NeuSB: A Scalable Interconnect Architecture for Spiking Neuromorphic Hardware

Adarsha Balaji, Phu Khanh Huynh, Francky Catthoor, *Fellow, IEEE*, Nikil D. Dutt, *Fellow, IEEE*, Jeffrey L. Krichmar, *Senior Member, IEEE*, and Anup Das, *Senior Member, IEEE*

***Abstract*—Neuromorphic systems are typically designed as a tile-based architecture where inter-tile data communication is facilitated using a shared global interconnect. Congestion on this interconnect can increase both interconnect energy, which increases the total energy consumption of the hardware and latency, which impacts the performance e.g., accuracy of the application that is being executed on the hardware. Mesh-based Network-on-Chip (NoC) that is used in most hardware prototypes is not the optimal interconnect solution for neuromorphic systems. This is because of the following two reasons. First, power consumption and average latency of a NoC increases exponentially with the number of tiles in the hardware. Second, a NoC cannot exploit an application's data communication pattern efficiently. Once designed for a target hardware, the bandwidth on each NoC link stays the same, independent of the volume of data traffic between different tile pairs of the NoC. In other words, a NoC cannot be customized at a finer granularity based on an individual application running on the hardware. We show that these NoC limitations prevent opportunities to further improve energy and latency of a neuromorphic hardware. To address these limitations, we propose Dynamic Segmented Bus (SB) interconnect for neuromorphic systems. Here, a bus lane is partitioned into segments with each segment connecting a few tiles. Connection of tiles to segments and those between segments are bridged using our novel three-way segmentation switches that are programmed using the software before admitting an application to the hardware. We partition an application by analyzing its workload and place partitions intelligently onto segments. This exploits application characteristics to use the segments without any routing collisions while exploiting the latency and energy savings in the design-time mapping phase. At a high-level, our mapping algorithm places tiles that communicate the most on shorter segments utilizing fewer number of switches, thereby reducing network congestion. It can adjust the bandwidth by controlling the number of segments connected to a destination tile. At run time, our controller dynamically executes the predefined routing paths without requiring any additionally routing decisions, unlike a NoC. This allows us to improve both energy and latency. Using parallel segmented busses, our proposed interconnect architecture can support a large number of tiles without significantly increasing the design cost, energy, and latency. Simulation results show that compared to the most widely-used mesh-based NoC design, our interconnect architecture, which we call NeuSB, reduces the switch area by 20x, average interconnect energy by 6.2x, and latency by 23%.**

***Index Terms*—Segmented bus, neuromorphic, spiking neural networks, network-on-chip (NoC), non-volatile memory (NVM).**

Manuscript received 3 May 2022; revised 21 October 2022; accepted 16 January 2023. Date of publication 26 January 2023; date of current version 7 June 2023. This work was supported in part by the U.S. Department of Energy under Grant DE-SC0022014, in part by the National Science Foundation Award under Grant CCF-1937419 (RTML: Small: Design of System Software to Facilitate Real-Time Neuromorphic Computing) and in part by the National Science Foundation Faculty Early Career Development Award under Grant CCF-1942697 (CAREER: Facilitating Dependable Neuromorphic Computing: Vision, Architecture, and Impact on Programmability). *(Corresponding author: Anup Das.)*

Adarsha Balaji, Phu Khanh Huynh, and Anup Das are with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104 USA (e-mail: ab3586@drexel.edu; ph434@drexel.edu; anup.das@drexel.edu).

Francky Catthoor is with the IMEC, KU, 3000 Leuven, Belgium (e-mail: francky.catthoor@imec.be).

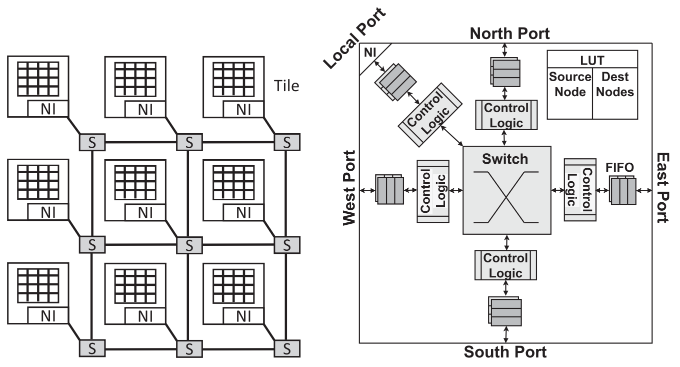
Nikil D. Dutt and Jeffrey L. Krichmar are with the Department of Cognitive Science, University of California, Irvine, CA 92697 USA (e-mail: dutt@ics.uci.edu; jkrichma@uci.edu).

Digital Object Identifier 10.1109/TETC.2023.3238708

I. Introduction

Hardware implementation of neuromorphic computing has shown significant promise to fuel the growth of machine learning [1], thanks to the low-power design of underlying computing circuitries [2], distributed implementation of computing and storage [3], and novel technology integration in the form of Non-Volatile Memories (NVMs) [4]. The aim is to build architectures that can execute machine learning applications designed using Spiking Neural Networks (SNNs). These are the third and more bio-inspired generation of neural networks [5], implemented as neuromorphic computing platforms that are inherently memory-centric, and which critically rely on in-memory compute and in-place synaptic storage. Unlike an artificial neural network (e.g., a convolutional neural network) where neural units communicate using tensors, communication between neurons in an SNN takes place via spikes. SNNs enable powerful computations due to their spatio-temporal information encoding capabilities [6].

To address scalability, neuromorphic systems are designed as a tile-based architecture, where each tile consists of circuitries to implement neurons and synapses. Typically, neurons and synapses in each tile are organized in a two dimensional grid, called crossbar [7]–[9]. Pre-synaptic neurons are implemented along rows of a crossbar and post-synaptic neurons are placed along columns. A memory cell is placed at the intersection of a



(a) A mesh-based NoC. (b) Detailed design of a switch.

Fig. 1. (a) A Network-on-Chip (NoC) with 9 tiles and 9 switches. These switches are organized in a 3 $\times$ 3 mesh. (b) The detailed design of a network switch with buffers and routing table (LUT). There are five ports – east, west, north, south, and local. A tile is connected to the local port of a switch, while the east, west, north, and south ports are used to connect to other switches in the NoC.

row and a column. Synaptic weights are programmed on these memory cells. A crossbar is not the only architecture to design a neuromorphic tile. There are also other designs proposed in literature such as the three layer μBrain architecture [10] and the tightly-coupled logic and memory design of Loihi [11].

Since the late 1980 s, major efforts have been in designing tile-based hardware platforms that can be deployed for inference. Examples include TrueNorth [12], Loihi [11], DYNAPs [13], and μBrain [10]. In all these systems, inter-tile data communication is facilitated using a Network-on-Chip (NoC) [14], which is an onchip communication infrastructure comprising the physical layer, the data link layer and the network layer of the Open Systems Interconnection (OSI) protocol stack [15]. Even though shared bus (e.g., AMBA [16]) has been the interconnect choice for multi-/many-core system-on-chip (SoC) due to its simplicity, it can lead to a high latency overhead to arbitrate the bus between different system components. NoCs are now replacing shared bus for modern SoCs that integrate a large number of system components. [17]–[21].

Inside a NoC, tiles are connected to channels via switches, which are organized as mesh (a Manhattan-like structure) [22]. Fig. 1(a) illustrates an example of a neuromorphic hardware with 9 tiles organized in a 3x3 mesh. Tiles (T) communicate with one another by sending data packets that are routed through the mesh network to reach their destinations. A network switch (S) buffers data packets and facilitates their routing using a lookup table (LUT), which is used to store the tile mapping information. We illustrate the design of a switch in Fig. 1(b). With the exception of the switches in the periphery, every other switch in the mesh is connected to four other network switches in the east, west, north, and south directions as shown in Fig. 1(a). A tile is connected to a switch using the local port as shown in Fig. 1(b). Data buffers and the LUT inside each switch are implemented using SRAMs. They constitute a significant fraction of the total power consumption of a NoC [23].

While a NoC eliminates the communication bottleneck introduced by a shared bus, it is not the optimal interconnect solution for neuromorphic systems for the following *three* reasons. First, for scaled machine learning applications where the data traffic (i.e., spikes) is high, the internal network contention in a NoC may 1) introduce variable (and high) packet delays, leading to inter-spike interval (ISI) distortion, and 2) reroute data packets, leading to spike arrival disorder at the receiving tile. ISI distortion and spike arrival disorder can lead to a significant loss of accuracy (quality in general) of an SNN-based machine learning application [24]–[26]. This is contrary to a shared bus, where the bus latency is the fixed wire-speed once the arbiter has granted control of the bus. However, the bus arbitration itself can become the critical latency bottleneck due to the presence of *multiple masters*, i.e., tiles that transmit data packets simultaneously on the bus.

Second, a NoC introduces a significant energy and area overhead due to 1) large buffers to store data packets and LUT to store routing table inside each switch, and 2) relatively long time-multiplexed connections that need to be near-continuously powered up and down, reaching from the ports of data producers/consumers (inside a tile or between different tiles) up to the ports of communication switches. The energy consumption in a NoC increases exponentially with an increase in the size of the mesh [27]–[29].

Third, once a NoC is designed for a neuromorphic hardware, there remains a limited opportunity to customize it based on the application executing on the hardware. Take for instance the data communication bandwidth, which remains the same for every link of a NoC. However, once an application is mapped to the hardware, the data communication between different tile pairs may vary based on the application characteristics. Therefore, some NoC links may remain underutilized due to low data traffic, while others may be bandwidth limited due to a high traffic volume, which may increase the latency. Another limitation of a NoC is its inability to manage power consumption at a granular level. For instance, it is not possible to power-down links that are unused by the application in order to save energy. This is particularly important for neuromorphic computing due to the following two reasons. First, communication energy constitute a large fraction (40%–50%) of the overall energy consumption as highlighted in many recent studies [30], [31]. Second, inter-tile communications in neuromorphic computing are sparse and therefore, full connectivity between the tiles is rarely exercised.

We propose a novel interconnect architecture for neuromorphic systems that leverages the simplicity of a shared bus yet addressing the three bottlenecks of a NoC and also the bottleneck associated with bus arbitration due to the presence of multiple masters on the bus. Our proposed solution is the following. We start with a bus lane consisting of multiple physical wires. We partition the bus lane into slices using our novel segmentation switches, which we describe in Section IV-C. Each switch has three directions – east, west, and north; east and west directions are reserved for interconnection with other switches on the bus lane, while the north direction is connected to a tile (see Fig. 4). Switches are controlled via the software. Slices can be combined using the switches to form segments, where each segment can have only one master tile and several slave tiles (see Fig. 4❶). In this way, each bus lane can support several segments that can be exercised in parallel and driven by their respective masters. Since no segment arbitration is needed at run-time after mapping an application to the hardware, the proposed segmented bus (SB) design saves significantly on the data communication latency and energy.

Following are our key *contributions.*

- We propose an application mapping (software) framework that partitions an application into clusters, minimizing the inter-cluster data communication. Clusters are then placed to tiles that are connected using the proposed SB interconnect.
- We design the segmented bus controller the uses the workload information to configure the segmentation switches based on the application running on the hardware such that each segment can have at most one master. Unused segments and switches are powered down to save energy.
- We propose a novel segmentation switch design, which eliminates the buffer and LUT of a NoC switche, thereby significantly reducing the area, energy, and latency overheads.
- We extend the framework to support multiple parallel segmented bus lanes to further increase the parallelism. The proposed architecture can support a large number of crossbars without significantly increasing the design cost, energy, and latency.

We integrate our proposed segmented bus-based interconnect architecture, which we call NeuSB, inside NeuroXplorer [32], a cycle-accurate simulator of neuromorphic systems. Evaluations with seven machine learning workloads show that compared to the most widely-used mesh-based NoC design, NeuSB reduces the switch area by 20x, average interconnect energy by 6.2x, and latency by 23%.

The concept of segmented bus for neuromorphic hardware was first introduced in our earlier works [10], [33], which focus only on the data plane. Here, we introduce the control plane, which is the core of NeuSB.

The remainder of the paper is organized as follows. A brief introduction to SNNs and their performance analysis is presented in Section II. The core concepts associated with segmented bus is introduced using a motivating example in Section III. The detailed design of segmented bus, including the mapping framework, segmented bus controller, and the segmentation switch design is presented in Section IV. Evaluation methodology is presented in Section V. Results are presented in Section VI. Related works are discussed in Section VII. Finally, the paper is concluded in Section VIII.

II. Background

SNNs are computation models designed using spiking neurons and synapses. A neuron can be implemented using some form of integrate and fire logic [34]. In an SNN, pre-synaptic neurons communicate information encoded in spike trains to post-synaptic neurons, via synapses. Performance, e.g., accuracy of an SNN model, is assessed in terms of the inter-spike interval (ISI), which is defined as inverse of the mean firing rate of the neurons.

On the hardware front, SNNs are implemented on NoC-based neuromorphic hardware, which is illustrated in Fig. 1(a). Here, each tile consists of a network interface (NI) to communicate

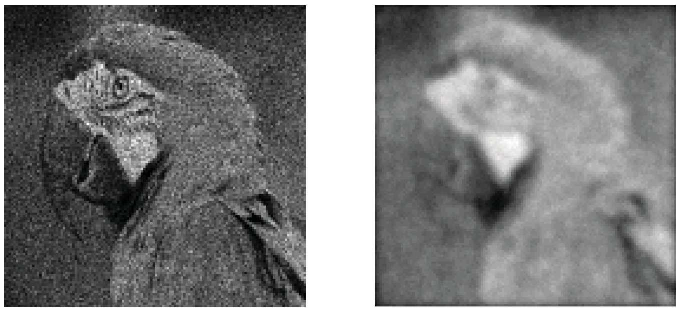

(a) No ISI distortion (PSNR = 20). (b) ISI distortion (PSNR = 10).

Fig. 2. Impact of ISI distortion on image smoothing application performed using an SNN. (a) Output of the SNN with no ISI distortion. (b) Output of the SNN with ISI distortion. The PSNR reduces due to ISI distortion.

data packets to the switch. A data packet consists of the address event representation (AER) of a spike generated from a neuron inside the tile. Fig. 1(b) illustrates the internal architecture of a switch. It has five ports (east, west, north, south, and local) to send and receive data. The local port connects a switch to the network interface of a tile. The three critical components of interest are 1) buffer space associated with each port to store data packets, 2) switch (crossbar) network to route data packets from an incoming to an outgoing port based on the routing algorithm and coordinates of destination tiles, and 3) a lookup table to store the mapping of neurons to tiles.

These switch components introduce significant area, energy, and latency overheads. We take the example of the LUT. To map n neurons to m tiles, the LUT size needed is $n \times \log_2 M$, where n could be in the order of millions. Such a large LUT consumes significant silicon area and power. It also takes a significant amount of time to access its content, thereby increasing the latency. In many recent designs such as DYNAPs and TrueNorth, architects address the LUT limitation of switches by enabling broadcast messages, where instead of routing incoming data packets to a specific output port, each packet is broadcasted on the NoC via all available ports. This eliminates the need for storing the neuron mapping table inside the LUT. However, the network traffic increases in this case.

Irrespective of the exact mechanism to route packets, a NoC interconnect does not scale when an SNN application with high data traffic is mapped to the hardware. This is because the internal network contention of a NoC can lead to some spikes encountering higher delays than others. This may lead to ISI distortion and spike disorder, which can lead to a loss in application quality. Fig. 2 shows this impact for image smoothing application implemented using an SNN [35]. Fig. 2(a) shows the output without ISI distortion. The peak signal-to-noise ration (PSNR) is 20. Fig. 2(b) shows the output of the image smoothing application with ISI distortion. The PSNR is 10. A reduction in PSNR indicates a loss in quality of image smoothing.

Spike disorder can also critically impact the performance. To illustrate this, Fig. 3 shows the impact of spike disorder. The top sub-figure shows a scenario where an output spike is not expected based on the spikes received from the three input neurons. Bottom sub-figure shows a scenario where the

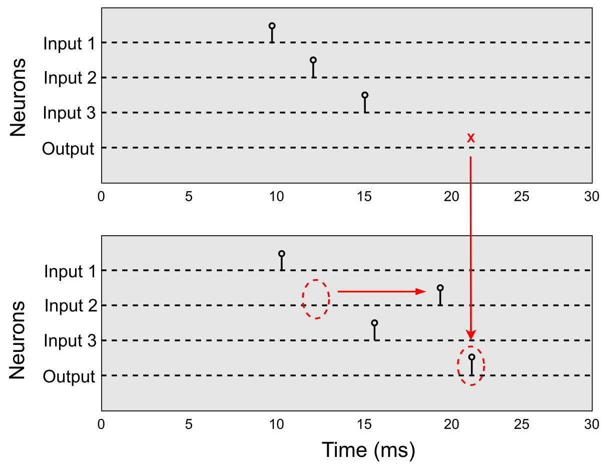


Fig. 3. Impact of spike disorder at the receiving neuron.

receiving order of spikes is altered via the NoC. This may result in a spike generation at the output. Extra spikes may result in misinterpretation of results, which may impact application quality.

III. A High-Level Overview of the Proposed Solution and a Motivating Example

A. A High-Level Overview

The proposed segmented bus framework consists of two logical abstractions – control plane and data plane.

Control Plane refers to functions and processes that determine the paths to send and receive data packets between different tiles. For a conventional NoC, the control plane is responsible for populating the routing table in the LUT of each NoC switch according to a given routing strategy. For the proposed segmented bus, the system software is responsible for finding routes at compile-time. Our solution for the control plane consists of the following key steps.

- Partitioning a machine learning model into clusters by analyzing its workload (Section IV-A).
- Mapping clusters to different tiles of the segmented bus interconnect (Section IV-B).
- Configuring segmentation switches to facilitate data communication between tiles (Section IV-C).

Data Plane refers functions and processes that forward data packets from one switch to another based on the control plane logic. For a conventional NoC, the routing table, forwarding table and the routing logic in each switch constitute the data plane function. For the proposed segmented bus, our novel segmentation switch design (Section IV-C) constitute the data plane.

B. A Motivating Example

Fig. 4 provides a high-level overview of how segmented bus can be used to overcome the critical area, energy, and latency limitations of a conventional shared bus and NoC. In ❶, a bus lane is partitioned into 6 slices using the segmentation switches (S1–S5). Tiles are connected to the bus lane using these switches. Imagine mapping an SNN application represented as the clustered graph shown in ❷. A cluster is a subset of the neurons and synapses of a given SNN application. In Section IV-A, we discuss how these clusters are generated from an SNN workload.

In ❷, there are 4 clusters (A, B, C, and D). Communication between these clusters are indicated using arcs with the arrow head pointing to the direction of communication. First, we consider mapping of the four clusters and only the arcs shown with solid lines to the segmented bus (later we consider the dashed arcs). To do this mapping, we generate two segments out of the bus lane by combining the slices. Segment 0 is formed using slices 1 and 2 while segment 1 is formed using slice 3. This is shown in ❸. To understand this behavior, we observe that there are two masters in ❷– A, which communicate data to B and C, and D, which communicate data to C. This is considering only the solid arcs. Since there can be at most one master in any bus segment, we form two segments – one connecting tiles that map clusters A, B, and C, while the other one connecting tiles that map clusters C and D. Correspondingly, ❸ also illustrates the actual data communication subsequent to mapping the four clusters to the hardware.

Imagine that the clustered SNN graph also includes the three dashed arcs to communicate data from cluster B (master) to clusters A, C, and D. These arcs cannot be mapped to the two existing segments because each of these segments is already assigned a master tile. Therefore, a second bus lane is introduced and a segment is formed out of it connecting tiles that map clusters A, B, C, and D, with the tile where B is mapped serving as the master. This is illustrated in ❹.

Finally, in ❺, we illustrate how the switches are configured in software to support data communication within the three segments. The internal architecture of a segmentation switch is introduced in Section IV-C. We note that some of the switch directions (ports) are disabled when admitting an application. This is because some slices of a bus lane may not be utilized by a particular mapping of clusters to tiles. By disabling switch ports through software we save energy as the unused slices are not driven unnecessarily as is the case with a NoC interconnect. We make the following four key observations that set the foundation of NeuSB.

1) Segmented bus allows multiple masters to communicate simultaneously on the same bus lane using different segments as shown in ❷. Here, tiles that map clusters A and D are the masters. This is contrary to a shared bus, where there can only be a single master at any given time.[1]
2) Communication patterns between clusters are analyzed at compile time using representative workloads. These patterns are then used to generate segments and map clusters to these segments such that each segment can have only one master. In this way, no run-time arbitration of segments is necessary, which significantly reduces energy and latency, compared to both a shared bus and a NoC (see our evaluations in Section VI).

[1]The concept of segmented bus is explored in prior works in the context of multi-/many-core systems [36]. Here, we optimize segmented bus design for neuromorphic computing.

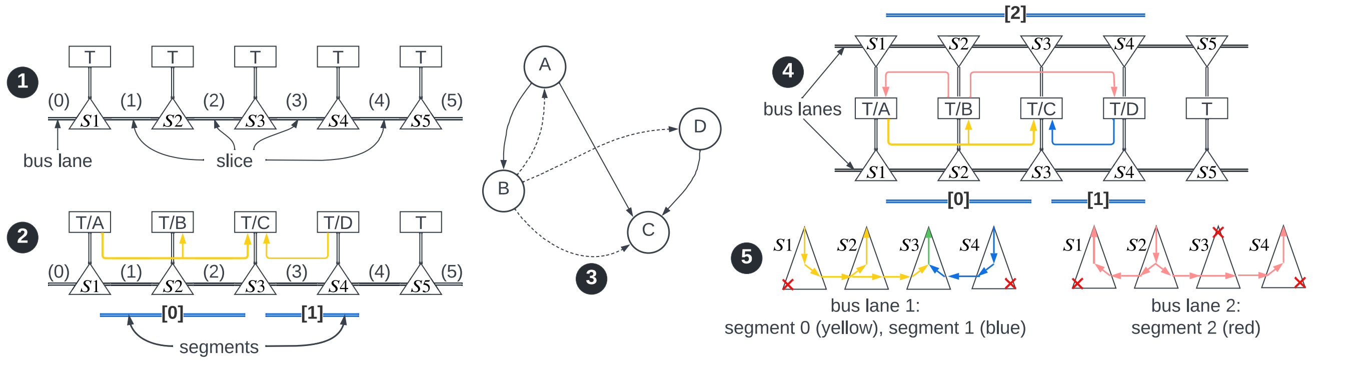


Fig. 4. Illustration of the proposed segmented bus-based interconnect solution using a simple example. ❶ A bus lane partitioned into 6 slices. ❷ Slices joined to form segments. ❸ An SNN partitioned into four clusters. ❹ Mapping of clusters to tiles. ❺ Switch configurations to support the inter-cluster communication.

3) The spike delay in a segmented bus is deterministic, and is equal to the sum of wire and switch delays on a segment. Therefore, there is a minimal ISI distortion. Moreover, data packets are serially communicated from the master tile on to a segment and therefore, there is no spike arrival disorder at the receiving tile. Finally, communication on a segmented bus is localized to a few segments, i.e., multicast compared to broadcasting mechanism, which is used in a shared bus and in some flavors of NoC. Multicasting reduces network congestion. Moreover, the interconnect energy is significantly reduced as not all tiles need to be activated to receive data packets all times.

4) Switch ports in the proposed segmented bus interconnect architectures are programmed during compile time before admitting an application. This is performed using the workload information by analyzing the communication pattern between the tiles. In this way, unused switch ports and bus slices are disabled to save energy.

C. Key Considerations and Future Directions

Following are the critical considerations and opportunities to further improve energy and latency of NeuSB.

- The worst-case mapping scenario of NeuSB is the one where each SNN cluster transmits spikes to all other clusters. Since, NeuSB requires a separate segment for each master, such worst-case condition may increase the number of segments and bus lanes. However, we mitigate this situation in the SNN partitioning step, where clusters are generated from an SNN by minimizing the inter-cluster spike communication (see Algorithm 1).
- NeuSB uses training data to first generate workload and then use the workload to analyze connection pattern and map clusters to tiles of the segmented bus. During inference, new connection patterns may emerge. This is particularly the case with on-chip learning, which is not the primary focus of this work. Nevertheless, we can mitigate such scenario by finding the best route from source to destination tile at run-time via the segmentation switch that connects the bus lanes themselves. This is left as a future work.

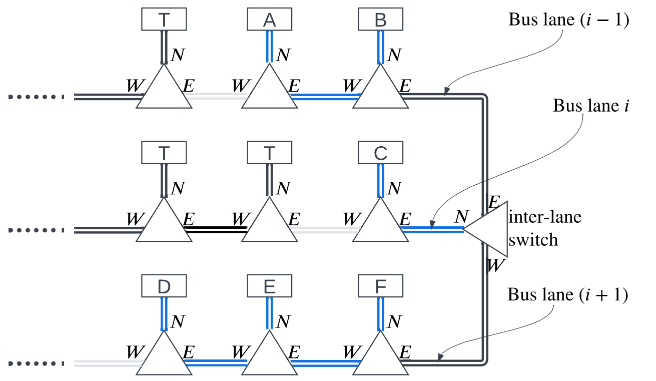


Fig. 5. A segment formed using tiles A-E, spanning across multiple bus lanes in the proposed segmented bus architecture. Here, bus lanes are interconnected using lane switches, which are essentially the segmentation switch that we present in Section IV-C.

- In NeuSB, a segment may span multiple bus lanes. In this scenario, our mapping algorithm which we describe in Section IV-B, enables the inter-lane switches. This is illustrated in Fig. 5 where a segment is formed using tiles A-E that span across three bus lanes. An inter-lane switch is essentially our proposed bus segmentation switch which we introduce in Section IV-C (see Fig. 8). The east, west, and north ports of the switch are connected to bus lane $(i-1)$, $(i+1)$, and (i), respectively as shown in the figure.

We now provide a detailed description of the segmented bus architecture architecture.

IV. Detailed Design of NeuSB

Our segmented bus design for neuromorphic hardware (NeuSB) consists of five key components – application clustering, segment generation, cluster mapping, switch design, and port configuration. We now describe these components.

A. Application Clustering

A tile in a neuromorphic hardware accommodates a fixed number of neurons and synapses. We take the example of

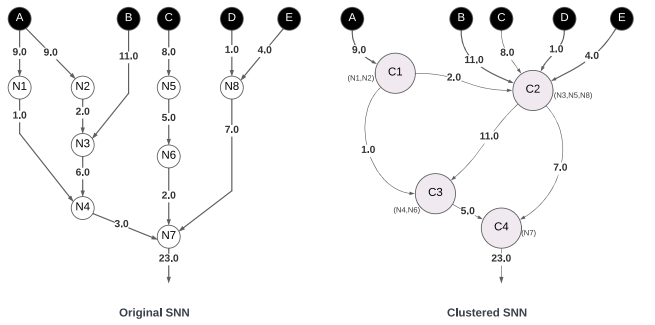


Fig. 6. Example of partitioning an SNN application with 8 neurons (left) into four clusters (right).

DYNAPs [13], where each tile can map a maximum of 128 input and 128 output neurons, and 16,384 synapses. On the other hand, each μBrain [10] tile can map 336 neurons and 17,408 synapses. A typical SNN application can have many neurons and synapses, well beyond the capacity of a single tile (see our evaluated SNN applications in Table V). Therefore, an SNN application needs to be partitioned, where each partition can then be mapped to a tile of the hardware. We call each partition a cluster. Fig. 6 illustrates the concept of partitioning. We start with an SNN (left) having 8 neurons (N1-8). Numbers on an edge between two neurons represent the number of spikes communicated between them. Using the proposed Hill Climbing-based clustering algorithm (Algorithm 1), we partition this SNN into 4 clusters (C1-4). Cluster C1 is formed with neurons N1 & N2, cluster C2 with N3, N5, & N8, cluster C3 with N4 & N6, and cluster C4 with N7. This is shown to the right of the figure. The number represented on each inter-cluster edge represents the total number of spikes generated from neurons inside the cluster.

Partitioning a large SNN application into clusters is essentially a graph partitioning problem and has been studied extensively in literature [37]. For non-trivial partitioning objectives (e.g., min cut), this is an NP-hard problem and therefore, heuristics are needed to solve this within a finite time bound. Recently, several graph partitioning approaches are proposed for SNNs targeting different optimization objectives. Examples include minimizing the number of clusters [38]–[40], minimizing inter-cluster spike communication [41], minimizing neuron and synapse circuit aging [42] and improving synaptic memory endurance [43]. See [44] for a comprehensive overview. Our objective is to reduce the number of channels between clusters. We introduce following definitions to formulate the partitioning problem.

Definition 1. (SNN) *An SNN application $G_{SNN} = (N, W)$ is a directed graph consisting of a finite set N of neurons and a finite set W of synapses between the neurons.*

Definition 2. (SYNAPSE) *A synapse $\mathbf{w}_{i,j}$ is a tuple $\langle n_i, n_j, s_{i,j}, \omega_{i,j}\rangle$ consisting of the source neuron n_i, the destination neuron n_j, the number of spikes ($s_{i,j}$) communicated from n_i to n_j for a given workload, and the synaptic strength ($\omega_{i,j}$) between neurons n_i and n_j.*

Definition 3. (CLUSTERED SNN) *A clustered SNN $G_{CSNN} = (C, L)$ is a directed graph consisting of a finite set C of clusters and a finite set L of links between the clusters. Each cluster $C_i \in C$ is a tuple $\langle N_i, W_i\rangle$, where $N_i \subseteq N$ is the set of neurons of the cluster and $W_i \subseteq W$ is the set of synapses of the cluster. Each link $L_{i,j} \in L$ is a channel connecting cluster C_i with cluster C_j, where C_i, $C_j \in C$.*

The partitioning problem is to perform the graph transformation of G_{SNN} to G_{CSNN}, i.e.,

$$f(\text{Optimization Problem}) = G_{SNN} \rightarrow G_{CSNN} \quad (1)$$

A link $L_{i,j} \in L$ represents a synaptic connectivity from a neuron n_s in the set $N_i \in C_i$ to a neuron n_d in the set $N_j \in C_j$. If there is no such connectivity, then $L_{i,j} = \emptyset$. The total number of spikes on this link is

$$\mathtt{spike}(L_{i,j}) = \sum_{s,d} s_{s,d} \mid n_s \in N_i \text{ and } n_d \in N_j \quad (2)$$

Once clustering is completed, the total number of inter-cluster spikes is given by

$$\mathcal{S}\text{ (total spikes)} = \sum_{i,j \in \{0,1,\ldots,|C|\}} \mathtt{spike}(L_{i,j}) \mid C_i, C_j \in C \quad (3)$$

We note that when clusters are mapped on to tiles, the inter-cluster data communication is mapped to the interconnect (shared bus, NoC, or segmented bus). Therefore, inter-cluster communication is the major source of energy and latency in a neuromorphic hardware. In fact, in our recent work [30] we have shown that data communication energy can be as high as 70% of the total energy consumption of a neuromorphic hardware. Accordingly, we set our optimization objective to minimize the total spikes S in (3).

Furthermore, since a cluster is to be placed on a tile of the hardware, the constraint of the optimization process is that the total number of neurons and synapses in each cluster must be less than or equal to the corresponding constraints of the hardware, i.e.,

$$|N_i| < N_T \text{ and } |W_i| < W_T, \text{ for } \langle N_i, W_i\rangle \in C_i \text{ and } \forall C_i \in C, \quad (4)$$

where N_T and W_T are the maximum number of neurons and synapses that can mapped to a tile, respectively.

Therefore, the application clustering problem is

$$\begin{aligned} f : \mathtt{Minimize}\ \mathcal{S} \\ \mathtt{s.t.}\ \text{ Equation 4 is satisfied} \end{aligned} \quad (5)$$

To solve the application clustering problem, we use an iterative algorithm that uses a Hill Climbing approach incorporating the Kernighan-Lin (KL) graph partitioning algorithm [45]. A conventional KL algorithm is a heuristic that partitions an undirected graph into two disjoint subsets in a way that minimize the total weights on all edges crossing one subset to another. We extend this as follows. We represent our SNN as an undirected graph where the neurons are the vertices and the synapses are the edges of the graph. The weight of each edge is the number of spikes communicated on that edge. We then iteratively partition each subset using the KL algorithm to smaller subsets such that each subset satisfies the hardware constraints ((4)). This

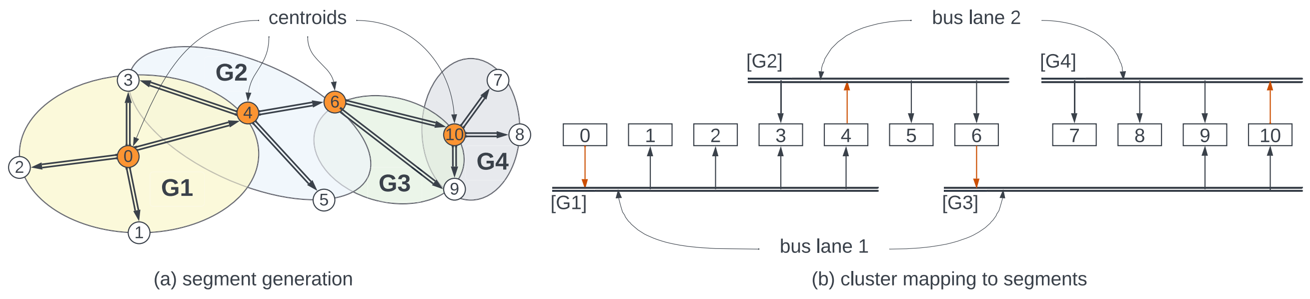


Fig. 7. Illustration of segment generation (left) and mapping of clusters to the segments (right).

generates the starting solution, which our Hill Climbing heuristic then improves upon.

Algorithm 1 provides the pseudo-code of our application clustering algorithm.

It starts with the KL block (lines 1-2), which generates an initial graph partitions using a modified KL algorithm. In the Hill Climbing exploration stage, it performs trial swaps of neurons from each pair ($C_i, C_j \in C$) of clusters (lines 4-16). In particular, each neuron $n_s \in C_i$ is swapped with another neuron $n_d \in C_j$ (line 5). A trial swap operation involves moving neuron n_s to cluster C_j and neuron n_d to cluster C_i. For this trial swap, the optimization objective is evaluated using (3) (line 7). A trial swap that leads to the minimum reduction of the optimization function is selected (line 11). In case of a tie, it selects a trial swap randomly. Once a swap is finalized, it makes the swap permanent by updating the clustering information (lines 12-15). The heuristic then iterates through the steps for the next pair of clusters. A pass in this heuristic consists of performing trial swaps for every pair of clusters. If during a pass, the optimization objective reduces, then another pass of the heuristic is performed (lines 17-18). Otherwise, the heuristic is said to be stuck at a local minimum. To come out of this local minimum, the clustering is perturbed, which involves making a fixed number of random swaps and restarting the iterative procedure (lines 19-21). Finally, the algorithm terminates after performing a fixed number of random perturbations (line 3). This is controlled using the user-defined parameter η, which is set empirically to 10.

Table V reports the number of clusters generated using our clustering technique for all evaluated applications.

Time Complexity Analysis: The time complexity of Algorithm 1 is computed as follows. The KL block (line 1) complexity is $O(r \cdot |N|^3)$, where r is the number of passes and $|N|$ is the total number of neurons [46]. The time complexity of lines 3-23 is $O(\eta \cdot |N| \cdot |C|^2)$, where $|C|$ is the total number of clusters. The overall time complexity is

$$O(Alg\,1) = O(r \cdot |N|^3) + O(\eta \cdot |N| \cdot |C|^2) \approx O(r \cdot |N|^3) \quad (6)$$

This is considering $r \approx \eta$ and $|C| < |N|$, i.e., the number of clusters is less than the number of neurons.

B. Bus Segmentation and Cluster Mapping

Algorithm 2 provides the pseudo-code for segment generation and cluster mapping step of our design flow. To explain this

Algorithm 1: Application Clustering Algorithm.

```
Input: $G_{SNN}(N, W)$
Output: $G_{CSNN}(C, L)$
1  $G_{CSNN}$ = modified-KL($G_{SNN}$);  /* Generate initial clustering using our iterative KL algorithm. */
2  $\mathcal{S}$ = Calculate inter-cluster spikes of $G_{CSNN}$ using Equation 3;
3  while number of perturbation $\leq \eta$ do  /* exit criteria */
4      for $C_i, C_j \in C$ do  /* For each pair of clusters */
5          for $n_s \in C_i$ and $n_d \in C_j$ do  /* For neuron pairs in the two clusters */
6              $G^{temp}_{CSNN}$ = swap($G_{CSNN}, n_s, n_d$);  /* Swap the neurons in the cluster. */
7              $\mathcal{S}^{temp}$ = Calculate inter-cluster spikes of $G^{temp}_{CSNN}$ using Equation 3;
8              $P$.append($\mathcal{S}^{temp}$);  /* Save the total spikes. */
9              $G$.append($G^{temp}_{CSNN}$);  /* Save the temporary clustering information. */
10         end
11         $x$ = argmin($P$);  /* Find the index to the minimum total spikes. */
12         if $P[x] < \mathcal{S}$ then  /* If the total spikes reduces. */
13             $\mathcal{S} = P[x]$;  /* Make the swap permanent. */
14             $G_{CSNN} = G[x]$; /* Update the clustered graph. */
15         end
16     end
17     if pass is successful then /* If the pass reduces the total spikes. */
18         Repeat steps 4-16;  /* Pass is successful and the steps 3-13 are repeated. */
19     else  /* If there is no improvement, the algorithm is stuck at a local minimum. */
20         $G_{CSNN}$ = Perturb;  /* Perturb the mapping by randomly swapping the clusters */
21         Repeat steps 3-23;  /* Hill Climbing pass is repeated. */
22     end
23 end
24 return $G_{CSNN}$;
```

algorithm, we provide a simple illustration in Fig. 7. Consider that the clustering step (Algorithm 1) generates 11 clusters as illustrated in Fig. 7(a). We start by grouping these clusters (lines 2-9). The grouping algorithm works as follows. If a cluster has an outgoing edge, we generate a group and assign the cluster as the centroid of the group. All clusters that are connected to an outgoing edge of the centroid are also added to the group (lines 5-7). In Fig. 7(a) we generate 4 groups (G1-4) from 11 clusters. The centroids of these groups are cluster 0, 4, 6, and 10, respectively. Groups may overlap as shown in this figure. For instance, groups G1 and G2 share clusters 3 and 4. We also note that each group will have one and only one centroid. This is essential to segmented bus implementation because every group

Algorithm 2. Segment Generation and Merging.

```
Input: $G_{CSNN}(C, L)$
Output: $\mathcal{B}$
$\mathcal{G} = \{\}$;   /* A data structure to store all groups. */
for $C_i \in C$ do                          /* For each cluster */
    $OutC$ = set of clusters on the outgoing edges of $C_i$;
    if $|OutC| \neq \emptyset$ then              /* OutC is not empty. */
        $g = \{C_i\} \cup OutC$;    /* Form a new group with $C_i$ and all clusters in $OutC$. */
        $g$.centroid($C_i$); /* Assign $C_i$ as the centroid of the group $g$. */
        $\mathcal{G}$.append($g$);        /* Save the group $g$ in $\mathcal{G}$. */
    end
end
$\mathcal{B} = \{\}$; /* A data structure to store all bus lanes. */
for $g_i \in \mathcal{G}$ do                        /* For each group */
    if $|\mathcal{B}| = 0$ then      /* If this is the first group. */
        $b = \{g_i\}$;      /* Create a new bus lane with the group. */
        $\mathcal{B}$.append($b$);    /* Append the new bus lane to $\mathcal{B}$. */
    else                    /* If is not the first group. */
        bus_lane_to_merge = $\emptyset$;
        for $b_j \in \mathcal{B}$ do                 /* For each bus lane */
            if $merge(G_i, b_j)$ then     /* If the group can be merged to the bus lane $b_j$. */
                bus_lane_to_merge = $b_j$;
                break;
            end
        end
        if $bus\_lane\_to\_merge \neq \emptyset$ then   /* If the group can be merged. */
            bus_lane_to_merge = bus_lane_to_merge $\cup g_i$
        else      /* If the group cannot be merged to an existing bus lane. */
            $b = \{g_i\}$;   /* Create a new bus lane with the group. */
            $\mathcal{B}$.append($b$);   /* Append the new bus lane to $\mathcal{B}$. */
        end
    end
end
return $\mathcal{B}$;
```

needs a segment to communicate, with the centroid of the group becoming the master of the segment. Therefore, by having only one centroid, we enforce only one master of the segment.

Without any further optimization, each group can form a bus segment and can be mapped to an individual bus lane. However, this will significantly increase the number of bus lanes, which is challenging for place and route. So, the next step of the algorithm is to merge segments to bus lanes such that the architecture has the minimum number of bus lanes needed to facilitate data communication. To this end, we use a greedy approach as follows. We start with the first group. Since there are no existing bus lanes yet, we assign G1 to bus lane 1 (lines 12-14). This is shown in Fig. 7(b). For all other groups, we check to see if the group can be merged with another group on a bus lane (lines 16-22). If the group can be merged, then we assign the group to the bus lane (lines 23-24). Otherwise, a new bus lane is created and the group is assigned to this new bus lane (lines 25-28).

Time Complexity: The time complexity of this algorithm is computed as follows. Lines 2-9 (the first for loop) iterates for the number of clusters and therefore, has a time complexity of $O(|C|)$. For lines 11-30, the outer for loop iterates for the number of clusters while the inner for loop (lines 17-22) iterates for the number of bus lanes. Therefore, the time complexity is $O(|B| \cdot |C|)$, where $|B|$ is the number of bus lanes. The overall time complexity is

$$O(Alg.2) = O(|C|) + O(|B| \cdot |C|) \approx O(|B| \cdot |C|) \quad (7)$$

The criteria for merging is explained using the example of Fig. 7. Referring to the groups in Fig. 7(a), we next consider group G2. Since G2 shares more than one clusters with G1 and one of the shared clusters (4) is the centroid of G2, we assign the new bus lane 2 for G2. Therefore,

$$\texttt{Criterion 1:} \mathrm{Merge}(G_j, G_i) \texttt{ iff } |G_i \cap G_j| < 2\,\& \\ \texttt{centroid}(G_j) \notin G_i \cap G_j \quad (8)$$

Next, we consider group G3. We note that G3 cannot be merged with G2 because the shared cluster 6 is the centroid of G3 (see criterion 1 in Eq. 8). However, groups G3 and G1 do not share any cluster. Therefore, G3 can be merged with G1 on bus lane 1. This is illustrated in Fig. 7(b). Finally, G4 cannot be merged with G3 as it violates criterion 1. However, G4 and G2 do not share any cluster and therefore, they can be merged on bus lane 2. Therefore, the two bus lanes are sufficient to allow communication between the 11 clusters of Fig. 7(a). For simplicity, we have not show the switches on the bus lanes. Table IV report the total number of bus lanes generated using our segmentation and cluster mapping algorithm.

We note that for large SNN applications there may be many clusters that are part of the same group. Therefore, merging groups (segments) using Algorithm 2 may lead to longer bus lanes, which can increase the spike delay on wires and switches. Therefore, we introduce a second merging criterion.

$$\texttt{Criterion 2:}\ \texttt{Merge}(G_j, G_i) \\ \texttt{iff}\ \left(\sum |b_i|\right) + |G_j| < B_s \quad (9)$$

Here, B_s is the constraint on the number of segmentation switches on each bus lane and $|G_j|$ is the total number of clusters of the group G_j. Observe that the total number of switches needed in implementing a group on a segment is equal to the number of clusters in the group. Furthermore, the number of bus slices (see Fig. 4❶) is one more than the number of switches. Therefore, setting a constraint on the number of switches per bus lane sets a constraint on the length of the bus lane. In (9), b_i is the bus lane of group G_i, i.e., $G_i \in b_i$. Once the group (and clusters) are assigned to bus lanes, switch configurations are set to enable the data communication. This is discussed next for our proposed segmentation switch design.

C. Segmentation Switch Design

Fig. 8 illustrates the internal architecture of a segmentation switch. There are three ports – north port (N), east port (E), and west port (W). This switch architecture can be used to connect a tile to a bus lane and also bridge multiple bus lanes. For connecting a tile to a bus lane, the north port of the switch is connected the tile, while the east and west ports are connected to the bus lane, i.e., to other switches. We illustrate this in Fig. 9(a). For bridging multiple bus lanes, the east, north, and west ports are connected to bus lanes $(i-1), i, (i+1)$, respectively as illustrated in Fig. 9(b).

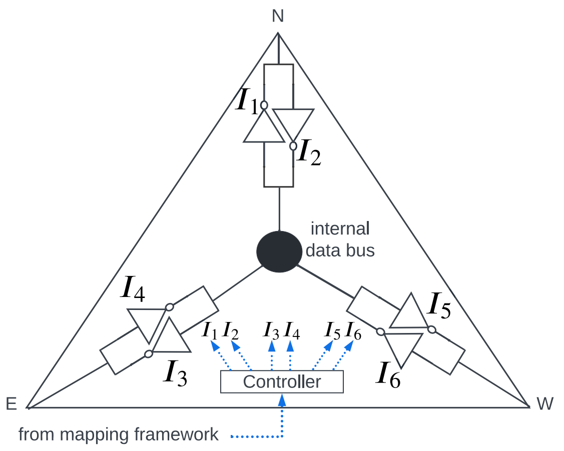
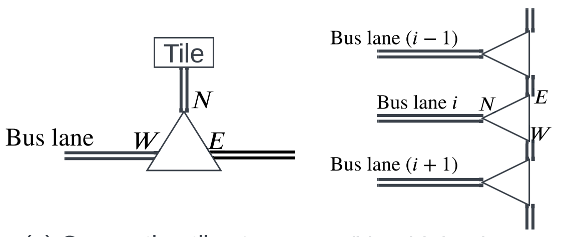


Fig. 8. Design of a segmentation switch with 6 inverters. The controller is used to configure (enable/disable) these inverters. The inverter configuration is loaded into this controller using the system software before admitting an application. The configuration is generated during the bus segmentation and cluster mapping step (Section IV-B). At run-time, the switch operates using this pre-loaded configuration.



(a) Connecting tiles to a bus lane (b) Bridging bus lanes

Fig. 9. Segmentation switch to connect a tile to a bus lane (a) and bridging multiple bus lanes (b).

Internally, each switch port is connected to two IO buffers that are used to route data to and from the internal data bus. These six buffers can be individually turned ON/OFF to perform the routing. Table I provides different buffer configurations and their corresponding actions. We note that configurations corresponding to simultaneously turning ON the two buffers connected to a port will lead to data corruption and therefore, they are marked as invalid. Similarly, turning OFF both the buffers of a port will disable the corresponding port.

Using the configurations from Table I, Table II reports the configurations of the switches on the two bus lanes, illustrated in Fig. 4❹–❺. We note that switch S3 on bus lane 1 is shared between segments 0 and 1, while switch S5 on bus lanes 1 and 2 are not part of any segments.

The configurations for IO buffers in each segmentation switch is generated from the controller that sits inside the switch as shown in Fig. 8. These configurations are generated during the mapping as describe in Section IV-B. Finally, we disable unused ports of a switch by setting both its buffers to OFF. This allows us to save energy.

TABLE I
CONFIGURATIONS OF THE INVERTERS IN THE SEGMENTATION SWITCH AND THEIR CORRESPONDING ACTION

Ports	Configurations	Actions
North	I_1 = OFF, I_2 = OFF	North port disabled
	I_1 = OFF, I_2 = ON	North port to internal data bus
	I_1 = ON, I_2 = OFF	Internal data bus to north port
	I_1 = ON, I_2 = ON	Invalid
East	I_3 = OFF, I_4 = OFF	East port disabled
	I_3 = OFF, I_4 = ON	East port to internal data bus
	I_3 = ON, I_4 = OFF	Internal data bus to east port
	I_3 = ON, I_4 = ON	Invalid
West	I_5 = OFF, I_6 = OFF	West port disabled
	I_5 = OFF, I_6 = ON	West port to internal data bus
	I_5 = ON, I_6 = OFF	Internal data bus to west port
	I_5 = ON, I_6 = ON	Invalid

Some switch configurations lead to data corruption and therefore, they are indicated as invalid in the table.

TABLE II
SWITCH CONFIGURATIONS FOR THE EXAMPLE OF FIG. 4❹–❺

Bus Lane	Segment No.	Switch	$\mathbf{I_1}$	$\mathbf{I_2}$	$\mathbf{I_3}$	$\mathbf{I_6}$	$\mathbf{I_5}$	$\mathbf{I_6}$
1	0	S1	OFF	ON	OFF	OFF	ON	OFF
		S2	ON	OFF	OFF	ON	ON	OFF
		S3	ON	OFF	OFF	ON	OFF	ON
	1	S3	ON	OFF	OFF	ON	OFF	ON
		S4	OFF	ON	ON	OFF	OFF	OFF
	–	S5	OFF	OFF	OFF	OFF	OFF	OFF
2	2	S1	ON	OFF	OFF	OFF	OFF	ON
		S2	OFF	ON	ON	OFF	ON	OFF
		S3	OFF	OFF	OFF	ON	ON	OFF
		S4	ON	OFF	OFF	ON	OFF	OFF
	–	S5	OFF	OFF	OFF	OFF	OFF	OFF

We show configurations for all 11 switches on the two bus lanes. There are three segments generated from the bus lanes. All these three segments can enable data communication in parallel. This improves latency.

TABLE III
MAJOR SIMULATION PARAMETERS EXTRACTED FROM [13]

Crossbar	28×28
Neuron technology	45nm CMOS
Synapse technology	Hfo2-based OxRAM
Supply voltage	1.0V
Energy per spike	50pJ at 30Hz spike frequency
Energy per routing	147pJ
Switch bandwidth	1.8G. Events/s

V. EVALUATION METHODOLOGY

A. Design Flow

Fig. 10 shows our design flow and evaluation framework. It incorporates machine learning applications designed using both convolutional neural networks (CNNs) and spiking neural networks (SNNs). For the former, our design flow integrates a frontend to integrated CNN models generated using Keras and

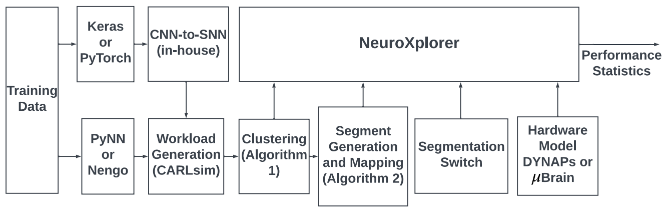


Fig. 10. Our design flow with NeuSB integrated inside NeuroXplorer [32].

PyTorch. Subsequently, a CNN model is converted to SNN using our in-house converter, which is described in [47], [48] before presenting it to the rest of flow. On the other hand, a native SNN can be specified using PyCARL [25] or Nengo [49].

Once an SNN is ready, it is simulated using CARLsim [35], which facilitates SNN simulations with neurobiological details at the neuron and synapse level using CPUs and multi-GPUs. We use CARLsim to generate the workload for a given training set, where a workload consists of

- *Spike Data:* the exact spike times of all neurons in the SNN model. We let $spk(i)$ represents a list of spike times of the i^{th} neuron in the model.
- *Weight Data:* the synaptic strength of all synapses in the SNN model. We let $w(i, j)$ represents the synaptic weight of the connection between the i^{th} and j^{th} neurons in the SNN model.

The workload information is used to generate the clusters using Algorithm 1, which we describe in Section IV-A. Next, we generate segments and map clusters to segments using Algorithm 2. We describe this in Section IV-B. Finally, we perform simulations using our cycle-accurate simulator NeuroXplorer [32]. This simulator incorporates our segmentation switch design, which we describe in Section IV-C and models for the neuromorphic hardware such as DYNAPs [13] and μBrain [10]. In this work, we show results for the DYNAPs hardware. However, the simulator can be easily configured for μBrain and other neuromorphic hardware such as TrueNorth [12] and Loihi [11]. Table III shows the major simulation parameters used in this work.

The segmentation switch is designed using Cadence Spectre with 45 nm technology libraries from [50]. The hardware configurations are as follows.

B. Evaluated Applications

Table V reports the machine learning applications that are used to evaluate NeuSB. For each of these applications, we apply a pruning technique [51] to eliminate near-zero weights. This is to keep the model size small. For each pruned model, we report the total number of neurons (column 2), synapses (column 3), and their baseline accuracy (column 4). We observe that the baseline accuracy numbers are lower than application-level simulation results. This is because of the ISI distortion and spike disorder, which reduces accuracy considerably. We have discussed this in Section II. Finally, column 5 reports the total number of clusters generated using the proposed clustering technique presented in Algorithm 1. We observe that the number of clusters of an application is higher for applications with higher number of neurons and synapses.

C. Evaluated Approaches

We evaluate the following approaches.

- NoC-Baseline: This is the DYNAPs hardware, which uses Neu-NoC [52], a hierarchical network-on-chip (NoC). Internally, it uses the X-Y routing algorithm to route packets via the switches. For this purpose, the routing table is stored in an LUT. SpiNeMap [24] is used to map applications to this hardware.
- NoC-Broadcast: This is NoC-Baseline, where broadcasting is used to communicate packets via a switch. Therefore, the routing table (LUT) in Fig. 1(b) is not needed. This saves on the dynamic energy of the switch. However, the congestion of the network is significantly increased.
- NeuSB: This is our proposed segmented bus architecture, where a single bus lane is partitioned into segments. The application mapping flow first partitions an application into clusters and then maps these clusters to segments of bus lanes. Unused ports and slices of bus lanes are disabled to save energy. The overall approach does not require bus arbitration and routing table inside switches, which reduces design area, energy, and latency.

VI. Results and Discussions

A. Interconnect Design Area

Table IV reports the different components that contribute to the area overhead of NoC and NeuSB. We make the following five key observations.

- First, the number of switches and wire slices for NoC-Baseline and NoC-Broadcast (see Fig. 1) increases as the size of an application increases. This is because larger applications need higher NoC sizes to accommodate their neurons and synapses. So, switches and slices needed for VGG-19 (which has more neurons, synapses, and clusters as reported in Table V) are much higher than for LeNet.
- Second, between NoC-Baseline and NoC-Broadcast, the switch area of NoC-Broadcast is significantly lower. This is because unlike NoC-Baseline, it uses broadcast mechanism to route data packets and therefore, routing tables are not needed inside the switches. This result also shows that the routing table is the biggest contributor of switch area in a NoC. Overall, the area overhead of NoC-Broadcast is on average 21.6x lower than NoC-Baseline.
- Third, NeuSB introduces more switches and slices than both these NoCs. However, the switch area is significantly lower. Individual switch area in NeuSB is 20x lower than NoC-Broadcast and over 1000x lower than NoC-Baseline. Therefore, the total switch area, which includes all switches on all bus lanes, is on average 2.2x lower than NoC-Broadcast and 55x lower than NoC-Baseline. We provide breakdown of the overhead of NeuSB in Section VI-E.

TABLE IV
DESIGN AREA COMPARISON NOC VERSUS NEUSB

Applications	NoC-Baseline				NoC-Broadcast				NeuSB (Proposed)				
	Switches	Wire Slices	Switch Area (B)	Total Area (KB)	Switches	Wire Slices	Switch Area (B)	Total Area (KB)	Switches	Total Wire Slices	Active Wire Slices	Switch Area (B)	Total Area (KB)
LeNet	165	469	2,230	359.3	165	469	160	35.5	986	1,422	163	8	7.7
AlexNet	256	736	3,520	880.0	256	736	160	54.8	2,480	19,500	4,259	8	19.4
HeartClass	440	1,278	11,294	4,852.9	440	1,278	160	144.2	7,293	50,504	27,471	8	57.0
ResNet-8	528	1,538	4,146	2,137.8	528	1,538	160	152.5	10,858	65,250	19,994	8	84.8
DenseNet-8	729	2,133	15,524	11,051.8	729	2,133	160	229.7	20,392	117,260	29,042	8	159.3
VGG-16	1,089	3,201	18,368	19,533.9	1,089	3,201	160	420.2	46,829	229,596	69,503	8	365.9
VGG-19	1,190	3,501	21,782	25,313.1	1,190	3,501	160	465.9	55,319	281,303	105,835	8	432.2

TABLE V
APPLICATIONS USED TO EVALUATE NEUSB

Applications	Neurons	Synapses	Baseline Accuracy	Clusters
LeNet	80,271	282,936	86.3%	157
AlexNet	127,894	3,873,222	66.4%	249
HeartClass [47]	219,070	4,892,102	63.7%	427
ResNet-8	266,799	5,391,616	57.4%	521
DenseNet-8	365,200	11,198,470	46.3%	714
VGG-16	554,059	22,215,209	81.4%	1082
VGG-19	603,198	28,948,582	75.8%	1176

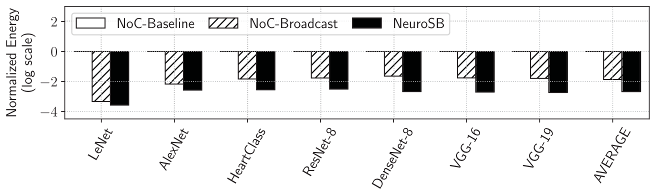


Fig. 11. Interconnect energy normalized to NoC-Baseline (log scale).

- Fourth, the switch area is the same for all applications for NeuSB and NoC-Broadcast, while it is application dependent for NoC-Baseline. This is because, each switch in NoC-Baseline requires a routing table which essentially contains information about mapping of neurons to tiles. Therefore, the bigger the size of the application, the larger is the routing table. Correspondingly, larger is the switch size.
- Finally, the total number of active wire slices in the proposed NeuSB is on average 70% lower than the total wire slices. This is because, using our control mechanism, which we describe in Section IV-C, we are able to disable segmentation switch ports such that the associated wire slices are not driven by voltages. This allows us to save a significant amount of energy (see Section VI-B).

B. Interconnect Energy

Fig. 11 plots the dynamic energy on the interconnect for the evaluated applications. Results are normalized to NoC-Baseline. We make the following two key observations.

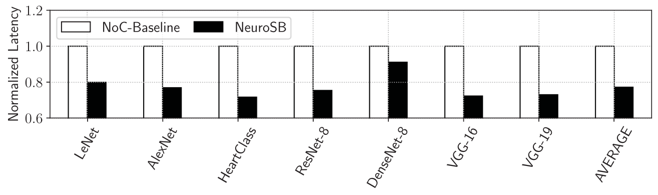


Fig. 12. Spike latency normalized to NoC-Baseline.

- First, between NoC-Baseline and NoC-Broadcast, NoC-Broadcast has an average 375x lower dynamic energy. This is because NoC-Broadcast does not need routing tables inside each switch, which require a significant amount of energy for table lookup.
- Second, the energy of NeuSB is on average 6.2x lower than NoC-Broadcast. This energy reduction is due to 1) the proposed segmented bus architecture, which does not require bus arbitration and packet routing at run-time, and 2) the proposed segmentation switch, which requires a simple design to route packets between its ports. Compared to NoC-Baseline, the energy of NeuSB is an average 478x lower.

C. Spike Latency

Fig. 12 plots the spike latency of NoC-Baseline and NeuSB for the evaluated applications. Results are normalized to NoC-Baseline.

We observe that the spike latency of NeuSB is on average 23% lower than NoC-Baseline. The reason for this improvement is three-fold.

- First, we partition bus lanes into segments and restrict inter-cluster data communication on a segment. These segments are smaller in size, which reduces the latency. On the other hand, data packets traverse longer distances (wire slices) within a NoC, which results in a higher latency than segmented bus.
- Second, as there is a single master for every segment and data packets are communicated sequentially by the master of the segment, there is no contention on the segment. On the other hand, there can be significant contention in a NoC, especially when the size of the model is large.

TABLE VI
ISI DISTORTION AND SPIKE ARRIVAL DISORDER OF NOC-BASELINE VERSUS NEUSB

Applications	ISI Distortion (cycles)		Spike Arrival Disorder	
	NoC-Baseline	NeuSB	NoC-Baseline	NeuSB
LeNet	29.65	23.73	0	0
AlexNet	147	113.23	0.013%	0
HeartClass	58.71	42.17	0.0046%	0
ResNet-8	41.87	31.63	0.0068%	0
DenseNet-8	14.16	12.91	0.0093%	0
VGG-16	153.79	111.42	0.0182%	0
VGG-19	217.93	159.4	0.0167%	0

- Finally, there is no arbitration required in a segmented bus because NeuSB analyzes data communication offline at compile time and use that information to map clusters to segments. This reduces the latency compared to NoC, where arbitration decisions are made at run-time by looking up the routing table within each switch.

D. ISI Distortion and Spike Disorder

Table VI reports the ISI distortion and spike arrival disorder of NoC-Baseline and NeuSB. ISI distortion is measure in number of cycles, while spike arrival disorder is measured as a fraction of the total number of spikes. We make the following two key observations.

- First, the ISI distortion of NeuSB is on average 22% lower than NoC-Baseline. This is because the latency of a spike on a segment in NeuSB is deterministic, being equal to the sum of latencies of the segmentation switches and the latency of wire slices. On the other hand, the latency of NoC-Baseline can be high and is dependent on network congestion. Therefore, some spikes may be delayed more than others, which increases the ISI distortion.
- Second, the spike arrival disorder of NeuSB is zero. This is because, spikes are serially communicated by the master to tiles in the segment. Therefore, spikes arrive in-order at the destination tiles. For NoC-Baseline, spikes from a source can reach destination tiles via different routes due to network congestion. So, the spike arrival disorder is high.

E. NeuSB-Related Key Insights

Table VII reports design details of NeuSB. On average, NeuSB introduces 135 bus lanes, 3 segments per bus lane, and 120 switches per segment.

We observe that the number of bus lanes is higher for large applications. This is to accommodate more neurons and synapses. Furthermore, we limit the number of switches per segment to 250 (the constraint B_s in (9)). This is to keep the spike latency small.

Fig. 13 plots the total energy of NeuSB, distributed into 1) dynamic energy in switches (average 24.4%), buffers (average 51.2%), and wires (average 13.7%), and 2) leakage energy (10.7%). We observe that the energy consumption in the switches is significantly low (compared to the energy of switches in NoC-Baseline) due to the elimination of routing tables. Buffers on tiles are the major contributor of energy. Finally, the wire and leakage currents are also considerably lower in NeuSB, compared to NoC-Baseline.

TABLE VII
BUS LANES, SEGMENTS, AND SWITCHES OF NEUSB

Applications	Bus Lanes	Avg. Segments/Bus	Avg. Switches/Segment
LeNet	9	2	109.5
AlexNet	78	1.47	31.8
HeartClass	118	3.52	62.3
ResNet-8	125	1.83	86.8
DenseNet-8	164	2.43	124.3
VGG-16	212	3.82	195.1
VGG-19	239	3.7	231.5

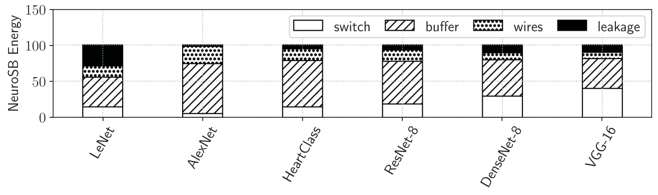


Fig. 13. Energy distribution of NeuSB.

TABLE VIII
COMPILATION TIME (IN SECONDS) OF NEUSB FOR THE PRE-PROCESSING ALGORITHMS

Applications	Algorithm 1 (s)	Algorithm 2 (s)
LeNet	25.861	0.126
AlexNet	104.598	0.318
HeartbeatClass	525.681	0.935
ResNet-8	949.568	1.39
DenseNet-8	2435.374	2.608
VGG16	8504.355	5.995
VGG19	10973.698	7.094

F. Program Compilation Time

Table VIII reports compilation time overhead for mapping different application on NeuSB. Algorithm 1 takes the majority of time in the compilation steps while the processing time for algorithm 2 is small. This corresponds to the time complexity analyses of the algorithms provided in Section IV-A and B. Algorithm 1 time complexity is proportional to the cube of the number of neurons while algorithm 2 is only proportional to the square of the number of clusters. Additionally, the number of clusters is more than two orders of magnitude lower than the number of neurons, as seen in Table V. Regardless, the total compilation time overhead for all the applications on NeuSB still stay within acceptable limit. Even for large application like VGG19 (with 603,198 neurons and 28,948,582 synapses), we only need approximately three hours to run both algorithms.

G. Energy Breakdown for NoC

Table IX reports interconnect energy consumption of NoC-Baseline for different applications, distributed into wires (static and dynamic), and buffers & switches (static and dynamic). We observe that the major contributor of energy in NoC-Baseline is buffers and switches. This is because of the significant amount of energy used for routing table lookup inside the switches, which increases exponentially with the increase of application size.

TABLE IX.
ENERGY DISTRIBUTION OF NOC-BASELINE.

Application	Wires (J)	Buffers & Switches (J)
LeNet	2.19E-07	1.182
AlexNet	1.17E-05	4.940
HeartbeatClass	2.63E-05	5.120
ResNet-8	3.22E-05	6.927
DenseNet-8	7.45E-05	8.405
VGG16	8.41E-05	13.894
VGG19	1.00E-04	15.305

VII. RELATED WORKS

This is the first work that addresses the design of segmented bus (both the data plane and the control plane) for neuromorphic hardware.

A. Other Interconnects Solutions for Neuromorphic Hardware

As the size of neuromorphic hardware is scaling up, interconnects is receiving growing attention in the research community. Multi-stage/hierarchical NoC is used in many neuromorphic hardware such as TrueNorth, DYNAPs, and Loihi. To this end, several new flavors of NoC have been proposed. Examples include Neu-NoC [52], 3D-NoC for multicompartment neurons [53], ring-based NoC [54], and restricted hierarchical NoC for the TrueNorth chip [55]. Among these NoC solutions, Neu-NoC is the most recent and relevant one. It is an optimized NoC solution for many-core neuromorphic hardware platforms. It outperforms many state-of-the-art NoC solutions in terms of both energy and latency. Here, we compare NeuSB against Neu-NoC and found that NeuSB significantly reduces design area, latency, and energy. The improvements over a classical NoC design is because 1) NeuSB enables more parallelism by partitioning bus lanes into segments and executing the segments in parallel, 2) NeuSB's switches are lightweight compared to a NoC switch, which helps to improve both area and energy, and 3) NeuSB configures the interconnect architecture by exploiting application characteristics and therefore, does not need to make any run-time routing decisions, which improves both energy and latency.

B. Interconnect Solutions for Conventional Multi-/Many-Core Systems

Several different interconnect architectures are proposed for multi-/many-core systems. Examples include shared bus, example AMBA [16], NoC [14], and segmented bus [36].

On the NoC front, both packet-switching flavor, e.g., HERMES [56] and circuit-switching flavor, e.g., SDM-NoC [57] are explored in literature. Although circuit-switching NoC can provide performance guarantee, it often leads to poor resource utilization as the reserved resources are assigned to only one transaction and remains reserved until the transaction is completed. We compare NeuSB against circuit-switching NoC because the large number of transactions (spike packets) in the context of neuromorphic computing will significantly increase the resource requirements (switches and wires) in a circuit-switching NoC with a correspondingly low utilization.

Segmented bus is previously studied for multi-/many-core systems [58]–[60]. Here, we describe segmented bus for neuromorphic computing. Specifically, we show how partitioning an application into clusters and mapping clusters intelligently onto segments of a segmented bus can lead to a significantly low area, energy, and latency. We also propose a very low overhead segmentation switch. Overall, we describe both the data and control plane of our interconnect solution for neuromorphic computing.

VIII. CONCLUSION

Neuromorphic computing platforms are inherently memory-centric, relying on technologies such as Non Volatile Memory (NVM) to enable in-memory compute and in-place synaptic storage. Crossbar interconnects and mesh-based Networks-on-Chip (NoCs) have been proposed to support these architectures, but suffer from increased power consumption and communication delays for scalable neuromorphic hardware designs. To support newer families of neuromorphic hardware, we presented NeuSB, an interconnect solution based on segmented buses, where a single bus lane is partitioned into segments, with each segment mapping a few tiles. Each segment in a segmented bus can have one and only one master tile, which communicates spikes to other tiles on the segment. In this way, we enforce 1) multi-casting of spike packets, which consumes less energy than broadcasting. The delays on a segment is deterministic, and is equal to the delay on the wires and switches on a segment. This allows to keep the inter-spike interval distortion low. Furthermore, spikes packets on a segment are communicated serially, which eliminates disorder of spikes at the receiving tile. A low inter-spike interval and spike arrival disorder leads to higher performance. Using multiple parallel bus, NeuSB allows parallelism in spike communication, which allows to scale up the hardware platform without significantly increasing the design area. Finally, NeuSB uses a novel segmentation switch hardware, which does not require routing tables, which lowers energy and latency. Overall, the design flow of NeuSB involves partitioning an SNN application into clusters and mapping the clusters to segments of parallel segmented bus. The communication patterns between clusters are analyzed at compile time to disable unused switch ports, saving energy. Since no run-time arbitration is necessary, the overall approach reduces energy and latency.

We evaluate NeuSB, using several SNN applications. Simulation results show that compared to the most widely-used mesh-based NoC design, NeuSB reduces the switch area by 20x, average interconnect energy consumption by 6.2x, and average spike latency by 23%.

References

[1] C. Mead, "Neuromorphic electronic systems," *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct. 1990.

[2] G. Indiveri, "A low-power adaptive integrate-and-fire neuron circuit," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2003, pp. IV–IV.

[3] 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*. Berlin, Germany: Springer, 2018.

[4] G. W. Burr et al., "Neuromorphic computing using non-volatile memory," *Adv. Phys. X*, vol. 2, no. 1, pp. 89–124, 2017.

[5] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997.

[6] H. Paugam-Moisy and S. M. Bohte, "Computing with spiking neuron networks," *Handbook Natural Comput.*, vol. 1, pp. 1–47, 2012.

[7] M. Hu et al., "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. Des. Automat. Conf.*, 2016, pp. 1–6.

[8] X. Zhang, A. Huang, Q. Hu, Z. Xiao, and P. K. Chu, "Neuromorphic computing with memristor crossbar," *Physica Status Solidi*, vol. 215, no. 13, 2018, Art. no. 1700875.

[9] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor crossbar-based neuromorphic computing system: A case study," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 10, pp. 1864–1878, Oct. 2014.

[10] M. L. Varshika, A. Balaji, F. Corradi, A. Das, J. Stuijt, and F. Catthoor, "Design of many-core big little μBrains for energy-efficient embedded neuromorphic computing," in *Proc. IEEE Design, Automat. Test Europe Conf. Exhib.*, 2022, pp. 1011–1016.

[11] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[12] M. V. DeBole et al., "TrueNorth: Accelerating from zero to 64 million neurons in 10 years," *Computer*, vol. 52, no. 5, pp. 20–29, May 2019.

[13] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.

[14] L. Benini and G. De Micheli, "Networks on chip: A new paradigm for systems on chip design," in *Proc. IEEE Des. Automat. Test Europe Conf. Exhib.*, 2002, pp. 418–419.

[15] J. R. Aschenbrenner, "Open systems interconnection," *IBM Syst. J.*, vol. 25, no. 3/4, pp. 369–379, 1986.

[16] D. Flynn, "AMBA: Enabling reusable on-chip designs," *IEEE Micro*, vol. 17, no. 4, pp. 20–27, Jul./Aug. 1997.

[17] D. Bertozzi and L. Benini, "Xpipes: A network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits Syst. Mag.*, vol. 4, no. 2, pp. 18–31, Sep. 2004.

[18] S. Kumar et al., "A network on chip architecture and design methodology," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI New Paradigms VLSI Syst. Des.*, 2002, pp. 117–124.

[19] B. Tobias and M. Shankar, "A survey of research and practices of network-on-chip," *ACM Comput. Surveys*, vol. 38, no. 1, 2006, Art. no. 1.

[20] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," in *Proc. IEEE Des. Automat. Test Europe Conf. Exhib.*, 2013, pp. 689–694.

[21] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surveys*, vol. 38, no. 1, pp. 1–es, 2006.

[22] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test Comput.*, vol. 22, no. 5, pp. 414–421, Sep./Oct. 2005.

[23] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: A power-performance simulator for interconnection networks," in *Proc. IEEE Int. Symp. Microarchit.*, 2002, pp. 294–305.

[24] A. Balaji et al., "Mapping spiking neural networks to neuromorphic hardware," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 76–86, Jan. 2020.

[25] A. Balaji et al., "PyCARL: A PyNN interface for hardware-software co-simulation of spiking neural network," in *Proc. Int. Joint Conf. Neural Netw.*, 2020, pp. 1–10.

[26] S. Song, A. Balaji, A. Das, and N. Kandasamy, "Design-technology co-optimization for NVM-based neuromorphic processing elements," *ACM Trans. Embedded Comput. Syst.*, vol. 21, pp. 1–27, 2022.

[27] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proc. Int. Symp. Low Power Electron. Des.*, 2005, pp. 387–392.

[28] A. Banerjee, P. T. Wolkotte, R. D. Mullins, S. W. Moore, and G. J. Smit, "An energy and performance exploration of network-on-chip architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 319–329, Mar. 2009.

[29] K. Lee, S.-J. Lee, and H.-J. Yoo, "Low-power network-on-chip for high-performance SoC design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 2, pp. 148–160, Feb. 2006.

[30] T. Titirsha, S. Song, A. Balaji, and A. Das, "On the role of system software in energy management of neuromorphic computing," in *Proc. ACM Int. Conf. Comput. Front.*, 2021, pp. 124–132.

[31] A. Das, Y. Wu, K. Huynh, F. Dell'Anna, F. Catthoor, and S. Schaafsma, "Mapping of local and global synapses on spiking neuromorphic hardware," in *Proc. Des. Automat. Test Europe Conf. Exhib.*, 2018, pp. 1217–1222.

[32] A. Balaji et al., "NeuroXplorer 1.0: An extensible framework for architectural exploration with spiking neural networks," in *Proc. Int. Conf. Neuromorphic Syst.*, 2021, pp. 1–9.

[33] A. Balaji, Y. Wu, A. Das, F. Catthoor, and S. Schaafsma, "Exploration of segmented bus as scalable global interconnect for neuromorphic computing," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 495–499.

[34] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I homogeneous synaptic input," *Biol. Cybern.*, vol. 95, no. 1, pp. 1–19, 2006.

[35] T.-S. Chou et al., "CARLsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters," in *Proc. Int. Joint Conf. Neural Netw.*, 2018, pp. 1–8.

[36] J. Chen, W.-B. Jone, J.-S. Wang, H.-I. Lu, and T.-F. Chen, "Segmented bus design for low-power systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 25–29, Mar. 1999.

[37] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," *Algorithm Eng.*, vol. 9220, pp. 117–158, 2016.

[38] F. Galluppi, S. Davies, A. Rast, T. Sharp, L. A. Plana, and S. Furber, "A hierachical configuration system for a massively parallel neural hardware platform," in *Proc. Conf. Comput. Front.*, 2012, pp. 183–192.

[39] A. Balaji et al., "Enabling resource-aware mapping of spiking neural networks via spatial decomposition," *Embedded Syst. Lett.*, vol. 13, no. 3, pp. 142–145, 2020.

[40] Y. Ji et al., "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints," in *Proc. Int. Symp. Microarchit.*, 2016, pp. 1–13.

[41] G. Urgese, F. Barchi, E. Macii, and A. Acquaviva, "Optimizing network traffic for spiking neural network simulations on densely interconnected many-core neuromorphic platforms," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 3, pp. 317–329, Third Quarter 2018.

[42] S. Song, A. Das, and N. Kandasamy, "Improving dependability of neuromorphic computing with non-volatile memory," in *Proc. Eur. Dependable Comput. Conf.*, 2020, pp. 17–24.

[43] A. Paul, S. Song, T. Titirsha, and A. Das, "On the mitigation of read disturbances in neuromorphic inference hardware," *IEEE Des. Test*, to be published, doi: 10.1109/MDAT.2022.3148967.

[44] P. K. Huynh, M. L. Varshika, A. Paul, M. Isik, A. Balaji, and A. Das, "Implementing spiking neural networks on neuromorphic architectures: A review," 2022, *arXiv:2202.08897*.

[45] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.

[46] S. Song, M. L. Varshika, A. Das, and N. Kandasamy, "A design flow for mapping spiking neural networks to many-core neuromorphic hardware," in *Proc. Int. Conf. Comput.-Aided Des.*, 2021, pp. 1–9.

[47] A. Balaji, F. Corradi, A. Das, S. Pande, S. Schaafsma, and F. Catthoor, "Power-accuracy trade-offs for heartbeat classification on neural networks hardware," *J. Low Power Electron.*, vol. 14, no. 4, pp. 508–519, 2018.

[48] S. Song, H. Chong, A. Balaji, A. Das, J. Shackleford, and N. Kandasamy, "DFSynthesizer: Dataflow-based synthesis of spiking neural networks to neuromorphic hardware," *ACM Trans. Embedded Comput. Syst.*, vol. 23, pp. 1–35, 2022.

[49] T. Bekolay et al., “Nengo: A python tool for building large-scale functional brain models,” *Front. Neuroinform.*, vol. 7, 2014, Art. no. 48.

[50] W. Zhao and Y. Cao, “Predictive technology model for Nano-CMOS design exploration,” *ACM J. Emerg. Technol. Comput. Syst.*, vol. 3, no. 1, pp. 1–es, 2007.

[51] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” 2015, *arXiv:1510.00149*.

[52] X. Liu, W. Wen, X. Qian, H. Li, and Y. Chen, “Neu-NoC: A high-efficient interconnection network for accelerated neuromorphic systems,” in *Proc. IEEE Asia South Pacific Des. Automat. Conf.*, 2018, pp. 141–146.

[53] S. Yang et al., “Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 1, pp. 148–162, Jan. 2020.

[54] Y. Qiu et al., “A novel ring-based small-world NoC for neuromorphic processor,” in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Archit. Processors*, 2021, pp. 234–241.

[55] P. A. Merolla et al., “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

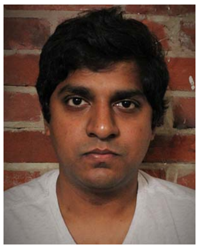
[56] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, “HERMES: An infrastructure for low area overhead packet-switching networks on chip,” *Integration*, vol. 38, no. 1, pp. 69–93, 2004.

[57] P. Marchal, D. Verkest, A. Shickova, F. Catthoor, F. Robert, and A. Leroy, “Spatial division multiplexing: A novel approach for guaranteed throughput on NoCs,” in *Proc. Int. Conf. Hardware/Softw. Codesign Syst. Synth.*, 2005, pp. 81–86.

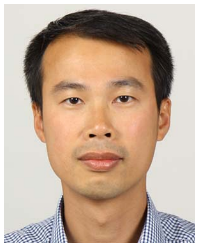
[58] T. Seceleanu, “Communication on a segmented bus,” in *Proc. IEEE Int. SOC Conf.*, 2004, pp. 205–208.

[59] W.-B. Jone, J.-S. Wang, H.-I. Lu, I. Hsu, and J.-Y. Chen, “Design theory and implementation for low-power segmented bus systems,” *ACM Trans. Des. Automat. Electron. Syst.*, vol. 8, no. 1, pp. 38–54, 2003.

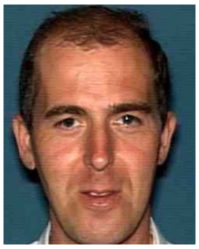
[60] T. Seceleanu, J. Plosila, and P. Lijeberg, “On-chip segmented bus: A self-timed approach [SoC],” in *Proc. IEEE Int. SOC Conf.*, 2002, pp. 216–220.



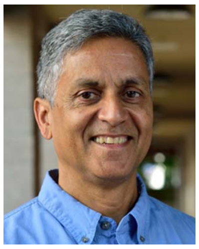
Adarsha Balaji received the bachelor’s degree from Visvesvaraya Technological University, India, in 2012 and the master’s degree from Drexel University, Philadelphia, PA, in 2017. He is currently working toward the PhD degree from the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA. His current research interests include design of neuromorphic computing systems, particularly data-flow and power optimization of spiking neural networks (SNN) hardware.



Phu Khanh Huynh received the bachelor’s degree in electrical electronic engineering from the Fontys University of Applied Sciences, the Netherlands, in 2011, and the master’s degree in embedded systems from the Eindhoven University of Technology, the Netherlands, in 2016. Between 2017 and 2021, he worked as an Embedded Software engineer in Veldhoven, the Netherlands. He is currently working toward the PhD degree from the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA. His research interests include designing, exploration, and optimization of neuromorphic computing systems, with focus on spiking neural networks (SNN).



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



Nikil D. Dutt (Fellow, IEEE) received the PhD in computer science from the University of Illinois at Urbana-Champaign in 1989, and is currently a distinguished professor of Computer Science, Cognitive Sciences, and EECS with the University of California, Irvine. His research interests are in embedded systems, electronic design automation (EDA), computer systems architecture and software, healthcare IoT, and brain-inspired architectures and computing. He received more than a dozen best paper awards and nominations at premier EDA and embedded systems conferences. He has extensive service on the steering, organizing, and program committees of several premier EDA and Embedded System Design conferences and workshops, and also serves or has served on the advisory boards of ACM SIGBED, ACM SIGDA, *ACM Transactions on Embedded Computing Systems, IEEE Embedded Systems Letters (ESL)*, and the ACM Publications Board. He is a fellow of the ACM and recipient of the IFIP Silver Core Award.

Jeffrey L. Krichmar (Senior Member, IEEE) received the BS degree in computer science from the University of Massachusetts at Amherst in 1983, the MS degree in computer science from The George Washington University in 1991, and the PhD degree in computational sciences and informatics from George Mason University in 1997. His research interests include neurorobotics, embodied cognition, biologically plausible models of learning and memory, neuromorphic applications and tools, and the effect of neural architecture on neural function. He is a Senior Member of the Society for Neuroscience.

Anup Das (Senior Member, IEEE) received the PhD degree in embedded systems from the National University of Singapore in 2014. He is an associate professor with Drexel University. Following his PhD, he was a postdoctoral fellow with the University of Southampton and a researcher with IMEC. His research focuses on neuromorphic computing and architectural exploration. He received the United States National Science Foundation CAREER Award in 2020 and the Department of Energy CAREER Award in 2021 to investigate reliability and security of neuromorphic hardware.