NeuroXplorer 1.0: An Extensible Framework for Architectural Exploration with Spiking Neural Networks

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

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

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

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

Jeffrey L. Krichmar
jkrichma@uci.edu
University of California, Irvine
Irvine, CA, USA

Nikil Dutt
dutt@ics.uci.edu
University of California, Irvine
Irvine, CA, USA

James Shackleford
jas64@drexel.edu
Drexel University
Philadelphia, PA, USA

Nagarajan Kandasamy
nk78@drexel.edu
Drexel University
Philadelphia, PA, USA

Francky Catthoor
Francky.Catthoor@imec.be
Imec
Leuven, Belgium

ABSTRACT

Recently, both industry and academia have proposed many different neuromorphic architectures to execute applications that are designed with Spiking Neural Network (SNN). Consequently, there is a growing need for an extensible simulation framework that can perform architectural explorations with SNNs, including both platform-based design of today's hardware, and hardware-software co-design and design-technology co-optimization of the future. We present NeuroXplorer, a fast and extensible framework that is based on a generalized template for modeling a neuromorphic architecture that can be infused with the specific details of a given hardware and/or technology. NeuroXplorer can perform both low-level cycle-accurate architectural simulations and high-level analysis with data-flow abstractions. NeuroXplorer's optimization engine can incorporate hardware-oriented metrics such as energy, throughput, and latency, as well as SNN-oriented metrics such as inter-spike interval distortion and spike disorder, which directly impact SNN performance. We demonstrate the architectural exploration capabilities of NeuroXplorer through case studies with many state-of-the-art machine learning models.

CCS CONCEPTS

• **Hardware → Neural systems**; **Emerging languages and compilers**; **Emerging tools and methodologies**.

KEYWORDS

Spiking Neural Networks (SNN), Neuromorphic Computing, Non Volatile Memory (NVM), Platform-Based Design, Hardware-Software Co-Design, Design-Technology Co-Optimization

ACM Reference Format:
Adarsha Balaji, Shihao Song, Twisha Titirsha, Anup Das, Jeffrey L. 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 *International Conference on Neuromorphic Systems 2021 (ICONS 2021), July 27–29, 2021, Knoxville, TN, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3477145.3477156

ICONS 2021, July 27–29, 2021, Knoxville, TN, USA

ACM ISBN 978-1-4503-8691-3/21/07…$15.00
https://doi.org/10.1145/3477145.3477156

1 INTRODUCTION

The term neuromorphic computing was coined in the 90s to describe integrated circuits that mimic the neuro-biological architecture of the central nervous system [42]. These circuits employ variants of integrate-and-fire (I&F) neurons [28] as computational units and analog weights as synaptic storage. I&F neurons use spikes to encode information, where each spike is a voltage or current pulse in the physical world, typically of ms duration [40]. Recently, both industry and academia have proposed many different neuromorphic platforms to execute applications that are designed with Spiking Neural Network (SNN). Examples of such platforms include SpiNNaker [27], Neurogrid [12], TrueNorth [23], DYNAPs [44], Tianji [54], Loihi [20], and ODIN [26], among others [52].

To cope with the growing complexity of neuromorphic systems[1], challenges in integrating emerging non-volatile memory technologies, and faster time-to-market pressure, efficient design methodologies are needed [10]. We highlight the following three key concepts that are likely to address the design issues postulated above.

- **Platform-based Design:** In this design methodology, a hardware platform is abstracted from its system software using the Application Programming Interface (API), making

[1]The complexity of a neuromorphic system can be expressed in terms of the number of neurons and synapses, and their interconnection.

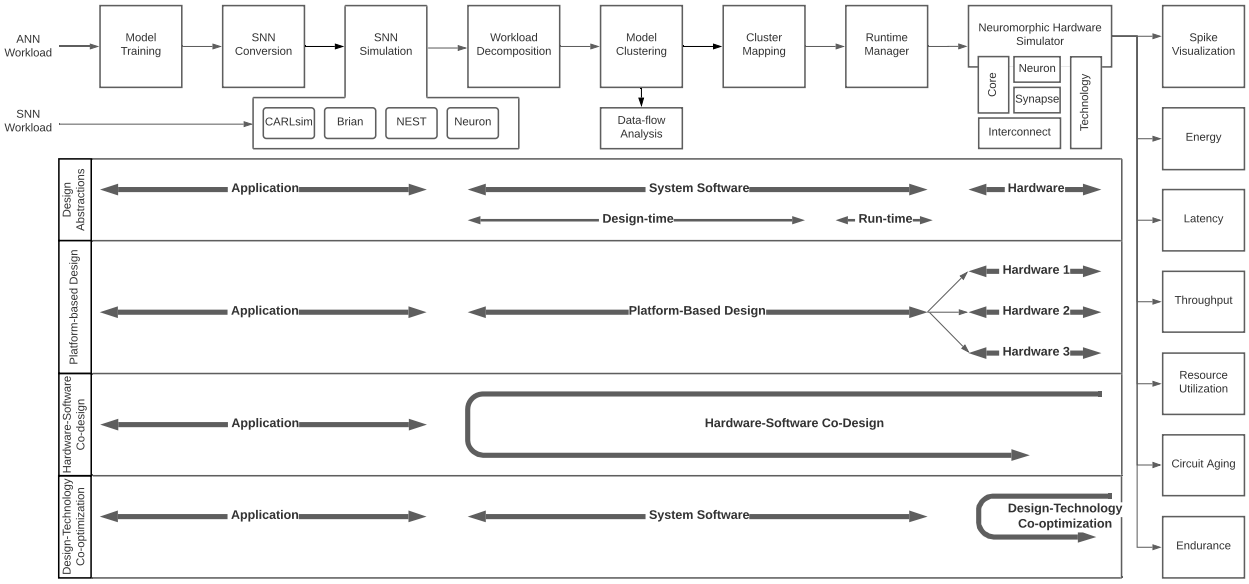


Figure 1: High-level overview of NeuroXplorer. The framework supports the following functionalities: 1) application quality exploration, 2) platform-based design, 3) hardware-software co-design, and 4) design-technology co-optimization.

the hardware and software development orthogonal to allow more effective exploration of alternative solutions [35]. Platform-based design methodology facilitates the reuse of the system software for many different hardware platforms.

- **Hardware-Software Co-design:** In this design methodology, a hardware platform and its system software are concurrently designed to exploit their synergism in order to achieve system-level design objectives [22]. The system software in this case is tailored for the hardware platform.
- **Design-Technology Co-optimization:** In this design methodology, system-level design metrics are applied to explore the choices in hardware design and process technology to enable scaling at advanced technology nodes [75].

Consequently, there is a growing need for an extensible hardware simulator and an application mapper that can perform architectural explorations with SNNs, including platform-based design, hardware-software co-design, and design-technology co-optimization. We present **NeuroXplorer**, a fast and extensible framework that is based on a *generalized template* for modeling a neuromorphic architecture that can be infused with the specific details of a given hardware and/or technology.

NeuroXplorer is released under the permissive MIT open license and it provides a user with the following high-level functionalities, which we will elaborate in the following sections.

- A design optimization engine that can incorporate hardware design metrics such as energy, latency, throughput, and reliability, as well as SNN-oriented metrics such as inter-spike interval distortion and spike disorder.
- A generalized and optimized system software framework, facilitating mapping of SNN-based applications to different neuromorphic hardware platforms.
- A cycle-accurate model of neuromorphic hardware utilizing a generalized template, which can be configured with hardware- and technology-specific details from industrial and academic manufacturers of neuromorphic systems.
- A design space exploration framework using data flow abstractions to represent machine learning models for execution on neuromorphic hardware, allowing estimation of key system-level performance metrics early in the system design stage.
- A framework to analyze different technological alternatives for neuron and synapse circuits, and the impact of scaling in neuromorphic hardware, facilitating optimization of key system-level design metrics.

In addition to these architecture-centric functionalities, NeuroXplorer also facilitates functional simulations via SNN simulators such as CARLsim [15], Brian [29], NEST [25], and Neuron [32], supporting different degrees of neuro-biological details and learning modalities. Thus, NeuroXplorer allows to explore the design-space of application performance alongside architecture development.

NeuroXplorer is developed over a period of five years and is supported by three National Science Foundation research grants and one Department of Energy grant from the United States, and one Horizon 2020 research grant from Europe.

2 NEUROXPLORER: HIGH-LEVEL DESIGN

Figure 1 illustrates the key components of NeuroXplorer. At a high-level, NeuroXplorer supports three layers of abstraction – the **application** layer, the **system software** layer, and the **hardware** layer, similar to the abstractions in a classical von-Neumann computing system. Internally, the system software layer is divided into a **design-time** or **compile-time** methodology, where a machine learning model is converted into an intermediate form for mapping

to a specific neuromorphic hardware, and a **run-time** methodology, which allocates hardware resources to admit and execute the model on the hardware. NeuroXplorer can work with both Artificial Neural Networks (ANNs) and biology-inspired Spiking Neural Networks (SNNs). NeuroXplorer interfaces with ANN workloads that are specified in high-level frameworks such as Keras with TensorFlow backend [1, 30] and PyTorch [48]. To map an ANN workload to an event-driven neuromorphic hardware, the workload is first converted to an SNN using the **SNN Conversion** unit and then, the SNN is simulated using the **SNN Simulation** unit of NeuroXplorer.

SNN workloads can be specified in PyNN [21], which is a Python interface to many SNN simulators such as CARLsim [15], Brian [29], NEST [25], and Neuron [32]. These simulators model neural functions at various levels of detail and therefore have different requirements for computational resources. User can also specify an SNN model directly using these simulators. NeuroXplorer allows exploration of application quality using these simulators.

NeuroXplorer incorporates the spike timing information obtained from simulating an SNN model with representative training data. Such information is used to map the model to the neuromorphic hardware using the system-software framework, which consists of **Workload Decomposition**, **Model Clustering**, **Cluster Mapping**, and **Runtime Management** units.

Without loss of generality, we describe NeuroXplorer where ANN workloads are specified using Keras and SNN workloads using a combination of PyNN and PyCARL [2], a Python wrapper for SNN simulations using CARLsim. Additionally, we use our previously proposed SNN converter [3] for SNN conversion of ANN workloads in order to map them to hardware. NeuroXplorer can be trivially **extended** to work with other SNN simulation tools such as GeNN [74] and Spyketorch [46], and with other SNN conversion approaches such as [43, 50, 51].

Figure 1 illustrates the three design methodologies supported by NeuroXplorer – 1) platform-based design, 2) hardware-software co-design, and 3) design-technology co-optimization. We have used NeuroXplorer to optimize for system-level design metrics, including energy [6, 19, 70], latency [4, 17], throughput [57, 58], resource utilization [5, 9, 56], circuit aging [8, 37, 59, 62], inferen lifetime [67], and endurance [68, 69, 71].

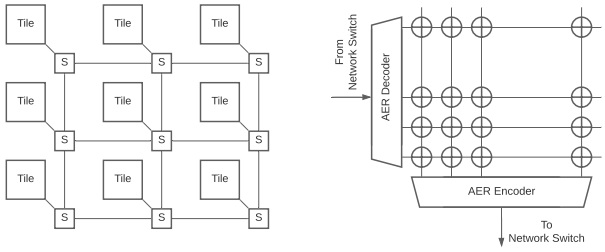
3 DETAILED DESIGN OF SYSTEM SOFTWARE

We now detail the system software of NeuroXplorer.

3.1 Platform Description

We consider a tile-based neuromorphic hardware as shown in Figure 2a. Each tile consists of a neuromorphic core, which can accommodate a certain number of neurons and synapses. A common approach to implementing a neuromorphic core is one where the synaptic cells are organized in a two-dimensional grid, known as crossbar. We illustrate a crossbar in Figure 2b.[2]

Typically, system designers limit the size of a crossbar to reduce energy consumption [3] and mitigate the high parasitic voltage drops



(a) A tile-based neuromorphic hardware [14]. A tile communicates spikes via the network switches (S) using a shared interconnect such as Networks-on-chip (NoC) [38] and Segmented Bus [11].

(b) A crossbar architecture. Spikes are encoded into Address Event Representation (AER). A crossbar peripheral circuitry consists of AER encoder and decoder.

Figure 2: A running example of a tile-based neuromorphic hardware. Each tile contains a neuromorphic core, which in its simplest form can be a crossbar.

within a crossbar (see Figure 12). Therefore, a large machine learning model must be partitioned into *local synapses*, those that map within the crossbar of a tile, and *global synapses*, those that map on the shared interconnect [19]. To effectively address this partitioning, NeuroXplorer's system software performs the following four key functionalities to map a machine learning workload to the hardware: workload decomposition, model clustering, cluster mapping, and runtime. We now describe these functions.

3.2 Workload Decomposition

We note that each $N \times N$ crossbar in a tile can accommodate up to N pre-synaptic connections per post-synaptic neuron, with typical value of N set between 128 (in DYNAPs) and 256 (in TrueNorth). Figure 3 illustrates an example of mapping a) one 4-input, b) one 3-input, and c) two 2-input neurons on a 4×4 crossbar. Unfortunately, neurons with more than 4 pre-synaptic connections per post-synaptic neuron cannot be mapped to this crossbar.

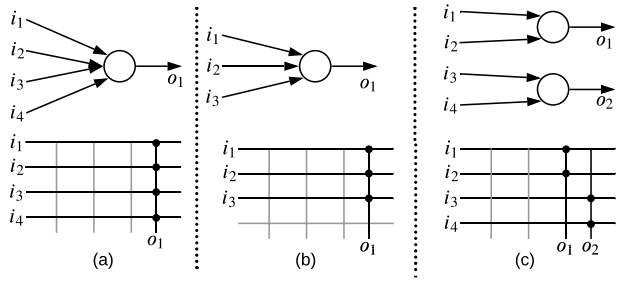


Figure 3: Example mapping of a) one 4-input, b) one 3-input, and c) two 2-input neurons to a 4×4 crossbar.

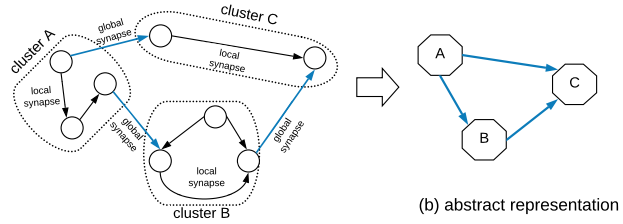
We take the example architecture of DYNAPs, where each crossbar can accommodate a maximum of 128 pre-synaptic connections. In many complex machine learning models such as LeNet, AlexNet, VGG, ResNet, and DenseNet, the number of pre-synaptic connections per post-synaptic neuron is much higher than what a crossbar in DYNAPs can accommodate.

To address this resource limitation, we have previously proposed workload decomposition, which exploits the firing principle of LIF

[2] Although NeuroXplorer provides a generalized template for crossbar-based neuromorphic tiles, NeuroXplorer can be easily extended to support many different types of processing elements such as [39, 49].

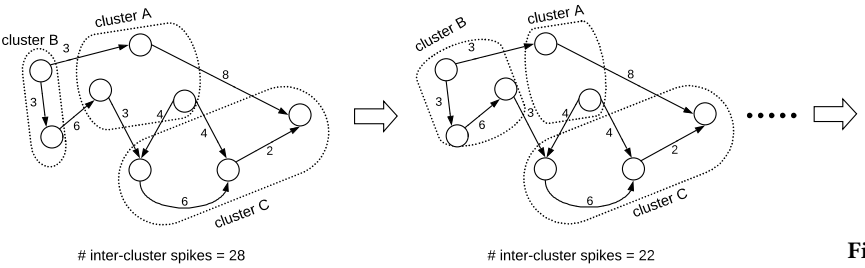
[3] Energy consumption in a crossbar scales proportional to M^2, where M is the input/output dimension of a crossbar.

neurons, decomposing each neuron with many pre-synaptic connections into a sequence of homogeneous fanin-of-two (FIT) neural units [9]. Figure 4 illustrates the spatial decomposition using a small example of a 3-input neuron shown in Figure 4(a). We consider the mapping of this neuron to 2x2 crossbars. Since each crossbar can accommodate a maximum of two pre-synaptic connections per neuron, the example 3-input neuron cannot be mapped to the crossbar directly. The most common solution is to eliminate a synaptic connection, which may lead to accuracy loss. Figure 4(b) illustrates the decomposition mechanism, where the 3-input neuron is implemented using two FIT neural units connected in sequence as shown in Figure 4(b). Each FIT unit is similar to a 2-input neuron and it exploits the leaky integrate behavior in hardware to maintain



(a) SNN clusters with local and global synapse

(b) abstract representation



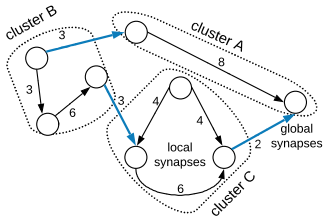
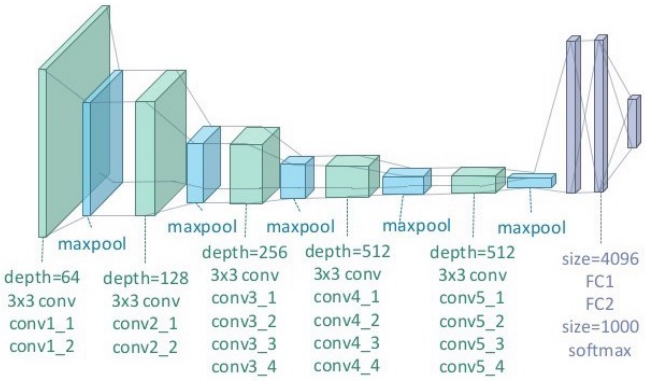
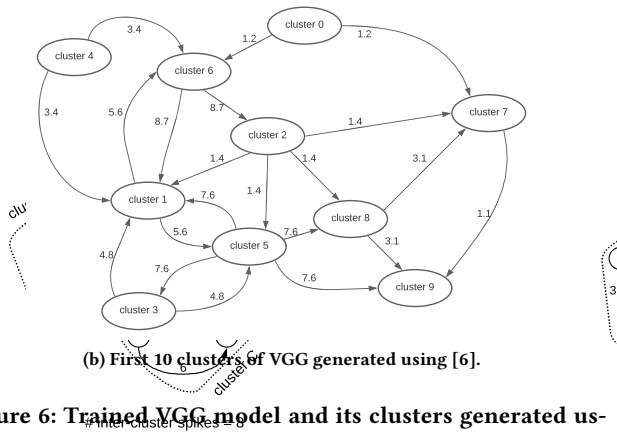


Figure 5: An SNN partitioned into three clusters.

The SNN partitioning problem is essentially a graph partitioning problem, which is NP-complete. Therefore, heuristics are typically used to find solutions. NeuroXplorer currently supports two heuristics – Particle Swarm Optimization (PSO) [33] and Kernighan-Lin Graph Partitioning algorithm [34]. NeuroXplorer uses these heuristics to minimize 1) the number of clusters (as in [9]), which reduces the hardware requirement, and 2) the number of inter-cluster spikes (as in [6, 19]), which reduces both energy and latency when the machine learning model is mapped to hardware. NeuroXplorer can be easily extended to use other heuristics such as Hill Climbing [53] and Simulated Annealing [72], as well as other optimization objectives such as application quality and hardware reliability.



(a) VGG Convolution Neural Network (CNN).



(b) First 10 clusters of VGG generated using [6].

Figure 6: Trained VGG model and its clusters generated using model partitioning tool such as SpiNeMap [6].

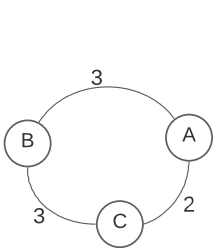
Figure 6a shows the architecture of VGG for CIFAR-10 classification. Figure 6b shows the first 10 clusters generated using SpiNeMap [6].[4] The figure illustrates the connections between these clusters, with the number on edge representing the average number of spikes communicated between the source and destination clusters when processing an image during inference. The inter-cluster links are the global synapses for mapping purposes.

3.4 Cluster Mapping

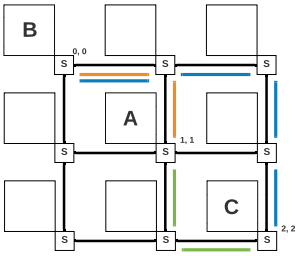
The cluster mapping step of NeuroXplorer is used to reserve computing resources of the hardware for a given machine learning

[4]SpiNeMap [6] generates 95,452 clusters from the VGG model trained on CIFAR-10 dataset. For simplicity, we illustrate only the first 10 clusters and their interconnection.

model and execute the model by placing its clusters onto the physical cores. Figure 7b illustrates the placement of a clustered SNN of Figure 7a to a neuromorphic hardware with 9 cores organized in a mesh architecture. The position of each core in the hardware is specified by a pair of Cartesian coordinates. In this example, cluster A is placed at coordinate (1,1), cluster B at (0,0), and cluster C at (2,2). All spikes between A and B, and between A and C are communicated via two interconnect segments and one hop, while all spikes between B and C are communicated via four interconnect segments and three hops. Clearly, the latency and energy on the shared interconnect depends on the placement of the clusters on the physical cores located in the Cartesian coordinate system.



(a) An example clustered SNN with three clusters – A, B, and C.



(b) An example placement of the three clusters to a mesh architecture. The three different colors (orange, blue, and green) indicate the the distance (in hops) spikes travel in the mesh architecture between source and destination clusters.

Figure 7: Example of mapping a clustered SNN on a mesh architecture.

NeuroXplorer uses an instance of PSO to optimize the placement of clusters of a machine learning model to the physical cores of the hardware, improving both latency and energy consumption. The placement solution of NeuroXplorer aims to place the clusters that communicate the most to nearby cores. NeuroXplorer can be extended to use other placement heuristics.

3.5 Runtime Manager

To illustrate the significance of a run-time manager, Figure 8 plots the spike firing rate of 100 randomly-selected neurons in AlexNet [36], a state-of-the-art CNN used for Imagenet classification. We report results for two randomly-selected training and test images.

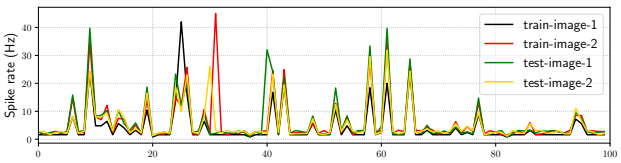


Figure 8: Spike rate of 100 randomly-selected neurons in AlexNet for 2 training images and 2 test images.

We observe that spike firing rates of neurons depend on the image presented to the AlexNet CNN. Therefore, energy and reliability improvement strategies based on design-time analysis with training examples may not be optimal when they are applied at run-time to process in-field data. Therefore, in addition to cluster placement when admitting a machine learning model to hardware, NeuroXplorer also supports monitoring key performance statistics collected from the hardware during model execution. Such statistics can uncover bottlenecks, allowing improving system-level metrics such as energy [7] and circuit aging [8, 66] through remapping of the neurons and synapses to the hardware.

4 DETAILED DESIGN OF NEUROMORPHIC HARDWARE SIMULATOR

Figure 9 shows the high-level overview of the proposed neuromorphic hardware simulator, which facilitates cycle-accurate simulation of the interconnect and the processing tiles. Each tile models 1) a processing element, which is a neuromorphic core, 2) a router for routing spike AER packets on the shared interconnect, 3) a local memory to store cluster parameters, 4) buffer space for spike packets, and 5) AER encoder and decoder. NeuroXplorer's hardware simulator can perform exploration with transistor technologies such as CMOS and FinFET that are used for the neurons and the peripheral circuitry in each tile, and Non-Volatile Memory (NVM) technologies such as Phase-Change Memory (PCM) [13], Oxide-Based Resistive Random Access Memory (OxRRAM) [41], Ferroelectric RAM (FeRAM) [47], and Spin-Transfer Torque Magnetic or Spin-Orbit-Torque RAM (STT/ SoT-MRAM) [73] used for synaptic weight storage.[5] We now describe the simulator.

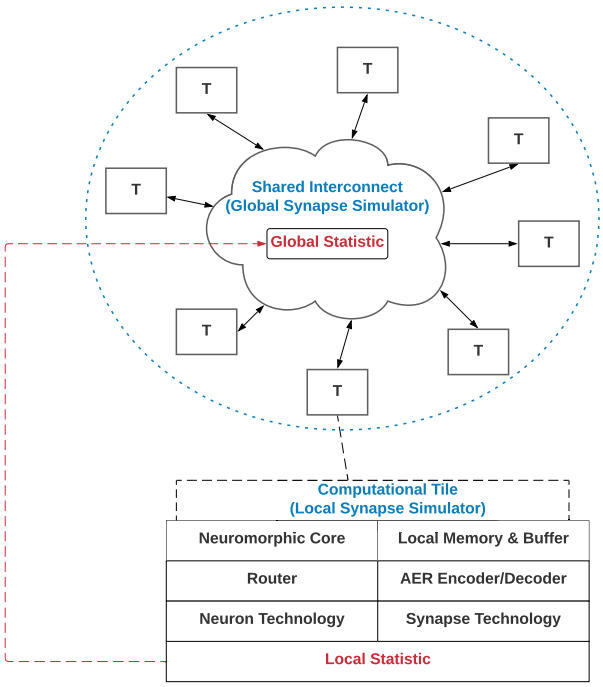


Figure 9: Architecture simulator of NeuroXplorer.

[5]Beside neuromorphic computing, NVMs are also used as main memory in conventional computers to improve performance and energy efficiency [60, 61, 63–65].

4.1 Cycle-Accurate Interconnect Simulator

Figure 10 illustrates the internal architecture of the interconnect (global synapse) simulator of NeuroXplorer in UML conventic Key components of this simulator are the following

- Spike Routing Strategy: This is the generalization class the following routing strategies: Dyad, Negative First, Nor Last, Odd Even, Table-based, West-First, and XY.
- Spike Traffic Model: This is the generalization class of t following traffic models: Random, Transpose Matrix, B Reversal, Butterfly, Shuffle, and Table-based.
- Configuration Manager: This is the generalization class f loading simulator parameters such as network topology, n work size, traffic type, routing strategy, and simulation tin

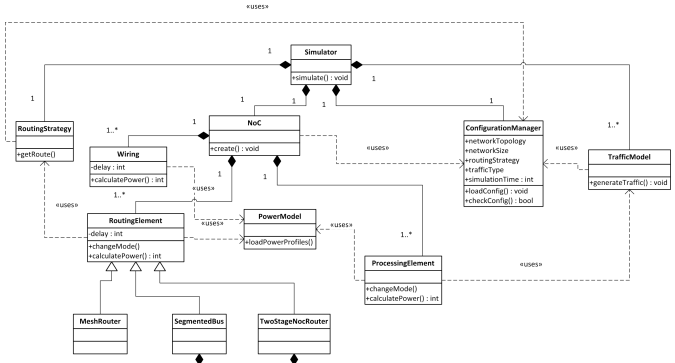


Figure 10: Class diagram of NeuroXplorer's hardware simulator using UML convention.

Figure 11 shows the capabilities of the hardware simulator of NeuroXplorer. For example, when changing the network topology, the user can select between the three interconnect types: Mesh, Segmented bus, and Two-stage NoC. The user can also input spike traffic generated from the application-level simulator at the frontend of NeuroXplorer to run hardware network simulation.

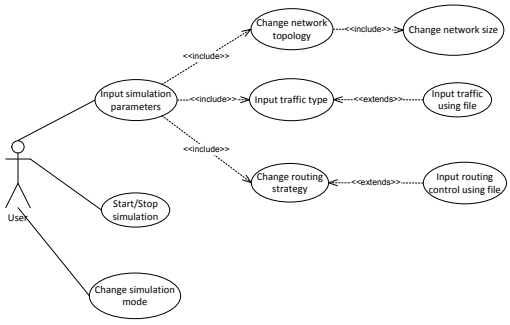


Figure 11: Use-case diagram of the hardware simulator of NeuroXplorer.

4.2 Cycle-Accurate Tile Simulator

On the computing tile front, NeuroXplorer supports detailed model of crossbars with PCM and OxRRAM-based synaptic cells. Figure 12 shows the detailed circuit model of a crossbar in NeuroXplorer with all of its parasitic components. Such parasitic components cause variable delays on the current paths inside the crossbar. For simplicity, we have only shown the current on the shortest and the longest paths in the crossbar, where the length of a current

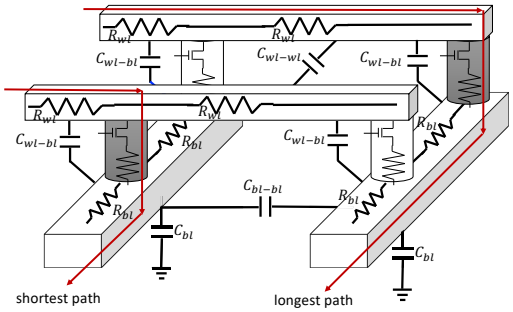


Figure 12: Detailed model of a crossbar in NeuroXplorer.

Table 1 shows the template for specifying the parasitic components in NeuroXplorer for a specific technology node.

Table 1: Generalized template for specifying the parasitic components of a crossbar in NeuroXplorer.

R_{wl}	unit wordline resistance
R_{bl}	unit bitline resistance
C_{wl}	unit wordline capacitance
C_{bl}	unit bitline capacitance
C_{wl-wl}	unit wordline to wordline capacitance
C_{wl-bl}	unit wordline to bitline capacitance
C_{bl-bl}	unit bitline to bitline capacitance

On the technology front, we briefly discuss the OxRRAM technology, as an instance of a technology that can be used for the synaptic cell. An RRAM cell is composed of an insulating film sandwiched between conducting electrodes forming a metal-insulator-metal (MIM) structure (see Figure 13). Recently, 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. 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 13 shows the High-Resistance State (HRS) and the Low-Resistance State (LRS) of an RRAM cell. In NeuroXplorer, the RRAM cell can also be programmed into intermediate low-resistance states, allowing its multilevel operations. For instance, to implement two bits per synapse we can program the RRAM cell for one HRS and three LRS states.

NeuroXplorer also supports implementing many variants of Integrate & Fire (I&F) neuron. Table 2 provides the template for specifying the parameters for a neuron and synaptic cell in a crossbar.

The generalized template of Tables 1 and 2 can be infused with the specific details of a present-day neuromorphic chip and evaluate the impact of technology scaling on system-level metrics such as energy, latency, and reliability. We now present the evaluation of NeuroXplorer by configuring it with the parameters of the DYNAPs

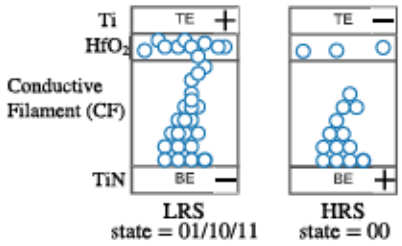


Figure 13: Operation of an RRAM cell with the HfO_2 layer sandwiched between the metals Ti (top electrode) and TiN (bottom electrode). The left subfigure shows the formation of LRS states with the formation of conducting filament (CF). This represents logic states 01, 10, and 11. The right subfigure shows the depletion of CF on application of a negative voltage on the TE. This represents the HRS state or logic 00.

Table 2: Generalized template for specifying the parameters of a neuron and synaptic cell in NeuroXplorer.

Neuron technology	CMOS or FinFET
Technology node	65nm, 45nm, 32nm, and 16nm
Supply voltage	1.0V
Energy per spike	50pJ at 30Hz spike frequency
Synapse technology	OxRRAAM or PCM
Access device	Diode or FET or NMOS
Resistance states	1-bit/synapse or 2-bit/synapse

neuromorphic hardware [44] at 45nm node with 2-bit synapses implemented using OxRRAM-based 1T1R cells.

5 EVALUATION OF NEUROXPLORER

Recently, SNNs are used to improve the quality of machine learning applications [18, 31, 45, 55]. We select 10 machine learning programs which 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, the number of neurons and synapses, the number of spikes per image, and the baseline accuracy of these applications on the DYNAPs neuromorphic hardware.

Table 3: Applications used to evaluate NeuroXplorer.

Class	Applications	Dataset	Neurons	Synapses	Avg. Spikes/Frame	Accuracy
CNN	LeNet	MNIST	23,687	275,110	724,565	85.1%
	DenseNet	CIFAR-10	17,450	198,470	1,250,976	42.8%
	AlexNet	ImageNet	259,604	3,873,222	7,055,109	69.8%
	ResNet	CIFAR-10	267,488	35,391,616	7,339,322	57.4%
	VGG	ImageNet	623,635	12,215,209	12,826,673	90.7 %
	HeartClass [16]	Physionet	170,292	1,049,249	2,771,634	63.7%
MLP	MLPDigit	MNIST	894	79,400	26,563	91.6%
	EdgeDet [15]	CARLsim	7,268	114,057	248,603	100%
	ImgSmooth [15]	CARLsim	5,120	9,025	174,872	100%
RNN	RNNDigit [24]	MNIST	1,191	11,442	30,508	83.6%

We evaluate the following three configurations of NeuroXplorer.

- **PyCARL** [2]: This is our default configuration, where a machine learning model is clustered arbitrarily. Clusters are also mapped arbitrarily to the crossbars of a hardware.
- **SpiNeMap** [6]: In this configuration, NeuroXplorer clusters a machine learning model to minimize the inter-cluster spike communication. Clusters are mapped to the crossbars to reduce energy consumption on the shared interconnect.
- **DecomposedSNN** [9]: In this configuration, NeuroXplorer decomposes a machine learning model to pack its neurons and synapses densely into each crossbar. The clusters are mapped to the crossbars to reduce spike latency and energy consumption on the shared interconnect.

5.1 Software Exploration: Cluster Count

Figure 14 plots the cluster count for each evaluated application for three different configurations of NeuroXplorer, normalized to PyCARL. For reference, the number of clusters obtained using PyCARL is indicated for each application. We observe that different configurations of NeuroXplorer lead to different cluster counts. DecomposedSNN, which maximizes the neuron and synapse utilization within each cluster, generates the lowest cluster count (44.5% lower than PyCARL and 50.1% lower than SpiNeMap).

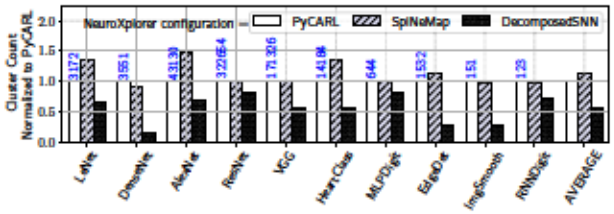


Figure 14: Cluster count using NeuroXplorer.

5.2 Software Exploration: Spike Count

Figure 15 plots the total number of spikes on the shared interconnect (called global spikes) for each evaluated application for three different configurations of NeuroXplorer, normalized to PyCARL. We observe that SpiNeMap has the lowest number of global spikes (6% lower than PyCARL and 34% lower than DecomposedSNN), which reduces both spike latency and communication energy due to reduction of the congestion on the interconnect.

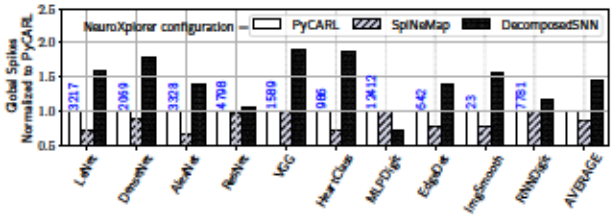


Figure 15: Global spikes using NeuroXplorer.

5.3 Hardware Exploration: Energy and ISI

Figure 16 and 17 plot respectively, the communication energy and inter-spike interval (ISI) of each evaluated application using four NoC routing techniques of NeuroXplorer normalized to XY routing. For reference, the communication energy and ISI at 45nm technology node with XY routing is also indicated.

5.4 Technology Exploration: Inference Lifetime

Non-Volatile Memories (NVMs) suffer from read endurance problem, where an NVM cell can switch its state upon repeated access. Therefore, the programmed synaptic weights of the NVM cells in a

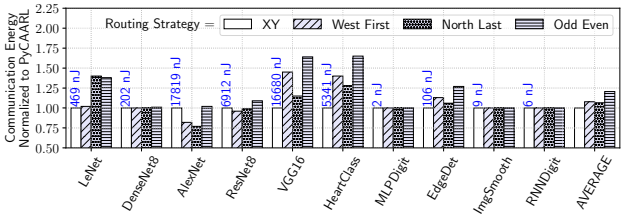


Figure 16: Communication energy using NeuroXplorer.

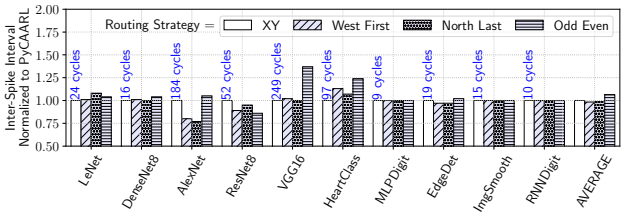


Figure 17: Inter-spike interval (ISI) using NeuroXplorer.

neuromorphic hardware needs to reprogrammed periodically. To this end, inference lifetime refers to how many images can be successfully inferred using the hardware before reprogramming of the synaptic weights becomes necessary. Figure 18 plots the impact of technology scaling on the inference lifetime of a neuromorphic hardware. At scaled nodes, the read endurance of NVMs reduces, which lowers the inference lifetime.

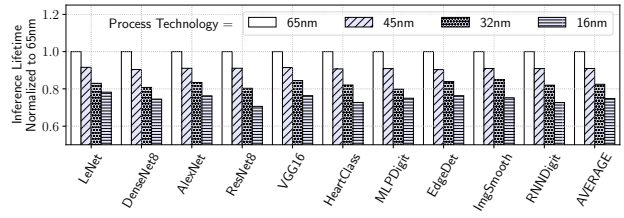


Figure 18: Technology exploration using NeuroXplorer.

6 CONCLUSIONS

We propose NeuroXplorer, an extensible framework for architectural exploration with Spiking Neural Networks (SNN). NeuroXplorer is based on a generalized template and can be infused with specific details of a neuromorphic hardware and technology. NeuroXplorer can perform platform-based design, hardware-software co-design, and design-technology co-optimization, enabling system designers to explore a variety of both application as well as platform design configurations to meet the needs of emerging workloads as well as newer design technologies. In addition to these architecture-centric functionalities, NeuroXplorer also facilitates functional simulations via SNN simulators supporting different degrees of neuro-biological details and learning modalities, allowing exploration of application quality.

ACKNOWLEDGMENTS

This work is supported by 1) 1) the US DOE CAREER Award DE-SC0022014 (Architecting the Hardware-Software Interface for Neuromorphic Computers), 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] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*.

[2] 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*.

[3] 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).

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

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

[6] 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).

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

[8] 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).

[9] 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).

[10] Adarsha Balaji, Salim Ullah, Anup Das, and Akash Kumar. 2019. Design methodology for embedded approximate artificial neural networks. In *GLSVLSI*.

[11] 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*.

[12] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. 2014. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* (2014).

[13] 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).

[14] 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*.

[15] 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*.

[16] 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*.

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

[18] 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).

[19] 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*.

[20] Mike Davies, Narayan Srinivasa, Tsung Han Lin, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* (2018).

[21] Andrew P Davison, Daniel Brüderle, Jochen M Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. 2009. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics* (2009).

[22] G De Michell and Rajesh K Gupta. 1997. Hardware/software co-design. *Proc. IEEE* (1997).

[23] Michael V. Debole, Brian Taba, Arnon Amir, et al. 2019. TrueNorth: Accelerating from zero to 64 million neurons in 10 years. *Computer* (2019).

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

[25] Jochen M Eppler, Moritz Helias, Eilif Muller, Markus Diesmann, and Marc-Oliver Gewaltig. 2009. PyNEST: a convenient interface to the NEST simulator. *Frontiers in Neuroinformatics* (2009).

[26] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. 2018. 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).

[27] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. 2014. The SpiNNaker project. *Proc. IEEE* (2014).

[28] Stefano Fusi and Maurizio Mattia. 1999. Collective behavior of networks with linear (VLSI) integrate-and-fire neurons. *Neural Computation* (1999).

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

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

[31] Kathleen Hamilton, Prasanna Date, Bill Kay, and Catherine Schuman D. 2020. Modeling epidemic spread with spike-based models. In *ICONS*.

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

[33] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *ICNN*.

[34] Brian W Kernighan and Shen Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* (1970).

[35] Kurt Keutzer, A Richard Newton, Jan M Rabaey, and Alberto Sangiovanni-Vincentelli. 2000. System-level design: Orthogonalization of concerns and platform-based design. *TCAD* (2000).

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*.

[37] 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*.

[38] 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*.

[39] De Ma, Juncheng Shen, Zonghua Gu, Ming Zhang, Xiaolei Zhu, Xiaoqiang Xu, Qi Xu, Yangjing Shen, and Gang Pan. 2017. Darwin: A neuromorphic hardware co-processor based on spiking neural networks. *JSA* (2017).

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

[41] 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*.

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

[43] Rivu Midya, Zhongrui Wang, Shiva Asapu, Saumil Joshi, Yunning Li, Ye Zhuo, Wenhao Song, Hao Jiang, Navnidhi Upadhay, Mingyi Rao, et al. 2019. Artificial neural network (ANN) to spiking neural network (SNN) converters based on diffusive memristors. *Advanced Electronic Materials* (2019).

[44] 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).

[45] Ethan J Moyer and Anup Das. 2020. Machine learning applications to DNA subsequence and restriction site analysis. In *SPMB*.

[46] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, and Timothée Masquelier. 2019. Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron. *Frontiers in Neuroscience* (2019).

[47] H Mulaosmanovic, J Ocker, S Müller, M Noack, J Müller, P Polakowski, T Mikolajick, and S Slesazeck. 2017. Novel ferroelectric FET based synapse for neuromorphic systems. In *VLSIT*.

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *arXiv* (2019).

[49] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. 2015. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience* (2015).

[50] Bodo Rueckauer and Shih-Chii Liu. 2018. Conversion of analog to spiking neural networks using sparse temporal coding. In *ISCAS*.

[51] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. 2016. Theory and tools for the conversion of analog to spiking convolutional neural networks. *arXiv* (2016).

[52] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. 2017. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963* (2017).

[53] Bart Selman and Carla P Gomes. 2006. Hill-climbing search. *En. of Cog. Sc.* (2006).

[54] Luping Shi, Jing Pei, Ning Deng, Dong Wang, Lei Deng, Yu Wang, Youhui Zhang, Feng Chen, Mingguo Zhao, Sen Song, et al. 2015. Development of a neuromorphic computing system. In *IEDM*.

[55] J Darby Smith, William Severa, Aaron J Hill, Leah Reeder, Brian Franke, Richard B Lehoucq, Ojas D Parekh, and James B Aimone. 2020. Solving a steady-state PDE using spiking networks and neuromorphic hardware. In *ICONS*.

[56] Shihao Song et al. 2021. A Design Flow for Mapping Spiking Neural Networks to Many-Core Neuromorphic Hardware. In *ICCAD*.

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

[58] 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).

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

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

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

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

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

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

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

[66] 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).

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

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

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

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

[71] 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).

[72] Peter JM Van Laarhoven and Emile HL Aarts. 1987. Simulated annealing. In *Simulated annealing: Theory and applications*.

[73] Adrien F Vincent, Jérôme Larroque, Nicolas Locatelli, Nesrine Ben Romdhane, Olivier Bichler, Christian Gamrat, Wei Sheng Zhao, Jacques-Olivier Klein, Sylvie Galdin-Retailleau, and Damien Querlioz. 2015. Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems. *TBCAS* (2015).

[74] Esin Yavuz, James Turner, and Thomas Nowotny. 2016. GeNN: a code generation framework for accelerated brain simulations. *Scientific Reports* (2016).

[75] Greg Yeric, Brian Cline, Saurabh Sinha, David Pietromonaco, Vikas Chandra, and Rob Aitken. 2013. The past present and future of design-technology co-optimization. In *CICC*.