Mapping Spiking Neural Networks to Neuromorphic Hardware

Adarsha Balaji, Anup Das, *Senior Member, IEEE*, Yuefeng Wu, Khanh Huynh, Francesco G. Dell'Anna, Giacomo Indiveri, Jeffrey L. Krichmar, *Senior Member, IEEE*, Nikil D. Dutt, *Fellow, IEEE*, Siebren Schaafsma, and Francky Catthoor, *Fellow, IEEE*

***Abstract*— Neuromorphic hardware implements biological neurons and synapses to execute a spiking neural network (SNN)-based machine learning. We present SpiNeMap, a design methodology to map SNNs to crossbar-based neuromorphic hardware, minimizing spike latency and energy consumption. SpiNeMap operates in two steps: SpiNeCluster and SpiNePlacer. SpiNeCluster is a heuristic-based clustering technique to partition an SNN into clusters of synapses, where intracluster local synapses are mapped within crossbars of the hardware and intercluster global synapses are mapped to the shared interconnect. SpiNeCluster minimizes the number of spikes on global synapses, which reduces spike congestion and improves application performance. SpiNePlacer then finds the best placement of local and global synapses on the hardware using a metaheuristic-based approach to minimize energy consumption and spike latency. We evaluate SpiNeMap using synthetic and realistic SNNs on a state-of-the-art neuromorphic hardware. We show that SpiNeMap reduces average energy consumption by 45% and spike latency by 21%, compared to the best-performing SNN mapping technique.**

***Index Terms*— Interspike interval (ISI), neuromorphic computing, spiking neural network (SNN).**

I. Introduction

NEUROMORPHIC hardware, such as TrueNorth [1], Loihi [2], and DYNAP-SE [3], can implement machine learning tasks [4]–[6] using spiking neural networks (SNNs) [7]–[9]. A typical neuromorphic hardware consists of artificial neurons, which generates spikes, when a neuron's action potential exceeds a threshold, and crossbars, which stores synaptic weights.

Manuscript received May 3, 2019; revised August 6, 2019; accepted September 3, 2019. Date of publication November 26, 2019; date of current version December 27, 2019. This work was supported by the National Science Foundation under Award CCF-1937419 (RTML: Small: Design of System Software to Facilitate Real-Time Neuromorphic Computing). *(Corresponding author: Anup Das.)*

A. Balaji and A. Das are with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104 USA (e-mail: anup.das@drexel.edu).

Y. Wu, K. Huynh, F. G. Dell'Anna, and S. Schaafsma are with Neuromorphic Division, Stichting IMEC Nederland, 5656 Eindhoven, The Netherlands.

G. Indiveri is with the Department of Neuroinformatics, University of Zurich, 8006 Zürich, Switzerland.

J. L. Krichmar is with the Department of Cognitive Sciences, University of California at Irvine, Irvine, CA 92697 USA.

N. D. Dutt is with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA.

F. Catthoor is with the Neuromorphic Division, imec, 3001 Leuven, Belgium.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2951493

To reduce energy consumption, the size of a crossbar is constrained, accommodating a limited number of synapses per neuron. To build a large chip, multiple crossbars are integrated together using a shared interconnect, such as Networks on Chip (NoCs) [10]. A large SNN must, therefore, be partitioned into synapses that are mapped inside crossbars (local synapses) and those that are mapped on the shared interconnect (global synapses) of the hardware. Unfortunately, a shared interconnect introduces latency, which distorts interspike intervals (ISIs) [11]. ISI distortion affects application performance, such as latency and accuracy (see Section II).

Recent works, such as [12]–[17], uses a single large crossbar to map SNNs. In Section V, we demonstrate the limitations of these techniques when used to map SNNs to a multicrossbar neuromorphic hardware, such as the DYNAP-SE. Techniques that explicitly address mapping to multicrossbar hardware are PACMAN [18], NEUTRAMS [19], and PSOPART [20].

Compared to PACMAN and NEUTRAMS that minimize crossbar usage, PSOPART minimizes the number of spikes on the shared interconnect. This optimization strategy reduces spike congestion and ISI distortion, which improves application performance. Unfortunately, PSOPART does not address the placement of local and global synapses to the physical resources of neuromorphic hardware. PSOPART is, therefore, limited to crossbars with shared bus interconnect.

A shared bus is a fundamental latency and energy bottleneck for large neuromorphic hardware, those that can map over a million synapses [21]. In recent years, many scalable interconnects are proposed. Examples include multistage NoC for TrueNorth [1] and segmented bus for DYNAP-SE [3]. For these emerging interconnects, PSOPART presents two key limitations. First, the synapse partitioning approach of PSOPART does not scale to large SNNs. Second, the synapse placement problem is not addressed in PSOPART, which contributes significantly to latency and energy consumption.

We present SpiNeMap, a comprehensive design methodology to map SNNs to multicrossbar neuromorphic hardware, minimizing energy consumption and spike latency on the shared interconnect and improving application performance.

Contributions: Following are our novel contributions.

1) *SpiNeCluster:* We propose a heuristic-based approach to partition SNNs into local and global synapses, reducing the number of spikes on the shared interconnect.
2) *SpiNePlacer:* We propose a metaheuristic-based approach to place local and global synapses on physical

TABLE I
SpiNeMap VERSUS STATE-OF-THE-ART APPROACHES

Techniques	Partitioning	Placement	Objective
[12]–[17]	×	×	Maximize single crossbar utilization
NEUTRAMS [19]	√	×	Minimize number of crossbars
PSOPART [20]	√	×	Minimize spikes on global synapses
SpiNeMap	√	√	Minimize energy consumption and latency of neuromorphic hardware

√ Optimized × Not optimized

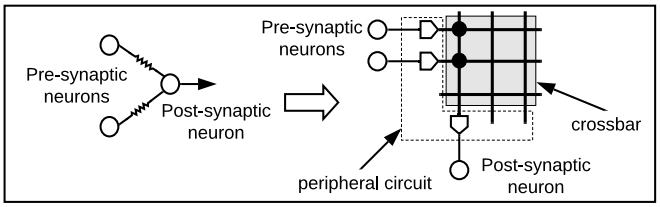


Fig. 1. Mapping an SNN to a crossbar.

resources of neuromorphic hardware, reducing energy consumption and spike latency.

3) We evaluate SpiNeMap on the DYNAP-SE neuromorphic hardware using synthetic and realistic SNNs.
4) We evaluate different interconnect topologies and spike routing algorithms for emerging neuromorphic hardware.

Table I compares our contributions against state-of-the-art techniques. We evaluate SpiNeMap with SNN-based applications on the DYNAP-SE hardware. We show that SpiNeMap reduces energy consumption by 45% and spike latency by 21% compared to the best-performing state-of-the-art techniques.

This article is organized as follows. We provide background in Section II. We describe the design methodology of SpiNeMap in Section III. We present our evaluation setup in Section IV and results in Section V. We describe related works in Section VI. We conclude this article in Section VII with an outlook on the design of future neuromorphic platforms.

II. BACKGROUND

Fig. 1 illustrates the mapping of an SNN to a crossbar. Spikes from a presynaptic neuron inject current into the crossbar, which is the product of spike voltage applied (i.e., input activation x_i) along the row with the conductance of the synaptic element at the cross point (i.e., synaptic weight w_{ij}). Current summations along columns are performed in parallel and implement the sums $\sum_j w_{ij}x_i$, needed for forward propagation of neuron excitation x_i. We focus on supervised machine learning tasks, where an SNN is first trained with representative examples and then deployed for inference with in-field data. Performance is measured using accuracy, which is assessed using ISIs [22]–[26].

To define ISI, we consider an SNN with N neurons and S synapses, which are excited with an input over some finite interval of time $[0, T]$. Neural activities in this time interval generate K spikes, which we organize based on their generation time and the source neuron as

$$\left\{t_1^1, t_2^1, \ldots, t_{k_1}^1\right\}, \left\{t_1^2, t_2^2, \ldots, t_{k_2}^2\right\}, \ldots, \left\{t_1^N, t_2^N, \ldots, t_{k_N}^N\right\} \quad (1)$$

where t_i^n is the time of the ith spike generated by the nth neuron and $K = \sum_{i=1}^{N} k_i$. The ISI of this spike train is [22]

$$I_i^n = t_i^n - t_{i-1}^n. \quad (2)$$

An application-level simulator, such as CARLsim [27], allows extracting the precise spike times from neurons and calculates the ISI using (2). However, such simulators do not incorporate hardware latencies. When an SNN is mapped to a neuromorphic hardware, ISI will be affected by: 1) the fixed latency within a crossbar to propagate current through synaptic elements and 2) the variable latency of time multiplexing on the shared interconnect. To incorporate these hardware latencies, we extract spike times at the synapse level rather than at the neuron level. This is because a synapse can encounter different latencies depending on whether it is mapped inside a crossbar (i.e., local synapse) or on the shared interconnect (i.e., global synapse). We represent the spike times on synapses as

$$\left\{\tau_1^1, \tau_2^1, \ldots, \tau_{k_1}^1\right\}, \left\{\tau_1^2, \tau_2^2, \ldots, \tau_{k_2}^2\right\}, \ldots, \left\{\tau_1^S, \tau_2^S, \ldots, \tau_{k_S}^S\right\} \quad (3)$$

where τ_j^s is the jth spike on sth synapse and spike timings in the set $\{\tau_j^s\}$ are obtained from spike timings in the set $\{t_i^n\}$. The ISI of this spike train is

$$I_j^s = \tau_j^s - \tau_{j-1}^s. \quad (4)$$

We use the notation δ_j^s to represent the latency of the jth spike on the sth synapse. The new ISI due to these latencies is

$$I_j^s|_{\text{new}} = \tau_j^s + \delta_j^s - \tau_{j-1}^s - \delta_{j-1}^s. \quad (5)$$

The change in ISI (called ISI distortion) is

$$I_j^s|_{\text{distortion}} = I_j^s|_{\text{new}} - I_j^s = \delta_j^s - \delta_{j-1}^s. \quad (6)$$

For local synapses, which are mapped within crossbars, all spikes have the same latency, i.e., $\delta_j^s = \delta_{j-1}^s$. Thus, the ISI distortion is zero. For global synapses, different spikes of the same synapse can have different latencies due to the varying congestion and routing paths on the shared interconnect. These are the synapses that contribute to ISI distortion, that is

$$I_j^s|_{\text{distortion}} = \begin{cases} 0, & \text{if } s \text{ is mapped inside a crossbar} \\ \delta_j^s - \delta_{j-1}^s, & \text{if } s \text{ is mapped on the shared interconnect.} \end{cases} \quad (7)$$

ISI distortion leads to unacceptable accuracy loss (see Section V). By reducing the number of spikes on global synapses, spike congestion can be lowered, which would reduce ISI distortion and improve application performance. This is precisely the intuition behind the optimization strategy in PSOPART [20] and also this article. The difference is that this article also addresses the placement problem, which further improves energy consumption and spike latency.

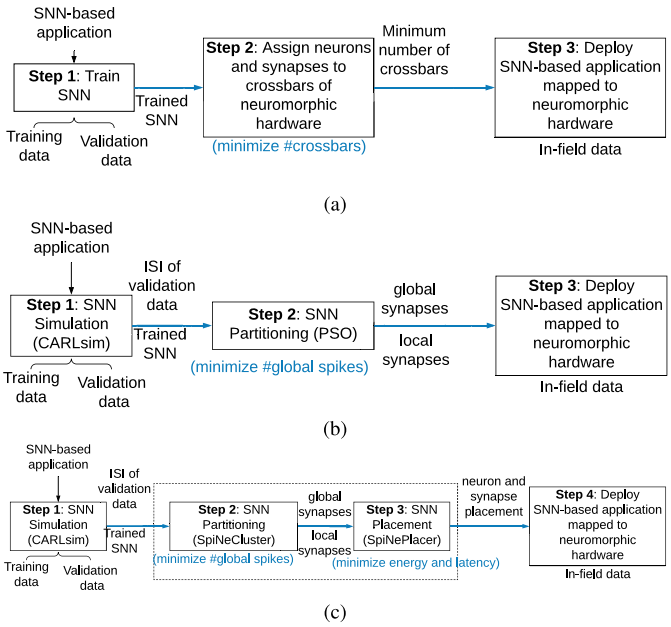


Fig. 2. High-level overview of our SpiNeMap mechanism and its difference with state of the art. (a) State-of-the-art approaches, e.g., NEUTRAMS [19]. (b) Our previous approach PSOPART [20]. (c) Our proposed design methodology SpiNeMap.

III. SpiNeMap: Mapping SNNs to Neuromorphic Hardware

A. High-Level Overview and Difference With State of the Art

Fig. 2(a) illustrates the design methodology of NEUTRAMS [19] and PACMAN [18], consisting of three steps: step 1) training an SNN model; step 2) mapping synapses to the hardware to minimize the number of crossbars; and step 3) deploying the SNN for inference.

Fig. 2(b) illustrates PSOPART [20], which minimizes the number of spikes on the shared interconnect in step 2 using an instance of the particle swarm optimization (PSO) [28].

Fig. 2(c) illustrates the proposed SpiNeMap methodology. SpiNeMap extracts the precise times of spikes by simulating an SNN in CARLsim. This spike information (called spike trace) is first used by SpiNeCluster to partition the SNN into local and global synapses, minimizing the number of spikes on the shared interconnect. The partitioned SNN and the spike trace are then used in SpiNePlacer to minimize the latency and energy consumption. Overall, the SNN partitioning and placement steps jointly improve the application performance, energy consumption, and spike latency.

B. Detailed Design of SNN Partitioning via SpiNeCluster

Fig. 3 illustrates an SNN partitioned into three clusters: A, B, and C. The number of spikes communicated between a pair of neurons is indicated on its synapse. We also indicate the local synapses in black and the global ones in blue in Fig. 3. The number of spikes on global synapses is 8.

We introduce the following notations for SpiNeCluster. Let $\mathcal{G}(\mathcal{N}, \mathcal{S})$ be an SNN with a set $\mathcal{N}$ of neurons and a set $\mathcal{S}$ of synapses. A synapse $s_{i,j}$ connects neuron n_i with n_j.

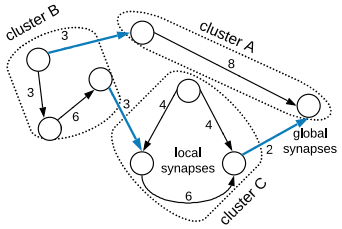


Fig. 3. SNN partitioned into local and global synapses.

Algorithm 1 SNN Clustering Algorithm

```
1  foreach C_i, C_j ∈ C do
       /* iterate over all cluster pairs                    */
       /* begin 2-part procedure                            */
2      gs = total spikes between C_i and C_j;
3      while True do
4          foreach n_i ∈ C_i and n_j ∈ C_j do
5              if n_i and n_j are not previously selected then
6                  Move n_i to C_j and calculate gs_1;
7                  Move n_j to C_i and calculate gs_2;
8                  Swap n_i and n_j and calculate gs_3;
9                  Select the option which lowers gs;
10                 Return new partitions C'_i, C'_j;
11             end
12         end
13         gs' = total spikes between C'_i and C'_j;
14         if gs' < gs then
15             gs = gs' and break;
16         end
17     end
       /* end 2-part procedure                              */
18 end
```

We partition this SNN into k clusters. Let $\mathcal{H}(\mathcal{C}, \mathcal{E})$ be the partitioned SNN with a set $\mathcal{C}$ of clusters and a set $\mathcal{E}$ of global synapses. Transforming $\mathcal{G}(\mathcal{N}, \mathcal{S}) \rightarrow \mathcal{H}(\mathcal{C}, \mathcal{E})$ is a classical graph partitioning problem [29] and has been applied in many contexts, including task mapping on multiprocessor systems [30]. Graph partitioning is an NP-complete problem [31], [32]; heuristics are typically used to find solutions. PSOPART [20] uses an instance of PSO [33] to solve this problem. However, the search space soon becomes intractable as the size of the SNN increases. To address this limitation, we propose an alternative greedy approach, roughly based on the Kernighan–Lin graph partitioning algorithm [29], which we show to be scalable to large SNNs.

We set $k = \lceil (|\mathcal{N}|)/(n_c) \rceil$, where n_c is the average number of neurons that can be accommodated within a crossbar. Next, we evenly (and arbitrarily) distribute neurons to these k clusters. Next, we iteratively swap neurons between clusters to minimize the number of spikes on global synapses.

We formalize these steps in Algorithm 1. The algorithm applies a two-part procedure (lines 2–17) to every cluster pair (with a total of $\binom{k}{2}$ iterations). In the two-part procedure, we first calculate the total number of the intercluster spike (gs) with the two clusters (line 2). Next, we select a pair of neurons n_i and n_j from the two selected clusters C_i and C_j, respectively, such that neither n_i nor n_j is selected in the previous iterations (lines 4 and 5). We then perform three operations: 1) move $n_i \in C_i$ to cluster C_j (if C_j can accommodate more neurons) (line 6); 2) move $n_j \in C_j$ to cluster C_i (if C_i can accommodate more neurons) (line 7); and 3) swap n_i and n_j (line 8). We calculate the number of intercluster spike for each

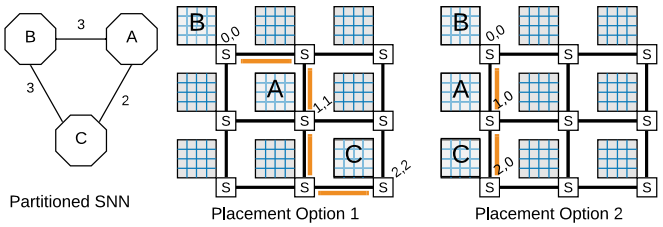


Fig. 4. Illustrating the impact of different placements of clusters of a partitioned SNN on a neuromorphic hardware.

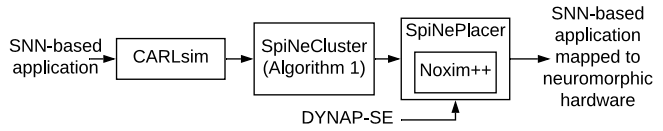


Fig. 5. Our design methodology: SpiNeMap.

of these operations and select the option that generates the maximum reduction of an intercluster spike compared to gs (line 9). We return the new clusters (line 10). We repeat the procedure (lines 4–13), while the number of intercluster spike continues to be reduced (lines 14–16).

1) Time Complexity: Lines 2–17 are executed $\binom{k}{2}$ times, with lines 4–16 executed at every iteration. Since a cluster can accommodate n_c neurons, the time complexity of Algorithm 1 is $O(\binom{k}{2} \times n_c * n_c) = O(k^2 \times n_c^2) = O(|\mathcal{N}|^2)$.

C. Detailed Design of SNN Placement via SpiNePlacer

Fig. 4 illustrates two alternate placements of a partitioned SNN (from SpiNeCluster) to the hardware. We show nine crossbars arranged in a 3×3 mesh topology. Different placements of clusters lead to different utilizations of interconnect segments, which impacts both energy consumption and latency. Clearly, cluster placement problem can no longer be ignored for large neuromorphic hardware (a common limitation of NEUTRAMS [19], PACMAN [18], Eyeriss [34], and PSOPART [20]).

To perform design-space explorations for cluster placement, we extend the Noxim [35] simulator to support: 1) simulation of spike traces from CARLsim; 2) simulation of current and emerging interconnect topologies of neuromorphic hardware; 3) simulation of different routing algorithms; and 4) technology-specific energy and latency of interconnect wires and switches. We call our new framework *Noxim++*.

Fig. 5 illustrates our design methodology SpiNeMap. Noxim++ is integrated in the SpiNePlacer and configured to model the DYNAP-SE neuromorphic hardware [3].

To formalize the optimization problem of SpiNePlacer, we consider the mapping of a clustered SNN $\mathcal{H}(\mathcal{C}, \mathcal{E})$ to the neuromorphic hardware $\mathcal{A}(\mathcal{V}, \mathcal{I})$, where $\mathcal{V}$ is the set of crossbars in the hardware and $\mathcal{I}$ is the set of connections of these crossbars for a given interconnect topology.

Mapping $M : \mathcal{H}(\mathcal{C}, \mathcal{E}) \rightarrow \mathcal{A}(\mathcal{V}, \mathcal{I})$ is specified by a logical matrix $(m_{ij}) \in \{0, 1\}^{|\mathcal{C}| \times |\mathcal{V}|}$, where m_{ij} is defined as

$$m_{ij} = \begin{cases} 1, & \text{if cluster } c_i \in \mathcal{C} \text{ is mapped to crossbar } v_j \in \mathcal{V} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

The mapping constraints are the following.

1) A cluster can be mapped to only one crossbar, that is

$$\sum_j m_{ij} = 1 \quad \forall i. \tag{9}$$

2) A crossbar can accommodate at most one cluster, that is

$$\sum_i m_{ij} \leq 1 \quad \forall j. \tag{10}$$

In our design methodology, Noxim++ is used to generate mapping that minimizes spike latency and energy consumption on the interconnect. These are computed as follows.

1) *Average spike latency:* This is the average delay experienced by spikes on the interconnect, that is

$$L = \sum_{i=1}^{N_s} [(h_i - 1) * l_w + h_i * l_s]/N_s \tag{11}$$

where h_i is the number of hops a spike traverses between the source and destination, l_w is the interconnect segment delay, and l_s is the delay of the hop.

2) *Total energy consumption:* This is the total energy consumed by all spikes on the interconnect, that is

$$E = \sum_{i=1}^{N_s} [(h_i - 1) * e_w + h_i * e_s] \tag{12}$$

where e_w and e_s are the energy consumption on the wires and hops, respectively.

To minimize the latency and energy consumption, we minimize the average number of hops that spikes communicate before reaching their destination [36]. This is obtained using Noxim++ for mapping M_i as $\mathcal{L}_i = \text{Noxim++}(M_i) = \sum_{j=1}^{N_s} h_j/N_s$. This is the fitness function of SpiNePlacer, which finds the mapping with minimum average hop count, that is

$$\mathcal{L}_{\min} = \mathcal{L}_a, \text{ where } a = \arg\min\{\text{Noxim++}(M_i) | i \in 1, 2, \cdots\}. \tag{13}$$

We use an instance of PSO [28] to find the optimum mapping. We instantiate n_p swarm particles. The positions of these particles are solutions to the fitness functions, and they represent cluster mappings, i.e., M's in (13). Each particle also has a velocity with which it moves in the search space to find the optimum solution. During the movement, a particle updates its position and velocity according to its own experience (closeness to the optimum) and also experience of its neighbors. We introduce the following notations:

$$\begin{aligned} D &= |\mathcal{C}| \times |\mathcal{V}| = \text{dimensions of the search space} \\ \Theta &= \{\theta_l \in \mathbb{R}^D\}_{l=0}^{n_p-1} = \text{positions of particles in the swarm} \\ \mathbf{V} &= \{\mathbf{v}_l \in \mathbb{R}^D\}_{l=0}^{n_p-1} = \text{velocity of particles in the swarm.} \end{aligned} \tag{14}$$

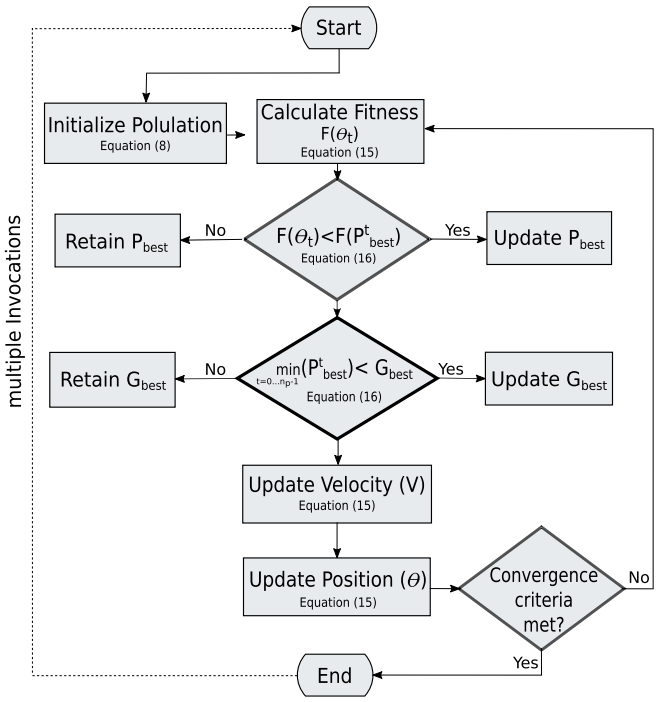


Fig. 6. Flowchart of our PSO algorithm.

Position and velocity of swarm particles are updated, and the fitness function is computed as

$$\begin{aligned}\Theta(t+1) &= \Theta(t) + \mathbf{V}(t+1)\\ \mathbf{V}(t+1) &= \mathbf{V}(t) + \varphi_1 \cdot (P_{\text{best}} - \Theta(t)) + \varphi_2 \cdot (G_{\text{best}} - \Theta(t))\\ F(\theta_l) &= \mathcal{L}_l = \text{Noxim++}(M_l)\end{aligned} \tag{15}$$

where t is the iteration number, φ_1 and φ_2 are constants, and P_{best} (and G_{best}) is the particles own (and neighbors) experience. Finally, local and global bests are updated as

$$\begin{aligned}P^l_{\text{best}} &= F(\theta_l) \text{ if } F(\theta_l) < F\left(P^l_{\text{best}}\right)\\ G_{\text{best}} &= \min_{l=0,\ldots,n_p-1} P^l_{\text{best}}.\end{aligned} \tag{16}$$

Due to the binary formulation of the mapping problem [see (8)], we need to binarize the velocity and position of (14), which we illustrate in the following:

$$\begin{aligned}\hat{\mathbf{V}} &= \texttt{sigmoid}(\mathbf{V}) = \frac{1}{1+\mathrm{e}^{-\mathbf{V}}}\\ \hat{\Theta} &= \begin{cases}0, & \text{if } \texttt{rand()} < \hat{\mathbf{V}}\\ 1, & \text{otherwise .}\end{cases}\end{aligned} \tag{17}$$

In finding a new position of a PSO particle, we use the two constraints: (9) and (10).

1) PSO Algorithm: Fig. 6 illustrates the PSO algorithm. The algorithm first initializes positions of the PSO particles (8) satisfying constraints (9) and (10). Next, the algorithm runs for n_{ISO} iterations. At each iteration, the PSO algorithm evaluates the fitness function (F) and updates its position based on the local and global best positions [see (15)], binarizing these updates using (17). The time complexity of the PSO algorithm is, therefore, $O(n_{\text{ISO}} \times$ operations in each iteration), where the operations in each iteration are proportional to the PSO dimension $D = |\mathcal{C}| \times |\mathcal{V}|$ and the number of particles n_p. The overall time complexity is $O(n_{\text{ISO}} \times n_p \times |\mathcal{C}| \times |\mathcal{V}|)$.

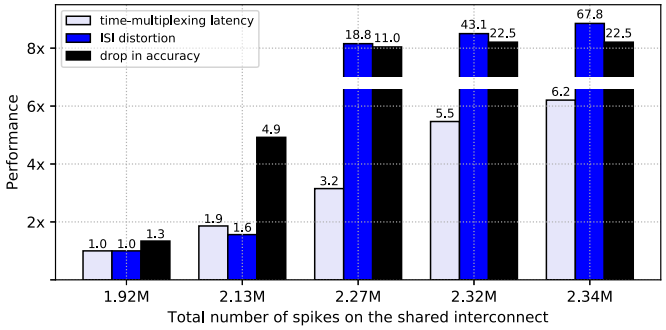


Fig. 7. Latency, ISI distortion, and accuracy as a function of the number of spikes on the shared interconnect for the handwritten digit-recognition example.

D. Justification of SpiNeMap's Design Choices

In this section, we motivate SpiNeMap's design choices.

1) Minimize Spike Count at the Partitioning Stage: SpiNeMap minimizes the number of spikes at the partitioning stage. To motivate this optimization objective, Fig. 7 plots the latency, ISI distortion, and drop in accuracy of the handwritten digit-recognition application for different mapping strategies generating a different number of spikes on the shared interconnect. The baseline hardware is the DYNAP-SE, with four crossbars organized in a 2×2 mesh with the XY routing algorithm. Each crossbar can accommodate 256 neurons.

We observe that as the number of spikes on the shared interconnect increases, the latency increases, increasing the ISI distortion. This lowers the application accuracy. We observe similar behavior for other applications as well.

2) Integration of Noxim++ Within PSO: The average spike hop count depends on: 1) the cluster mapping M and 2) the routing algorithm that dynamically routes spikes on the interconnect to avoid the congestion of interconnect links. Our PSO incorporates cluster mapping in the fitness function. Due to the dynamic nature of spike routing for congestion avoidance, we need to simulate the cycle-accurate behavior of interconnect for every mapping with the spike trace generated from CARLsim. This allows us to accurately compute the hop distance that each spike traverses before reaching its destination. This motivates our strategy to integrate Noxim++ within PSO to minimize the average hop count.

3) Using PSO Only for SpiNePlacer: PSOPART uses PSO for SNN partitioning (equivalent of SpiNeCluster). In this article, we use PSO only for SpiNePlacer and a greedy approach for SpiNeCluster. The rationale behind this is as follows. Had PSO been used for SpiNeCluster, the total number of dimensions for each particle in the PSO would be $D = |\mathcal{N}| \times |\mathcal{C}|$. The total number of dimensions of each particle in the PSO of SpiNePlacer is $D = |\mathcal{C}| \times |\mathcal{V}|$. In Table II, we compare these dimensions for different SNN sizes, with a fixed neuromorphic hardware (16 256-neuron crossbars).

As we can clearly see from Table II, the PSO problem of partitioning soon becomes intractable for a modest-sized SNN

TABLE II

DIMENSIONS OF PSO TO SOLVE PARTITIONING AND PLACEMENT PROBLEMS FOR DIFFERENT SNN SIZES ON A FIXED NEUROMORPHIC HARDWARE WITH 16 CROSSBARS AND 256 NEURONS EACH

# of SNN neurons	PSO dimensions (D) for	
	SNN partitioning	SNN placement
1,000	16,000	64
2,000	32,000	128
3,000	48,000	192
4,000	64,000	256

even if we restrict to 1000 particles (each with dimensions D) in the swarm. To keep the solution time reasonable, we, therefore, use PSO only for the placement problem (namely, SpiNePlacer) and greedy approach instead of the partitioning problem (namely, SpiNeCluster).

IV. EVALUATION METHODOLOGY

We build SpiNeMap with the following system components.

1) *CARLsim [27]:* A GPU-accelerated simulator used to train and test SNN-based applications. CARLsim reports spike times for every synapse in the SNN.
2) *Noxim++ [35]:* A trace-driven and cycle-accurate interconnect simulator for multiprocessor systems. We extend it: 1) to incorporate crossbar-based architectures; 2) to communicate spikes packets; and 3) to generate key performance statistics, such as energy, latency, and ISI distortion. Noxim++ uses spike traces from CARLsim to compute these statistics.
3) *DYNAP-SE [3]:* We use Noxim++ to model DYNAP-SE, with 256-neuron crossbars interconnected using the multistage NoCs. Technology parameters are obtained from [37] for 45-nm technology node [38].

A. Simulation Environment

We conduct all experiments on a system with eight CPUs, 32-GB RAM, and NVIDIA Tesla GPU, running Ubuntu 16.04.

B. Evaluated Applications

Table III reports seven synthetic and eight realistic SNN applications used for evaluation. The synthetic applications are indicated with the letter "S" followed by a number (e.g., S_1000), where the number represents the total number of neurons in the application. Column 3 reports the number of synapses in these applications. Column 4 reports the SNN topology.

The realistic applications are image smoothing (ImgSmooth) [27] on 64 × 64 images, edge detection (EdgeDet) [27] on 64 × 64 images using the difference-of-Gaussian, multilayer perceptron (MLP)-based handwritten digit recognition (MLP-MNIST) [9] on 28 × 28 images of handwritten digits, ECG-based heart-rate estimation (HeartEstm) [39], ECG-based heart-beat classification (HeartClass) [40], CNN-based digit classification (CNN-MNIST) [41], [42], CNN-based digit classification with LeNet (LeNet-MNIST) [42], and CNN-based CIFAR image classification with LeNet (LeNet-CIFAR) [42]. The last three applications are part of the MLPerf benchmark suite [42] and developed for analog computation model. We converted these applications into spike-based model using the CNN-to-SNN conversion tool N2D2 [43], [44].

TABLE III

APPLICATIONS USED FOR EVALUATING SPINEMAP

Category	Applications	Synapses	Topology	Spikes
synthetic	S_1000	240,000	FeedForward (400, 400, 100)	5,948,200
	S_1500	300,000	FeedForward (500, 500, 500)	7,208,000
	S_2000	640,000	FeedForward (800, 400, 800)	45,807,200
	S_2500	1,440,000	FeedForward (900, 900, 700)	66,972,600
	S_3000	2,000,000	FeedForward (1000, 1000, 1000)	155,123,000
	S_3500	2,500,000	FeedForward (1000, 1000, 1500)	46,476,000
	S_4000	3,750,000	FeedForward (1500, 1500, 1000)	149,580,500
realistic	ImgSmooth [27]	136,314	FeedForward (4096, 1024)	17,600
	EdgeDet [27]	272,628	FeedForward (4096, 1024, 1024, 1024)	22,780
	MLP-MNIST [9]	79,400	FeedForward (784, 100, 10)	2,395,300
	HeartEstm [39]	636,578	Recurrent	3,002,223
	HeartClass [40]	2,396,521	CNN[1]	1,036,485
	CNN-MNIST [41]	159,553	CNN[2]	97,585
	LeNet-MNIST [41]	1,029,286	CNN[3]	165,997
	LeNet-CIFAR [41]	2,136,560	CNN[4]	589,953

[1] Input(82x82) - [Conv, Pool]*16 - [Conv, Pool]*16 - FC*256 - FC*6
[2] Input(24x24) - [Conv, Pool]*16 - FC*150 - FC*10
[3] Input(32x32) - [Conv, Pool]*6 - [Conv, Pool]*16 - Conv*120 - FC*84 - FC*10
[4] Input(32x32x3) - [Conv, Pool]*6 - [Conv, Pool]*6 - FC*84 - FC*10

C. Evaluated State-of-the-Art Techniques

We evaluate the following four approaches.

1) The baseline [19] minimizes the use of crossbars.
2) The SCO [15] balances crossbar occupancy.
3) The PSOPART minimizes the total number of spikes on the shared interconnect.
4) The SpiNeMap uses: 1) SpiNeCluster to partition SNNs into clusters and 2) SpiNePlacer to place these clusters to crossbars of the hardware. SpiNeMap minimizes energy consumption and latency on the shared interconnect.

D. Evaluated Metrics

We evaluate the following metrics.

1) *Total number of spikes:* This is the number of spikes (N_s) on the shared interconnect post crossbar placement.
2) *Spike latency:* This is computed using (11).
3) *Energy consumption:* This is computed using (12).
4) *Average ISI distortion:* This is computed using (7), averaged over all spikes, that is

$$I = \sum_{i=1}^{N_s} I_i|_{\text{distortion}}/N_s. \tag{18}$$

V. RESULTS AND DISCUSSION

A. Summary of Results

Table IV summarizes our results.

B. Energy Consumption on the Shared Interconnect

Fig. 8 reports the energy consumption of each of our applications for each of our evaluated systems normalized to the baseline. We make the following three observations.

TABLE IV
SUMMARY OF RESULTS

SpiNeMap	Energy Consumption (Sec. V-B)	Spike Latency (Sec. V-C)	ISI Distortion (Sec. V-D)	Application Accuracy (Sec. V-E)
vs. Baseline [19]	45%	21%	36%	12%
vs. SCO [15]	40%	27%	39%	20%
vs. PSOPART [20]	20%	13%	23%	5%

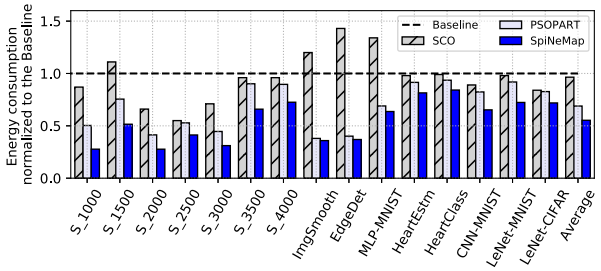


Fig. 8. Energy consumption normalized to the baseline.

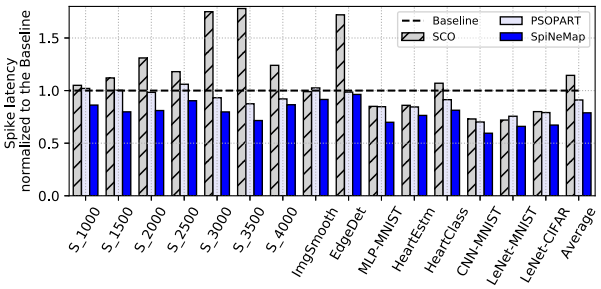


Fig. 9. Spike latency normalized to the baseline.

First, the average energy consumption of SCO is similar to the baseline. Second, PSOPART has an average 31% lower energy consumption than the baseline. This reduction is because PSOPART minimizes the total number of global spikes, which reduces the energy consumption on the shared interconnect [see (12)]. Third, SpiNeMap has the lowest energy consumption of all our evaluated systems (on average, 45% lower than baseline, 40% lower than SCO, and 20% lower than PSOPART). These improvements are because of SpiNeMap's optimization policies: 1) SpiNeCluster that reduces the total number of spikes on the shared interconnect and 2) SpiNePlacer that places these clusters on crossbars to minimize energy consumption.

C. Spike Latency on the Shared Interconnect

Fig. 9 reports the spike latency of each of our applications for each of our evaluated systems normalized to the baseline. We make the following three observations.

First, the average spike latency of SCO is 14% higher than the baseline. Second, PSOPART has 9% lower average spike latency than baseline. This improvement is because PSOPART reduces the total number of spikes on the shared interconnect, which reduces spike congestion and latency. Third, SpiNeMap has the lowest average spike latency among all our evaluated

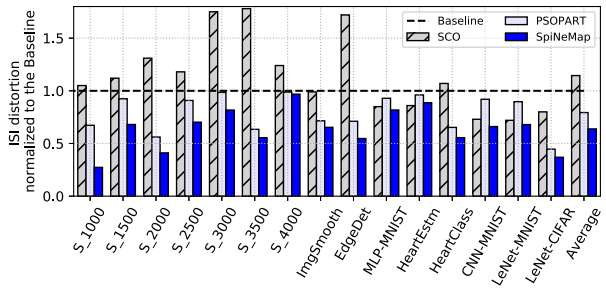


Fig. 10. ISI distortion normalized to the baseline.

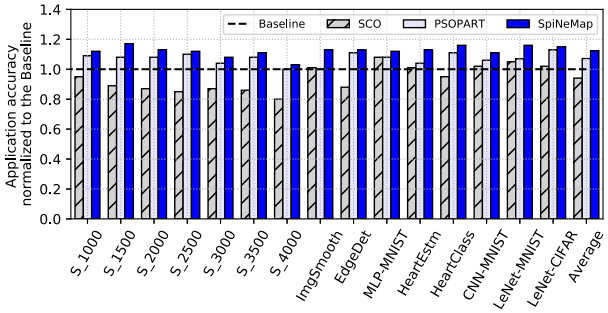


Fig. 11. Application accuracy normalized to the baseline.

systems (21% lower than baseline, 27% lower than SCO, and 13% lower than PSOPART). These improvements are due to SpiNeMap's optimization policies: 1) SpiNeCluster that reduces the number of spikes and 2) SpiNeCluster that minimizes the average number of hop counts [see (11)].

D. ISI Distortion on the Shared Interconnect

Fig. 10 compares the ISI distortion of each of our applications for each of our evaluated systems normalized to the baseline. We make the following three observations.

First, ISI distortion of SCO is, on average, 12% higher than the baseline. Second, PSOPART has 21% lower average ISI distortion than baseline. This reduction is due to the reduction of the number of spikes (see Section V-F). Third, SpiNeMap has the lowest ISI distortion of all our evaluated systems (36% lower than baseline, 39% lower than SCO, and 23% lower than PSOPART). The improvement with respect to PSOPART is because of our new SpiNePlacer step (see Fig. 2), which reduces ISI distortion by reducing spike latency.

E. Application Accuracy

Fig. 11 reports the accuracy of each of our applications for each of our evaluated systems normalized to the baseline. We observe that the accuracy results directly correlate with ISI distortion (see Section V-D). Accuracy of SCO is lower than baseline by an average 6%. PSOPART has an average 7% higher accuracy than baseline due to the 17% reduction in ISI distortion. SpiNeMap has the highest accuracy among all our evaluated systems (on average, 12% higher than baseline, 20% higher than SCO, and 5% higher than PSOPART).

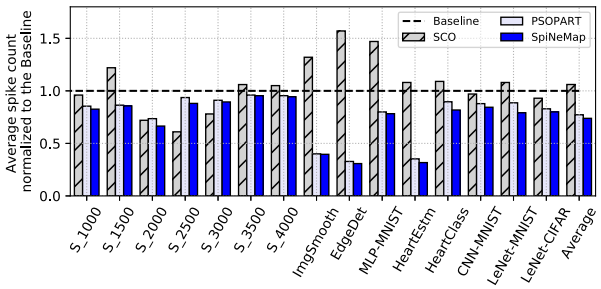


Fig. 12. Spike count normalized to the baseline.

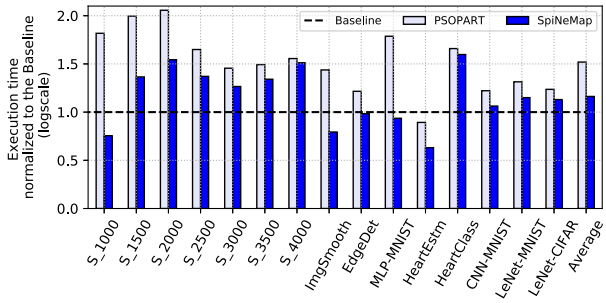


Fig. 13. Execution time normalized to the baseline.

F. Spike Count on the Shared Interconnect

Fig. 12 reports the total number of spikes communicated on the shared interconnect of each of our applications for each of our evaluated systems normalized to the baseline. We make the following three observations.

First, SCO has an average 6% higher spike count compared to baseline. These extra spikes increases energy consumption (see Section V-B). Second, PSOPART has, on average, 23% lower spikes than baseline due to its PSO-based clustering. Third, SpiNeMap generates the lowest number of spikes (26% lower than baseline, 24% lower than SCO, and 9% lower than PSOPART) The improvement over PSOPART is due to the greedy approach of Algorithm 1, which outperforms PSO for large application use cases.

G. Optimization Time

Fig. 13 compares execution time of our new clustering algorithm (see Algorithm 1) against the PSO-based clustering of PSOPART normalized to the baseline. We observe that SpiNeCluster has an average 3× lower execution time than PSOPART. Moreover, SpiNeCluster generates lower spikes on the interconnect and reduces energy consumption and latency. We conclude that SpiNeCluster is scalable and better than PSO for solving the clustering problem.

H. Interconnect Design-Space Explorations

Fig. 14 illustrates explorations of interconnect for neuromorphic hardware. We compare the NoC interconnect with XY (used in DYNAP-SE), NorthLast, and WestFirst routing and the segmented bus [45] interconnect (used in the next generation of DYNAP-SE) for all our evaluated workloads.

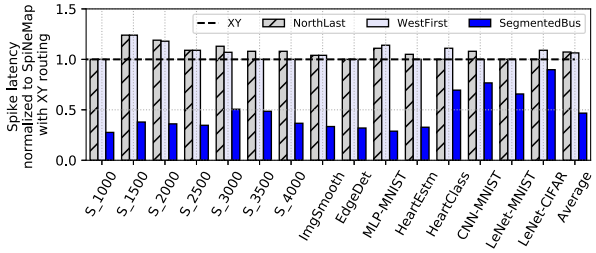


Fig. 14. Interconnect exploration using SpiNeMap.

We observe that NorthLast and WestFirst routings have an average 7% and 4% higher latency than XY routing, respectively. The segmented bus has the lowest spike latency among all (average 54% lower than NoC with XY routing). Lower spike latency leads to lower energy consumption and higher application performance. We have open-sourced SpiNeMap to allow design-space explorations on emerging interconnect strategies and routing algorithms for neuromorphic hardware.

VI. Related Works

This is the first work that jointly addresses the partitioning and placement of SNNs on crossbar-based neuromorphic hardware, minimizing the energy consumption, spike latency, and ISI distortion and improving application accuracy.

A. SNN-Based Machine Learning

Recently, machine learning tasks are designed using SNNs to improve energy efficiency. Verstraeten *et al.* [46] propose reservoir computing with SNNs for speech recognition. Grzyb *et al.* [47] use spiking liquid-state machine for facial recognition. Diehl and Cook [9] propose handwritten digit recognition using SNNs. Das *et al.* [39] propose spiking liquid-state machine for heart-rate estimation. We evaluate SpiNeMap using these applications.

Analog neural networks, such as convolutional neural networks (CNNs), have been immensely successful in computer vision tasks. The machine learning database MLPerf [42] provides a comprehensive collection of these applications. We converted these applications to spike model using N2D2 [43] and use them to evaluate SpiNeMap.

B. Neuromorphic Hardware

Recently, several research groups are investigating crossbar-based neuromorphic hardware with nonvolatile memory technologies. Ramasubramanian *et al.* [48] use spin-transfer torque magnetic RAM (STT MRAM), Burr *et al.* [49] use phase-change memories (PCMs), while Mallik *et al.* [50] use oxide-based resistive RAM (OxRAM) to design neuromorphic crossbars. While all these orthogonal works focus on the design of a crossbar, we focus on the architecture of a neuromorphic chip integrating multiple such crossbars. Examples of commercial neuromorphic chips include TrueNorth [1], Loihi [2], and DYNAP-SE [3]. We evaluate SpiNeMap on DYNAP-SE. Khan *et al.* [51] propose SNN mapping strategy for SpiNNaker. Ji *et al.* [19] propose NEUTRAMS for

crossbar-based neuromorphic hardware. In Section V, we compare SpiNeMap against NEUTRAMS (i.e., the baseline) and found that SpiNeMap is significantly better in terms of energy, latency, and application accuracy.

C. SNN Simulators

SpiNeMap uses CARLsim [27] due to its detailed STDP and homeostasis models, parameter tuning, and multiGPU support to accelerate the simulation. SpiNeMap can be combined with any other SNN simulators [52]–[56].

D. Related Concepts in the Domain of Embedded Systems

Graph partitioning problem has been extensively used for embedded multiprocessor systems, where an application task graph is partitioned to map tasks on the processing cores [57]–[59]. These mapping techniques cannot be directly used for clustering because of the new metric ISI distortion that is specific to SNN. We chose the clustering technique in SpiNeCluster because it is scalable and generates a good starting solution for the SpiNePlacer.

VII. Conclusion

We introduced SpiNeMap, a design methodology to map SNN-based applications to crossbar-based neuromorphic hardware. SpiNeMap completed the mapping in two steps. In step 1 (SpiNeCluster), we used a heuristic-based clustering algorithm to partition SNNs into local and global synapses, with local synapses mapped within crossbars and global synapses to the shared interconnect. SpiNeCluster minimized spikes on the shared interconnect, reducing spike congestion and ISI distortion. In step 2 (SpiNePlacer), we used an instance of the PSO to place clusters on physical crossbars in the hardware, optimizing energy consumption and spike latency on the shared interconnect.

We evaluated SpiNeMap using synthetic and realistic SNN applications. We have shown that SpiNeMap reduces energy consumption by 45% and spike latency by 21%, compared to the best of state-of-the-art techniques. These improvements reduced ISI distortion by 36%, improving application accuracy by 12%.

We have open-sourced our framework to enable future work based on SpiNeMap [60].

A. Future Outlook

We now describe how SpiNeMap can be used to advance the field of neuromorphic computing.

Mapping New Machine Learning Approaches to Hardware: In this article, we used supervised machine learning tasks to evaluate SpiNeMap. Emerging machine learning approaches, such as [61]–[66], can also be mapped to the neuromorphic hardware using SpiNeMap by first simulating the application in CARLsim and then using the spike trace to partition and place clusters to hardware.

We demonstrated SpiNeMap for spike-based model. Machine learning tasks designed with analog model, such as CNN or MLP, can also be used in our design methodology by first converting them to spike-based model before presenting to SpiNeMap. In this article, we demonstrated this using three analog CNN-based applications. We converted these applications to spike-based model using the N2D2 framework.

For rate model, information is encoded as an average firing rate of neurons. ISI distortion due to congestion on the interconnect does not always lead to performance loss as long as the average number of spikes received within a given time interval remains the same. A relevant metric for the rate model is the spike disorder. We provide a proper intuition behind spike disorder as follows. We consider a source neuron generating spikes at time $t = 0$, 5, and 25 ns. Spike rates of the source neuron are 200 and 50 MHz, respectively. These three spikes need to be communicated to a destination neuron. We consider a scenario where spikes 0 and 2 are received at time $t = 5$ and 30 ns and spike 1 is rerouted due to congestion, reaching the destination neuron at $t = 35$ ns. Spike rates observed at the destination neuron is 40 and 200 MHz, respectively. This is spike disorder, which can lead to performance loss. We formulate spike disorder as follows. Let $F^i = \{F^i_1, \ldots, F^i_{n_i}\}$ be the expected spike arrival rate at neuron i (from CARLsim) and $\hat{F}^i = \{\hat{F}^i_1, \ldots, \hat{F}^i_{n_i}\}$ be the actual spike rate considering hardware latencies. The spike disorder is computed as

$$\text{spike disorder} = \sum_{j=1}^{n_i} \left[\left(F^i_j - \hat{F}^i_j\right)^2\right]/n_i. \tag{19}$$

SpiNeCluster can be extended to support spike disorder.

Using SpiNeMap for Other Neuromorphic Hardware: SpiNeMap is a general-purpose design methodology for mapping SNN-based applications to crossbar-based neuromorphic hardware. We evaluated SpiNeMap for DYNAP-SE. Our future work will demonstrate integration of SpiNeMap with Loihi and TrueNorth.

In this article, we use Noxim [35] for cycle-accurate simulation of neuromorphic interconnect. Noxim allows significant advantage in terms of trace-driven simulations, extensions to other interconnect types. SpiNeMap can be used with other interconnect simulators, such as [67]–[69], which also supports cycle-accurate and trace-driven simulation.

References

[1] M. V. DeBole *et al.*, "TrueNorth: Accelerating from zero to 64 million neurons in 10 years," *Computer*, vol. 52, no. 5, pp. 20–29, 2019.
[2] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
[3] 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.
[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[6] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. Conf. Acoust., Speech Signal Process.*, 2013, pp. 6645–6649.
[7] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997.
[8] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *Int. J. Comput. Vis.*, vol. 113, no. 1, pp. 54–66, 2015.

[9] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Front. Comput. Neurosci.*, vol. 9, p. 99, Aug. 2015.

[10] L. Benini and G. De Micheli, "Networks on chip: A new paradigm for systems on chip design," in *Proc. Conf. Design, Autom. Test Eur.*, 2002, pp. 418–419.

[11] T. Sauer, "Interspike interval embedding of chaotic signals," *Chaos, Interdiscipl. J. Nonlinear Sci.*, vol. 5, no. 1, pp. 127–132, 1995.

[12] A. Ankit, A. Sengupta, and K. Roy, "Neuromorphic computing across the stack: Devices, circuits and architectures," in *Proc. Workshop Signal Process. Syst.*, 2018, pp. 1–6.

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

[14] Q. Xia and J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature Mater.*, vol. 18, no. 4, pp. 309–323, 2019.

[15] M. K. F. Lee *et al.*, "A system-level simulator for RRAM-based neuromorphic computing chips," *Trans. Archit. Code Optim.*, vol. 15, no. 4, 2019, Art. no. 64.

[16] P. Wijesinghe, A. Ankit, A. Sengupta, and K. Roy, "An all-memristor deep spiking neural computing system: A step toward realizing the low-power stochastic brain," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 5, pp. 345–358, Oct. 2018.

[17] W. Wen *et al.*, "An EDA framework for large scale hybrid neuromorphic computing systems," in *Proc. Design Autom. Conf.*, 2015, pp. 1–6.

[18] 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. Frontiers*, 2012, pp. 183–192.

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

[20] 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. Conf. Design, Autom. Test Eur.*, 2018, pp. 1217–1222.

[21] Y. Orii *et al.*, "Advanced interconnect technologies in the era of cognitive computing," in *Proc. Pan Pacific Microelectron. Symp.*, 2016, pp. 1–6.

[22] S. Grün and S. Rotter, *Analysis of Parallel Spike Trains*, vol. 7. New York, NY, USA: Springer, 2010.

[23] R. P. N. Rao and T. J. Sejnowski, "Spike-timing-dependent Hebbian plasticity as temporal difference learning," *Neural Comput.*, vol. 13, no. 10, pp. 2221–2237, 2001.

[24] D. P. Phillips and S. A. Sark, "Separate mechanisms control spike numbers and inter-spike intervals in transient responses of cat auditory cortex neurons," *Hearing Res.*, vol. 53, no. 1, pp. 17–27, 1991.

[25] R. Brette, "Philosophy of the spike: Rate-based vs. spike-based theories of the brain," *Frontiers Syst. Neurosci.*, vol. 9, p. 151, Nov. 2015.

[26] Y. Dan and M.-M. Poo, "Spike timing-dependent plasticity of neural circuits," *Neuron*, vol. 44, no. 1, pp. 23–30, 2004.

[27] 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.

[28] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. Symp. Micro Mach. Hum. Sci.*, 1995, pp. 39–43.

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

[30] A. Das, A. Kumar, and B. Veeravalli, "Communication and migration energy aware task mapping for reliable multiprocessor systems," *Future Gener. Comput. Syst.*, vol. 30, pp. 216–228, Jan. 2014.

[31] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified *NP*-complete problems," in *Proc. Symp. Theory Comput.*, 1974, pp. 47–63.

[32] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. Design Autom. Conf.*, 1982, pp. 175–181.

[33] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*. New York, NY, USA: Springer, 2010, pp. 760–766.

[34] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[35] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Proc. Conf. Appl.-Specific Syst., Archit. Process.*, 2015, pp. 162–163.

[36] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Design Autom. Electron. Syst.*, vol. 12, no. 3, p. 23, Aug. 2007.

[37] G. Indiveri, F. Corradi, and N. Qiao, "Neuromorphic architectures for spiking deep neural networks," in *Proc. Int. Electron Devices Meeting*, 2015, pp. 2–4.

[38] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Trans. Electron Devices*, vol. 53, no. 11, pp. 2816–2823, Nov. 2006.

[39] A. Das *et al.*, "Unsupervised heart-rate estimation in wearables with liquid states and a probabilistic readout," *Neural Netw.*, vol. 99, pp. 134–147, Mar. 2018.

[40] 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.

[41] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," 2014. *arXiv:1412.6806*. [Online]. Available: https://arxiv.org/abs/1412.6806

[42] *MLPerf: Fair and Useful Benchmarks for Measuring Training and Inference Performance of ML Hardware, Software, and Services*. Accessed: 2019. [Online]. Available: https://mlperf.org/training-overview/#overview

[43] *N2D2: Neural Network Design and Deployment*. Accessed: 2017. [Online]. Available: https://github.com/CEA-LIST/N2D2

[44] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *Proc. Conf. Rebooting Comput.*, 2016, pp. 1–8.

[45] 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.

[46] D. Verstraeten, B. Schrauwen, and D. Stroobandt, "Reservoir-based techniques for speech recognition," in *Proc. Int. Joint Conf. Neural Netw.*, 2006, pp. 1050–1053.

[47] B. J. Grzyb, E. Chinellato, G. M. Wojcik, and W. A. Kaminski, "Facial expression recognition based on liquid state machines built of alternative neuron models," in *Proc. Int. Joint Conf. Neural Netw.*, 2009, pp. 1011–1017.

[48] S. G. Ramasubramanian, R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "SPINDLE: Spintronic deep learning engine for large-scale neuromorphic computing," in *Proc. Int. Symp. Low Power Electron. Design*, 2014, pp. 15–20.

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

[50] A. Mallik *et al.*, "Design-technology co-optimization for OxRRAM-based synaptic processing unit," in *Proc. Symp. VLSI Technol.*, 2017, pp. T178–T179.

[51] M. M. Khan *et al.*, "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *Proc. Int. Joint Conf. Neural Netw.*, 2008, pp. 2849–2856.

[52] D. F. M. Goodman and R. Brette, "The Brian simulator," *Front. Neurosci.*, vol. 3, p. 26, Sep. 2009.

[53] A. P. Davison *et al.*, "PyNN: A common interface for neuronal network simulators," *Frontiers Neuroinform.*, vol. 2, p. 11, Jan. 2009.

[54] E. Yavuz, J. Turner, and T. Nowotny, "GeNN: A code generation framework for accelerated brain simulations," *Sci. Rep.*, vol. 6, Jan. 2016, Art. no. 18854.

[55] M.-O. Gewaltig and M. Diesmann, "NEST (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[56] O. Bichler, D. Roclin, C. Gamrat, and D. Querlioz, "Design exploration methodology for memristor-based spiking neuromorphic architectures with the Xnet event-driven simulator," in *Proc. Symp. Nanosc. Archit.*, 2013, pp. 7–12.

[57] A. Das, A. K. Singh, and A. Kumar, "Energy-aware dynamic reconfiguration of communication-centric applications for reliable MPSoCs," in *Proc. Workshop Reconfigurable Commun.-Centric Syst.-Chip*, 2013, pp. 1–7.

[58] A. Das, M. Walker, A. Hansson, B. Al-Hashimi, and G. Merrett, "Hardware-software interaction for run-time power optimization: A case study of embedded Linux on multicore smartphones," in *Proc. Int. Symp. Low Power Electron. Design*, 2015, pp. 165–170.

[59] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. Design Autom. Conf.*, 2013, pp. 1–10.

[60] A. Balaji. (2019). *SpiNeMap: Mapping Spiking Neural Networks to Neuromorphic Hardware*. [Online]. Available: https://github.com/drexel-DISCO/SpiNeMap
[61] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng, "Zero-shot learning through cross-modal transfer," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 935–943.
[62] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, Apr. 2006.
[63] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[64] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[65] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, pp. 2531–2560, Nov. 2002.
[66] D. L. Silver, Q. Yang, and L. Li, "Lifelong machine learning systems: Beyond learning algorithms," in *Proc. AAAI Spring Symp. Ser.*, 2013.
[67] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2013, pp. 86–96.
[68] L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, and A. Narayanan, "NIRGAM: A simulator for NoC interconnect routing and application modeling," in *Proc. Conf. Design, Autom. Test Eur.*, 2007, pp. 16–20.
[69] K. Huynh, "Exploration of dynamic communication networks for neuromorphic computing," M.S. thesis, Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2016.

Adarsha Balaji received the bachelor's degree from Visvesvaraya Technological University, Bengaluru, India, in 2012, and the master's degree from Drexel University, Philadelphia, PA, USA, in 2017, where he is currently working toward the Ph.D. degree at the Department of Electrical and Computer Engineering.

His current research interests include the design of neuromorphic computing systems, particularly data flow and power optimization of spiking neural networks (SNN) hardware.

Anup Das (SM'18) received the Ph.D. degree in embedded systems from the National University of Singapore, Singapore, in 2014.

He was a Postdoctoral Fellow with the University of Southampton, Southampton, U.K., and a Researcher with imec, Leuven, Belgium. He is currently an Assistant Professor with Drexel University, Philadelphia, PA, USA. His research focuses on neuromorphic computing and architectural exploration.

Yuefeng Wu receiving the bachelor's degree from Tianjin University, Tianjin, China. He is currently working toward the joint master's degree with the KTH Royal Institute of Technology, Stockholm, Sweden, and the Eindhoven University of Technology, Eindhoven, The Netherlands.

He worked at imec The Netherlands, Eindhoven, for his master's thesis and researched on the communication mechanisms of neuromorphic computing.

Khanh Huynh, photograph and biography not available at the time of publication.

Francesco G. Dell'Anna received the B.E. degree in computer engineering and the M.E. degree in embedded systems from Polytechnic University of Turin, Turin, Italy, in 2014 and 2016, respectively. He is currently working toward the Ph.D. degree with the Department of Micro- and Nanotechnology Systems, University of South-Eastern Norway, Notodden, Norway.

In 2016, he attended the Electrical Engineering Master Program at KU Leuven, Leuven, Belgium, working on a neuromorphic simulator at imec, Leuven. He is currently a Researcher with the Department of Micro- and Nanotechnology Systems, University of South-Eastern Norway.

Giacomo Indiveri is currently a Professor with the Faculty of Science, University of Zurich, Zürich, Switzerland, the Director of the Institute of Neuroinformatics (INI), University of Zurich, and ETH Zurich, Zürich, and the Head of the Neuromorphic Cognitive Systems Group, INI. He is interested in the study of real and artificial neural processing systems and is building hardware neuromorphic cognitive systems, using full-custom analog and digital VLSI technologies.

Dr. Indiveri was awarded an ERC Starting Grant in 2011 and an ERC Consolidator Grant in 2017.

Jeffrey L. Krichmar (SM'17) received the B.S. degree in computer science from the University of Massachusetts at Amherst, Amherst, MA, USA, in 1983, the M.S. degree in computer science from The George Washington University, Washington, DC, USA, in 1991, and the Ph.D. degree in computational sciences and informatics from George Mason University, Fairfax, VA, USA, in 1997.

He spent 15 years as a Software Engineer on projects ranging from the PATRIOT Missile System at Raytheon Corporation, Bedford, MA, USA, to Air Traffic Control for the Federal Systems Division, IBM, Gaithersburg, MD, USA. From 1999 to 2007, he was a Senior Fellow in Theoretical Neurobiology with The Neurosciences Institute. He is currently a Professor with the Department of Cognitive Sciences and the Department of Computer Science, University of California at Irvine, Irvine, CA, USA.

Dr. Krichmar is a Senior Member of the Society for Neuroscience.

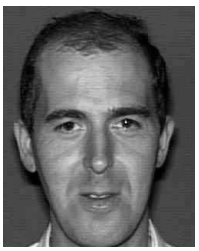
Nikil D. Dutt (F'08) received the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1989.

He is currently a Distinguished Professor of Computer Science, Cognitive Sciences, and Electrical Engineering and Computer Sciences (EECS) with the University of California at Irvine, Irvine, CA, USA. He is also a Distinguished Visiting Professor with the Department of Computer Science and Engineering (CSE), IIT Bombay, Mumbai, India. His research interests are in embedded systems, electronic design automation (EDA), computer systems architecture and software, healthcare IoT, and brain-inspired architectures and computing.

Dr. Dutt is a Fellow of the ACM. He was a recipient of the IFIP Silver Core Award.

Siebren Schaafsma received the two master's degrees in nuclear physics and in computer science from the Rijks Universiteit Groningen (RUG), Groningen, The Netherlands, in 1988 and 1989, respectively, and the Ph.D. (Dr.) degree from the Biophysics Department, University of Nijmegen, Nijmegen, The Netherlands. His second master's degree dissertation addresses the neural networks' implementation on a transputer cluster.

He is currently an Research and Development Manager with the IoT Unit, imec The Netherlands (Imec-nl), Eindhoven, The Netherlands.



Francky Catthoor (F'05) received the Ph.D. degree in electrical engineering from Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 1987.

From 1987 to 2000, he headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000, he has been strongly involved in other activities at imec, Leuven, including deep submicrometer technology aspects, IoT and biomedical platforms, and smart photovoltaic modules. He is currently an imec Fellow. He is also a part-time Full Professor with the Department of Electrical Engineering, KU Leuven.

Dr. Catthoor has been an associate editor for several IEEE and ACM journals.