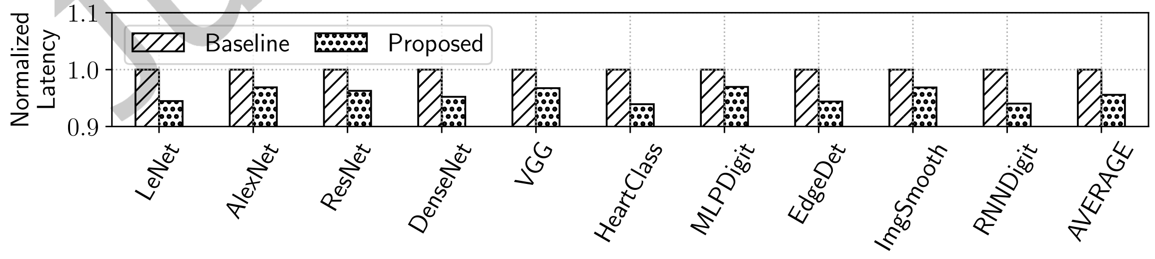


Fig. 18. Crossbar latency normalized to Baseline.

We observe that the crossbar latency using the proposed approach is on average 4.5% lower than the Baseline. This reduction is because the proposed approach places synapses with the HRS state on shorter current paths, which lowers the overall spike latency on those synapses. We have elaborated this in Section 3.2.

7.4 Inference Quality

Figure 19 shows the improvement in inference quality using the proposed approach, normalized to Baseline. We observe that the image quality improves by an average of 4%. This is due to the reduction in ISI distortion caused by a reduction of the latency variation in neuromorphic PEs using the proposed changes, which we have analyzed in Section 7.2. In addition, the improvement of inference quality with PSNR and SSIM metrics for EdgeDet and ImgSmooth is higher than other inference tasks with accuracy metrics. This is because PSNR and SSIM metrics are computed on individual images where we see a large improvement in quality. For accuracy-based tasks, we observe that feature representation in hidden layers of these models changes due to ISI distortion, but not all such changes lead to misclassification. So the accuracy of these inference tasks is comparable to Baseline.

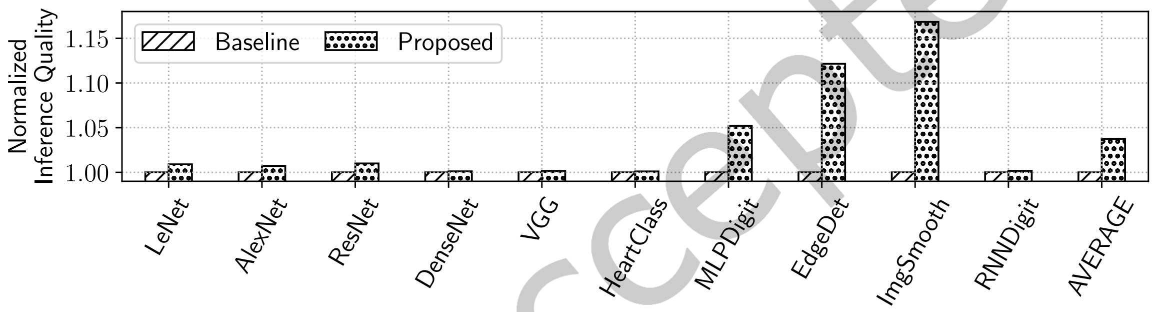


Fig. 19. Inference quality normalized to Baseline.

7.5 Single vs. Double Control Design

Figure 20 plots the energy efficiency of the proposed design with single control signal and the default, which uses two control signals for each PE. We observe that using single control, energy reduces by only 2% compared to Baseline. This is because most crossbars are operated in the expanded mode due to limited scope to collapse the crossbar. Our default design leads to 14.4% lower energy than with single control. This is because in the default design, a crossbar can be collapsed along X- and Y- dimensions independently, leading to three collapsed array configurations. Therefore, the system software has a higher probability to use the collapsed mode, leading to a reduction in energy.

7.6 Die Area Analysis

Adding an isolation transistor to the bitline increases the height of the crossbar, whereas that on the wordline increases the width. Without the isolation transistors, the height of a baseline crossbar is equal to the sum of height of the memory cells and the sense-amplifier, while the width is equal to the sum of the width of the memory cells. For RRAM-based neuromorphic PEs, a sense amplifier in the peripheral circuit and a isolation transistor is approximately 384x and 9.6x taller than an individual RRAM cell, respectively [18, 65, 97]. In terms of width, an isolation transistor is only 1.3x wider than an RRAM cell. Therefore, for a crossbar with 128 RRAM

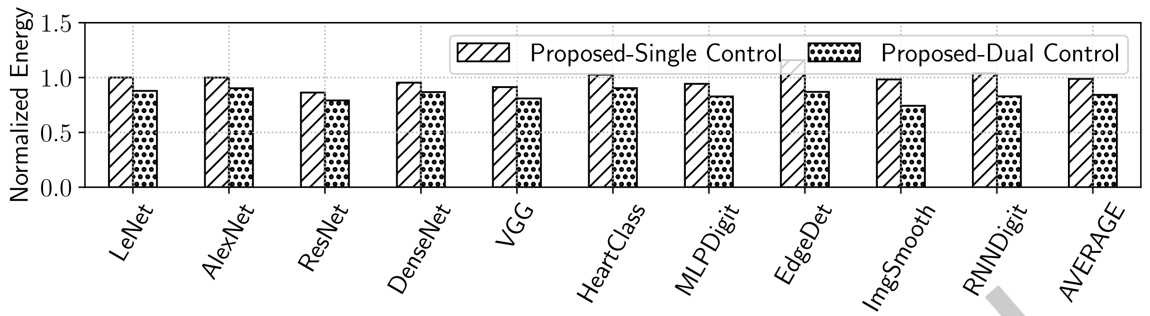


Fig. 20. Partitioned PE architecture with single and double control.

cells per bitline and wordline (i.e., 128×128 array), the overhead along the height of the crossbar is $\frac{9.6}{384+128} = 1.83\%$, and the overhead along the width of the crossbar is $\frac{1.3}{128} = 1.01\%$.

8 CONCLUSIONS

We present a design-technology co-optimization approach to implement energy-efficient machine-learning inference on NVM-based neuromorphic processing elements (PEs). First, we optimize the NVM resistance state such that the state that takes the longest time to sense is placed on current paths with fewer parasitics, and hence incurs lower propagation delay, and vice versa. Second, we use isolation transistors to partition a PE into collapsed and far regions such that the NVM cells of the far region can be opportunistically power-gated to save both energy and latency. Finally, we use the system software to exploit the design changes, maximizing the utilization of the collapsed region of each PE in the hardware. Our system software uses the far region only when it is absolutely necessary to do so, otherwise it improves both latency and energy by operating the PE in the collapsed mode. We evaluate our design-technology co-optimization approach for a state-of-the-art neuromorphic architecture. Evaluations with different machine-learning inference tasks show that the proposed approach improves both latency and energy without incurring significant cost-per-bit.

ACKNOWLEDGMENTS

This work is supported by 1) U.S. Department of Energy under Award Number DE-SC0022014, 2) the National Science Foundation Award CCF-1937419 (RTML: Small: Design of System Software to Facilitate Real-Time Neuromorphic Computing) and 3) the National Science Foundation Faculty Early Career Development Award CCF-1942697 (CAREER: Facilitating Dependable Neuromorphic Computing: Vision, Architecture, and Impact on Programmability).

REFERENCES

[1] [n.d.].
[2] Arnon Amir, Pallab Datta, William P Risk, Andrew S Cassidy, Jeffrey A Kusnitz, Steve K Esser, Alexander Andreopoulos, Theodore M Wong, Myron Flickner, Rodrigo Alvarez-Icaza, et al. 2013. Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores. In *IJCNN*.
[3] Aayush Ankit, Abhronil Sengupta, and Kaushik Roy. 2017. TraNNsformer: Neural network transformation for memristive crossbar based neuromorphic system design. In *ICCAD*.
[4] Adarsha Balaji, Prathyusha Adiraju, Hirak J Kashyap, Anup Das, Jeffrey L Krichmar, Nikil D Dutt, and Francky Catthoor. 2020. PyCARL: A PyNN interface for hardware-software co-simulation of spiking neural network. In *IJCNN*.

[5] Adarsha Balaji, Federico Corradi, Anup Das, Sandeep Pande, Siebren Schaafsma, and Francky Catthoor. 2018. Power-accuracy trade-offs for heartbeat classification on neural networks hardware. *JOLPE* (2018).

[6] Adarsha Balaji and Anup Das. 2019. A Framework for the Analysis of Throughput-Constraints of SNNs on Neuromorphic Hardware. In *ISVLSI*.

[7] Adarsha Balaji and Anup Das. 2020. Compiling Spiking Neural Networks to Mitigate Neuromorphic Hardware Constraints". In *IGSC Workshops*.

[8] Adarsha Balaji, Anup Das, Yuefeng Wu, Khanh Huynh, Francesco G. Dell'anna, Giacomo Indiveri, Jeffrey L. Krichmar, Nikil D. Dutt, Siebren Schaafsma, and Francky Catthoor. 2020. Mapping spiking neural networks to neuromorphic hardware. *TVLSI* (2020).

[9] Adarsha Balaji, Thibaut Marty, Anup Das, and Francky Catthoor. 2020. Run-time mapping of spiking neural networks to neuromorphic hardware. *JSPS* (2020).

[10] Adarsha Balaji, Shihao Song, Anup Das, Nikil Dutt, Jeff Krichmar, Nagarajan Kandasamy, and Francky Catthoor. 2019. A framework to explore workload-specific performance and lifetime trade-offs in neuromorphic computing. *CAL* (2019).

[11] Adarsha Balaji, Shihao Song, Anup Das, Jeffrey Krichmar, Nikil Dutt, James Shackleford, Nagarajan Kandasamy, and Francky Catthoor. 2020. Enabling Resource-Aware Mapping of Spiking Neural Networks via Spatial Decomposition. *ESL* (2020).

[12] Adarsha Balaji, Shihao Song, Twisha Titirsha, Anup Das, Jeffrey Krichmar, Nikil Dutt, James Shackleford, Nagarajan Kandasamy, and Francky Catthoor. 2021. NeuroXplorer 1.0: An Extensible Framework for Architectural Exploration with Spiking Neural Networks. In *ICONS*.

[13] Adarsha Balaji, Yuefeng Wu, Anup Das, Francky Catthoor, and Siebren Schaafsma. 2019. Exploration of segmented bus as scalable global interconnect for neuromorphic computing. In *GLSVLSI*.

[14] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. 2014. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics* (2014).

[15] Sumon Bose, Jyotibdha Acharya, and Arindam Basu. 2019. Is my neural network neuromorphic? Taxonomy, recent trends and future directions in neuromorphic engineering. *ACSSC* (2019).

[16] Geoffrey W. Burr, Robert M. Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, Lucas L. Sanches, Irem Boybat, Manuel Le Gallo, Kibong Moon, Jiyoo Woo, Hyunsang Hwang, and Yusuf Leblebici. 2017. Neuromorphic computing using non-volatile memory. *Advances in Physics: X* (2017).

[17] Francky Catthoor, Srinjoy Mitra, Anup Das, and Siebren Schaafsma. 2018. Very large-scale neuromorphic systems for biological signal processing. In *CMOS Circuits for Biological Sensing and Processing*.

[18] Pai-Yu Chen, Zhiwei Li, and Shimeng Yu. 2016. Design tradeoffs of vertical RRAM-based 3-D cross-point array. *TVLSI* (2016).

[19] Pai-Yu Chen and Shimeng Yu. 2015. Compact modeling of RRAM devices and its applications in 1T1R and 1S1R array design. *TED* (2015).

[20] Yangyin Chen. 2020. ReRAM: History, status, and future. *TED* (2020).

[21] Yi-Hsuan Chiu, Yi-Bo Liao, Meng-Hsueh Chiang, Chia-Long Lin, Wei-Chou Hsu, Pei-Chia Chiang, Yen-Ya Hsu, Wen-Hsing Liu, Shyh-Shyuan Sheu, Keng-Li Su, et al. 2010. Impact of resistance drift on multilevel PCM design. In *ICDT*.

[22] Ting Chou, Hirak Kashyap, Jinwei Xing, Stanislav Listopad, Emily Rounds, Michael Beyeler, Nikil Dutt, and Jeffrey Krichmar. 2018. CARLsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters. In *IJCNN*.

[23] Dennis V Christensen, Regina Dittmann, Bernabé Linares-Barranco, Abu Sebastian, Manuel Le Gallo, Andrea Redaelli, Stefan Slesazeck, Thomas Mikolajick, Sabina Spiga, Stephan Menzel, et al. 2021. 2021 Roadmap on Neuromorphic Computing and Engineering. *arXiv* (2021).

[24] Serena Curzel, Nicolas Bohm Agostini, Shihao Song, Ismet Dagli, Ankur Limaye, Cheng Tan, Marco Minutoli, Vito Giovanni Castellana, Vinay Amatya, Joseph Manzano, et al. 2021. Automated Generation of Integrated Digital and Spiking Neuromorphic Machine Learning Accelerators. In *ICCAD*.

[25] Anup Das, Francky Catthoor, and Siebren Schaafsma. 2018. Heartbeat classification in wearables using multi-layer perceptron and time-frequency joint distribution of ECG. In *CHASE*.

[26] Anup Das and Akash Kumar. 2018. Dataflow-Based Mapping of Spiking Neural Networks on Neuromorphic Hardware. In *GLSVLSI*.

[27] A. Das, P. Pradhapan, W. Groenendaal, P. Adiraju, R.T. Rajan, F. Catthoor, S. Schaafsma, J.L. Krichmar, N. Dutt, and C. Van Hoof. 2018. Unsupervised heart-rate estimation in wearables with Liquid states and a probabilistic readout. *Neural Networks* (2018).

[28] Anup Das, Yuefeng Wu, Khanh Huynh, Francesco Dell'Anna, Francky Catthoor, and Siebren Schaafsma. 2018. Mapping of local and global synapses on spiking neuromorphic hardware. In *DATE*.

[29] Mike Davies, Narayan Srinivasa, Tsung Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* (2018).

[30] Michael V. Debole, Brian Taba, Arnon Amir, Filipp Akopyan, Alexander Andreopoulos, William P. Risk, Jeff Kusnitz, Carlos Ortega Otero, Tapan K. Nayak, Rathinakumar Appuswamy, Peter J. Carlson, Andrew S. Cassidy, Pallab Datta, Steven K. Esser, Guillaume J. Garreau, Kevin L. Holland, Scott Lekuch, Michael Mastro, Jeff Mckinstry, Carmelo Di Nolfo, Jun Sawada, Brent Paulovicks, Kai Schleupen, Benjamin G. Shaw, Jennifer L. Klamo, Myron D. Flickner, John V. Arthur, and Dharmendra S. Modha. 2019. TrueNorth: Accelerating from zero to 64 million neurons in 10 years. *Computer* (2019).

[31] Peter Diehl and Matthew Cook. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. in Comp. Neuroscience* (2015).

[32] J Doevenspeck, R Degraeve, A Fantini, S Cosemans, A Mallik, P Debacker, D Verkest, R Lauwereins, and W Dehaene. 2021. OxRRAM-Based Analog in-Memory Computing for Deep Neural Network Inference: A Conductance Variability Study. *TED* (2021).

[33] B Rasitha Fernando, Yangjie Qi, Chris Yakopcic, and Tarek M Taha. 2020. 3D Memristor Crossbar Architecture for a Multicore Neuromorphic System. In *IJCNN*.

[34] Mohammed E Fouda, Ahmed M Eltawil, and Fadi Kurdahi. 2017. Modeling and analysis of passive switching crossbar arrays. *TCAS I* (2017).

[35] Mohammed E Fouda, Jongeun Lee, Ahmed M Eltawil, and Fadi Kurdahi. 2018. Overcoming crossbar nonidealities in binary neural networks through learning. In *NANOARCH*.

[36] Mohammed E Fouda, Sugil Lee, Jongeun Lee, Gun Hwan Kim, Fadi Kurdahi, and Ahmed Eltawil. 2020. IR-QNN Framework: An IR Drop-Aware Offline Training Of Quantized Crossbar Arrays. *IEEE Access* (2020).

[37] Mohammed E Fouda, E Neftci, Ahmed Eltawil, and F Kurdahi. 2019. Effect of asymmetric nonlinearity dynamics in RRAMs on spiking neural network performance. In *ACSSC*.

[38] Charlotte Frenkel. 2020. *Bottom-up and top-down neuromorphic processor design: Unveiling roads to embedded cognition.* Ph.D. Dissertation. PhD thesis, UCL-Université Catholique de Louvain.

[39] Francesco Galluppi, Sergio Davies, Alexander Rast, Thomas Sharp, Luis A Plana, and Steve Furber. 2012. A hierachical configuration system for a massively parallel neural hardware platform. In *Computing Frontiers*. 183–192.

[40] Dan FM Goodman and Romain Brette. 2009. The brian simulator. *Frontiers in Neuroscience* (2009).

[41] Roshan Gopalakrishnan, Yansong Chua, Pengfei Sun, Ashish Jith Sreejith Kumar, and Arindam Basu. 2020. HFNet: A CNN Architecture Co-designed for Neuromorphic Hardware With a Crossbar Array of Synapses. *Frontiers in Neuroscience* (2020).

[42] Antonio Gulli and Sujit Pal. 2017. *Deep learning with Keras*.

[43] Yintao He, Ying Wang, Xiandong Zhao, Huawei Li, and Xiaowei Li. 2020. Towards state-aware computation in ReRAM neural networks. In *DAC*.

[44] Michael L Hines and Nicholas T Carnevale. 1997. The NEURON simulation environment. *Neural Computation* (1997).

[45] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *ICPR*.

[46] Miao Hu, Hai Li, Yiran Chen, Qing Wu, Garrett S Rose, and Richard W Linderman. 2014. Memristor crossbar-based neuromorphic computing system: A case study. *TNNLS* (2014).

[47] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *DAC*.

[48] Phu Khanh Huynh, M. Lakshmi Varshika, Ankita Paul, Murat Isik, Adarsha Balaji, and Anup Das. 2022. Implementing Spiking Neural Networks on Neuromorphic Architectures: A Review. *arXiv* (2022).

[49] Giacomo Indiveri. 2003. A low-power adaptive integrate-and-fire neuron circuit. In *ISCAS*.

[50] YeonJoo Jeong, Mohammed A Zidan, and Wei D Lu. 2017. Parasitic effect analysis in memristor-array-based neuromorphic systems. *TNANO* (2017).

[51] Yu Ji, YouHui Zhang, ShuangChen Li, Ping Chi, CiHang Jiang, Peng Qu, Yuan Xie, and WenGuang Chen. 2016. NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *MICRO*.

[52] Yongtae Kim, Yong Zhang, and Peng Li. 2012. A digital neuromorphic VLSI architecture with memristor crossbar synaptic array for machine learning. In *SOCC*.

[53] Yongtae Kim, Yong Zhang, and Peng Li. 2015. A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing. *JETC* (2015).

[54] Olga Krestinskaya, Aidana Irmanova, and Alex Pappachen James. 2019. Memristive non-idealities: Is there any practical implications for designing neural network chips?. In *ISCAS*.

[55] Shamik Kundu, Kanad Basu, Mehdi Sadi, Twisha Titirsha, Shihao Song, Anup Das, and Ujjwal Guin. 2021. Special Session: Reliability Analysis for ML/AI Hardware. In *VTS*.

[56] Minh Le and Son Ngoc Truong. 2021. Memristor Crossbar Circuits for Neuromorphic pattern Recognition. In *ISOCC*.

[57] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. 2013. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *HPCA*.

[58] Tianjian Li, Xiangyu Bi, Naifeng Jing, Xiaoyao Liang, and Li Jiang. 2017. Sneak-path based test and diagnosis for 1R RRAM crossbar using voltage bias technique. In *DAC*.

[59] Yesheng Li and Kah-Wee Ang. 2021. Hardware Implementation of Neuromorphic Computing Using Large-Scale Memristor Crossbar Arrays. *Advanced Intelligent Systems* (2021).

[60] C-Y Liao, K-Y Hsiang, F-C Hsieh, S-H Chiang, S-H Chang, J-H Liu, C-F Lou, C-Y Lin, T-C Chen, C-S Chang, et al. 2021. Multibit Ferroelectric FET Based on Nonidentical Double $HfZrO_2$ for High-Density Nonvolatile Memory. *EDL* (2021).

[61] Chit-Kwan Lin, Andreas Wild, Gautham N. Chinya, Tsung-Han Lin, Mike Davies, and Hong Wang. 2018. Mapping Spiking Neural Networks onto a Manycore Neuromorphic Architecture. In *PLDI*.

[62] Chenchen Liu, Bonan Yan, Chaofei Yang, Linghao Song, Zheng Li, Beiye Liu, Yiran Chen, Hai Li, Qing Wu, and Hao Jiang. 2015. A spiking neuromorphic design with resistive crossbar. In *DAC*.

[63] Xiaoxiao Liu, Wei Wen, Xuehai Qian, Hai Li, and Yiran Chen. 2018. Neu-NoC: A high-efficient interconnection network for accelerated neuromorphic systems. In *ASP-DAC*.

[64] Wolfgang Maass. 1997. Networks of spiking neurons: The third generation of neural network models. *Neural Networks* (1997).

[65] A Mallik, D Garbin, A Fantini, D Rodopoulos, R Degraeve, J Stuijt, AK Das, S Schaafsma, P Debacker, G Donadio, et al. 2017. Design-technology co-optimization for OxRRAM-based synaptic processing unit. In *VLSIT*.

[66] Carver Mead. 1990. Neuromorphic electronic systems. *Proc. of the IEEE* (1990).

[67] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. 2017. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *TBCAS* (2017).

[68] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *IMW*.

[69] Onur Mutlu and Lavanya Subramanian. 2015. Research problems and opportunities in memory systems. *SUFI* (2015).

[70] Nishant S Nukala, Niranjan Kulkarni, and Sarma Vrudhula. 2014. Spintronic threshold logic array (STLA)—A compact, low leakage, non-volatile gate array architecture. *JPDC* (2014).

[71] Ankita Paul, Shihao Song, and Anup Das. 2021. Design Technology Co-Optimization for Neuromorphic Computing. In *IGSC Workshop*.

[72] Ankita Paul, Shihao Song, Twisha Titirsha, and Anup Das. 2022. On the Mitigation of Read Disturbances in Neuromorphic Inference Hardware. *D&T* (2022).

[73] Bipin Rajendran, Abu Sebastian, Michael Schmuker, Narayan Srinivasa, and Evangelos Eleftheriou. 2019. Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *Signal Processing Magazine* (2019).

[74] M Rakka, ME Fouda, R Kanj, Ahmed Eltawil, and FJ Kurdahi. 2020. Design Exploration of Sensing Techniques in 2T-2R Resistive Ternary CAMs. *TCAS II: Express Briefs* (2020).

[75] Wonbo Shim, Yandong Luo, Jae-sun Seo, and Shimeng Yu. 2020. Impact of read disturb on multilevel RRAM based inference engine: Experiments and model prediction. In *IRPS*.

[76] Shihao Song, Adarsha Balaji, Anup Das, Nagarajan Kandasamy, and James Shackleford. 2020. Compiling spiking neural networks to neuromorphic hardware. In *LCTES*.

[77] Shihao Song, Harry Chong, Adarsha Balaji, Anup Das, James Shackleford, and Nagarajan Kandasamy. 2021. DFSynthesizer: Dataflow-based synthesis of spiking neural networks to neuromorphic hardware. *TECS* (2021).

[78] Shihao Song and Anup Das. 2020. A case for lifetime reliability-aware neuromorphic computing. In *MWSCAS*.

[79] Shihao Song and Anup Das. 2020. Design Methodologies for Reliable and Energy-efficient PCM Systems. In *IGSC Workshops*.

[80] Shihao Song, Anup Das, and Nagarajan Kandasamy. 2020. Exploiting Inter- and Intra-Memory Asymmetries for Data Mapping in Hybrid Tiered-Memories. In *ISMM*.

[81] Shihao Song, Anup Das, and Nagarajan Kandasamy. 2020. Improving dependability of neuromorphic computing with non-volatile memory. In *EDCC*.

[82] Shihao Song, Anup Das, Onur Mutlu, and Nagarajan Kandasamy. 2019. Enabling and Exploiting Partition-Level Parallelism (PALP) in Phase Change Memories. *TECS* (2019).

[83] Shihao Song, Anup Das, Onur Mutlu, and Nagarajan Kandasamy. 2020. Improving Phase Change Memory Performance with Data Content Aware Access. In *ISMM*.

[84] Shihao Song, Anup Das, Onur Mutlu, and Nagarajan Kandasamy. 2021. Aging-Aware Request Scheduling for Non-Volatile Main Memory. In *ASP-DAC*.

[85] Shihao Song, Jui Hanamshet, Adarsha Balaji, Anup Das, Jeff Krichmar, Nikil Dutt, Nagarajan Kandasamy, and Francky Catthoor. 2021. Dynamic reliability management in neuromorphic computing. *JETC* (2021).

[86] Shihao Song, Lakshmi Varshika Mirtinti, Anup Das, and Nagarajan Kandasamy. 2021. A Design Flow for Mapping Spiking Neural Networks to Many-Core Neuromorphic Hardware. In *ICCAD*.

[87] Shihao Song, Twisha Titirsha, and Anup Das. 2021. Improving Inference Lifetime of Neuromorphic Systems via Intelligent Synapse Mapping. In *ASAP*.

[88] Sherin A Thomas, Sahibia Kaur Vohra, Rahul Kumar, Rohit Sharma, and Devarshi Mrinal Das. 2021. Analysis of Parasitics on CMOS based Memristor Crossbar Array for Neuromorphic Systems. In *MWSCAS*.

[89] Twisha Titirsha and Anup Das. 2020. Reliability-Performance Trade-offs in Neuromorphic Computing. In *IGSC Workshops*.

[90] Twisha Titirsha and Anup Das. 2020. Thermal-Aware Compilation of Spiking Neural Networks to Neuromorphic Hardware. In *LCPC*.

[91] Twisha Titirsha, Shihao Song, Adarsha Balaji, and Anup Das. 2021. On the Role of System Software in Energy Management of Neuromorphic Computing. In *CF*.

[92] Twisha Titirsha, Shihao Song, Anup Das, Jeffrey Krichmar, Nikil Dutt, Nagarajan Kandasamy, and Francky Catthoor. 2021. Endurance-Aware Mapping of Spiking Neural Networks to Neuromorphic Hardware. *TPDS* (2021).

[93] Shikhar Tuli, Marco Rios, Alexandre Levisse, and David Atienza ESL. 2020. RRAM-VAC: A variability-aware controller for RRAM-based memory architectures. In *ASP-DAC*.

[94] M Lakshmi Varshika, Adarsha Balaji, Federico Corradi, Anup Das, Jan Stuijt, and Francky Catthoor. 2022. Design of Many-Core Big Little μBrains for Energy-Efficient Embedded Neuromorphic Computing. In *DATE*.

[95] Zhehui Wang, Huaipeng Zhang, Tao Luo, Weng-Fai Wong, Anh Tuan Do, Paramasivam Vishnu, Wei Zhang, and Rick Siow Mong Goh. 2020. NCPower: Power Modelling for NVM-based Neuromorphic Chip. In *ICONS*.

[96] Parami Wijesinghe, Aayush Ankit, Abhronil Sengupta, and Kaushik Roy. 2018. An all-memristor deep spiking neural computing system: A step toward realizing the low-power stochastic brain. *TETCI* (2018).

[97] Cong Xu, Xiangyu Dong, Norman P Jouppi, and Yuan Xie. 2011. Design implications of memristor-based RRAM cross-point structures. In *DATE*.

[98] Cheng-Xin Xue, Wei-Hao Chen, Je-Syu Liu, Jia-Fang Li, Wei-Yu Lin, Wei-En Lin, Jing-Hong Wang, Wei-Chen Wei, Ting-Wei Chang, Tung-Cheng Chang, et al. 2019. 24.1 a 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In *ISSCC*.

[99] Steven R Young, Pravallika Devineni, Maryam Parsa, J Travis Johnston, Bill Kay, Robert M Patton, Catherine D Schuman, Derek C Rose, and Thomas E Potok. 2019. Evolving energy efficient convolutional neural networks. In *Big Data*.

[100] Shimeng Yu, Yexin Deng, Bin Gao, Peng Huang, Bing Chen, Xiaoyan Liu, Jinfeng Kang, Hong-Yu Chen, Zizhen Jiang, and H-S Philip Wong. 2014. Design guidelines for 3D RRAM cross-point architecture. In *ISCAS*.

[101] Xinjiang Zhang, Anping Huang, Qi Hu, Zhisong Xiao, and Paul K Chu. 2018. Neuromorphic computing with memristor crossbar. *Physica Status Solidi (a)* (2018).

[102] Wei Zhao and Yu Cao. 2007. Predictive technology model for nano-CMOS design exploration. *JETC* 1 (2007).

[103] Zhenhua Zhu, Jilan Lin, Ming Cheng, Lixue Xia, Hanbo Sun, Xiaoming Chen, Yu Wang, and Huazhong Yang. 2018. Mixed size crossbar based RRAM CNN accelerator with overlapped mapping method. In *ICCAD*.

A SPIKING NEURAL NETWORKS

Spiking Neural Networks (SNNs) enable powerful computations due to their spatio-temporal information encoding capabilities [64]. An SNN consists of neurons, which are connected via synapses. A neuron can be implemented as an integrate-and-fire (IF) logic, which is illustrated in Figure 21 (left). Here, an input current $U(t)$ (i.e., spike from a pre-synaptic neuron) raises the membrane voltage of the neuron. When this voltage crosses a threshold V_{th}, the IF logic emits an output spike, which propagates to is post-synaptic neuron. Figure 21 (middle) illustrates the membrane voltage of the IF neuron due to an input spike train. The moment of threshold crossing is illustrated in Figure 21 (right). These are the firing times of the output spike train of the neuron.

SNNs can implement many machine learning approaches such as supervised learning, unsupervised learning, reinforcement learning, and lifelong learning. We focus on supervised machine learning, where an SNN is pre-trained with representative data. Machine learning **inference** refers to feeding live data points to this trained SNN to generate the corresponding output.

B QUALITY OF INFERENCE

The **quality** of machine learning inference can be expressed in terms of accuracy [5], Mean Square Error (MSE) [27], Peak Signal-to-Noise Ratio (PSNR) [22], and Structural Similarity Index Measure (SSIM) [45]. While accuracy is commonly used for assessing the quality of supervised learning, e.g., using Convolution Neural Networks (CNNS), there are also applications such as edge detection, where the quality is assessed using other metrics such as SSIM. In our prior work [8], we have shown that these quality metrics are a function of the

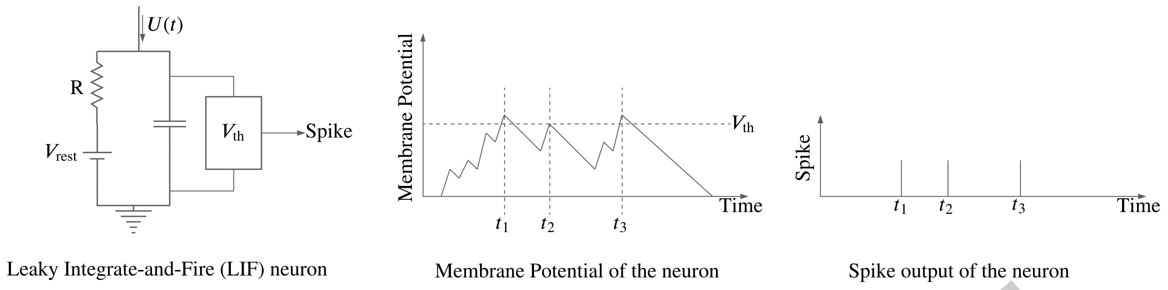


Fig. 21. A leaky integrate-and-fire (LIF) neuron with current input $U(t)$ (left). The membrane potential over time of the neuron (middle). The spike output of the neuron representing its firing time (right).

inter-spike interval (ISI) between neurons. Therefore, any deviation of ISI (called ISI distortion) from its trained value may lead to quality loss. To describe ISI, let $\{t_1, t_2, \cdots, t_K\}$ denote a neuron's firing times in the time interval $[0, T]$, the average ISI of this spike train is

$$\mathcal{I} = \sum_{i=2}^{K}(t_i - t_{i-1})/(K-1). \tag{7}$$

To illustrate how a change in ISI, called **ISI distortion**, impacts inference quality, we use a small SNN in which three input neurons are connected to an output neuron. Figure 22 illustrates the impact of ISI distortion on the output spike. In the top sub-figure, a spike is generated at the output neuron at 22μs due to spikes from the input neurons. In the bottom sub-figure, the second spike from input 3 is delayed, i.e., it has an ISI distortion. Due to this distortion, there is no output spike generated. Missing spikes can impact inference quality, as spikes encode information in SNNs.

Figure 23 shows the impact of ISI distortion on the quality of image smoothing implemented using an SNN [22]. Figure 23a shows the input image, which is fed to the SNN. Figure 23b shows the output of the image smoothing application with no ISI distortion. PSNR of the output with reference to the input is 20. Figure 23c shows the output with ISI distortion due to variation in latency within neuromorphic PEs of the hardware. PSNR of this output with respect to the input is 19. A reduction in PSNR indicates that the output image quality with ISI distortion is lower than the one without distortion. In fact, image quality deteriorates with increase in ISI distortion. We use ISI distortion as a measure of the quality of machine learning inference [8]. Our aim is to improve this inference quality via technological and architectural enhancements that reduce ISI distortion when the inference task is implemented on neuromorphic PEs of a hardware.

C HARDWARE IMPLEMENTATION OF MACHINE LEARNING INFERENCE

Most neuromorphic hardware platforms are implemented as tiled-based architectures [17, 29, 30, 38, 73, 94], where the tiles are interconnected via a shared interconnect such as Network-on-Chip [63] and Segmented Bus [13]. Figure 24 illustrates a tile-based neuromorphic hardware platform, where the tiles can communicate concurrently. Each tile includes 1) a neuromorphic PE, which consists of neuron and synapse circuitries and 2) a network interface, which encodes spikes into Address Event Representation (AER) and communicates these AER packets to the switch for routing to their destination tiles. A common design practice is to use analog crossbars to implement a neuromorphic PE [3, 8, 46, 53, 56, 59, 62, 101]. Within a crossbar, a pre-synaptic neuron circuit acts as a current driver and is placed on a wordline, while a post-synaptic neuron circuit acts as a current sink and is placed on a bitline as illustrated in Figure 1 (left).

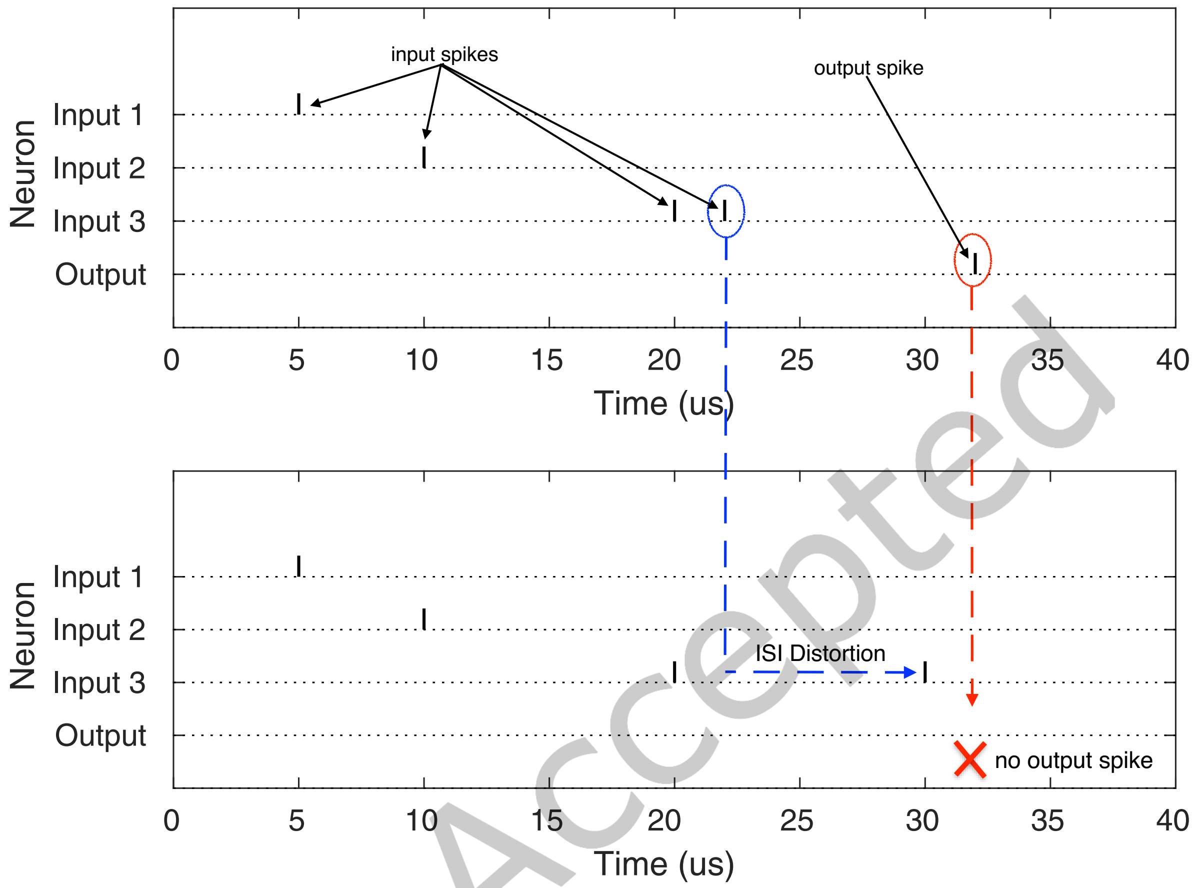
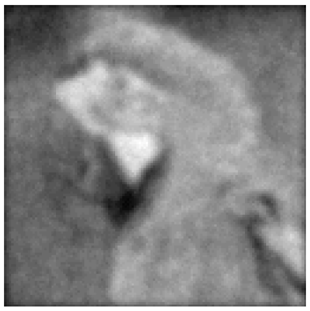


Fig. 22. Impact of ISI distortion on accuracy [4]. Top sub-figure shows a scenario where an output spike is generated based on the spikes received from the three input neurons. Bottom sub-figure shows a scenario where the second spike from neuron 3 is delayed. There are no output spikes generated.

Since a crossbar can accommodate only a limited number of neurons and synapses, a machine learning model is first partitioned into clusters, where each cluster can be implemented on a crossbar of the hardware. Partitioned clusters are then mapped to different crossbars when admitting the model to the hardware platform. To this end, several heuristic approaches are proposed in literature. PSOPART [28] minimizes spike latency on the shared interconnect, SpiNeMap [8] minimizes interconnect energy, DFSynthesizer [77] maximizes throughput, DecomposedSNN [11] maximizes crossbar utilization, EaNC [91] minimizes overall energy of a machine learning task by targeting both computation and communication energy, TaNC [90] minimizes the average temperature of each crossbar, eSpine [92] maximizes NVM endurance in a crossbar, RENEU [81] minimizes the circuit aging in a crossbar's peripheral circuits, and NCil [87] reduces read disturb issues in a crossbar, improving the inference lifetime. Beside these techniques, there are also other software frameworks [2, 6, 7, 10, 12, 24, 26, 39, 48, 51, 55, 61, 72, 76, 78, 79, 86, 89] and run-time approaches [9, 85], addressing one or more of these optimization objectives.

We investigate the internal architecture of a crossbar and find that the parasitic components introduce delay in propagating current from a pre-synaptic neuron to a post-synaptic neuron as illustrated in Figure 1 (right). This

(a) Original Image.


(b) Output with no ISI distortion (PSNR = 20).

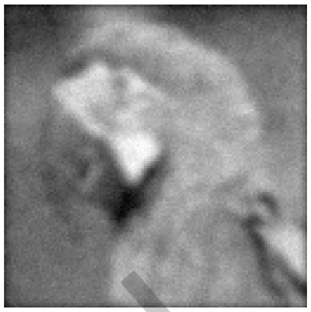

(c) Output with ISI distortion (PSNR = 19).

Fig. 23. Impact of ISI distortion on image smoothing.

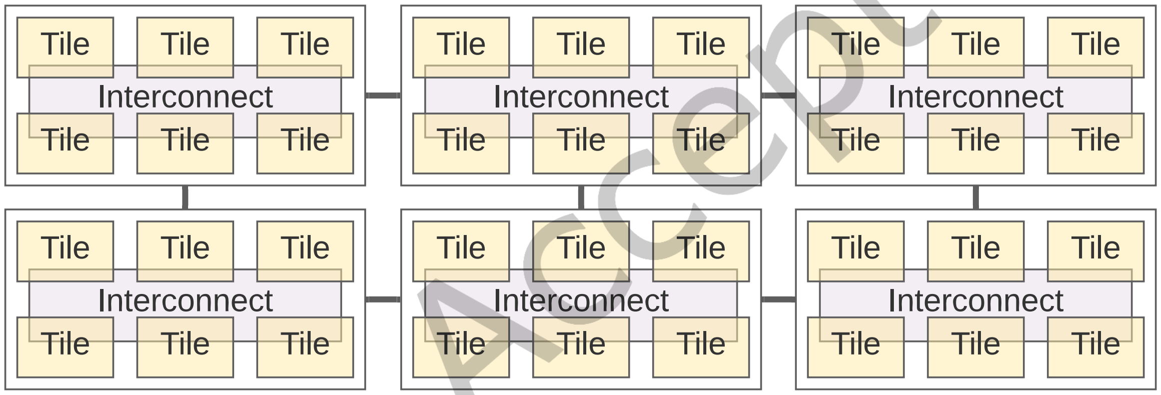


Fig. 24. Tile-based neuromorphic hardware, representative of hardware platforms such as TrueNorth [30], Loihi [29], DYNAPs [67], and μBrain [94].

delay depends on the specific current path used in the mapping. Higher the number of parasitic components on a current path, larger is its propagation delay. Parasitic components on bitlines and wordlines are a major source of latency at scaled process technology nodes and they create significant **latency variation** in a crossbar. Specifically, the latency of a synaptic connection in an SNN depends precisely on the memory cell in the crossbar that is used to implement it. Such latency variation can introduce ISI distortion (Section B), which may impact the quality of an inference task.

D NON-VOLATILE MEMORY TECHNOLOGY

RRAM technology presents an attractive option for implementing memory cells of a crossbar due to its demonstrated potential for low-power multilevel operation and high integration density [65]. An RRAM cell is composed of an insulating film sandwiched between conducting electrodes forming a metal-insulator-metal (MIM) structure (see Figure 25). Recently, conducting filament-based metal-oxide RRAM implemented with transition-metal-oxides such as HfO_2, ZrO_2, and TiO_2 has received considerable attention due to their low-power and CMOS-compatible scaling.

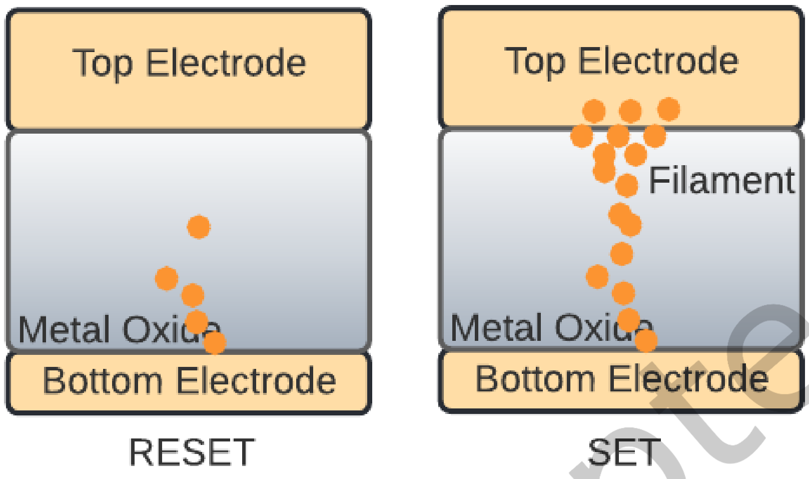


Fig. 25. Operation of an RRAM cell with the HfO_2 layer sandwiched between the metals Ti (top electrode) and TiN (bottom electrode). The right subfigure shows the formation of LRS/SET state. The left subfigure shows the HRS/RESET state.

Synaptic weights are represented as conductance of the insulating layer within each RRAM cell. To program an RRAM cell, elevated voltages are applied at the top and bottom electrodes, which re-arranges the atomic structure of the insulating layer. Figure 25 shows the High-Resistance State (HRS) and the Low-Resistance State (LRS) of an RRAM cell. An RRAM cell can also be programmed into intermediate low-resistance states, allowing its multilevel operations [19].