Title

Mapping Spiking Neural Networks to Neuromorphic Hardware

Permalink

https://escholarship.org/uc/item/0fn2443s

Journal

IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 28(1)

ISSN

1063-8210

Authors

Balaji, Adarsha
Das, Anup
Wu, Yuefeng
et al.

Publication Date

2020

DOI

10.1109/tvlsi.2019.2951493

Mapping Spiking Neural Networks to Neuromorphic Hardware

Adarsha Balaji, Anup Das, Yuefeng Wu, Khanh Huynh, Francesco Dell'Anna, Giacomo Indiveri,
Jeffrey L. Krichmar, Nikil Dutt, Siebren Schaafsma, and Francky Catthoor

***Abstract*—Neuromorphic hardware platforms implement biological neurons and synapses to execute spiking neural networks (SNNs) in an energy-efficient manner. 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 SNNs into clusters of synapses, where intra-cluster local synapses are mapped within crossbars of the hardware and inter-cluster global synapses are mapped to the shared interconnect. SpiNeCluster minimizes the number of spikes on global synapses, which reduces spike congestion on the shared interconnect, improving application performance. SpiNePlacer then finds the best placement of local and global synapses on the hardware using a meta-heuristic-based approach to minimize energy consumption and spike latency. We evaluate SpiNeMap using synthetic and realistic SNNs on the DynapSE neuromorphic hardware. We show that SpiNeMap reduces average energy consumption by 45% and average spike latency by 21%, compared to state-of-the-art techniques.**

I. Introduction

Spiking Neural Networks (SNNs) [1] are typically used for machine learning on energy-constrained devices [2]–[4]. Neuromorphic platforms such as TrueNorth [5], Loihi [6], and DynapSE [7] implement biological neurons and synapses, making them efficient in executing SNNs. Typically, these platforms consist of multiple crossbars with a shared time-multiplexed interconnect. A crossbar is a two-dimensional arrangement with n rows, n columns, and memory elements (to store synaptic weights) at every cross-point. Each crossbar can map at most n synapses per neuron, meaning that a large SNN must be partitioned into synapses that map inside different crossbars (*local synapses*) and those that map on the shared interconnect (*global synapses*).

A crossbar's size is usually kept small to reduce the energy consumed in driving high voltages through n^2 connections of a $n \times n$ crossbar. For the DynapSE platform, with $n = 256$, a crossbar consumes 17pJ at 1.3V supply with SRAM-based synapses. This number is expected to reduce significantly when using non-volatile memory (NVM) synapses [8]. The shared interconnect in a neuromorphic hardware introduces spike congestion and latency to communicate spikes from one crossbar to another due to time-multiplexing, which impacts the inter-spike interval (ISI) [9]. This reduces application performance such as accuracy (see Section II).

Many recent works demonstrate mapping of SNNs to a single crossbar [10]–[15]. In Section V we show how these techniques can be inefficient when applied to a multi-crossbar neuromorphic platform such as the DynapSE. There are only a few works that address SNN mapping to multi-crossbar neuromorphic hardware. These include the PACMAN [16], NEUTRAMS [17], and PSOPART [18].

Compared to PACMAN and NEUTRAMS, which minimize crossbar usage, PSOPART partitions an SNN into local and global synapses, minimizing the number of spikes on the shared interconnect. This optimization strategy reduces spike congestion and changes in ISI, which improves performance. PSOPART is designed for the *shared bus* interconnect and it does not address the placement of local and global synapses to the neuromorphic hardware.

Unfortunately, the shared bus becomes the latency and energy bottleneck for large SNNs, those with more than a million synapses [19]. In recent years many new interconnects are explored for large-scale neuromorphic computing. Examples include the multi-stage networks-on-chip for the new TrueNorth platform [20] and the segmented bus for the new DynapSE platform [21]. For these new neuromorphic interconnects, the PSOPART technique has two limitations. First, the synapse partitioning approach is not scalable for large number of neurons and synapses. Second, different synapse placement strategies lead to different latency and energy consumption, which we show in Section V. Therefore, the placement problem can no longer be left unaddressed.

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

Contributions : Following are our novel contributions:

- **SpiNeCluster**: We propose a heuristic-based approach to partition an SNN into local and global synapses, reducing the number of spikes communicated on the shared interconnect.
- **SpiNePlacer**: We propose a meta-heuristic-based approach to place local and global synapses on physical resources of a neuromorphic hardware, reducing energy consumption and spike latency.
- We evaluate SpiNeMap on the DynapSE neuromorphic hardware using synthetic and realistic SNNs.
- 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 DynapSE hardware. We show that SpiNeMap reduces energy consumption by 45% and spike latency by 21% compared to state-of-the-art techniques.

Techniques	Partitioning	Placement	Optimization Objective
[10]–[15]	×	×	Maximize single crossbar utilization
NEUTRAMS [17]	√	×	Minimize number of crossbars
our PSOPART [18]	√	×	Minimize spikes on global synapses
SpiNeMap	√	√	Minimize energy consumption and latency of neuromorphic hardware

√ Optimized by these approaches
× Not optimized by these approaches

TABLE I: Contributions of SpiNeMap over the state-of-the-art approaches and our earlier work [18].

This paper is organized as follows. We provide background in Section II. We describe our 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 the paper in Section VII with an outlook on the design of future neuromorphic platforms.

II. Background

Figure 1 illustrates how a small SNN with two pre-synaptic neurons connected to a post-synaptic neuron is mapped to a crossbar. Spikes from a pre-synaptic neuron injects 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}) following Ohm's law. Current summations along columns are performed in parallel following Kirchhoff's current law, and implement the sums $\sum_j w_{ij}x_i$, needed for forward propagation of neuron excitation x_i. Beyond this supervised approach, recent works [22] have also developed peripheral structures necessary to implement online synaptic updates such as spike timing dependent plasticity (STDP) [23].

We demonstrate our design methodology for supervised machine learning approaches, where an SNN is first trained with examples from the field and then deployed for inference with in-field data. Performance is measured using *accuracy*, which is assessed using inter-spike intervals (ISIs) [24].

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

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

where t_i^n is the time of the i^{th} spike generated by the n^{th} neuron in the time interval $[0, T]$ and $K = \sum_{i=1}^N k_i$. The ISI of this spike train is given by [25]

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

For a feedforward architecture [26], (spiking) neurons are organized into layers, with one input layer, one or more hidden layers, and one output layer. For these architectures, accuracy is assessed from ISI of neurons in the (output) decision layer. For other architectures such as the Liquid State Machine (LSM) [27], ISI of critical neurons contribute to the accuracy.

Using CARLsim [28] we can simulate different machine learning approaches and neural architectures, and extract ISI from any neuron in the architecture. This makes CARLsim our ideal starting point. However, CARLsim is an application-level simulator meaning that hardware latencies are not incorporated. In a realistic scenario, ISI will be affected due to hardware latency arising from two sources – 1) the *fixed* latency within a crossbar to propagate current through synaptic elements and 2) the *variable* latency of time multiplexing in the shared interconnect. In Section III we describe our framework SpiNeMap to obtain these latencies, starting from the application-level simulation results using CARLsim.

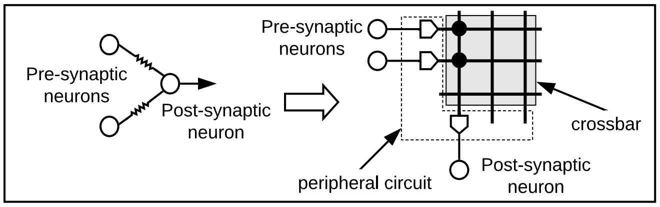


Fig. 1: Overview of how SNNs are mapped to a crossbar in a neuromorphic hardware.

To incorporate hardware latency in ISI computation, Equation 1 needs to be represented considering spike times at individual synapse-level. This is because different synapses have different latencies in neuromorphic hardware based on whether they are mapped within crossbars (i.e., local synapses) or on the shared interconnect (i.e., global synapses).

The spike times on synapses are

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

where τ_j^s is the j^{th} spike on s^{th} synapse and spike timings in the set $\{\tau_j^s\}$ are obtained from spike timings in the set $\{t_i^n\}$. We can similarly define ISI for this spike as

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

We use the notation δ_j^s to represent the latency of the j^{th} spike on s^{th} 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 given by

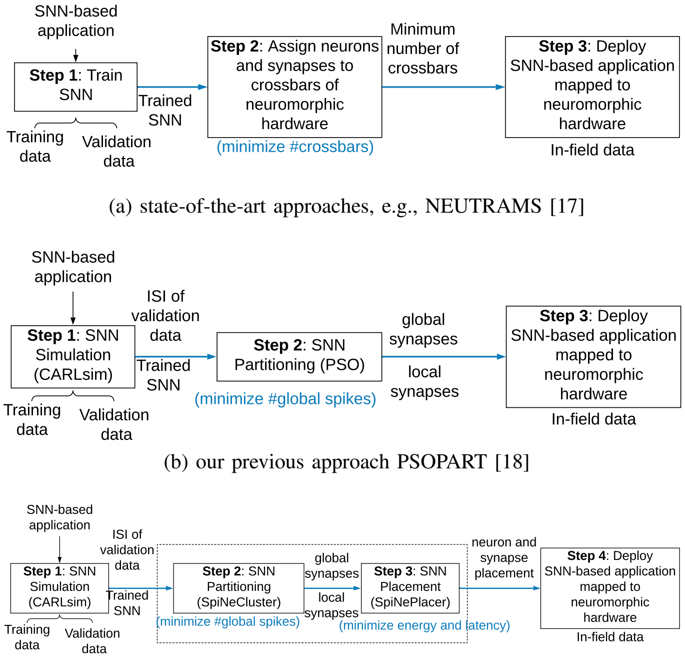
$$I_j^s|_{\text{distortion}} = I_j^s|_{\text{new}} - I_j^s = \delta_j^s - \delta_{j-1}^s \quad (6)$$

For local synapses, which are mapped within crossbars, all spikes have the same latency, i.e., $\delta_j^s = \delta_{j-1}^s$. So, 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, i.e.,

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

ISI distortion due to the interconnect latency can lead to unacceptable accuracy loss. Existing techniques [10]–[14] minimize the latency inside crossbar, leaving the optimization of the interconnect latency to system designers. In this work, we reduce the *average ISI distortion* of spikes on all global synapses. Our framework can also perform other optimizations such as minimizing the *maximum ISI distortion*.

As we can clearly see from Equation 7, ISI distortion is due to the latency to time-multiplex spikes on the shared


SNN-based application
Step 1: Train SNN
Training data
Validation data
Trained SNN
Step 2: Assign neurons and synapses to crossbars of neuromorphic hardware
(minimize #crossbars)
Minimum number of crossbars
Step 3: Deploy SNN-based application mapped to neuromorphic hardware
In-field data

(a) state-of-the-art approaches, e.g., NEUTRAMS [17]

SNN-based application
Step 1: SNN Simulation (CARLsim)
Training data
Validation data
ISI of validation data
Trained SNN
Step 2: SNN Partitioning (PSO)
(minimize #global spikes)
global synapses
local synapses
Step 3: Deploy SNN-based application mapped to neuromorphic hardware
In-field data

(b) our previous approach PSOPART [18]

SNN-based application
Step 1: SNN Simulation (CARLsim)
Training data
Validation data
ISI of validation data
Trained SNN
Step 2: SNN Partitioning (SpiNeCluster)
(minimize #global spikes)
global synapses
local synapses
Step 3: SNN Placement (SpiNePlacer)
(minimize energy and latency)
neuron and synapse placement
Step 4: Deploy SNN-based application mapped to neuromorphic hardware
In-field data

(c) our proposed design methodology SpiNeMap

Fig. 2: A high-level overview of our SpiNeMap mechanism and its difference with state-of-the-art.

interconnect. This latency depends on the number of spikes that must be communicated via the shared interconnect at any given time (i.e., *spike congestion*). Therefore, by reducing the number of spikes on global synapses we can reduce spike congestion, which would reduce ISI distortion and improve application performance. This is precisely the intuition behind our optimization strategy for the partitioning approach in our prior work PSOPART [18] and this current work. The difference is that the partitioning approach in this work is scalable to larger problem sizes than PSOPART (see Section V-G for comparison with PSOPART).

III. SpiNeMap: Mapping Spiking Neural Networks to Neuromorphic Hardware

A. High-Level overview and difference with state-of-the-art

In Figure 2, we illustrate our SpiNeMap methodology and its differences with state-of-the-art. In Figure 2(a), we illustrate how NEUTRAMS [17] and PACMAN [16] can be used to deploy SNN-based application on neuromorphic hardware. These approaches use 3 steps: *Step 1)* train the SNN using training data and validate the trained model, *Step 2)* pack neurons and synapses to crossbars, minimizing the resource requirements, and *Step 3)* deploy the trained SNN mapped to the neuromorphic hardware for inference with in-field data.

Our previously-proposed PSOPART [18] also uses 3 steps to deploy SNN-based applications for inference (see Figure 2(b)). The difference in our prior approach is that we minimize the number of spikes on the global synapses to reduce ISI distortion, which improves application performance. To do so, we extract the spike count on every synapse of the SNN corresponding to the validation data used in SNN simulation. The spike count information is used by PSOPART to partition the SNN into local and global synapses using an instance of the particle swarm optimization (PSO) [29].

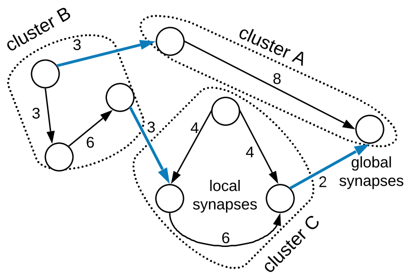


Fig. 3: An example illustrating how an SNN with 8 neurons is partitioned into 3 clusters with local and global synapses.

Algorithm 1: SNN Clustering algorithm.

```
foreach $C_i, C_j \in \mathcal{C}$ do
    /* iterate over all cluster pairs                    */
    /* begin 2-part procedure                            */
    $gs$ = total spikes between $C_i$ and $C_j$;
    while True do
        foreach $n_i \in C_i$ and $n_j \in C_j$ do
            if $n_i$ and $n_j$ are not previously selected then
                Move $n_i$ to $C_j$ and calculate $gs_1$;
                Move $n_j$ to $C_i$ and calculate $gs_2$;
                Swap $n_i$ and $n_j$ and calculate $gs_3$;
                Select the option which lowers $gs$;
                Return new partitions $C_i', C_j'$;
            end
        end
        $gs'$ = total spikes between $C_i'$ and $C_j'$;
        if $gs' < gs$ then
            $gs = gs'$ and break;
        end
    end
    /* end 2-part procedure                              */
end
```

In Figure 2(c), we illustrate our SpiNeMap. The key difference with our previously-proposed PSOPART is that we propose a 4-step methodology, with the new *SNN Placement* step explicitly minimizing energy consumption and latency on the shared interconnect. This step is necessary for SNN mapping to large neuromorphic architectures with many crossbars. To do so, we extract not only the spike count on different synapses, but also their precise timing information by simulating the SNN in CARLsim. These information about spikes, also called *spike trace*, are then used in SpiNePlacer to simulate the exact latency and energy consumption, considering spike traffic on the shared interconnect. Overall, the *SNN Partitioning* and *Placement* steps jointly improve application performance, energy consumption, and spike latency.

B. Detailed design of SNN Partitioning via SpiNeCluster

In Figure 3, we illustrate 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 this figure. In this example, the total number of spikes on global synapses is 8. To understand how SpiNeCluster partitions an SNN, we introduce the following notations.

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, and communicates $w_{i,j}$ spikes. Our objective is to 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. This problem of transforming $\mathcal{G}(\mathcal{N},\mathcal{S}) \rightarrow \mathcal{H}(\mathcal{C},\mathcal{E})$ is a classic graph partitioning problem [30], and has been applied in many context, including task mapping on multiprocessor systems [31]. The graph partitioning problem is already NP-complete [32], so heuristics are typically used to solve them [33]. In our earlier work PSOPART [18], we use an instance of particle-swarm optimization (PSO) [34] to solve this problem. However, the approach becomes *intractable* as the size of the SNN increases. Here we propose a greedy approach, roughly based on the Kernighan-Lin Graph Partitioning algorithm [30], which we show to be scalable to large SNNs.

We set $k = \lceil \frac{|\mathcal{N}|}{n_c} \rceil$, where n_c is the number of neurons that can be accommodated per crossbar. We make this choice because *by utilizing the minimum number of crossbars, the overall energy consumption of the hardware can be minimized* [10]–[14]. Next, we evenly (and arbitrarily) distribute neurons to these k clusters. Starting from this arbitrary assignment, we analyze the change in the number of spikes on global synapses by moving a single neuron from one cluster to another, tracking and enforcing those changes that lead to minimum number of spikes on global synapses.

We formalize these steps in Algorithm 1. The algorithm applies a 2-part procedure (lines 2-17) to every cluster pair (with a total of $\binom{k}{2}$ iterations). In the 2-part procedure, we first calculate the total number of inter-cluster 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 4-5). 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 inter-cluster spike for each of these operations, and select the option that generates the maximum reduction of inter-cluster spike compared to gs (line 9). We return the new clusters (line 10). We repeat the procedure (lines 4-13) while the number of inter-cluster spike continues to be reduced (lines 14-16).

1) Time complexity: We compute the time complexity of Algorithm 1 as follows: Line 2-17 are executed $\binom{k}{2}$ times. At each iteration, lines 4-16 is iterated for every neurons of the a cluster with each cluster accommodating a maximum of n_c neurons. This time complexity is therefore

$$\text{time complexity} = O\left(\binom{k}{2} \times n_c * n_c\right) = O\left(k^2 \times n_c^2\right) = O\left(|\mathcal{N}|^2\right) \tag{8}$$

where $k = \lceil \frac{|\mathcal{N}|}{n_c} \rceil$.

C. Detailed design of SNN Placement via SpiNePlacer

In Figure 4, we illustrate how a partitioned SNN (obtained via SpiNeCluster) can be placed on to the hardware, which consists of three crossbars arranged in a mesh topology. Different placement alternatives lead to different interconnect lengths traversed by spikes to reach their destinations. This impacts both energy consumption and latency, meaning that the placement problem is no longer a trivial one, especially for large neuromorphic architectures (a limitation of NEUTRAMS [17], PACMAN [16], Eyeriss [35], and PSOPART [18]).

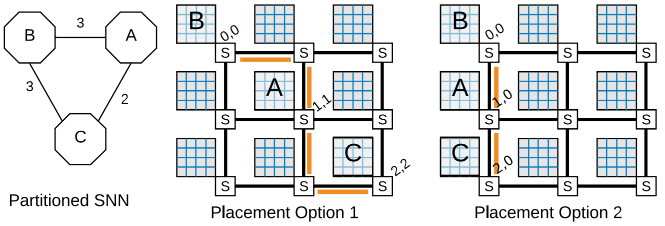


Fig. 4: Illustrating the impact of different placements of clusters of a partitioned SNN on a neuromorphic hardware.

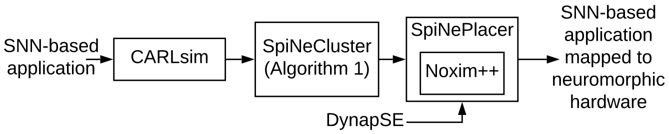

SNN-based application
CARLsim
SpiNeCluster (Algorithm 1)
SpiNePlacer
Noxim++
DynapSE
SNN-based application mapped to neuromorphic hardware

Fig. 5: Our design methodology SpiNeMap.

To accurately estimate the energy and latency impact of different placement alternatives, we have extended the Noxim [36], a cycle-accurate interconnect simulator, to support 1) simulation of *spike traces* from CARLsim containing information (generation time, source neuron, and destination neuron) of every spike in an SNN and clustered to generate information about communication on global synapses, i.e., broadcast, multicast and one-to-one, 2) simulation of current and emerging interconnect topologies of neuromorphic architectures, 3) simulation of different routing algorithms, and 4) technology-specific energy and latency of interconnect wires and switches. We call our new framework *Noxim++*. In Figure 5 we present our design methodology SpiNeMap, illustrating how Noxim++ is integrated in the SpiNePlacer and configured to model the DynapSE neuromorphic platform [7]. We now formalize the optimization problem of our SpiNePlacer.

We consider the mapping of a clustered SNN $\mathcal{H}(\mathcal{C},\mathcal{E})$ to the neuromorphic architecture $\mathcal{A}(\mathcal{V},\mathcal{I})$, where $\mathcal{V}$ is the set of crossbars in the architecture 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 represented by a logical matrix $(m_{ij}) \in \{0,1\}^{|\mathcal{C}|\times|\mathcal{V}|}$, where m_{ij} is defined as

$$m_{ij} = \begin{cases} 1 & \text{if cluster } c_i \in \mathcal{C} \text{ is mapped to crossbar } v_j \in \mathcal{V} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The constraints in this formulation are the following:

1. A cluster can be mapped to only one crossbar, i.e.,

$$\sum_j m_{ij} = 1 \quad \forall i \tag{10}$$

2. A crossbar can accommodate at most one cluster, i.e.,

$$\sum_i m_{ij} \leq 1 \quad \forall j \tag{11}$$

We use our Noxim++ framework to evaluate a mapping in terms of the optimization objective of SpiNePlacer, i.e., to minimize energy consumption and spike latency on the interconnect. In has been shown in many prior works such as [37] that minimizing these metrics is equivalent to minimizing the *average number of hops* that spikes communicate before reaching their destination (see also the formulations in Section IV-D). Let $\mathcal{L}_i$ be the average hop count for the cluster mapping M_i obtained using Noxim++, i.e., $\mathcal{L}_i = \text{Noxim++}(M_i)$. The optimization objective of our SpiNePlacer is to find the mapping with the minimum average hop count, i.e.,

$$\mathcal{L}_{\text{min}} = \mathcal{L}_{\text{a}}, \text{ where } a = \arg\min\{\text{Noxim++}(M_i)|i \in 1, 2, \cdots, N_m\}, \tag{12}$$

where N_m is the total number of mappings evaluated. Of different techniques to generate and evaluate cluster mappings, we use an instance of PSO, which we describe next.

In general, PSO finds the optimum solution to a fitness function F. There can be several particles in the swarm. The position of these particles are solutions to the fitness functions, and they represent cluster mappings, i.e., M's in Equation 12. 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 for PSO.

$$\begin{aligned} D &= \text{dimensions of the search space} \\ n_p &= \text{number of particles in the swarm} \\ \mathbf{\Theta} = \{\theta_l \in \mathbb{R}^D\}_{l=0}^{n_p-1} &= \text{positions of particles in the swarm} \\ \mathbf{V} = \{\mathbf{v}_l \in \mathbb{R}^D\}_{l=0}^{n_p-1} &= \text{velocity of particles in the swarm} \end{aligned} \tag{13}$$

Here θ_l is the position of the l^{th} particle in the swarm, and translates to the mapping M_l. D is therefore the dimension of the logical mapping matrix M, i.e., $D = |\mathcal{C}| \times |\mathcal{V}|$.

Position and velocity updates are performed according to the following equation.

$$\begin{aligned} &\mathbf{\Theta}(t+1) = \mathbf{\Theta}(t) + \mathbf{V}(t+1) \\ &\mathbf{V}(t+1) = \mathbf{V}(t) + \varphi_1 \cdot \Big(P_{\text{best}} - \mathbf{\Theta}(t)\Big) + \varphi_2 \cdot \Big(G_{\text{best}} - \mathbf{\Theta}(t)\Big) \end{aligned} \tag{14}$$

where t is the iteration number, φ_1, φ_2 are constants and P_{best} (and G_{best}) is the particles own (and neighbors) experience. The fitness function is then

$$F(\theta_l) = \mathcal{L}_l = \text{Noxim++}(M_l) \tag{15}$$

Once the fitness function is computed for all particles in the *swarm*, the personal best position of each particle (P^t_{best}) and the global best position of the swarm (G_{best}) are updated using Equation 16.

$$\begin{aligned} P^t_{\text{best}} &= F(\theta_t) \text{ if } F(\theta_t) < F(P^t_{\text{best}}) \\ G_{\text{best}} &= \min_{t=0,\dots n_p-1} P^t_{\text{best}} \end{aligned} \tag{16}$$

Due to the binary formulation of the mapping problem (see Equation 9), we need to binarize the velocity and position of Equation 13, which we illustrate below.

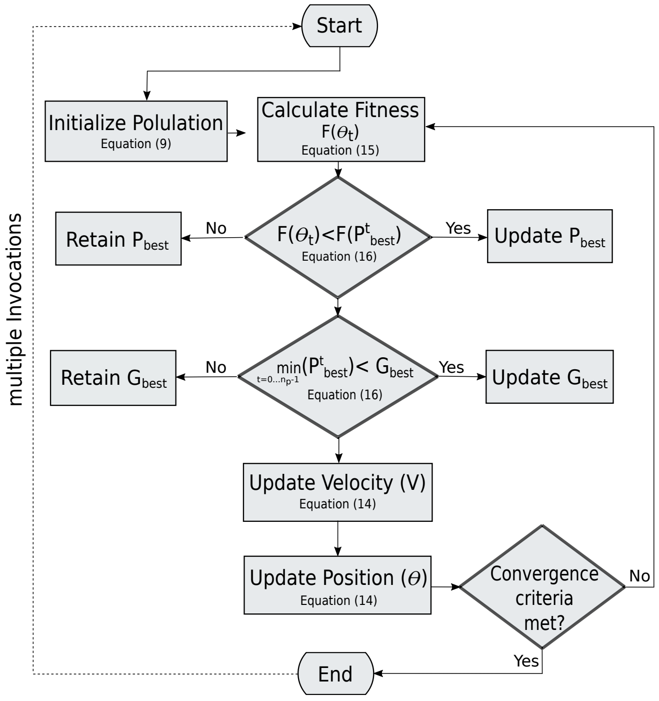


Fig. 6: Flow chart of our PSO algorithm.

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

In finding a new position of a PSO particle, we use the two constraints in Equations 10 & 11.

1) PSO Algorithm: In Figure 6, we describe our iterative PSO algorithm that uses the analytical formulations we introduced in Equations 10-17. The algorithm begins by initializing the position of the PSO particles that satisfies constraints 10 & 11. Then the algorithm runs for n_{ISO} iterations. At each iteration, the PSO algorithm evaluates the fitness function (Equation 15) and updates its position based on the local and global best positions (Equation 14), binarizing these updates using Equation 17. The time complexity of the PSO algorithm is therefore $O(n_{\text{ISO}} \times \text{operations in each iteration})$, where operations in each iteration is proportional to the PSO dimension $D = |\mathcal{C}| \times |\mathcal{V}|$ and the number of particles n_p. We represent the time complexity as

$$\text{time complexity of PSO} = O\left(n_{\text{ISO}} \times n_p \times |\mathcal{C}| \times |\mathcal{V}|\right) \tag{18}$$

D. Justification of SpiNeMap's design choices

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

1) Minimize spike count at the partitioning stage: To justify our optimization objective of minimizing the number of spikes at the partitioning stage of our design methodology, we conducted an experiment with the hand written digit

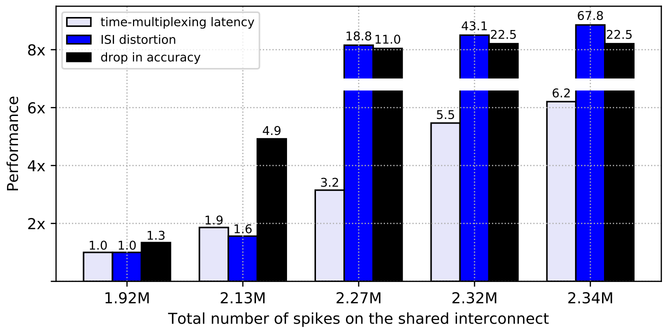


Fig. 7: Latency, ISI distortion, and accuracy as a function of the number of spikes on the global interconnect for the handwritten digit recognition example.

recognition example, where as the number of spikes on the shared interconnect is increased, the latency and average ISI distortion on the time-multiplexed interconnect, and the classification accuracy are recorded. We use the hardware configuration of DynapSE, with four crossbars organized in a 2x2 mesh with XY routing algorithm. Each crossbar can accommodate 256 neurons. We report these results in Figure 7, with the latency and ISI distortion normalized to the case with minimum number of spikes on the shared interconnect. The drop in accuracy is calculated with respect to the accuracy obtained when the number of spikes on the shared interconnect in the minimum.

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 a similar behavior for all our evaluated applications.

2) Integration of Noxim++ within PSO: The average hop count of spikes communicated between clusters (i.e., crossbars) on the shared interconnect depends on 1) the cluster mapping M and 2) the routing algorithm that *dynamically* routes spikes on the interconnect to avoid 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 the interconnect for every mapping with the spike trace generated from CARLsim 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: We use binary particle swarm optimization (PSO) [29], an evolutionary computing technique inspired by social behaviors such as bird flocking and fish schooling. Evolutionary computing techniques, in general, are efficient in avoiding being stuck at local optima. Additionally, PSO is computationally less expensive with faster convergence compared to its counterparts such as genetic algorithm (GA) or simulated annealing (SA).

In our earlier work [18], we use PSO for SNN partitioning (equivalent of SpiNeCluster). In this work 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 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 crossbars, with 256 neurons each).

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

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.

As we can clearly see from Table II, the PSO problem of partitioning soon becomes intractable with modest size SNNs, 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 (viz. SpiNePlacer), and use a greedy approach instead for the partitioning problem (viz. SpiNeCluster).

IV. Evaluation Methodology

We build SpiNeMap with the following system components.

- **CARLsim** [28] : A GPU accelerated simulator used to train and test SNN-based applications. CARLsim reports spike times for every synapse in the SNN.
- **Noxim++** [36] : A trace-driven and cycle-accurate interconnect simulator for multiprocessor systems. We extend it (1) to incorporate crossbar-based neuromorphic hardware, (2) to communicate spikes (rather than data packets), and (3) to generate key performance statistics such as energy consumption, spike latency and ISI distortion. In our SpiNeMap mechanism, the spike timing information from CARLsim is used as trace and is input to Noxim++ to generate the performance statistics.
- **DynapSE** [7]: We configure Noxim++ to model the DynapSE neuromorphic hardware at 65nm technology nodes with 256 neurons per crossbar. The crossbars are interconnected using a multi-stage networks-on-chip (NoCs) [38]. We extract the latency and energy numbers of each crossbar from silicon data [39]. We also use the analytical performance and energy model of the interconnect network at 65nm technology. Finally, we use predictive technology mapping (PTM) [40] to scale technology parameters to 28nm nodes.

A. Simulation environment

We conduct all experiments on a system with 8 CPUs, 32GB RAM, and NVIDIA Tesla GPU, running Ubuntu 16.04.

B. Evaluated applications

In order to evaluate the effectiveness of SpiNeMap, we use 7 synthetic and 8 realistic SNN applications. These applications

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 [28]	136,314	FeedForward (4096, 1024)	17,600
	EdgeDet [28]	272,628	FeedForward (4096, 1024, 1024, 1024)	22,780
	MLP-MNIST [41]	79,400	FeedForward (784, 100, 10)	2,395,300
	HeartEstm [2]	636,578	Recurrent	3,002,223
	HeartClass [42]	2,396,521	CNN[1]	1,036,485
	CNN-MNIST [43]	159,553	CNN[2]	97,585
	LeNet-MNIST [43]	1,029,286	CNN[3]	165,997
	LeNet-CIFAR [43]	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

TABLE III: 7 synthetic and 8 realistic applications we use to evaluate SpiNeMap.

are described in Table III. We indicate the synthetic applications with the letter 'S' followed by a number (e.g., S_1000), where the number represents the total number of neurons in the synthetic SNN. We use 7 synthetic SNN applications with number of neurons between 1000 to 4000. In column 3 of this table, we indicate the number of synapses in the networks, while in column 4 we describe the corresponding SNN topology. The total number of synapse in these synthetic applications ranges from 240,000 in S_100 to 3.75M in S4000.

We use eight realistic applications: *image smoothing* (ImgSmooth) [28] on 64x64 images, *edge detection* (EdgeDet) [28] on 64x64 images using difference-of-Gaussian, *multi-layer perceptron (MLP)-based handwritten digit recognition* (MLP-MNIST) [41] on 28x28 images of handwritten digits, *ECG-based heart-rate estimation* (HeartEstm) [2], *ECG-based heart-beat classification* (HeartClass) [42], *CNN-based digit classification* (CNN-MNIST) [43], [44], *CNN-based digit classification with LeNet* (LeNet-MNIST) [43], and *CNN-based CIFAR image classification with LeNet* (LeNet-CIFAR) [43]. We note that the last three applications are part of the MLPerf benchmark suite [43] and developed using analog computation model. We convert these applications into spike-based model using the CNN-to-SNN conversion tool N2D2 [45].

Finally, in the last column of Table III we report the total number of spikes for these applications obtained through simulation of the representative validation data using CARLsim [28]. The spike trace from CARLsim is clusted using SpiNeCluster, and placed on crossbars using SpiNePlacer.

C. Evaluated state-of-the-art techniques

We evaluate the following four approaches.

- Baseline: The Baseline uses NEUTRAMS [17] to cluster SNNs, minimizing the use of crossbars.
- SCO: The SCO approach uses the framework of [13] to balance the utilization of crossbars in the hardware.
- PSOPART: Our previously-proposed PSOPART [18] clusters SNNs to minimize the total number of spikes on the shared interconnect.
- SpiNeMap: Our SpiNeMap uses (1) SpiNeCluster to partition SNNs into clusters to minimize the total number of spikes on the shared interconnect and (2) SpiNePlacer to optimize the placement of clusters to crossbars of the neuromorphic hardware to minimize energy consumption and latency on the shared interconnect.

SpiNeMap	Energy Consumption (Sec. V-B)	Spike Latency (Sec. V-C)	ISI Distortion (Sec. V-D)	Application Accuracy (Sec. V-E)
vs. Baseline [17]	45%	21%	36%	12%
vs. SCO [13]	40%	27%	39%	20%
vs. PSOPART [18]	20%	13%	23%	5%

TABLE IV: Average improvement summary using SpiNeMap for all our evaluated applications.

D. Evaluated metrics

We evaluate all four approaches in terms of the following metrics for every application.

- Total number of spikes on the shared interconnect: This is the number of spikes (N_s) on the shared interconnect obtained after mapping synapse clusters to crossbars of the neuromorphic hardware.
- Spike latency on the shared interconnect: This is the delay experienced by spikes before reaching their destination, averaged over all spikes [37], i.e.,

$$L = \sum_{i=1}^{N_s}[(h_i - 1) * l_w + h_i * l_s]/N_s, \tag{19}$$

  where h_i is the number of hops that spikes traverses between the source crossbar and destination crossbar, l_w is the delay on the wires connecting two crossbars, and l_s is the delay of the hop.
- Energy consumption on the shared interconnect: This is the total energy consumed by all spikes on the shared interconnect [37], i.e.,

$$E = \sum_{i=1}^{N_s}[(h_i - 1) * e_w + h_i * e_s], \tag{20}$$

  where e_w and e_s are the energy consumption on the wires and hops, respectively.
- Average ISI distortion: This is motivated in Section II and computed using Equation 7, averaged over all spikes, i.e.,

$$I = \sum_{i=1}^{N_s} I_i|_{distortion}/N_s, \tag{21}$$

V. Results and Discussions

A. Improvement summary

In Table IV, we summarize the average improvements of SpiNeMap against the Baseline [17], SCO [13], and our previously-proposed PSOPART [18].

We now describe these results in details.

B. Energy consumption on the shared interconnect

In Figure 8, we report the energy consumption 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.

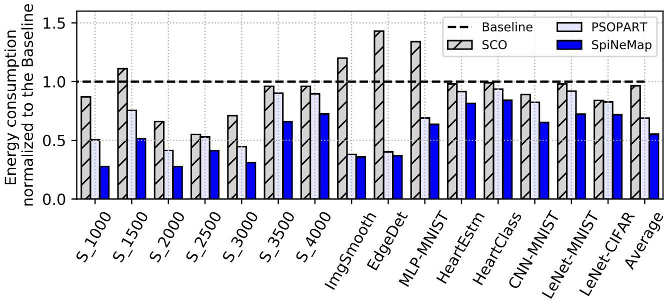


Fig. 8: Energy consumption on the shared interconnect normalized to the Baseline.

First, the average energy consumption of SCO is very similar to that of the Baseline. For some workloads such as S_2500 it achieves 45% lower energy consumption, while for other workloads such as EdgeDet it has 43% higher energy consumption than the Baseline. These differences are due to the earlier discussed distinction in the optimization objective for these two approaches. *Second*, PSOPART has 31% lower average energy consumption than the Baseline. This reduction is because PSOPART minimizes the total number of global spikes, which also reduces the energy consumption on the shared interconnect. *Third*, SpiNeMap has the lowest energy consumption of all our evaluated systems (45% lower average energy consumption than the Baseline, 40% lower than SCO, and 20% lower than PSOPART). These improvements are because of SpiNeMap's optimization policies: 1) SpiNeCluster reduces the total number of spikes on the shared interconnect, which lowers energy consumption, and 2) SpiNePlacer places the clusters on crossbars of the hardware to minimize both latency and energy consumption on the shared interconnect.

C. Spike latency on the shared interconnect

In Figure 9, we report the spike latency of the global synapses 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.

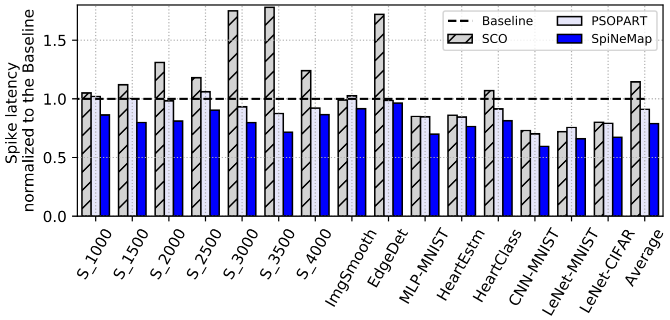


Fig. 9: Spike latency on the shared interconnect normalized to the Baseline.

First, the average spike latency of SCO is 14% higher than the Baseline. This increase is because SCO balances the crossbar utilization in the hardware and in doing so it can place certain synapses with large number of spikes on the shared interconnect, increasing the congestion and therefore the latency. *Second*, PSOPART has 9% lower average spike latency than the Baseline. This improvement is because PSOPART reduces the total number of spikes on the shared interconnect, which reduces spike congestion, improving the latency. *Third*, SpiNeMap has the lowest average spike latency among all our evaluated systems (21% lower average spike latency than the Baseline, 27% lower than SCO, and 13% lower than PSOPART). These improvements are due to SpiNeMap's optimization policies: 1) SpiNeCluster, which reduces the number of spikes on the shared interconnect, reducing congestion and latency and 2) SpiNeCluster, which minimizes latency by minimizing the average number of hop counts that spike traverses before reaching their destination.

D. Average ISI distortion of spikes on the shared interconnect

In Figure 10, we compare the average inter-spike interval (ISI) distortion 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.

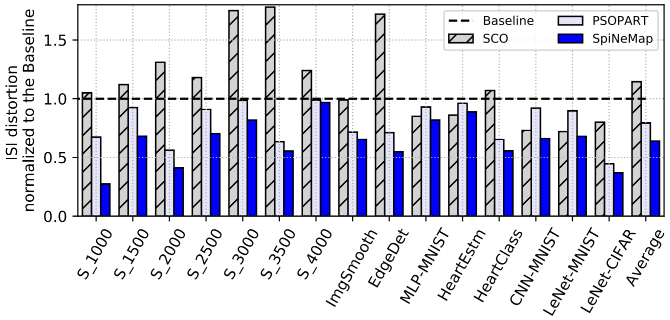


Fig. 10: Average ISI distortion normalized to the Baseline.

First, the ISI distortion of SCO is 12% higher than the Baseline. This increase is due to the increase in total spikes on the shared interconnect, which increases spike congestion and ISI distortion. *Second*, PSOPART has 21% lower average ISI distortion than the Baseline. This reduction is due to the reduction of the number of spikes on the shared interconnect. *Third*, SpiNeMap has the lowest ISI distortion of all our evaluated systems (36% lower average ISI distortion than the Baseline, 39% lower than SCO, and 23% lower than PSOPART). The improvement with respect to PSOPART is because of our new SpiNePlacer step (see Figure 2), which further reduces the ISI distortion while reducing the spike latency.

E. Application accuracy

In Figure 11, we report the application 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 the ISI distortion results we presented in Section V-D. Specifically, the accuracy using SCO is lower than the Baseline by an average 6% due to the 12% increase in ISI distortion. PSOPART increases the accuracy by 7% due to the 17% reduction of ISI distortion. Finally, SpiNeMap achieves the highest accuracy among all our evaluated systems (12% higher average accuracy than the Baseline, 20% higher than SCO, and 5% higher than PSOPART).

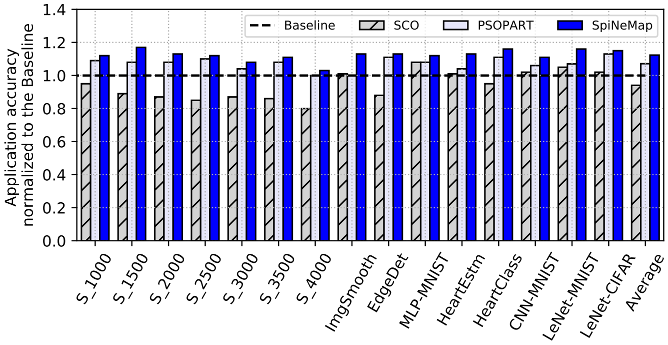


Fig. 11: Application accuracy normalized to the Baseline.

F. Evaluation of SpiNeCluster in terms of spike count

In Figure 12, we compare 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.

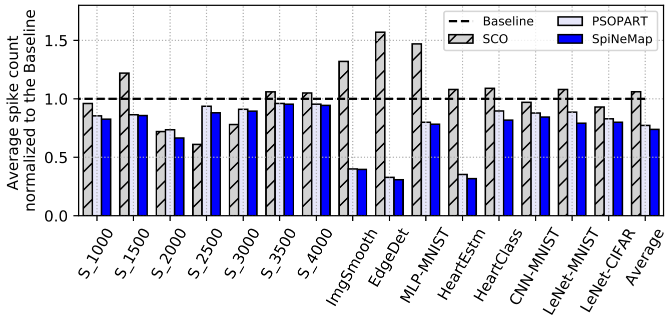


Fig. 12: Number of spike on the shared interconnect normalized to the Baseline.

First, SCO has an average 6% higher number of spikes on the shared interconnect compared to the Baseline. These extra spikes increases the energy consumption on the shared interconnect, which we presented in Section V-B. *Second*, PSOPART has 23% lower number of spikes due to the PSO approach, which explicitly minimizes the total number of spikes on the shared interconnect. *Third*, SpiNeMap generates the lowest number of spikes on the shared interconnect (26% lower than the 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 the PSO, especially for the large application use-cases.

G. Evaluation of SpiNeCluster in terms of optimization time

In Figure 13, we compare the execution time of our new clustering algorithm (Algorithm 1) against the PSO-based clustering approach of PSOPART normalized to the Baseline.

We observe that our SpiNeCluster has an average 3x lower execution time than our previously-proposed PSO-based PSOPART. Additionally, we have shown in Section V-F that Algorithm 1 generates an average 9% lower number of spikes than the PSO-based solution, improving energy consumption, spike latency, and application accuracy. We conclude that our new clustering algorithm is scalable and generates better results than our previously-proposed PSO-based approach.

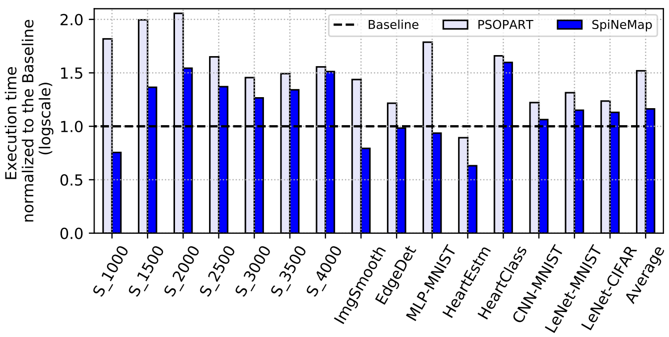


Fig. 13: Execution time normalized to the Baseline.

H. Interconnect design explorations

In Figure 14, we illustrate how our design methodology can be used for explorations on interconnect for neuromorphic hardware. In this figure, we compare XY routing, which is used in DynapSE against NorthLast and WestFirst routing algorithms. Finally, we evaluate our previously-proposed segmented bus [46] as an alternative to the multi-stage NoC used in the DynapSE neuromorphic platform. We evaluate these alternatives for all our evaluated workloads. We make the following two observations.

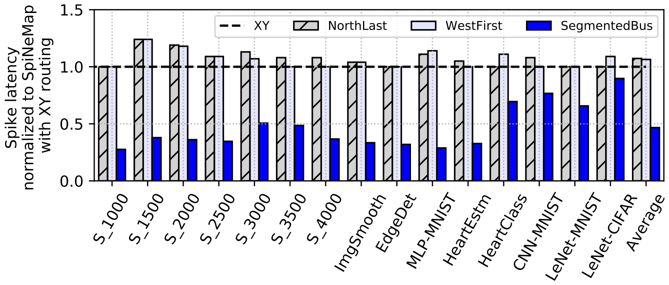


Fig. 14: Exploration of interconnects and routing algorithms using SpiNeMap

First, the NorthLast and WestFirst routing algorithms have an average 7% and 4% higher latency than the default XY routing algorithms, meaning that the XY routing algorithm is the most suitable one for the applications. *Second*, the segmented bus interconnect has the lowest spike latency among all our evaluated routing algorithms (average 54% lower for all these three routing algorithms). Lower spike latency leads to lower energy consumption and higher application performance.

Our SpiNeMap design methodology allows simulating NoCs, segmented bus, and other interconnect topologies, facilitating future research on scalable interconnect for neuromorphic computing. Our continuing work is to extend the SpiNeMap with architecture of TrueNorth, Loihi, and other neuromorphic hardware platforms.

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

Machine learning techniques such as neural networks [26] have proved to be immensely successful in many domains such as computer vision [47] and natural language processing [48]. The machine learning database MLPerf [43] provides a comprehensive collection of these applications. We demonstrate the performance of SpiNeMap using applications from MLPerf benchmark suite. Compared to analog and rate models, machine learning techniques implemented with spike model [49] and brain-inspired learning algorithms [50], e.g., spiking neural networks [1], have ultra-low power footprint when executed on neuromorphic hardware such as DYNAP-SE [7], TrueNorth [**?**], and Loihi [6]. This makes spike-based computation model attractive for implementing machine learning applications on these devices. Verstraeten et al. propose reservoir computing with SNNs for speech recognition [51]. Grzyb et al. use spiking liquid state machine for facial recognition [52]. Diehl et al. propose hand-written digit recognition using SNNs [41]. We have previously proposed a liquid state machine approach for heart-rate estimation from ECG signals [2]. We demonstrate the performance of SpiNeMap using some of these applications.

Recent works have demonstrated techniques to convert operations of analog computation model to spike model. One example is the N2D2 tool [45]. Using this tool we have previously demonstrated the SNN implementation of convolutional neural networks (CNN)-based heart-beat classification [42].

B. Neuromorphic hardware

Recently, several research initiatives are undertaken to develop crossbar-based neuromorphic hardware using the emerging non-volatile memory technologies. Ramasubramanian et al. propose to use Spin-transfer torque magnetic RAM (STT MRAM) to build neuromorphic crossbars [53]. Burr et al. propose to use phase-change memories (PCM) to design neuromorphic crossbars [22]. Mallik et al. propose to use oxide-based resistive RAM (OxRAM) as alternative [54]. 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. To this end, Khan et al. propose a mapping strategy for SNNs on the SpiNNaker platform [55]. Ji et al. propose NEUTRAMS for mapping neural networks on crossbar-based neuromorphic hardware [17]. 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 and neuromorphic simulators

SpiNeMap is a technique that maps trained SNNs on the neuromorphic hardware. To this end, there are several choices for application-level SNN simulators that can generate trained SNNs. PyNN [56] is a high level, simulator-independent interface used for building neuronal models by providing high level abstractions allowing the access of low-level details like neuron and synapse models of the computing back-end. There are also other simulators such as Brian [57], GeNN [58], and NEST [59]. We use CARLsim [28] due to its detailed STDP and homeostasis models, parameter tuning, and multi-GPU support to accelerate the simulation. Nevertheless, SpiNeMap can be combined with any other SNN simulators.

D. Related concepts in similar domains

Graph partitioning problem has been extensively used for multiprocessor systems, where an application task graph is partitioned to map tasks on the processing cores. The survey paper [60] provides an overview of different mapping techniques and optimization objectives that have been proposed for multiprocessor systems. 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 and Future Outlook

This paper introduces SpiNeMap, a design methodology to map SNN-based applications to crossbar-based neuromorphic hardware. SpiNeMap completes the mapping in two steps. In Step 1 (SpiNeCluster), we use 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. Our objective is to minimize the number of spikes on the shared interconnect, which reduces spike congestion, leading to a reduction of the ISI distortion. In Step 2 (SpiNePlacer), we use an instance of the particle swarm optimization (PSO) to place clusters on physical crossbars in the hardware, optimizing energy consumption and spike latency on the shared interconnect.

Our optimization strategies in the two steps also improves application accuracy. We evaluate SpiNeMap using synthetic and realistic SNN applications. SpiNeMap reduces energy consumption on the shared interconnect by 45% and spike latency by 21%, compared to the state-of-the-art techniques. This reduces ISI distortion by 36%, which improves application accuracy by 12% over state-of-the-art approaches.

We believe that SpiNeMap is an end-to-end design methodology to map SNN applications on neuromorphic hardware. Our SpiNeMap framework is open-sourced and can be downloaded from the url `https://github.com/drexel-DISCO/SpiNeMap`.

A. Future Outlook

In this section, we describe how our design methodology SpiNeMap can be used to advance neuromorphic computing.

1) Mapping new machine learning approaches to hardware: Supervised machine learning approaches are usually limited when remembering and dealing with rare events. Advanced machine learning approaches are therefore investigated. Many of these new proposals are based on spiking events. Examples include the liquid state machine [27], zero-shot learning

[61], one-shot learning [62], lifelong learning [63], transfer learning [64], and deep reinforcement learning [65] among others. All these new approaches can be mapped to hardware using SpiNeMap, by first simulating the application behavior in CARLsim, and then using the spike trace to partition and place the clusters on to hardware. In fact, in this work we demonstrate the mapping of one such emerging machine learning approach viz the liquid state machine implemented in the *HeartEstm* application.

From the computational neuroscience models front, we have demonstrated our design methodology SpiNeMap using the *spike-based model*. Machine learning algorithms designed with the *analog model* such as CNN or MLP can also be used in our design methodology by first converting the analog model to a spike-based model before presenting the application to SpiNeMap. In this work, we demonstrate this using three analog CNN-based applications. We converted these applications to spike-based model using the N2D2 tool [45].

For the *rate model*, information is encoded as average firing rate of neurons in the SNN. 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 to capture the effect of spike congestion on the shared interconnect is the *spike disorder*. We provide a proper intuition behind spike disorder as follows: We consider that a source neuron generates three spikes at time t = 0ns, 5ns, 25ns and 50ns. The spike rate of the source neuron are 200MHz and 50MHz, respectively. These three spikes need to be communicated to a destination neuron. We consider a scenario where spike 0 and 2 are received earlier at the destination neurons at time t = 5ns and 30ns, and spike 1 is re-routed due to congestion, reaching the destination neuron at t = 35ns. The spike rate received at the destination is therefore 40MHz and 200MHz, respectively. This spike disorder can lead to performance loss. We can formalize the definition of spike disorder as follows. Let $F^i = \{F_1^i, \cdots, F_{n_i}^i\}$ be the expected spike arrival rate at neuron i (from CARLsim) and $\hat{F}^i = \{\hat{F}_1^i, \cdots, \hat{F}_{n_i}^i\}$ be the actual spike rate considering hardware latencies. The spike disorder is computed as

$$\text{spike disorder} = \sum_{j=1}^{n_i} [(F_j^i - \hat{F}_j^i)^2]/n_i \qquad (22)$$

Our SpiNeCluster can be trivially extended with minimum effort to compute and minimize spike disorder.

2) Using SpiNeMap for other neuromorphic platforms: Our design methodology uses CARLsim to extract neural activity on every synapse of SNNs. CARLsim's support for built-in biologically realistic neuron, synapse, and computation models, designing new machine learning approaches and online learning algorithms, and continuous integration and testing, make it an easy to use and powerful simulator of biologically-plausible neural network models. The present release allows for the simulation using multiple GPUs and multiple CPU cores concurrently in a heterogeneous computing cluster. Benchmarking results demonstrate simulation of 8.6 million neurons and 0.48 billion synapses using 4 GPUs and up to 60x speedup for multi-GPU implementations over a single-threaded CPU implementation, making CARLsim 4 well- suited for large-scale SNN models in the presence of real-time constraints. Additionally, the present release adds new features, such as leaky-integrate-and-fire (LIF), 9-parameter Izhikevich, multi-compartment neuron models, and fourth order Runge Kutta integration.

SpiNeMap is a *general-purpose design methodology* for mapping SNN-based applications to neuromorphic hardware. We have seamlessly integrated SpiNeMap with both open-sourced SNN simulators such as Brian [57] and proprietary simulators such as XNet [66]. As the input for SpiNeMap is the precise time of neural activity on every synapse, SpiNeMap can be extended with minimum effort to consider any SNN simulator that allows extracting spike timing information.

Our SpiNePlacer uses the Noxim [36] simulator for cycle-accurate simulation of neuromorphic interconnect. To this end, we have previously evaluated many other simulators such as BookSim2 [67] and NIRGAM [68] for neuromorphic computing. Noxim allows significant advantage in terms of trace-driven simulations, extensions to other interconnect types, etc. See our prior work [69] for discussion of these alternatives. Our design-methodology SpiNeMap can be trivially extended to consider other interconnect simulators as long as they support 1) cycle-accurate simulation, and 2) trace-driven simulation. The former requirement is necessary to precisely compute the spike latency, which impacts performance (such as accuracy) of spike-based computation model. The second requirement is necessary to simulate application-level spike behavior in hardware considering delays on the interconnect.

Finally, our SpiNeMap is demonstrated to work with the DynapSE neuromorphic hardware [7]. Our continuing work is to support other neuromorphic architectures including TrueNorth [20] and Loihi [6]. We have open-sourced our framework to foster future research in neuromorphic computing.

REFERENCES

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

[2] A. Das, P. Pradhapan, W. Groenendaal, P. Adiraju, R. T. Rajan, F. Catthoor, S. Schaafsma, J. L. Krichmar, N. Dutt, and C. Van Hoof, "Unsupervised heart-rate estimation in wearables with liquid states and a probabilistic readout," *Neural Networks*, 2018.

[3] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[4] 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 *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[5] F. Akopyan, J. Sawada *et al.*, "TrueNorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[6] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

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

[8] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, and others, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017.

[9] T. Sauer, "Interspike interval embedding of chaotic signals," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 5, no. 1, pp. 127–132, 1995.

[10] A. Ankit, A. Sengupta, and K. Roy, "Neuromorphic computing across the stack: Devices, circuits and architectures," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2018, pp. 1–6.

[11] X. Zhang, A. Huang, Q. Hu, Z. Xiao, and P. K. Chu, "Neuromorphic computing with memristor crossbar," *physica status solidi (a)*, vol. 215, no. 13, p. 1700875, 2018.

[12] Q. Xia and J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature materials*, vol. 18, no. 4, p. 309, 2019.

[13] M. K. F. Lee, Y. Cui, T. Somu, T. Luo, J. Zhou, W. T. Tang, W.-F. Wong, and R. S. M. Goh, "A system-level simulator for rram-based neuromorphic computing chips," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, p. 64, 2019.

[14] 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 Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 345–358, 2018.

[15] W. Wen, C.-R. Wu, X. Hu, B. Liu, T.-Y. Ho, X. Li, and Y. Chen, "An eda framework for large scale hybrid neuromorphic computing systems," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[16] 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 *International Conference on Computing Frontiers*, 2012.

[17] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen, "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2016.

[18] 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 *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, March 2018, pp. 1217–1222.

[19] Y. Orii, A. Horibe, K. Matsumoto, T. Aoki, K. Sueoka, S. Kohara, K. Okamoto, S. Yamamichi, K. Hosokawa, and H. Mori, "Advanced interconnect technologies in the era of cognitive computing," in *Pan Pacific Microelectronics Symposium (Pan Pacific)*, 2016.

[20] M. V. DeBole, B. Taba, A. Amir, F. Akopyan, A. Andreopoulos, W. P. Risk, J. Kusnitz, C. O. Otero, T. K. Nayak, R. Appuswamy, and others, "TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years," *Computer*, vol. 52, no. 5, pp. 20–29, 2019.

[21] A. Balaji, Y. Wu, A. Das, F. Catthoor, and S. Schaafsma, "Exploration of segmented bus as scalable global interconnect for neuromorphic computing," in *Great Lakes Symposium on VLSI*. ACM, 2019.

[22] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017.

[23] R. P. N. Rao and T. J. Sejnowski, "Spike-timing-dependent Hebbian plasticity as temporal difference learning," *Neural computation*, 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 research*, vol. 53, no. 1, pp. 17–27, 1991.

[25] S. Grün and S. Rotter, *Analysis of parallel spike trains*. Springer, 2010, vol. 7.

[26] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[27] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, pp. 2531–2560, 2002.

[28] T. Chou, H. J. Kashyap, J. Xing, S. Listopad, E. L. Rounds, M. Beyeler, N. Dutt, and J. L. Krichmar, "Carlsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters," in *2018 International Joint Conference on Neural Networks (IJCNN)*, July 2018, pp. 1–8.

[29] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *International Symposium on Micro Machine and Human Science (MHS)*. IEEE, 1995, pp. 39–43.

[30] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[31] A. Das, A. Kumar, and B. Veeravalli, "Communication and migration energy aware task mapping for reliable multiprocessor systems," *Future Generation Computer Systems*, vol. 30, pp. 216–228, 2014.

[32] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *Proceedings of the sixth annual ACM symposium on Theory of computing*. ACM, 1974, pp. 47–63.

[33] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Design Automation Conference*. IEEE, 1982, pp. 175–181.

[34] J. Kennedy, "Particle swarm optimization," *Encyclopedia of machine learning*, pp. 760–766, 2010.

[35] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, pp. 127–138, 2017.

[36] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2015.

[37] 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 Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 12, no. 3, p. 23, 2007.

[38] L. Benini and G. De Micheli, "Networks on chip: A new paradigm for systems on chip design," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 2002, pp. 418–419.

[39] G. Indiveri, F. Corradi, and N. Qiao, "Neuromorphic architectures for spiking deep neural networks," in *International Electron Devices Meeting (IEDM)*. IEEE, 2015, pp. 4–2.

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

[41] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, 2015.

[42] A. Balaji, F. Corradi, A. Das, S. Pande, S. Schaafsma, and F. Catthoor, "Power-accuracy trade-offs for heartbeat classification on neural networks hardware," *Journal of Low Power Electronics*, vol. 14, 2018.

[43] *MLPerf: Fair and useful benchmarks for measuring training and inference performance of ML hardware, software, and services. https://mlperf.org/training-overview/*.

[44] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[45] *N2D2: Neural Network Design and Deployment. https://github.com/CEA-LIST/N2D2*.

[46] A. Balaji, Y. Wu, A. Das, F. Catthoor, and S. Schaafsma, "Exploration of Segmented Bus As Scalable Global Interconnect for Neuromorphic Computing," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019.

[47] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[48] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, 2013, pp. 6645–6649.

[49] R. Brette, "Philosophy of the spike: rate-based vs. spike-based theories of the brain," *Frontiers in systems neuroscience*, vol. 9, p. 151, 2015.

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

[51] D. Verstraeten, B. Schrauwen, and D. Stroobandt, "Reservoir-based techniques for speech recognition," in *International Joint Conference on Neural Network Proceedings*. IEEE, 2006, pp. 1050–1053.

[52] 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 *2009 International Joint Conference on Neural Networks*. IEEE, 2009, pp. 1011–1017.

[53] S. G. Ramasubramanian, R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "Spindle: Spintronic deep learning engine for large-scale neuromorphic computing," in *International symposium on Low power electronics and design*. ACM, 2014, pp. 15–20.

[54] A. Mallik, D. Garbin, A. Fantini, D. Rodopoulos, R. Degraeve, J. Stuijt, A. Das, S. Schaafsma, P. Debacker, G. Donadio *et al.*, "Design-technology co-optimization for oxrram-based synaptic processing unit," in *2017 Symposium on VLSI Technology*. IEEE, 2017, pp. T178–T179.

[55] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2008.

[56] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in neuroinformatics*, 2009.

[57] D. F. Goodman and R. Brette, "The brian simulator," *Frontiers in neuroscience*, vol. 3, p. 26, 2009.

[58] E. Yavuz, J. Turner, and T. Nowotny, "Genn: a code generation framework for accelerated brain simulations," *Scientific reports*, vol. 6, p. 18854, 2016.

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

[60] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–10.

[61] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng, "Zero-shot learning through cross-modal transfer," in *Advances in neural information processing systems*, 2013, pp. 935–943.

[62] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[63] D. L. Silver, Q. Yang, and L. Li, "Lifelong machine learning systems: Beyond learning algorithms," in *2013 AAAI spring symposium series*, 2013.

[64] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[65] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[66] 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 *2013 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2013, pp. 7–12.

[67] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, "Booksim 2.0 users guide," *Standford University*, 2010.

[68] L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, and A. Narayanan, "Nirgam: a simulator for noc interconnect routing and application modeling," in *Design, Automation and Test in Europe Conference*, 2007, pp. 16–20.

[69] K. Huynh, "Exploration of dynamic communication networks for neuromorphic computing," 2016.

Adarsha Balaji Adarsha Balaji received a Bachelors degree from Visvesvaraya Technological University, India, in 2012 and a Master's degree from Drexel University, Philadelphia, PA, in 2017. He is currently pursuing a Ph.D. degree from the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA. His current research interests include design of neuromorphic computing systems, particularly data-flow and power optimization of spiking neural networks (SNN) hardware.

Anup Das Dr. Anup Das is an Assistant Professor at Drexel University. He received a Ph.D. in Embedded Systems from National University of Singapore in 2014. Prior to his Ph.D., he was a research engineer for more than 7 years at ST Microelectronics (India and Grenoble) and LSI Corporation (India). Following his Ph.D., he was a post-doctoral fellow at the University of Southampton and a researcher at IMEC. His research focuses on neuromorphic computing and architectural exploration. He is a senior member of the IEEE.

Yuefeng Wu Yuefeng was enrolled in a joint master program of KTH, Royal Institute of Technology, Stockholm, Sweden and Technology University of Eindhoven after receiving his bachelor degree from Tianjin University. He worked at IMEC NL for his master thesis and researched on the communication mechanisms of neuromorphic computing. He designed and implemented the simulator for communication based on Noxim. He joined ING Groep N.V. as a management trainee in the track of IT after graduation and currently works as an information architect.

Khanh Huynh Biography not available.

Francesco G. Dell'Anna Francesco G. Dell'Anna was born in Gallipoli, Italy, on 14 January 1993. He received the BE degree in computer engineering and the ME degree in embedded systems from Polytechnic of Turin in 2014 and 2016 respectively. In 2016 he attended the electrical engineering master program at KULeuven, working on a neuromorphic simulator in IMEC (Belgium). He is currently a researcher in the Institute of Applied Micro-Nano Systems Technology, Key Laboratory of Micro-Nano System Technology and Smart Transduction, Chongqing Technology and Business University, Chongqing, China, and a Ph.D. student in the department of Micro- and Nanotechnology systems at the university college of southeast Norway. In 2018 he then joined Omnivision Technology as a Digital Designer in Oslo. His research interests include image sensors, neural networks, piezoelectric energy harvesters and low power electronic designs.

Giacomo Indiveri Giacomo Indiveri is a Professor at the Faculty of Science of the University of Zurich, Switzerland, director of the Institute of Neuroinformatics (INI) of the University of Zurich and ETH Zurich, and head of the Neuromorphic Cognitive Systems group at INI. Indiveri was awarded an ERC starting grant in 2011, and an ERC consolidator grant in 2017. 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 technology. The circuits he develops are designed to emulate the physics of computation of biological neural processing systems and are aimed at building autonomous agents that can learn and reason about the actions to take in response to the combinations of external stimuli, internal states, and behavioral objectives. These ”neuromorphic cognitive agents” are used to validate brain inspired computational paradigms in real-world scenarios, and to develop a new generation of fault-tolerant event-based neuromorphic computing technologies.

Jeffrey L. Krichmar Jeffrey L. Krichmar received a B.S. in Computer Science in 1983 from the University of Massachusetts at Amherst, a M.S. in Computer Science from The George Washington University in 1991, and a Ph.D. in Computational Sciences and Informatics from George Mason University in 1997. He spent 15 years as a software engineer on projects ranging from the PATRIOT Missile System at the Raytheon Corporation to Air Traffic Control for the Federal Systems Division of IBM. From 1999 to 2007, he was a Senior Fellow in Theoretical Neurobiology at The Neurosciences Institute. He currently is a professor in the Department of Cognitive Sciences and the Department of Computer Science at the University of California, Irvine. Krichmar has nearly 20 years experience designing adaptive algorithms, creating neurobiologically plausible network simulations, and constructing brain-based robots whose behavior is guided by neurobiologically inspired models. He has over 100 publications and holds 7 patents. His research interests include neurorobotics, embodied cognition, biologically plausible models of learning and memory, neuromorphic applications and tools, and the effect of neural architecture on neural function. He is a Senior Member of IEEE and the Society for Neuroscience.

Nikil D. Dutt Nikil D. Dutt (F) received a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1989, and is currently a Distinguished Professor of Computer Science, Cognitive Sciences, and EECS at the University of California, Irvine. He is also a Distinguished Visiting Professor in the CSE department at IIT Bombay, India. Dutts research interests are in embedded systems, electronic design automation (EDA), computer systems architecture and software, healthcare IoT, and brain-inspired architectures and computing. He received over a dozen best paper awards and nominations at premier EDA and embedded systems conferences and is coauthor of 7 books on topics covering hardware synthesis, memory and computer architecture specification and validation, and on-chip networks. Dutt has served as Editor-in-Chief of ACM TODAES and as Associate Editor for ACM TECS and IEEE TVLSI. He has extensive service on the steering, organizing, and program committees of several premier EDA and Embedded System Design conferences and workshops, and also serves or has served on the advisory boards of ACM SIGBED, ACM SIGDA, ACM TECS, IEEE Embedded Systems Letters (ESL), and the ACM Publications Board.

He is a Fellow of the ACM, Fellow of the IEEE, and recipient of the IFIP Silver Core Award.

Siebren Schaafsma Dr. Siebren Schaafsma is an R&D manager in the IoT unit of Imec The Netherlands (Imec-nl). This unit is part of the Holst Center in Eindhoven. He is responsible for two teams of Analog and Digital IC designers building new state of the art Radio ICs and sub GHz Radar (BT-LE, Wifi 11.ah, subGHz, etc). He is also responsible for a team of embedded hardware and software engineers working in the domain of IoT and Artificial Intelligence. He received two masters (Nuclear physics in 1988 and computer science in 1989) at the Rijks Universiteit Groningen (RUG). His dissertation in the latter one addresses a neural networks implementation on a transputer cluster. He received his Ph.D. (Dr.) from the University of Nijmegen (KUN) in the Biophysics Department. His dissertation addresses the coding of optic flow in the visual cortex. He holds two patents on his research inventions from his period in research at Ericsson Telecommunications.

Francky Catthoor Dr. Francky Catthoor received a Ph.D. in EE from the Katholieke Univ. Leuven, Belgium in 1987. Between 1987 and 2000, he has headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000 he is strongly involved in other activities at IMEC including deep submicron technology aspects, IoT and biomedical platforms, and smart photovoltaic modules, all at IMEC Leuven, Belgium. Currently he is an IMEC fellow. He is also part-time full professor at the EE department of the KULeuven.

He has been associate editor for several IEEE and ACM journals. He was elected IEEE fellow in 2005.