Clustering and Allocation of Spiking Neural Networks on Crossbar-Based Neuromorphic Architecture

Ilknur Mustafazade
Drexel University
Philadelphia, PA, USA
im445@drexel.edu

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

Anup Das
Drexel University
Philadelphia, PA, USA
ad3639@drexel.edu

ABSTRACT

Neuromorphic hardware, designed to mimic the neural structure of the human brain, offers an energy-efficient platform for implementing machine-learning models in the form of Spiking Neural Networks (SNNs). Achieving efficient SNN execution on this hardware requires careful consideration of various objectives, such as optimizing utilization of individual neuromorphic cores and minimizing inter-core communication. Unlike previous approaches that overlooked the architecture of the neuromorphic core when clustering the SNN into smaller networks, our approach uses architecture-aware algorithms to ensure that the resulting clusters can be effectively mapped to the core. We base our approach on a crossbar architecture for each neuromorphic core. We start with a basic architecture where neurons can only be mapped to the columns of the crossbar. Our technique partitions the SNN into clusters of neurons and synapses, ensuring that each cluster fits within the crossbar's confines, and when multiple clusters are allocated to a single crossbar, we maximize resource utilization by efficiently reusing crossbar resources. We then expand this technique to accommodate an enhanced architecture that allows neurons to be mapped not only to the crossbar's columns but also to its rows, with the aim of further optimizing utilization. To evaluate the performance of these techniques, assuming a multi-core neuromorphic architecture, we assess factors such as the number of crossbars used and the average crossbar utilization. Our evaluation includes both synthetically generated SNNs and spiking versions of well-known machine-learning models: LeNet, AlexNet, DenseNet, and ResNet. We also investigate how the structure of the SNN impacts solution quality and discuss approaches to improve it.

CCS CONCEPTS

• **Software and its engineering** → *Compilers*; • **Computer systems organization** → **Other architectures**.

KEYWORDS

Neuromorphic Architectures, Spiking Neural Network, Clustering

CF '24, May 7–9, 2024, Ischia, Italy

ACM ISBN 979-8-4007-0597-7/24/05
https://doi.org/10.1145/3649153.3649199

ACM Reference Format:
Ilknur Mustafazade, Nagarajan Kandasamy, and Anup Das. 2024. Clustering and Allocation of Spiking Neural Networks on Crossbar-Based Neuromorphic Architecture . In *21st ACM International Conference on Computing Frontiers (CF '24), May 7–9, 2024, Ischia, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3649153.3649199

1 INTRODUCTION

Neuromorphic systems that emulate the human brain's neural structure can be used to implement a variety of machine-learning tasks while achieving very high energy efficiency, thanks to their event-driven asynchronous operation which requires no clock signal; low-power analog/mixed-signal design of neurons; crossbar-based arrangement of synapses; and distributed placement of neurons and crossbars which overcomes the memory-bandwidth bottlenecks endemic to conventional computing systems. Several prototypes have been developed, such as Loihi [9], DYNAP-SE [17], μBrain [24], and FPGA-based implementations [6].

Neuromorphic architectures execute *spiking neural networks* (*SNNs*), which are networks of spiking neurons interconnected via synapses [11, 13]. Neurons communicate with each other by sending short impulses called spikes via synapses. Such spiking neurons can be organized into feed-forward layers or in a recurrent topology as SNNs, and used to solve classification problems in image and signal processing. The computation performed within the SNN is based on asynchronously occurring *spike trains* in which the location and frequency of spikes occurring within the network guide the execution.

Running a neuromorphic program on dedicated hardware requires several major steps: compilation, resource allocation, and run-time mapping. While these steps are well established for traditional Von Neumann architectures, they are still being refined for neuromorphic computing due to the distinct differences between the two paradigms. When a developer wishes to map the SNN onto neuromorphic hardware, they must partition the SNN into clusters containing smaller subnetworks of neurons and synapses. These clusters are then assigned to specific neuromorphic cores. At first glance, clustering and allocation of SNNs may appear to be a solved problem, and, indeed, numerous studies have tackled this topic (as discussed in Section 6). However, most efforts do not take into account the architecture of the target neuromorphic core onto which the clusters will be mapped. A common approach involves using some variation of the Kernighan-Lin (KL) graph partitioning algorithm to cluster the SNN in a way that minimizes inter-cluster communication, as measured by the number of spikes. This approach operates under the assumption that the formed clusters can be readily mapped onto the neuromorphic core. But this may be a flawed assumption. *If the target architecture or topology is not*

explicitly considered during clustering, the result may be clusters that do not make optimal use of the underlying hardware resources or, even worse, are incompatible with the target architecture.

We develop a clustering and allocation approach that is aware of the target neuromorphic architecture. This ensures that the clusters that are generated can be efficiently mapped onto the cores. An $n \times n$ crossbar capable of housing a predetermined number of neurons and synapses acts as the computational unit within each neuromorphic core. Spikes are communicated between cores through a network interconnect. Crossbar-based designs are popular for neuromorphic computing [1, 12, 16, 26]. Crossbar arrays can naturally represent synaptic connections in an SNN; by adjusting the resistance or conductance of the crosspoints, the strength of the connections between neurons can be emulated. Other reasons include parallelism, since each crosspoint junction can store and process information simultaneously, so that many neurons and synapses can operate in parallel; low-latency operations, since the computations are performed directly at the crosspoints without the need for data transfer between memory and processing units; and energy efficiency, since analog calculations can be performed directly at the crosspoints, without needing digital-to-analog and analog-to-digital conversions.
Our work makes the following contributions.

- Beginning with a foundational architecture in which neurons can only be mapped to the crossbar's columns, we develop a method that partitions the SNN into clusters while ensuring that each cluster fits within the parameters of the crossbar. When several clusters are assigned to one crossbar, we maximize resource utilization by strategically reusing the available crossbar resources.
- Our initial technique is then augmented to adapt to a more advanced crossbar design that permits neurons to be mapped to both columns and rows. This adaptation aims to further improve the utilization of crossbar resources.
- The efficacy of these techniques is assessed using both synthetically constructed SNNs and spiking versions of well-known machine learning models: LeNet, DenseNet, and ResNet. Evaluation metrics include the number of crossbars used and the average crossbar utilization. Our architecture-aware clustering algorithms significantly improve average crossbar utilization compared to the baseline case which uses the KL method. We also study the influence of the SNN's inherent structure on the quality of the clustering results and explore strategies to improve the outcomes generated by our techniques.

The remainder of the paper is organized as follows. Section 2 provides the necessary background on neuromorphic computing. Sections [**?**] and 4 develop the proposed clustering and allocation techniques, and the evaluation results are presented in Section 5. Related work is discussed in Section 6 and we conclude the paper in Section 7. The figures in this paper are best viewed in color.

2 PRELIMINARIES

This section familiarizes the reader with the basic concepts that underpin neuromorphic computing.

2.1 Spiking Neural Network Model

An SNN can be represented as a graph $G = (V, E)$, where the vertices $V = \{v_1, v_2, \ldots, v_n\}$ represent neurons and the edges $E = \{e_{ij} \mid v_i, v_j \in V\}$ represent the synapses between neurons. If neurons i and j are connected, an edge e_{ij} connects v_i and v_j. Since synapses have a direction, G is a digraph. Thus, e_{ij} and e_{ji} are distinct edges and can potentially both exist. The weight $w(e_{ij})$ along each edge represents the strength of the synaptic connection between neurons i and j.

Leaky integrate-and-fire is a simple model for a spiking neuron [5]. Here, the neuron's membrane potential integrates the incoming synaptic currents. This potential is modeled as a capacitor that is charged by incoming synaptic currents and discharges over time due to a leak current that represents the gradual loss of charge across the membrane. When the membrane potential reaches a certain threshold, the neuron fires a spike and the potential is reset. After firing a spike, the neuron enters a refractory period during which it cannot fire another spike. Figure 1(a) illustrates the connection of presynaptic and postsynaptic neurons through synapses.

2.2 Neuromorphic Architecture

Though neuromorphic platforms differ in their operation, crossbar-based architectures are common. For example, Loihi simulates spiking neurons using a digital architecture, whereas DYNAP uses analog circuits for the same purpose. A crossbar is a two-dimensional arrangement of synapses, with n^2 synapses for n input neurons. Figure 2 shows the organization of a crossbar in more detail. The top electrodes (TEs) and bottom electrodes (BEs) constitute the rows and columns, respectively, of the crossbar. A synaptic cell is connected at a crosspoint via an access transistor. Synaptic weights are specified in terms of the conductivity of nonvolatile memory cells (NVM), which allows these cells to act as computational units through analog summation of the current flowing through them [4]. The NVM is shown as a resistive element in Fig. 2. Presynaptic neurons are mapped along the TEs and postsynaptic neurons along the BEs. The weight between a presynaptic neuron and a postsynaptic neuron is programmed as the conductance of the corresponding synaptic cell at the crosspoint. The voltage of a presynaptic neuron v, applied on the TE, is multiplied by the conductance to generate a current according to Ohm's law. Current summation occurs on each BE according to Kirchoff's Current Law, when integrating excitation from other pre-synaptic neurons. The figure shows the integration of the input excitation of two presynaptic neurons into one postsynaptic neuron via synaptic weights w_1 and w_2, respectively. Following Kirchhoff's law, the current summation along the column implements the sum $w_1v_1 + w_2v_2$ needed for forward propagation of neuron activation. These current summations are performed along each column in parallel.

The NVM device of a synaptic cell can be implemented using phase-change memory, oxide-based memory, or spin-based magnetic memory [4]. To read or program a cell, its peripheral circuit drives current through it using a bias voltage generated by on-chip charge pumps built using CMOS devices.

Scaling the size of a crossbar increases the number of synapses per neuron, which exponentially increases the dynamic and leakage energy. Therefore, the size is limited to accommodate only a fixed

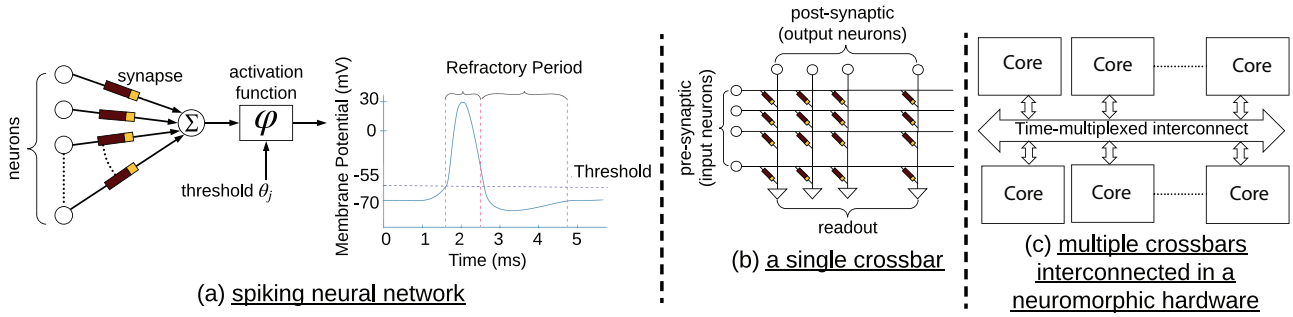


Figure 1: (a) connection of presynaptic and postsynaptic neurons via synapses in an SNN; (b) crossbar organization with fully connected presynaptic and postsynaptic neurons; and (c) multi-core neuromorphic architecture.

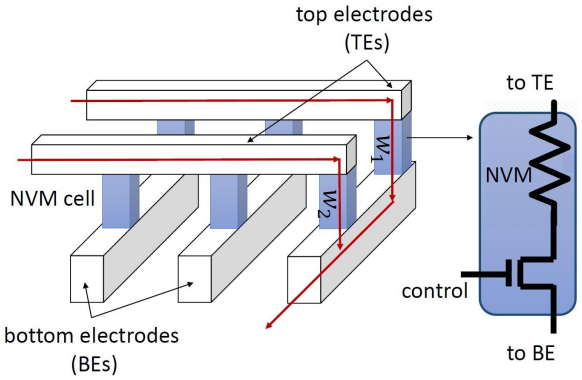


Figure 2: Crossbar organization showing the top and bottom electrodes. Each synaptic cell consists of an NVM device (resistive element) and an access transistor.

number of synapses per neuron. Therefore, to build a large system, multiple crossbars are integrated using a shared interconnect, as shown in Fig. 1(c). Each neuromorphic core includes neuron and synapse circuits, peripheral logic to encode and decode spikes into Address Event Representation (AER), and a network interface to send and receive AER packets over the interconnect. Switches are placed on the interconnect to route AER packets to their destination cores. The AER communication protocol is used to transmit the addresses of neurons that spike within the SNN. When a neuron spikes, it generates a packet with its unique address, which is transmitted in real time on the interconnect. At the receiver, this address is used to convey the spike to the target neuron, thus stimulating it. Since only addresses of active neurons are transmitted, this sparse representation enables very efficient intercore communication, especially in SNNs where only a small fraction of neurons might be active at any given time. This hardware abstraction fits most neuromorphic systems, and therefore will be used in this work.

Given an SNN, a mapping of the synapses to crossbars and crossbars to charge pumps is generated for the specific neuromorphic hardware, which maximizes crossbar utilization while minimizing the maximum latency incurred by spikes transmitted over the interconnect. Performance is verified in terms of the spike rate and inter-spike interval at the output neurons using a cycle-accurate simulator with a detailed hardware model [7]. Figure 3a shows the portion of an SNN whose neurons are mapped to the 4×4 crossbar

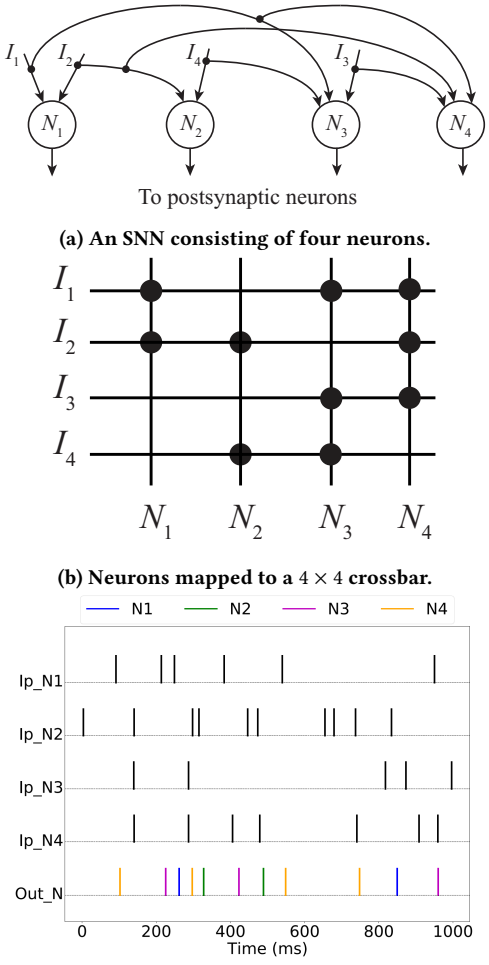


Figure 3: Simulating a small SNN mapped to a 4×4 crossbar.

in Fig. 3b, where neuron N_1 is mapped to column 1, N_2 to column 2, etc. Figure 3c shows a simulation result for this SNN obtained using CARLsim [18], which is an SNN simulator. The plot shows spikes from presynaptic neurons supplied to the crossbar's rows and the resulting spikes along the columns that propagate to any

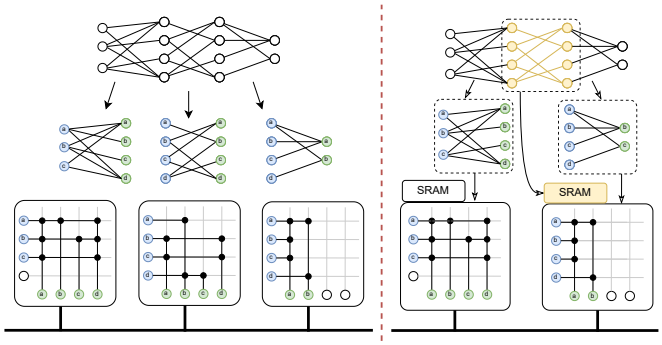


Figure 4: Comparison of mapping solutions obtained for the CROSS and CROSS+ architectures.

postsynaptic neurons. Each output spike is color coded to indicate the neuron that generates it.

3 ASSUMED CROSSBAR ARCHITECTURES

We will develop clustering algorithms for two types of crossbar architectures.

The first model is called CROSS which implements neurons only along the crossbar's columns. The implication is that each layer within the SNN is mapped onto a crossbar. Figure 4 (left) shows the mapping solution that decomposes the SNN into three subnetworks. Spikes generated by postsynaptic neurons within a subnetwork are transmitted over the interconnect to the appropriate crossbar using AER, where they become inputs. This architecture is simple, but requires more crossbars since each neuron must be duplicated — it acts as a postsynaptic neuron in one crossbar and as a presynaptic neuron in the other.

Figure 4 (right) shows the second model called CROSS+ which accommodates neurons along both rows and columns of a crossbar. To accommodate this setup, each neuromorphic core maintains local memory (SRAM) to store synaptic weights (in addition to weights programmed into the crossbar). Neurons are either placed on the crossbar's columns (and receive inputs from the crossbar itself) or placed on the crossbar's rows (and receive inputs and weights from the interconnect and scratch memory, respectively).

The clustering and allocation methodology presented in the subsequent section yields solutions tailored for both CROSS and CROSS+ architectures. In particular, it uses the solution derived for CROSS as the foundation for the solution for CROSS+.[1]

4 CLUSTERING AND ALLOCATION

Many existing studies on the mapping of SNNs onto neuromorphic hardware neglect to consider the inherent topological constraints of the target crossbar architecture. The presumption is that the generated clusters can be seamlessly deployed onto the target. Mere restrictions on neuron counts fall short when striving for viable mappings. The limitation lies in the emphasis on only clustering neurons, even though the essence of crossbar functionality lies in the synapses or edges mapped onto it. We explain this issue using the example shown in Fig. 5. Suppose that we wish to map the SNN onto a set of 3×3 crossbars (this capacity is based on the maximum fan-in of the SNN). We use the clustering technique developed by Song et al. [21], which falls in the category of work using the KL algorithm to minimize inter-cluster traffic, to obtain the solution shown in Fig. 5b. However, simply grouping neurons $N3$, $N5$, and $N8$ does not provide a feasible mapping solution. The first problem is the totality of their inputs (B, C, D, E) exceeds the assumed crossbar capacity. Also, *since the crossbar operation is determined by edges rather than by nodes, a neuron-centric clustering approach can result in infeasible solutions.*

Our approach explicitly takes into account synaptic connections during SNN clustering. By clustering neurons and their associated synapses into subgraphs, we ensure that these subgraphs align with the topological requirements of the target crossbar architecture. Figure 5c shows the solution obtained that also highlights the synaptic inputs for each cluster. The subsequent mapping to crossbars is shown in Fig. 5d. *Emphasis on synapses ensures that our clustering method consistently produces mappable clusters. Multiple such clusters can also be merged in a way that maximizes reuse of crossbar resources and, thereby maximizes utilization.*

4.1 Overview

We assume the availability of $n \times n$ crossbars. Major steps in our methodology, shown in Fig. 6, are summarized as follows:

- Given a trained SNN graph $G = (V, E)$, we use CARLsim [19] to extract the following details needed for our clustering method: spike-traffic information, timestamps per spike, and total spike count.
- The SNN is partitioned into minimal sub-graphs guaranteed to be mappable onto the target (Algorithm 1).

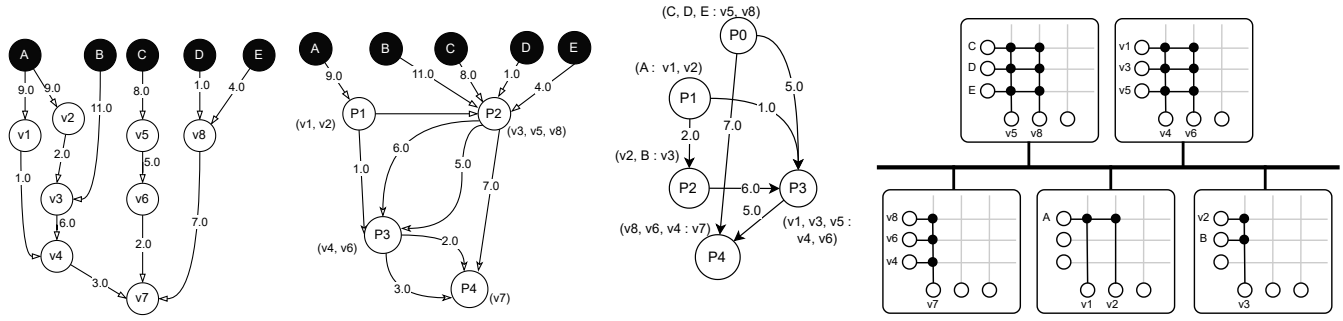
$$\mathcal{S} = \{S_1, S_2, S_3, ..., S_i \mid S_i \subseteq G\}$$

- Subgraphs are merged together to maximize available overlaps in their inputs until crossbar capacity is reached (Algorithm 3). By maximizing overlap in inputs, we can reuse inputs along the crossbar's rows for multiple neurons, improving crossbar utilization
- Combinations of subgraphs that would satisfy the crossbar capacity are packed into the clusters obtained from the previous step; these subgraphs are not required to be overlapping and sharing neurons (Algorithm 2). This step aims to further improve crossbar utilization.
- The above steps generate a feasible solution for the CROSS architecture. Given this initial solution, we place the available candidates pairs as input and output neurons in the CROSS+ architecture.

4.2 Generating Solution for CROSS Architecture

Algorithm 1 partitions the SNN into subgraphs. For each neuron v, it considers all incoming edges and generates the induced subgraph (line 3). This step ensures that this subgraph — provided the number of incoming edges into it is less than n — can be mapped to the target crossbar. We term these subgraphs as *fundamental blocks*, which are provided as input to Algorithm 2. It merges blocks together to generate every possible pair that is still mappable onto the target

[1]The main focus of this paper is to develop clustering and allocation algorithms for SNNs. As such, a detailed development of the hardware circuitry needed to realize the CROSS and CROSS+ architectures is beyond its scope.



(a) The input SNN **(b) Infeasible solution** **(c) Feasible solution** **(d) Final mapping onto crossbars**

Figure 5: Comparison of clustering techniques to motivate the need for an architecture-aware approach.

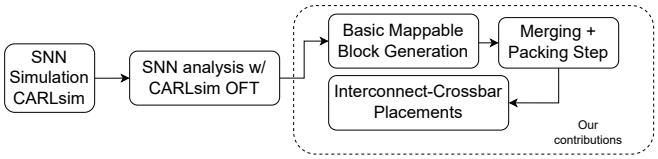


Figure 6: Key steps in our clustering and allocation method.

Algorithm 1: Basic Block Generation

Input: $G(V, E)$
Output: $starterSubgraphs$
1 $starterSubgraphs = \emptyset$
2 **for** $v \in V$ **do**
3 $\quad subgraph = DiGraph(IncomingEdges(v))$
4 $\quad starterBlocks.add(subgraph)$
5 **end**
6 **return** $starterSubgraphs$

Algorithm 2: Candidate Solution Generation

Input: $currentSubgraphs$
Output: $nextSolutions$
1 $nextSolutions = \emptyset$
2 $finalGraphs = \emptyset$
3 **for** $this\_sg \in currentSubgraphs$ **do**
4 $\quad$ **for** $other\_sg \in neighbours(currentSubgraphs)$ **do**
5 $\quad\quad$ **if** $mappable(this\_sg, other\_sg)$ **then**
6 $\quad\quad\quad nextSolutions.add(Solution(this\_sg, other\_sg))$
7 $\quad\quad$ **end**
8 $\quad$ **end**
9 $\quad$ **if** $notMatched(subgraph)$ **then**
10 $\quad\quad finalGraphs.add(subgraph)$
11 $\quad$ **end**
12 **end**
13 **return** $nextSolutions$

(lines 3–12). Figure 7 illustrates this merging process. If a subgraph cannot be merged with any other subgraph due to crossbar-capacity limits, it is considered a finalized cluster of nodes and edges.

Algorithm 3 iterates over these combinations, evaluating them in terms of the percentage of neurons on a crossbar that are shared,

Algorithm 3: Generation of Best Solutions

Input: $solutions$
Output: $graphs$
1 $graphs = \emptyset$
2 **while** $solutions \neq \emptyset$ **do**
3 $\quad solution = max(solutions)$
4 $\quad$ **if** $valid(solution)$ **then**
5 $\quad\quad graphs.add(subgraph(solution))$
6 $\quad$ **end**
7 **end**
8 **return** $graphs$

and in greedy fashion combines them to obtain even larger solutions. If a subgraph is already part of another solution, it is invalidated to ensure no overlaps between solutions. Lines 2–7 are repeated until no more new solutions can be generated. Figure 8 illustrates how shared synaptic inputs between neurons in the SNN affect the mapping solution.

We then repeat a similar step to pack any clusters that may fit in the same crossbar, regardless of connectivity. This means that if a crossbar can house the subgraph candidates while meeting the constraint requirements, then we can pack these two candidates into the same crossbar 11b.

These two steps conclude the mapping procedure for CROSS, with neurons mapped to the crossbar's columns.

4.3 Adapting Solution to CROSS+ Architecture

Using neuronal clusters generated for CROSS, we obtain the solution for CROSS+ by systematically mapping them to either rows or to columns. The following rules apply when deciding on the placement: if a cluster is placed on the rows, inputs to its presynaptic neurons must be placed in SRAM so that the generated outputs can be fed into the crossbar; and if a cluster is placed on the columns, its inputs must be from the crossbar, so that the outputs can be placed on the interconnect.

The mapping process starts by prioritizing clusters with the highest in-degree for placement on crossbars to optimize utilization. However, situations arise where two intercommunicating groups both land on crossbars, which is an infeasible configuration (Fig. 9a). To solve this, we adopt a two-step strategy. When crossbar-to-crossbar connections emerge, we transition the neurons from

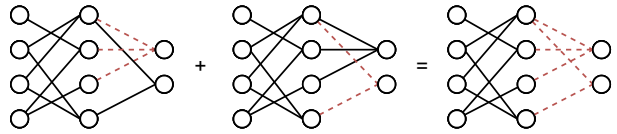
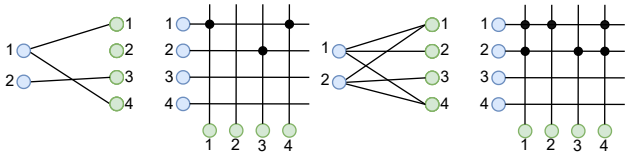
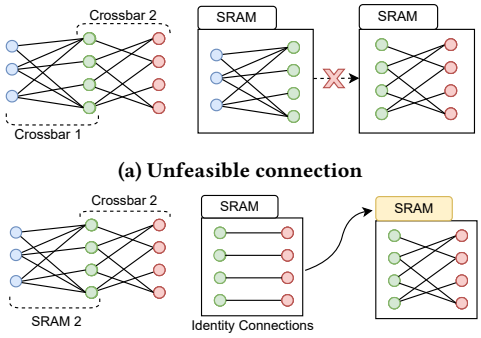


Figure 7: Merging minimum blocks.



(a) Example of low reuse **(b) Example of high reuse**

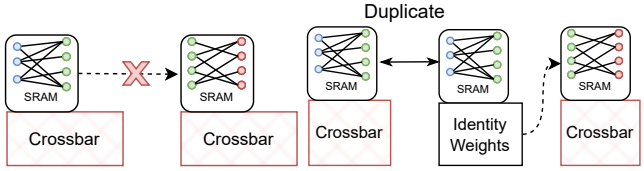
Figure 8: Examples of reuse of row inputs between neurons.



(a) Unfeasible connection

(b) Resolved connection

Figure 9: Resolving crossbar-to-crossbar communication.



(a) Unfeasible connection **(b) Resolved connection**

Figure 10: Resolving SRAM-to-SRAM communication.

the incoming crossbar's domain to the SRAM associated with the current crossbar. Subsequently, the crossbar linked with the incoming cluster is configured with an identity matrix. This arrangement ensures forward propagation of values from the SRAM (Fig. 9b). Conversely, when two communicating clusters both inhabit the SRAM of different crossbars, we instantiate a new node. In this scenario, source neurons are positioned on the SRAM, complemented by identity matrix connections on the crossbar. This configuration permits neurons values to be placed at the crossbar's output, facilitating routing to the SRAM of the target neurons (Fig. 10).

The complexity of generating the candidate solutions is $O(N^2)$, as we compare each pair of candidates. The complexity of greedily finding the best solution: is $O(N \log N)$ to insert solutions into max priority queue, $O(N)$ to loop through all solutions, and $O(1)$ to get the next best solution. So in total, our algorithm is of $O(N^2)$ time complexity.

Table 1: Number of 1024×1024 crossbars used by our method compared to baseline for various machine learning models. The architecture of each crossbar assumes the CROSS model.

Model	SpineMap (KL)	Ours	Utilization ratio
LeNet	316	211	**1.49x**
AlexNet	5276	5208	**1.43x**
ResNet	968	327	**2.82x**
DenseNet8	1074	703	**1.83x**

5 RESULTS

SNNs used to obtain the results reported in this section are generated as follows. The machine learning model of interest, implemented in Tensorflow, is converted to its SNN variant using snntoolbox. This SNN is simulated using PyCARL [3], which consists of a PyNN-based [10] frontend along with a CARLsim backend [19], to generate statistics related to spike traffic along the SNN's edges. These files are then processed by our mapping routines which are written in Python.

5.1 Comparison to KL-based Methods

Table 1 summarizes and compares the performance of the proposed clustering method with the baseline that uses KL clustering. Performance is quantified in terms of the number of 1024×1024 crossbars used by the competing methods under the CROSS model. *The architecture-aware clustering algorithm significantly reduces the number of required crossbars by an average of 1.9x.*

5.2 Detailed Evaluation of Proposed Method

The following metrics are used to measure quality of the solutions generated by our method:

- *Output-to-Input Ratio* (*OIR*) is the ratio, $N_{\text{output}}/N_{\text{input}}$, of the number of input and output neurons in the crossbar. This metric captures balanced usage of input and output ports in the crossbars. A low OIR indicates higher input-layer occupancy whereas a larger OIR indicates higher output-layer occupancy, A well-balanced crossbar increases the chances of merging it with other crossbars, while maintaining high utilization.
- *Average Pair-wise Jaccard Similarity* measures similarity in SRAM contents across all crossbars. When generating mapping for CROSS+, clusters placed on the input rows may have to be duplicated across multiple crossbars (Section 4.3). The Jaccard similarity between the SRAM contents of crossbars C_i and C_j can be calculated as

$$\text{JSim}(S(C_i), S(C_j)) = \frac{|S(C_i) \cap S(C_j)|}{|S(C_i) \cup S(C_j)|},$$

where $S(C_i)$ represents the SRAM contents of Cluster i. The average Jaccard Similarity between the SRAM contents of all pairs of crossbars is calculated as

$$\text{AJS} = \frac{1}{\frac{n(n-1)}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \text{JSim}(S(C_i), S(C_j)).$$

- *Crossbar Utilization* is the ratio of used neurons to total neurons in a crossbar, $N_{\text{used}}(C_i)/N_{\text{total}}(C_i)$.

Table 2: Results for various machine-learning models.

Model Information			# Clusters after each operation		
Name	Neurons	Synapses	Merging	CROSS	CROSS+
MLP-MNIST	894	79,400	2	1	1
LeNet	23,477	275,110	218	211	80
ResNet-8	126,056	5,155,078	366	128	116
AlexNet	256,570	3,843,103	5289	5208	2443
DenseNet	280,414	13,856,615	1007	703	274

- With *Jaccard Similarity* we also measure overlap between two subgraphs by checking the similarity in nodes. We use this cost function to promote solutions with large overlap, so that number of clusters can be minimized. The overlap can be calculated as

$$\text{Graph\_Overlap}(G_1, G_2) = \frac{|V(G_1) \cap V(G_2)|}{|V(G_1) \cup V(G_2)|},$$

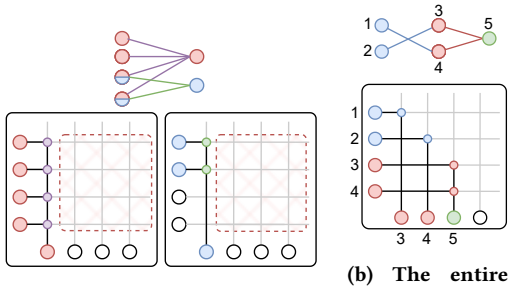
where $V(G_i)$ represents the set of vertices in graph G_i, and $|A|$ denotes the cardinality (size) of set A.

Table 2 summarizes the performance of our mapper for well-known SNNs when the capacity of each crossbar is fixed at 1024×1024. Number of crossbars needed to accommodate these SNNs on the CROSS and CROSS+ architectures is a function of network size, and the fan-in and fan-out degrees of individual neurons.

As a fundamental requirement, all presynaptic inputs incident on a neuron must be mapped onto a single crossbar. Given the nonlinear nature of a spiking neuron's activation function, we cannot simply accumulate activation values for a neuron if these inputs are provided by different clusters. Effective merging requires neighbouring neurons with a substantial overlap in their presynaptic inputs. When neighbouring neurons source their inputs from an large layer, the probability of overlapping input neurons is diminished due to the increased dimensionality of the input space. Consider a scenario with a crossbar designed to accommodate up to 256 neurons. Assume a fully connected SNN comprised of two layers, one with 512 neurons and the subsequent layer with 256 neurons. At first glance, a one-to-one mapping seems plausible. However, for each neuron in the second layer, we are dealing with 256 distinct input neurons. Selecting 256 neurons from a set of 512 equates to a total of $\binom{512}{256}$ possible combinations, thus reducing the likelihood of obtaining precise neuron matches. Even when partial overlaps are feasible, the constraints of the crossbar may prevent their co-location since the combined input count may exceed the capacity of 256. This illustrates an intrinsic limitation of crossbars when addressing neurons with vast input arrays. The most favorable scenario for SNN-to-hardware mapping arises when a crossbar has sufficient inputs and outputs, to house the entirety of the network (Fig. 11b). Nonetheless, many scenarios might culminate in crossbars operating at sub-optimal utilization levels (Fig. 11a).

One strategy to improve OIR involves systematic pruning of neurons exhibiting minimal outgoing traffic from the SNN, while balancing classification accuracy.[2] To ascertain the influence of pruning on the OIR of the resultant clusters, we perform a series of experiments. First, we compute the mean count of incoming edges for each neuron. For neurons surpassing this average, a specified

[2]In-depth exploration of this strategy is beyond the purview of this paper.



(a) Unable to merge neurons despite high overlap

(b) The entire SNN is mapped to the crossbar

Figure 11: Favorable and unfavorable mapping scenarios.

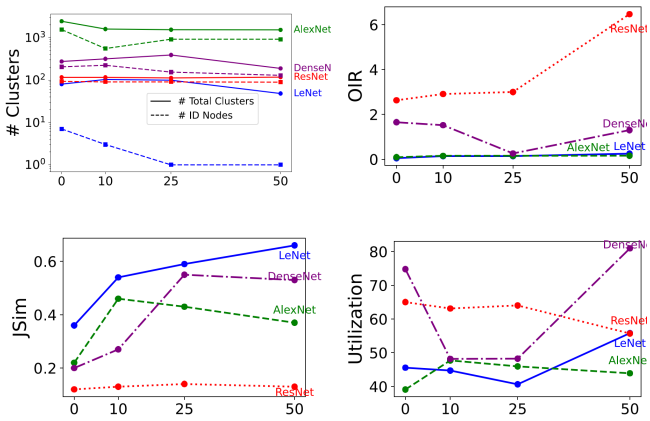


Figure 12: Effect of pruning on the performance metrics. X-axis shows the % of nodes removed from the original SNN.

percentage of their incoming connections are pruned. Notably, networks with higher input occupancy, such as LeNet or AlexNet, are more susceptible to this procedure compared to more evenly balanced networks like ResNet. The rationale behind this approach is to provide crossbars with more merging flexibility; reduced input connections naturally increase the potential for neuron merging.

Figure 12 (top left) shows the effect of pruning on the total number of generated clusters for each of the models (shown by solid lines). This includes the identity nodes as well (displayed separately using dashed lines). The reason behind the large number of identity nodes lies in the degree of outward communication for each cluster. Whenever a cluster placed in SRAM needs to communicate with another cluster, also in SRAM, an intermediary crossbar is needed to host the source cluster with an identity matrix so that the SRAM's contents can be passed through to the interconnect to the destination crossbar. Pruning leads to significant savings for LeNet. If SNN layers are comparable in size, it is likely to result in highly utilized crossbars. If utilization is initially low, pruning the fan-in of neurons may improve it. For a model like ResNet, however, with high crossbar utilization and OIR, pruning may actually decrease solution quality. So, careful analysis is needed before clustering to employ correct pruning techniques. The OIR also depends on the ratio of the neurons per each layer and pruning improves this metric. Finally, our mapping methodology is greedy in that it discards

sub-optimal local solutions in favor of optimal ones. However, sub-optimal local solutions may in fact lead to better global solutions. This explains the non-monotonic changes seen in our metrics, as final solutions may differ vastly based on the initial state.

6 RELATED WORK

Efforts dedicated to mapping SNNs to hardware predominantly use algorithms such as KL, particle swarm optimization, or other packing techniques. The primary objective of these methodologies is to cluster neurons cohesively, aiming to alleviate traffic congestion on the interconnect [2, 8, 21, 22]. However, these studies do not check to ensure that the resulting clusters can be effectively mapped onto the designated neuromorphic cores or crossbars. Li et al. develop an approach to progressively aggregate neurons within a core, but without considering the underlying topology of intra-cluster connections [15]. Xiao et al. further this line of research, accentuating interconnect mapping, but don't rectify the aforementioned limitation of ensuring mappable clusters [25]. Jin et al. focus their efforts on the placement and mapping of SNNs which are already partitioned onto network-on-chip systems, but not the basic task of mapping neurons to crossbars [14].

Sugiarto et al. develop a mapping framework for general-purpose applications represented as task graphs onto the SpiNNaker hardware [23]. Individual tasks are allocated to SpiNNaker chips with the goal of reducing inter-chip data traffic. To bolster fault tolerance, replicas of a task are executed across multiple SpiNNaker chips. An evolutionary algorithm aims to balance system load with minimized inter-chip data communication. The inherent limitation of these specialized methodologies is their tailoring to a particular hardware device. Dedicated compilers for specific platforms, which pursue hardware-tailored optimization routes, also exist. One example is NxTF for Intel's Loihi which uses a greedy, layer-wise optimization for its partitioning strategy, recognizing the impracticality of partitioning the vast combinatorial space [20]. NxTF improves hardware-resource efficiency by incorporating mechanisms like synapse and axon sharing, along with synapse compression. Contrarily, our pursuit is the development of a generalized algorithm adaptable to any crossbar-centric platform.

7 CONCLUSION

We developed an architecture-aware mapping methodology for SNNs which ensures that the generated clusters can be feasibly and effectively mapped onto crossbars. Our algorithms accommodate both a basic architecture, CROSS, where neurons can only be mapped to the crossbar's columns as well as CROSS+ that allows neurons to be mapped to both columns and rows. We evaluate performance using well-known machine-learning models, and discuss how the SNN's structure impacts solution quality along with approaches to improve it.

8 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No 2209745.

REFERENCES

[1] Aayush Ankit, Abhronil Sengupta, Priyadarshini Panda, and Kaushik Roy. 2017. RESPARC: A Reconfigurable and Energy-Efficient Architecture with Memristive Crossbars for Deep Spiking Neural Networks. In *Proceedings of the 54th Annual Design Automation Conference 2017 (DAC '17)*. ACM.
[2] Adarsha Balaji et al. 2020. Mapping Spiking Neural Networks to Neuromorphic Hardware. *IEEE Trans. VLSI Syst.* 28, 1 (2020), 76–86.
[3] Adarsha Balaji et al. 2020. PyCARL: A PyNN Interface for Hardware-Software Co-Simulation of Spiking Neural Network. In *IJCNN*.
[4] 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* 2, 1 (2017), 89–124.
[5] Elisabetta Chicca et al. 2003. A VLSI recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory. *IEEE Trans. Neural Networks* 14, 5 (2003).
[6] Federico Corradi et al. 2021. Gyro: A Digital Spiking Neural Network Architecture for Multi-Sensory Data Analytics. In *Proc. ACM Drone Syst. Eng. & Rapid Simulation & Perf. Eval.* 9–15.
[7] Anup Das et al. 2018. Mapping of local and global synapses on spiking neuromorphic hardware. In *Design, Automation Test in Europe Conference (DATE)*. 1217–1222.
[8] Anup Das et al. 2018. Mapping of local and global synapses on spiking neuromorphic hardware. In *DATE*. 1217–1222.
[9] Mike Davies et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.
[10] Andrew P Davison. 2008. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics* 2 (2008).
[11] Samanwoy Ghosh-Dastidar et al. 2009. Spiking neural networks. *J. Neural Systems* (2009), 295–308.
[12] Miao Hu et al. 2014. Memristor Crossbar-Based Neuromorphic Computing System: A Case Study. *IEEE Trans. Neural Networks & Learning Syst.* 25, 10 (2014), 1864–1878.
[13] E. M. Izhikevich. 2003. Simple model of spiking neurons. *IEEE Trans. Neural Networks* 14, 6 (Nov 2003), 1569–1572.
[14] Ouwen Jin et al. 2023. Mapping Very Large Scale Spiking Neuron Network to Neuromorphic Hardware. In *ACM ASPLOS*.
[15] Shiming Li et al. 2020. SNEAP: A Fast and Efficient Toolchain for Mapping Large-Scale Spiking Neural Network onto NoC-based Neuromorphic Platform. In *Proc. ACM Great Lakes VLSI Symp.*
[16] Chenchen Liu et al. 2015. A Spiking Neuromorphic Design with Resistive Crossbar. In *Proc. ACM Annual Design Automation Conference (DAC)*.
[17] S. Moradi et al. 2018. A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs). *IEEE Trans. Biomedical Circuits & Systems* 12, 1 (Feb. 2018), 106–122.
[18] Lars Niedermeier et al. 2022. CARLsim 6: An Open Source Library for Large-Scale, Biologically Detailed Spiking Neural Network Simulation. In *Int'l IEEE Joint Conf. Neural Networks (IJCNN)*. 1–10.
[19] Lars Niedermeier et al. 2022. CARLsim 6: An Open Source Library for Large-Scale, Biologically Detailed Spiking Neural Network Simulation. In *IJCNN*.
[20] Bodo Rueckauer et al. 2022. NxTF: An API and Compiler for Deep Spiking Neural Networks on Intel Loihi. *ACM J. Emerging Tech. Comput. Syst.* (2022).
[21] Shihao Song et al. 2021. A Design Flow for Mapping Spiking Neural Networks to Many-Core Neuromorphic Hardware. In *ICCAD*. 1–9.
[22] Shihao Song et al. 2022. DFSynthesizer: Dataflow-based Synthesis of Spiking Neural Networks to Neuromorphic Hardware. *ACM Trans. Embedded Comput. Syst.* (2022).
[23] Indar Sugiarto et al. 2017. Optimized Task Graph Mapping on a Many-core Neuromorphic Supercomputer. In *HPEC 2017*.
[24] Lakshmi Varshika et al. 2022. Design of Many-Core Big Little μBrain for Energy-Efficient Embedded Neuromorphic Computing. In *DATE*.
[25] Chao Xiao et al. 2022. Optimal Mapping of Spiking Neural Network to Neuromorphic Hardware for Edge-AI. *Sensors* (2022).
[26] Qi Xu et al. 2023. Reliability-Driven Memristive Crossbar Design in Neuromorphic Computing Systems. *IEEE Trans. Automation Science & Eng.* 20, 1 (2023), 74–87.