On the Role of System Software in Energy Management of Neuromorphic Computing

Twisha Titirsha*
tt624@drexel.edu
Drexel University
Philadelphia, PA, USA

Shihao Song*
ss3695@drexel.edu
Drexel University
Philadelphia, PA, USA

Adarsha Balaji*
ab3586@drexel.edu
Drexel University
Philadelphia, PA, USA

Anup Das
anup.das@drexel.edu
Drexel University
Philadelphia, PA, USA

ABSTRACT

Neuromorphic computing systems such as DYNAPs and Loihi have recently been introduced to the computing community to improve performance and energy efficiency of machine learning programs, especially those that are implemented using Spiking Neural Network (SNN). The role of a system software for neuromorphic systems is to cluster a large machine learning model (e.g., with many neurons and synapses) and map these clusters to the computing resources of the hardware. In this work, we formulate the energy consumption of a neuromorphic hardware, considering the power consumed by neurons and synapses, and the energy consumed in communicating spikes on the interconnect. Based on such formulation, we first evaluate the role of a system software in managing the energy consumption of neuromorphic systems. Next, we formulate a simple heuristic-based mapping approach to place the neurons and synapses onto the computing resources to reduce energy consumption. We evaluate our approach with 10 machine learning applications and demonstrate that the proposed mapping approach leads to a significant reduction of energy consumption of neuromorphic computing systems.

CCS CONCEPTS

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

KEYWORDS

Spiking Neural Network (SNN), Neuromorphic Computing, Non Volatile Memory (NVM), Energy Consumption, Static Power, Dynamic Power

*Authors contributed equally to this research.

CF '21, May 11–13, 2021, Virtual Conference, Italy

ACM ISBN 978-1-4503-8404-9/21/05…$15.00
https://doi.org/10.1145/3457388.3458664

ACM Reference Format:
Twisha Titirsha, Shihao Song, Adarsha Balaji, and Anup Das. 2021. On the Role of System Software in Energy Management of Neuromorphic Computing. In *Computing Frontiers Conference (CF '21), May 11–13, 2021, Virtual Conference, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3457388.3458664

1 INTRODUCTION

Neuromorphic computing describes the VLSI implementation of the neuro-biological architecture of the central nervous system [6, 14, 49]. Neuromorphic systems are energy efficient in executing Spiking Neural Networks (SNNs), which are considered as the third generation of neural networks [47]. SNNs use spike-based computations and bio-inspired learning algorithms in solving machine learning problems. In an SNN, pre-synaptic neurons communicate information encoded in spike trains to post-synaptic neurons, via the synapses. Performance of an SNN-based application can be assessed in terms of the inter-spike interval (ISI coding) or mean firing rate of the neurons (rate coding).

The hardware architecture of neuromorphic systems consists of neurosynaptic cores, which are interconnected via a shared interconnect [7]. Figure 1 illustrates the representative hardware architecture of many recent neuromorphic systems such as Loihi [30], TrueNorth [32], and DYNAPs [50].

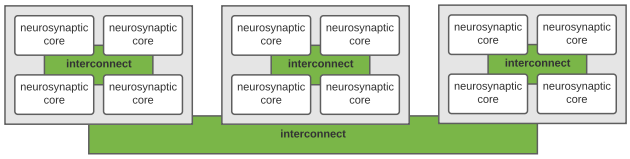


Figure 1: Hardware architecture of neuromorphic systems.

Neurosynaptic cores are the computing resources of a neuromorphic system. In many emerging architectures, a neurosynaptic core is essentially a crossbar array that can accommodate a fixed number of neurons and synapses. The role of a system software for neuromorphic systems is therefore, to partition an SNN with many neurons and synapses into clusters such that, the neurons and synapses of each cluster can be mapped on to a neurosynaptic core of the system. Therefore, the inter-cluster synapses are mapped on the shared interconnect of the system.

Recently, energy consumption of neuromorphic systems has come into spot light, specifically due to their application in power-constrained environments such as Embedded Systems and Internet-of-Things (IoT). To this end, several low-power analog neuron designs are proposed to implement neurosynaptic cores for neuromorphic systems [41, 53, 55, 73, 74]. Another research direction is the shift from Static RAM (SRAM)-based synapse designs to implementations that use Non-Volatile Memory (NVM) – Phase Change Memory (PCM) [13], Oxide-based Resistive RAM (OxRRAM) [48], Ferroelectric RAM (FeRAM) [52], and Spin-Transfer Torque Magnetic or Spin-Orbit-Torque RAM (STT/ SoT-MRAM) [72].[1]

In addition to the hardware-oriented energy reduction techniques, we argue that the system software also plays a pivotal role in the energy consumption of neuromorphic systems. We show that the energy consumption of a neuromorphic hardware depends on 1) how an SNN model is partitioned into clusters, 2) how the clusters are mapped to the neurosynaptic cores, and 3) how the neurons and synapses of a cluster are placed inside each core. Following are our key **contributions**.

- We formulate the energy consumption of a neuromorphic hardware, considering the energy consumed inside each neurosynaptic core and the energy consumed in communicating spikes on the shared interconnect.
- We show that by not considering all the sources of energy loss, existing system software approaches leave a significant energy improvement opportunities.
- We propose a heuristic to minimize the total energy consumption in neuromorphic computing without significantly increasing the spike latency. This leads to only a very marginal impact on performance.
- We evaluate our mapping approach with 10 machine learning workloads on a cycle-accurate simulator of a state-of-the-art neuromorphic hardware.

Results demonstrate that the current system software frameworks for neuromorphic systems miss significant energy improvement opportunities. By explicitly incorporating energy consumption of different hardware units, the proposed mapping approach significantly minimizes the energy consumption of neuromorphic systems.

2 BACKGROUND

There are many recent initiatives to map machine learning workloads to neuromorphic hardware. PACMAN [37] is used to map SNNs to SpiNNaker hardware [36]. Corelet [1] is used to map workloads to TrueNorth [32]. NEUTRAM [42] is a mapping approach for digital neuromorphic chips such as TIANJI [59]. PyNN [31], which started as a front-end to many back-end SNN simulators such as Brian [39], NEURON [40], and NEST [34], can now map SNN applications to many neuromorphic hardware such as Loihi [30] and Neurogrid [12]. A recent extension of PyNN, called PyCARL [8], can simulate SNN applications using the back-end CARLsim simulator [15], allowing mapping of these applications to the DYNAPs neuromorphic hardware [50]. There are also other propritary approaches to mapping SNN applications to emerging neuromorphic chips such as BrainScaleS [56], Braindrop [54], and ODIN [35].

DecomposedSNN [11] uses spatial decomposition technique to unroll each neuron with many fanin connections into smaller atomic units that are connected sequentially. This allows to densely pack each crossbar in a neuromorphic hardware leading to a significant improvement of resource utilization and a reduction of hardware area overhead. PSOPART [29] and SpiNeMap [9] are mapping approaches that minimize spike communication energy on the shared interconnect by lowering the spike volume and spike latency, respectively. SPINERTM [10] is proposed to remap SNN applications to neuromorphic hardware at tun-time by monitoring the performance degradation. DFSynthesizer [64] uses data flow models to analyze performance of SNN workloads on crossbar-based neuromorphic hardware. There are also other dataflow-based technique reported in literature [2, 3, 18]. These approaches are demonstrated with many SNN applications, such as the liquid state machine (LSM)-based heart-rate estimation [16]; spiking ResNet architecture for ImageNet classification [57]; deep learning architecture for DNA sequence analysis [51]; heart-rate classification using spiking CNN architecture using ECG data [4, 28]; lateral inhibition-based digit recognition [33]; recurrent architecture-based predictive visual pursuit [43]; spiking architecture for seizure classification using EEG data [38]; among others.

RENEU [66] is a recent technique proposed to map SNN applications to hardware improving the circuit aging of the peripheral circuitry in crossbars, which is caused due to their high-voltage exposure. There are also other approaches targeting circuit aging [5, 62]. ESPINE [71] is an approach to map SNN applications to neuromorphic hardware, improving the endurance of its Non-Volatile Memory synapses. There are also other mapping approaches that target temperature optimization [70] and releiability-performance trade-offs [45, 69]. We compare our Hill Climbing approach against PyCARL and SpiNeMap, and found it to perform significantly better in terms of energy consumption.

3 PROBLEM FORMULATION

Unlike system software for conventional computers (e.g., the Operating System), the role of the system software for neuromorphic hardware is to cluster a machine learning model and map the clusters onto the crossbars of the neuromorphic hardware. Figure 2 illustrates the mapping concept using an example SNN shown in (❶). The number on a link represents the average number of spikes communicated between the source and destination neurons for a representative training data. We consider the mapping of this SNN to a hardware with 2×2 crossbars. Since a crossbar in this hardware can only accommodate a maximum of 2 pre-synaptic connections, we partition the SNN of (❶) into two clusters (shown in two different colors) in (❷). These clusters can then be mapped to the two crossbars as shown in (❸), with an average 8 spikes communicated between the crossbars.

In many neuromorphic applications, the number of pre-synaptic connections per neuron can well exceed the crossbar input limit (which is typically 128 or 256). For those applications, each neuron

[1] Beside neuromorphic computing, some of these memristor technologies are also used as main memory in conventional computers to improve performance and energy efficiency [61, 63, 65, 67, 68].

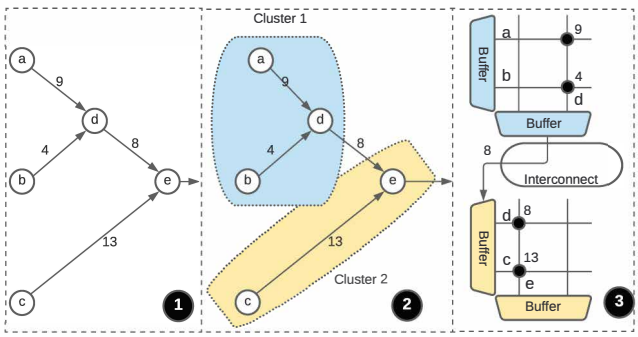


Figure 2: A typical clustering approach used by system software to map SNNs to neuromorphic hardware.

is first decomposed into smaller units with fewer pre-synaptic connections before they are clustered using the approach illustrated in Fig. 2 (see [11]).

Figure 3 illustrates the clusters 7 clusters of the LeNet Convolutional Neural Network (CNN) obtained using the clustering technique of SpiNeMap.

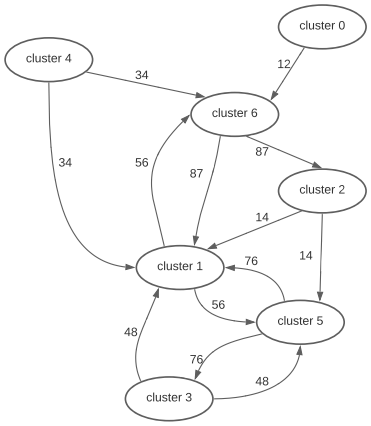


Figure 3: The clusters generated from LeNet CNN.

Formally, a clustered SNN graph is defined as follows.

Definition 1. *(Clustered SNN) A clustered SNNs* $G_{CSNN} = (\boldsymbol{C}, \boldsymbol{L})$ *is a directed graph consisting of a finite set* $\boldsymbol{C}$ *of clusters and a finite set* $\boldsymbol{L}$ *of connections between these clusters.*

Each cluster $C_i \in \mathbf{C}$ is a tuple $\langle In(C_i), Out(C_i), S(C_i), W(C_i)\rangle$, where $In(C_i)$ is the number of pre-synaptic neurons of the cluster, $Out(C_i)$ is the number of post-synaptic neurons of the cluster, $S(C_i)$ is the number of spikes generated inside the cluster, and $W(C_i)$ is the set of synaptic weights of the cluster. Each link $L_i \in \mathbf{L}$ of the graph has a value $Spk(L_i)$ attached to it representing the number of spikes communicated on the link between the source and the destination clusters.

The clusters of an SNN-based application are mapped to the tiles of a neuromorphic hardware, where a tile consists of a neurosynaptic core, e.g., a crossbar.

Formally, a neuromorphic hardware is defined as follows.

Definition 2. *(Neuromorphic Hardware) A neuromorphic hardware* $G_{NH} = (\boldsymbol{T}, \boldsymbol{I})$ *is a directed graph consisting of a finite set* $\boldsymbol{T}$ *of tiles and a finite set* $\boldsymbol{I}$ *of interconnect links.*

Each tile consists of a crossbar to map neurons and synapses, and input and output buffers to receive and send spikes over the interconnect, respectively. A tile $T_i \in \mathbf{T}$ is a tuple $\langle M, InB(T_i), OutB(T_i)\rangle$, where M is the dimension of a crossbar on the tile, i.e., the tile T_i can accommodate M pre-synaptic neurons, M post-synaptic neurons, and M^2 synaptic connections, $InB(T_i)$ is the input buffer size on the tile, and $OutB(T_i)$ is its output buffer size. Each interconnect link is bidirectional, representing two-way communication between the source and destination tiles with a fixed bandwidth BW.

When mapping the clusters to the tiles of the hardware, spikes from a tile (i.e., the cluster mapped to the tile) are broadcasted on the interconnect. The network interface (NI) logic on each tile ensures the delivery of the spikes to the intended recipient neurons mapped to these tiles.

To formalize the energy consumption, we consider the mapping of a clustered SNN $G_{CSNN} = (\mathbf{C}, \mathbf{L})$ to the neuromorphic hardware $G_{NH} = (\mathbf{T}, \mathbf{I})$.

Mapping $M : G_{CSNN} = (\mathbf{C}, \mathbf{L}) \rightarrow G_{NH} = (\mathbf{T}, \mathbf{I})$ is specified by a logical matrix $(m_{ij}) \in \{0,1\}^{|\mathbf{C}|\times|\mathbf{T}|}$, where m_{ij} is defined as

$$m_{ij} = \begin{cases} 1 & \text{if cluster } C_i \in \mathbf{C} \text{ is mapped to tile } T_j \in \mathbf{T} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

To simplify the discussion, we consider a neuromorphic hardware to have as many tiles as clusters of a given application. The energy formulation also holds when the tiles are time-multiplexed among the clusters [64].

4 ENERGY MODELING

In this section, we provide a comprehensive energy model for neuromorphic hardware executing machine learning workloads. We consider the following energy components.

4.1 Spike Energy

This is the total energy consumed on the tiles to generate all the spikes for a given SNN application working on a representative data.

Using the formalism of Section 3, the spike energy is

$$E_{spk} = \sum_{i=0}^{|\mathbf{C}|-1} \cdot \sum_{j=0}^{S(C_i)-1} e_{spk}(i,j), \tag{2}$$

where $e_{spk}(i,j)$ is the energy of j^{th} spike on tile C_i. Since each cluster is mapped to a tile of the hardware, the outer summation is for all the clusters of an application, while the inner summation is for all the spikes generated inside each cluster. $e_{spk}(i,j)$ comprises of two components – the energy to generate a spike by a pre-synaptic neuron (e_{neuron}), which remains the same for all the tiles (ignoring process variation for the moment), and the energy consumed on a synaptic cell due to the flow of current ($e_{synapse}(i,j)$). Therefore,

$$e_{spk}(i,j) = e_{neuron} + e_{synapse}(i,j). \tag{3}$$

In all previous work, the energy per spike is typically assumed to be constant. However, here we show this is not the case. In general, the synaptic energy depends on the specific NVM used to model the synaptic weights in a neurosynaptic core. We formulate this for the Phase-Change Memory (PCM). The **scope** of this work is on the inference phase, wherein a machine learning model is pre-trained offline, and the trained model is programmed on the neuromorphic

hardware. Therefore, we focus on the energy to read the synaptic weights stored in the PCM cells of a crossbar.

The energy consumed in propagating current through a PCM cell is given by Joule Heating [21, 24–27, 71]

$$e_{synapse} = I_{prog}^2 \cdot (R_{synapse} + R_{ON}) \cdot t_{spk}, \quad (4)$$

where I_{prog} is the current generated for the spike voltage ($\approx 50\mu A$), $R_{synapse}$ is the resistance of the PCM cell, R_{ON} is the ON resistance of the access transistor connecting the PCM cell and t_{spk} is the spike duration (typically a few ms). Considering $W(C_i)$ to be the synaptic weights of the cluster C_i, which are programmed as conductances, Eq. 5 can be written as

$$e_{synapse}(i,j) = I_{prog}^2 \cdot t_{spk} \cdot \left(R_{ON} + \frac{1}{w(i,j)}\right), \quad (5)$$

where $w(i,j) \in W(C_i)$ is the conductance of the PCM cell on the path of the j^{th} spike in the cluster C_i.

Figure 4 shows a simple 2-input and 1-output SNN. The neurons are shown as grayed circles, while the synaptic weights are shown on each connection. The number on a link represents the number of spikes for a given input.

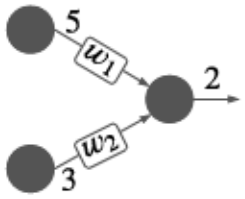


Figure 4: Example of calculating E_{spk} for a simple SNN.

The energy for the 5 spikes generated from the top pre-synaptic neuron = $5 \cdot \left|e_{neuron} + I_{prog}^2 \cdot t_{spk} \cdot \left(R_{ON} + \frac{1}{w_1}\right)\right|$. The energy for the 3 spikes from the bottom pre-synaptic neuron = $3 \cdot \left|e_{neuron} + I_{prog}^2 \cdot t_{spk} \cdot \left(R_{ON} + \frac{1}{w_2}\right)\right|$. Finally, the energy for the 2 spikes generated from the post-synaptic neuron is $2 \cdot e_{neuron}$. Therefore, the total spike energy is

$$\begin{aligned} E_{spk} = \; & 5 \cdot \left|e_{neuron} + I_{prog}^2 \cdot t_{spk} \cdot \left(R_{ON} + \frac{1}{w_1}\right)\right| + \\ & 3 \cdot \left|e_{neuron} + I_{prog}^2 \cdot t_{spk} \cdot \left(R_{ON} + \frac{1}{w_2}\right)\right| + \\ & 2 \cdot e_{neuron} \end{aligned}$$

From a crossbar perspective, the parasitic components on the rows and columns create asymmetry in current propagating through different NVM cells in the crossbar. Figure 5 shows the current variation in a 128x128 PCM crossbar.

Considering these current variations in a crossbar, the programming current I_{prog} is not a constant value for every spike generated in a crossbar. In fact, the programming current is higher for spikes propagating through a synaptic cell located at the bottom left corner than through a synaptic cell located at the top right corner (see Fig. 5). This is illustrated in Figure 6, which shows two different ways of mapping the SNN of Figure 6a to the crossbar. For Figure 6c, the programming current is higher than the mapping of Figure 6b.

Therefore, the spike energy for an SNN application on a neuromorphic hardware is

$$E_{spk} = \sum_{i=0}^{|C|-1} \cdot \sum_{j=0}^{S(C_i)-1} \left|e_{neuron} + I_{prog}^2(i,j) \cdot t_{spk} \cdot \left(R_{ON} + \frac{1}{w(i,j)}\right)\right| \quad (6)$$

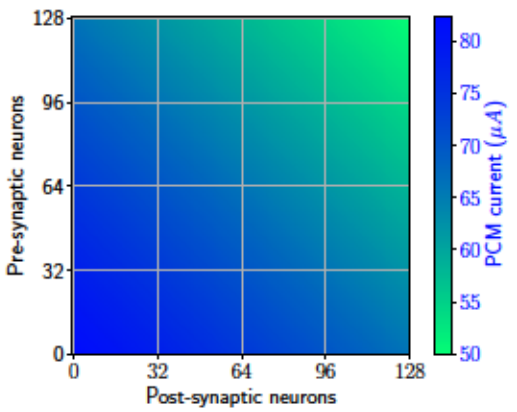


Figure 5: Current map in a 128x128 crossbar.

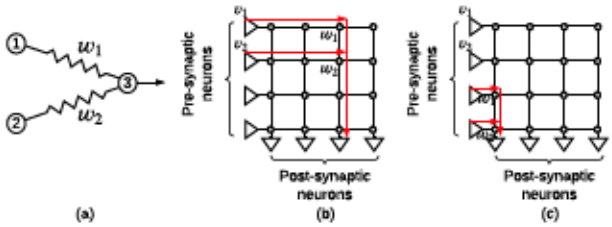


Figure 6: (a) A simple spiking neural network, (b) Mapping of the network to a crossbar, (c) A different mapping of the network to the hardware.

where $I_{prog}(i,j)$ is the current of the j^{th} spike in crossbar C_i and it depends on where the corresponding synaptic connection is mapped within a crossbar.

4.2 Communication Energy

This is the total energy consumed by all spikes on the interconnect of a neuromorphic hardware for a given SNN application working on a representative data.

In mapping clusters to tiles, the inter-cluster spikes are the ones that are communicated over the interconnect. Using the formalism of Section 3, the communication energy is

$$E_{comm} = \sum_{k=0}^{|\mathbf{L}|-1} Spk(L_k) \cdot e_{comm}(L_k) \quad (7)$$

where $e_{comm}(L_k)$ is the energy to communicate a spike on the link between the source and destination clusters of the link $L_k \in \mathbf{L}$. $e_{comm}(L_k)$ depends on the hardware architecture and how tiles are interconnected. In general,

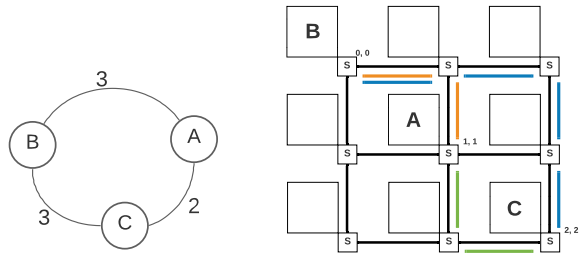
$$e_{comm}(L_k) = e_{switch} \cdot (h_k - 1) + e_{wire} \cdot h_k, \quad (8)$$

where h_k is the hop distance between the source and destination tiles of the connection L_k, e_{wire} is the energy consumed on the interconnect wires, and e_{switch} is the energy consumed on the switch [17, 19, 20, 22, 23, 58]. In the following, we consider a mesh-based organization of tiles with X-Y routing of spikes. For this interconnect architecture, the hop count between two tiles is their

Manhattan Distance. Specifically, if the source cluster of the link L_k, represented as $Src(L_k)$, is placed on a tile located at coordinate $\left(x_{src}(L_k), y_{src}(L_k)\right)$, while the destination cluster, represented as $Dst(L_k)$, is placed on a tile located at coordinate $\left(x_{dst}(L_k), y_{dst}(L_k)\right)$, then

$$\begin{aligned} h_k &= ManhattanDistance\Big(Src(L_k), Dst(L_k)\Big) \\ &= \Big|x_{src}(L_k) - x_{dst}(L_k)\Big| + \Big|y_{src}(L_k) - y_{dst}(L_k)\Big| \end{aligned} \tag{9}$$

Figure 7 illustrates the placement of an example clustered SNN to a mesh architecture. Cluster A in this example is placed at coordinate (1,1), cluster B at (0,0), and cluster C at (2,2). As can be seen from this figure, the hop distance between A and B is 2, between B and C is 4, and between C and A is 2. Therefore, the communication energy for spikes communicated between A and B $= 3 \cdot \Big[e_{switch} + 2 * e_{wire}\Big]$, that between B and C $= 3 \cdot \Big[3 \cdot e_{switch} + 4 * e_{wire}\Big]$, and that between A and C $= 2 \cdot \Big[e_{switch} + 2 * e_{wire}\Big]$.



(a) Example clustered SNN.

(b) Placement of the SNN to a mesh architecture..

Figure 7: Example of calculation E_{comm} for a clustered SNN placed on a mesh architecture.

Therefore, the total communication energy is

$$\begin{aligned} E_{comm} &= 3 \cdot \Big[e_{switch} + 2 * e_{wire}\Big] + \\ &\quad 3 \cdot \Big[3 \cdot e_{switch} + 4 * e_{wire}\Big] + \\ &\quad 2 \cdot \Big[e_{switch} + 2 * e_{wire}\Big] \end{aligned}$$

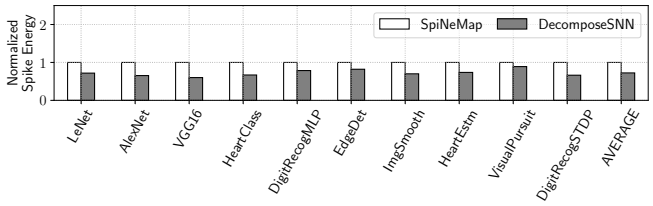
Overall, the communication energy for an SNN application mapped to a neuromorphic hardware is

$$\boxed{E_{comm} = \sum_{k=0}^{|\mathbf{L}|-1} Spk(L_k) \cdot \Big[e_{switch} \cdot (h_k - 1) + e_{wire} \cdot h_k\Big]} \tag{10}$$

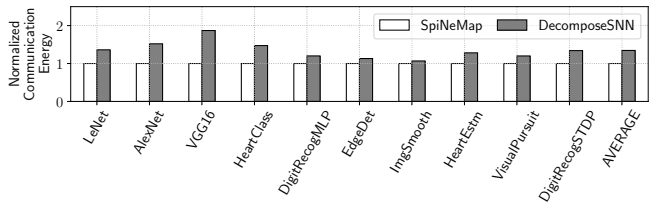
4.3 Energy Dependencies and the Role of System Software in Energy Consumption

From Equations 6 and 10, we can conclude that energy consumption depends on 1) how an SNN model is partitioned into clusters (determines the number of neurons and synapses in each cluster), 2) how the clusters are mapped to the neurosynaptic cores (determines the hop distances), and 3) how the neurons and synapses of a cluster are placed inside each core (determines the spikes propagation). All these factors can be controlled via a system software such as SpiNeMap [9] and DecomposeSNN [11].

Figure 8 compares SpiNeMap, which minimizes communication on the interconnect and DecomposeSNN, which maximizes crossbar utilization for 10 machine learning applications (see our evaluation methodology in Section 6).



(a) Spike Energy (E_{spk}) of SpiNeMap and DecomposeSNN.



(b) Communication Energy (E_{comm}) of SpiNeMap and DecomposeSNN.

Figure 8: Role of system software in energy management of neuromorphic computing.

We observe that SpiNeMap has 24% lower communication energy and 40% higher spike energy than DecomposeSNN. This is because SpiNeMap explicitly minimizes spike communication on the interconnect and therefore, has lower communication energy. On the other hand, DecomposeSNN maximizes crossbar utilization and therefore, generates fewer clusters than SpiNeMap, resulting in lower spike energy. These results are consistent with the reported results in the two corresponding publications.

To highlight the importance of neuron and synapse placement within each crossbar (see our motivation example in Figure 6), Figure 9 shows the variation between minimum and maximum spike energy for SpiNeMap and DecomposeSNN considering 100 random placements of synapses of clusters to the crossbars of a neuromorphic hardware.

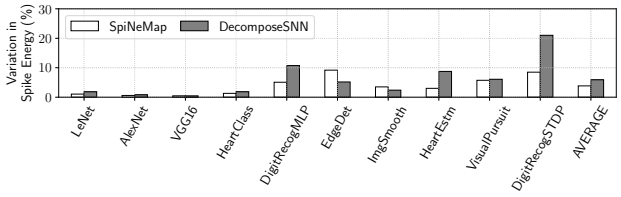


Figure 9: Variation in spike energy due to different synapse placement strategies on crossbars.

We observe that the spike energy of SpiNeMap varies by 3.8% and that of DecomposeSNN varies by 5.9% depending on how synapses are placed on crossbars.

We conclude that the system software of a neuromorphic hardware can play a key role in managing the energy consumption of neuromorphic computing.

5 ENERGY-AWARE SYSTEM SOFTWARE

Using the motivation presented in Section 4.3, we now present our energy-aware system software to map machine learning applications to neuromorphic hardware.

Figure 10 shows a neuromorphic system comprising of the application layer, the system software layer, and the hardware layer. The application layer at the top consists of the user space to run machine learning applications. In this illustration we show the execution of AlexNet for ImageNet classification. The hardware layer at the bottom consists of the neuromorphic hardware such as TrueNorth [32], Loihi [30], and DYNAPs [50]. At the middle is the system software layer, which interacts with both the application and hardware layers. The system software performs energy optimization using the iterative approach shown to the right.

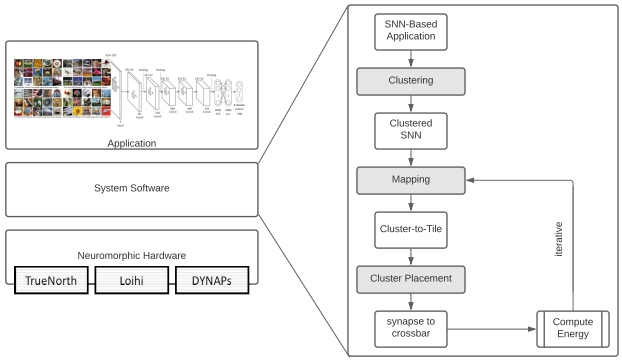


Figure 10: Our energy-aware system software.

The workflow of the system software involves clustering a machine learning application to generate clustered SNN graph. Next, the clusters are mapped to the tiles of the hardware using a mapping approach. Finally, the clusters are placed to crossbars using the placement step. Although the clustering step could potentially be incorporated inside the iterative loop, we placed it outside to limit the complexity of the design space exploration. In fact, clustering of applications is an NP-hard problem as shown in SpiNeMap [9]. Our clustering approach uses the graph partitioning algorithm of SpiNeMap, minimizing 1) inter-cluster communication (similar to SpiNeMap, and 2) maximizing cluster utilization (similar to DecomposeSNN).

We now describe the iterative approach of energy minimization starting from a clustered SNN using Algorithm 1. At the heart of this algorithm is the `CalculateEnergy` function, which calculates the total energy consumption using Equations 6 and 10. The `AssignCluster` function is a greedy heuristic to place a cluster to a crossbar. For this purpose, we first sort (in descending order) the synapses of a cluster in terms of their number of spikes. Next, the synapses are allocated to the crossbars, ensuring that the most activated one is placed towards the top right corner, where the spike current is lower.

Algorithm 1: Place neuron and synapse to neuromorphic hardware, minimizing the energy consumption.

Input: G_{CSNS}, G_{NH}
Output: M
1 **for** i *in* $MaxIter$ **do**
2 M_{init} = allocate clusters to tiles randomly;
3 AssignCluster();
4 E_{init} = `CalculateEnergy`(M_{init});
5 **for** $C_x, C_y \in C$ **do**
6 $M = M_{init}$;
7 Find T_i, T_j such that $m_{x,i} = m_{y,j} = 1$ /* Find the tiles to which C_x and C_y are mapped. */
8 Change M to set $m_{x,j} = m_{y,i} = 1$ and $m_{x,i} = m_{y,j} = 0$ /* Swap the tiles of C_x and C_y. */
9 AssignCluster();
10 E = `CalculateEnergy`(M) /* Calculate energy of the new mapping. */
11 **if** $E < E_{init}$ **then**
12 $M_{init} = M$ and $M_{min} = M$ /* If energy reduces then retain the new mapping. */
13 **end**
14 **end**
15 **end**
16 Return M_{min}

Algorithm 1 proceeds as follows. First, the clusters are randomly allocated to tiles (line 2). Next, the energy consumption is computed after assigning the synapses to the crossbars (lines 2-3). Then, for every cluster pair, the algorithm swaps their tile and compute the new energy (lines 6-10). If the energy improves, then the swapping is retained and the algorithm proceeds to the next cluster pair (lines 11-13). The algorithm is iterated for $MaxIter$ iterations, each time starting from a new random allocation of clusters (lines 1-15). The minimum energy mapping is returned.

In this algorithm, $MaxIter$ is a user-defined parameter and is used to explore the trade-offs between mapping time and the solution quality (see Section 6).

6 EVALUATION

We evaluated 10 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 summarized in Table 1. We simulate these applications on an in-house cycle-accurate neuromorphic hardware simulator. We model the DYNAPs neuromorphic hardware [50] with the following configurations.

Table 1: Evaluated applications.

Class	Applications	Synapses	Neurons	Topology	Accuracy
CNN	LeNet [46]	282,936	20,602	CNN	85.1%
	AlexNet [44]	38,730,222	230,443	CNN	90.7%
	VGG16 [60]	99,080,704	554,059	CNN	69.8 %
	HeartClass [4]	1,049,249	153,730	CNN	63.7%
MLP	DigitRecogMLP	79,400	884	FeedForward (784, 100, 10)	91.6%
	EdgeDet [15]	114,057	6,120	FeedForward (4096, 1024, 1024, 1024)	100%
	ImgSmooth [15]	9,025	4,096	FeedForward (4096, 1024)	100%
RNN	HeartEstm [16]	66,406	166	Recurrent Reservoir	100%
	VisualPursuit [43]	163,880	205	Recurrent Reservoir	47.3%
	DigitRecogSTDP [33]	11,442	567	Recurrent Reservoir	83.6%

- A tiled array of 4 tiles, each with a 128x128 crossbar. There are 65,536 synapses 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 cell implementing multi-bit synapse.

Table 2 reports the hardware parameters of DYNAP-SE.

Table 2: Major simulation parameters extracted from [50] and extrapolated for PCM technology.

Neuron technology	65nm CMOS
Synapse technology	PCM
Supply voltage	1.0V
E_{neuron}	50pJ at 30Hz spike frequency
$e_{switch} + 2 * e_{wire}$	147pJ
Switch bandwidth	1.8G. Events/s

6.1 Energy Consumption

Figure 11 reports the total energy consumption for each application for the evaluated approaches normalized to SpiNeMap. We make the following two observations.

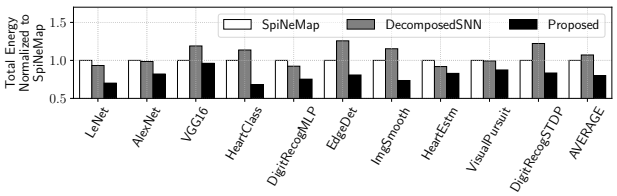


Figure 11: Total energy normalized to SpiNeMap.

First, between SpiNeMap and DecomposeSNN, the energy consumption of SpiNeMap is lower than DecomposeSNN for applications such as VGG16, HeartClass, and EdgeDet. For these applications, there is a significant number of spikes communicated on the interconnect. Therefore, by reducing inter-cluster communication, SpiNeMap also reduces energy consumption. For other applications such as LeNet and HeartEstm, the number of inter-cluster spikes is less, so DecomposeSNN, which maximizes cluster utilization improves on the total energy consumption. Second, compared to both these approaches, the proposed approach results in 20% lower energy than SpiNeMap and 24% lower energy than DecomposeSNN. The improvement over both these approaches is because the proposed approach explicitly incorporates both the spike and communication energy in finding a suitable mapping of clusters to the hardware.

To give further insight, Figure 12 reports the total energy, distributed into spike energy (E_{spk}) and communication energy (E_{comm}). We observe that communication energy constitute an average 58.8% of the total energy consumption and it depends on the total spikes generated in a workload.

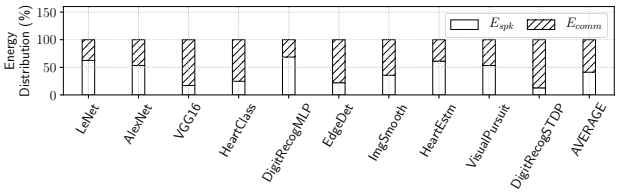


Figure 12: Total energy distributed into E_{spk} and E_{comm}.

6.2 Spike Latency and Model Performance

Figure 13 plots the spike latency for each evaluated applications for the evaluated approaches normalized to SpiNeMap. We make the following two observations.

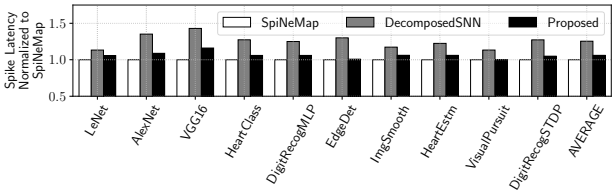


Figure 13: Spike latency normalized to SpiNeMap.

First, SpiNeMap has the lowest latency. This is because SpiNeMap minimizes spike congestion on the interconnect, which reduces spike delay. DecomposeSNN has the highest latency because of its optimization objective, which is to maximize utilization. Second, the proposed approach minimizes spike communication to reduce the communication energy. This also reduces the spike latency. Overall, the spike latency of the proposed approach is only 6% higher than SpiNeMap but 15% lower than DecomposeSNN. As described in [9], spike latency can lead to a loss in model performance. Therefore, by keeping the spike latency reasonably close to SpiNeMap, the performance of the proposed approach is similar to that reported in column 6 of Table 1.

6.3 Architecture Evaluation

Figure 14 report the energy consumption for three different hardware configurations – with 128x128, 256x256, and 512x512 crossbars, for the evaluated applications. Results are normalized to the energy consumption with 128x128 crossbars. We observe that the energy consumption is 13% and 28% lower when using 256x256 and 512x512 crossbars compared to using 128x128 crossbars. Energy consumption reduces when using larger crossbars because of the reduction in the total number of spikes on the interconnect.

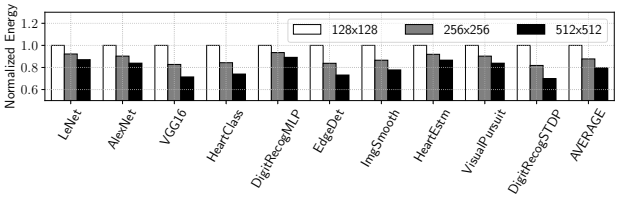


Figure 14: Energy consumption for different hardware configurations.

6.4 Solution Quality Evaluation

Table 3 reports the mapping time and the normalized energy obtained for three different settings of the parameter $MaxIter$. We observe that as $MaxIter$ is increased, the energy consumption reduces for all applications. This is because with the increase in the number of iterations, Algorithm 1 is able to find a better solution. However, the mapping time also increases. Finally, we observe that increasing $MaxIter$ from 100 to 1000 results in a significant increase in mapping time with a minimal improvement of the total energy. We conclude that setting $MaxIter = 100$ gives the best trade-off in terms of mapping time and the solution quality. Users can use this $MaxIter$ parameter to set a limit on the time of their algorithm by analyzing the complexity of their model against the ones we evaluate (see Table 1).

Table 3: Mapping time and solution trade-off.

Application	MaxIter = 10		MaxIter = 100		MaxIter = 1000	
	Mapping Time (sec)	Total Energy	Mapping Time (sec)	Total Energy	Mapping Time (sec)	Total Energy
LeNet	75	1.13	321	1.0	2700	0.99
AlexNet	200	1.1	2189	1.0	12008	1.0
VGG16	241	1.06	2989	1.0	34300	1.0
HeartClass	116	1.16	1008	1.0	10178	1.03
DigitRecogMLP	10	1.17	160	1.0	1600	0.97
EdgeDet	25	1.15	200	1.0	1898	0.98
ImgSmooth	50	1.11	400	1.0	1940	0.99
HeartEstm	15	1.08	156	1.0	1344	0.97
VisualPursuit	30	1.0	324	1.0	3003	1.0
DigitRecogSTDP	15	1.07	164	1.0	1615	0.9

7 CONCLUSION

In this work, we provide a comprehensive energy model for executing machine learning applications on neuromorphic hardware. Using this model we show that the system software for neuromorphic hardware plays a critical role in energy management of neuromorphic computing by controlling 1) how an SNN model is partitioned into clusters, 2) how the clusters are mapped to the neurosynaptic cores of the hardware, and 3) how the neurons and synapses of a cluster are placed inside each core. We then propose a heuristic to perform energy-aware application mapping to neuromorphic hardware, lowering the overall energy consumption. Using this heuristic, we show that the energy consumption can be reduced by an average 20% compared to a state-of-the-art for typical machine learning applications.

8 ACKNOWLEDGMENTS

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

REFERENCES

[1] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz *et al.*, "Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores," in *IJCNN*, 2013.
[2] A. Balaji and A. Das, "Compiling spiking neural networks to mitigate neuromorphic hardware constraints"," in *IGSC Workshops*, 2020.
[3] A. Balaji and A. Das, "A framework for the analysis of throughput-constraints of snns on neuromorphic hardware," in *ISVLSI*, 2019.
[4] A. Balaji, F. Corradi, A. Das, S. Pande, S. Schaafsma *et al.*, "Power-accuracy trade-offs for heartbeat classification on neural networks hardware," *Journal of Low Power Electronics (JOLPE)*, 2018.
[5] 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.
[6] A. Balaji, S. Ullah, A. Das, and A. Kumar, "Design methodology for embedded approximate artificial neural networks," in *GLSVLSI*, 2019.
[7] 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.
[8] A. Balaji, P. Adiraju, H. J. Kashyap, A. Das, J. L. Krichmar *et al.*, "PyCARL: A PyNN interface for hardware-software co-simulation of spiking neural network," in *IJCNN*, 2020.
[9] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'anna *et al.*, "Mapping spiking neural networks to neuromorphic hardware," *TVLSI*, 2020.
[10] A. Balaji, T. Marty, A. Das, and F. Catthoor, "Run-time mapping of spiking neural networks to neuromorphic hardware," *JSPS*, 2020.
[11] 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.
[12] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, 2014.
[13] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, 2017.
[14] F. Catthoor, S. Mitra, A. Das, and S. Schaafsma, "Very large-scale neuromorphic systems for biological signal processing," in *CMOS Circuits for Biological Sensing and Processing*, 2018.
[15] T. Chou, H. Kashyap, J. Xing, S. Listopad, E. Rounds *et al.*, "CARLsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters," in *IJCNN*, 2018.
[16] 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.
[17] A. Das and A. Kumar, "Fault-aware task re-mapping for throughput constrained multimedia applications on NoC-based MPSoCs," in *RSP*, 2012.
[18] A. Das and A. Kumar, "Dataflow-based mapping of spiking neural networks on neuromorphic hardware," in *GLSVLSI*, 2018.
[19] A. Das, A. Kumar, and B. Veeravalli, "Fault-tolerant network interface for spatial division multiplexing based Network-on-Chip," in *ReCoSoC*, 2012.
[20] A. Das, A. Kumar, and B. Veeravalli, "Communication and migration energy aware design space exploration for multicore systems with intermittent faults," in *DATE*, 2013.
[21] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," in *DATE*, 2013.
[22] A. Das, A. K. Singh, and A. Kumar, "Energy-aware dynamic reconfiguration of communication-centric applications for reliable MPSoCs," in *ReCoSoC*, 2013.
[23] A. Das, A. Kumar, and B. Veeravalli, "Communication and migration energy aware task mapping for reliable multiprocessor systems," *FGCS*, 2014.
[24] A. Das, A. Kumar, and B. Veeravalli, "Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs," in *DATE*, 2014.
[25] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar *et al.*, "Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems," in *DAC*, 2014.
[26] A. Das, A. Kumar, and B. Veeravalli, "Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems," *TPDS*, 2015.
[27] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems," *TECS*, 2016.
[28] 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.
[29] A. Das, Y. Wu, K. Huynh, F. Dell'Anna, F. Catthoor *et al.*, "Mapping of local and global synapses on spiking neuromorphic hardware," in *DATE*, 2018.
[30] M. Davies, N. Srinivasa, T. H. Lin *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, 2018.
[31] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller *et al.*, "PyNN: a common interface for neuronal network simulators," *Frontiers in Neuroinformatics*, 2009.
[32] M. V. Debole, B. Taba, A. Amir *et al.*, "TrueNorth: Accelerating from zero to 64 million neurons in 10 years," *Computer*, 2019.

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

[34] J. M. Eppler, M. Helias, E. Muller, M. Diesmann, and M.-O. Gewaltig, "Pynest: a convenient interface to the nest simulator," *Frontiers in Neuroinformatics*, 2009.

[35] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *TBCAS*, 2018.

[36] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, 2014.

[37] F. Galluppi, X. Lagorce, E. Stromatias, M. Pfeiffer, L. A. Plana *et al.*, "A framework for plasticity implementation on the SpiNNaker neural architecture," *Frontiers in Neuroscience*, 2015.

[38] S. Ghosh-Dastidar and H. Adeli, "Improved spiking neural networks for EEG classification and epilepsy and seizure detection," *Integrated Computer-Aided Engineering*, 2007.

[39] D. F. Goodman and R. Brette, "The brian simulator," *Frontiers in Neuroscience*, 2009.

[40] M. L. Hines and N. T. Carnevale, "The NEURON simulation environment," *Neural Computation*, 1997.

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

[42] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang *et al.*, "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints," in *MICRO*, 2016.

[43] 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.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems (NeurIPS)*, 2012.

[45] S. Kundu, K. Basu, M. Sadi, T. Titirsha, S. Song *et al.*, "Reliability analysis for ML/AI hardware," in *VTS*, 2021.

[46] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, 2015.

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

[48] 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.

[49] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, 1990.

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

[51] E. J. Moyer and A. Das, "Machine learning applications to DNA subsequence and restriction site analysis," in *SPMB*, 2020.

[52] H. Mulaosmanovic, J. Ocker, S. Müller, M. Noack, J. Müller *et al.*, "Novel ferroelectric FET based synapse for neuromorphic systems," in *VLSIT*, 2017.

[53] A. Natarajan and J. Hasler, "Hodgkin–huxley neuron and FPAA dynamics," *TBCAS*, 2018.

[54] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza *et al.*, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, 2018.

[55] A. Rubino, C. Livanelioglu, N. Qiao, M. Payvand, and G. Indiveri, "Ultra-low-power FDSOI neural circuits for extreme-edge neuromorphic intelligence," *TCAS I: Regular Papers*, 2020.

[56] J. Schemmel, A. Grübl, S. Hartmann, A. Kononov, C. Mayr *et al.*, "Live demonstration: A scaled-down version of the brainscales wafer-scale neuromorphic system," in *ISCAS*, 2012.

[57] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in Neuroscience*, 2019.

[58] R. A. Shafik, A. Das, S. Yang, G. Merrett, and B. M. Al-Hashimi, "Adaptive energy minimization of openmp parallel applications on many-core systems," in *PARMA-DITAM*, 2015.

[59] L. Shi, J. Pei, N. Deng, D. Wang, L. Deng *et al.*, "Development of a neuromorphic computing system," in *IEDM*, 2015.

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.

[61] S. Song and A. Das, "Design methodologies for reliable and energy-efficient PCM systems," in *IGSC Workshops*, 2020.

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

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

[64] S. Song, A. Balaji, A. Das, N. Kandasamy, and J. Shackleford, "Compiling spiking neural networks to neuromorphic hardware," in *LCTES*, 2020.

[65] S. Song, A. Das, and N. Kandasamy, "Exploiting inter- and intra-memory asymmetries for data mapping in hybrid tiered-memories," in *ISMM*, 2020.

[66] S. Song, A. Das, and N. Kandasamy, "Improving dependability of neuromorphic computing with non-volatile memory," in *EDCC*, 2020.

[67] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Improving phase change memory performance with data content aware access," in *ISMM*, 2020.

[68] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Aging-aware request scheduling for non-volatile main memory," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021.

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

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

[71] T. Titirsha, S. Song, A. Das, J. Krichmar, N. Dutt *et al.*, "Endurance-aware mapping of spiking neural networks to neuromorphic hardware," *TPDS*, 2021.

[72] 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.

[73] S. Woo, J. Cho, D. Lim, Y.-S. Park, K. Cho *et al.*, "Implementation and characterization of an integrate-and-fire neuron circuit using a silicon nanowire feedback field-effect transistor," *TED*, 2020.

[74] Z. Yang, Y. Huang, J. Zhu, and T. T. Ye, "Analog circuit implementation of LIF and STDP models for spiking neural networks," in *GLSVLSI*, 2020.