Journal of Signal Processing Systems (2020) 92:1293–1302
https://doi.org/10.1007/s11265-020-01573-8

Run-time Mapping of Spiking Neural Networks to Neuromorphic Hardware

Adarsha Balaji[1] · **Thibaut Marty**[2] · **Anup Das**[1] · **Francky Catthoor**[3]

Received: 20 September 2019 / Revised: 15 June 2020 / Accepted: 18 June 2020 / Published online: 28 July 2020

Abstract

Neuromorphic architectures implement biological neurons and synapses to execute machine learning algorithms with spiking neurons and bio-inspired learning algorithms. These architectures are energy efficient and therefore, suitable for cognitive information processing on resource and power-constrained environments, ones where sensor and edge nodes of internet-of-things (IoT) operate. To map a spiking neural network (SNN) to a neuromorphic architecture, prior works have proposed design-time based solutions, where the SNN is first analyzed offline using representative data and then mapped to the hardware to optimize some objective functions such as minimizing spike communication or maximizing resource utilization. In many emerging applications, machine learning models may change based on the input using some online learning rules. In online learning, new connections may form or existing connections may disappear at run-time based on input excitation. Therefore, an already mapped SNN may need to be re-mapped to the neuromorphic hardware to ensure optimal performance. Unfortunately, due to the high computation time, design-time based approaches are not suitable for remapping a machine learning model at run-time after every learning epoch. In this paper, we propose a design methodology to partition and map the neurons and synapses of online learning SNN-based applications to neuromorphic architectures at run-time. Our design methodology operates in two steps – step 1 is a layer-wise greedy approach to partition SNNs into clusters of neurons and synapses incorporating the constraints of the neuromorphic architecture, and step 2 is a hill-climbing optimization algorithm that minimizes the total spikes communicated between clusters, improving energy consumption on the shared interconnect of the architecture. We conduct experiments to evaluate the feasibility of our algorithm using synthetic and realistic SNN-based applications. We demonstrate that our algorithm reduces SNN mapping time by an average 780x compared to a state-of-the-art design-time based SNN partitioning approach with only 6.25% lower solution quality.

Keywords Spiking Neural Networks (SNN) · Neuromorphic computing · Internet of Things (IoT) · Run-time · Mapping

1 Introduction

Internet of things (IoT) is an emerging computing paradigm that enables the integration of ubiquitous sensors over a wireless network [3]. Recent estimates predict that over 50 billion IoT devices will be interconnected via the cloud over the next decade [22]. In a conventional IoT, data collected from sensors and actuators are transferred to the cloud and processed centrally [34]. However, with an increase in the number of connected IoT devices, processing on the cloud becomes the performance and energy bottleneck [41].

Edge computing is emerging as a scalable solution to process large volumes of data by executing machine learning tasks closer to the data source e.g. on a sensor or an edge node [40]. Processing on edge devices allows real-time data processing and decision making, and offers network scalability and privacy benefits as data transferred to the cloud over a possibly insecure communication channel is minimized [31, 32].

Spiking neural networks (SNNs) [29] are extremely energy efficient in executing machine learning tasks on event-driven neuromorphic architectures such as TrueNorth [2], DYNAP-SE [36], and Loihi [19], making them suitable for machine learning-based edge computing. A neuromorphic architecture is typically designed using *crossbars*, which can accommodate only a limited number of synapses per neuron to reduce energy consumption. To build a large neuromorphic chip, multiple crossbars are integrated using

✉ Adarsha Balaji
adarsha.balaji@drexel.edu

Extended author information available on the last page of the article.

a shared interconnect such as network-on-chips (NoC) [7]. To map an SNN to these architectures, the common practice is to partition the neurons and synapses of the SNN into clusters and map these clusters to the crossbars, optimizing hardware performance such as minimizing the number of spikes communicated between crossbar, which reduces energy consumption [16].

Most prior works on machine learning-based edge computing focus on supervised approaches, where neural network models are first trained offline with representative data from the field and then deployed on edge devices to perform inference in real-time [39]. However, data collected by IoT sensors constantly evolve over time and may not resemble the representative data used to train the neural network model. This change in the relation between the input data and an offline trained model is referred to as *concept drift* [23]. Eventually, the concept drift will reduce the prediction accuracy of the model over time, lowering its quality. Therefore, there is a clear need to periodically re-train the model using recent data with adaptive learning algorithms. Examples of such algorithms include transfer learning [38], lifelong learning [43] and deep reinforcement learning [33].

Mapping decisions for a supervised SNN are made at design-time before the initial deployment of the trained model. However, in the case of online learning, when the model is re-trained, (1) synaptic connections within the SNN may change, i.e. new connections may form and existing connection may be removed as new events are learned, and (2) weights of existing synaptic connections may undergo changes after every learning epoch. In order to ensure the optimal hardware performance at all times, a *run-time* approach is required that remaps the SNN to the hardware after every learning epoch. Prior methods to partition and map an SNN to neuromorphic hardware, such as PSOPART [16], SpiNeMap [6], PyCARL [4], NEUTRAMS [25] and DFSynthesizer [42] are design-time approaches that require significant exploration time to generate a good solution. Although suitable for mapping supervised machine learning models, these approaches cannot be used at run-time to remap SNNs frequently. For online learning, we propose an approach to perform run-time layer-wise mapping of SNNs on to crossbar-based neuromorphic hardware. The approach is implemented in two steps. First, we perform a layer-wise greedy clustering of the neurons in the SNN. Second, we use an instance of hill-climbing optimization (HCO) to lower the total number of spikes communicated between the crossbars.

Contributions: Following are our key contributions.

- We propose an algorithm to partition and map online learning SNNs on to neuromorphic hardware for IoT applications in run-time;
- We demonstrate suitability of our approach for online mapping in terms of the exploration time and total number of spikes communicated between the crossbars, when compared to a state-of-the-art design time approach.

The remainder of this paper is organized as follows, Section 2 presents the background, Section 3 discusses the problem of partitioning a neural network into clusters to map on to the crossbars neuromorphic hardware and describes our two-step approach. Section 5 presents the experimental results based on synthetic applications. Section 6 concludes the paper followed by a discussion in Section 7.

2 Background

Spiking neural networks are event-driven computational models inspired by the mammalian brain. Spiking neurons are typically implemented using Integrate-and-Fire (I&F) models [9] and communicate using short impulses, called *spikes*, via synapses. Figure 1a illustrates an SNN with *two* pre-synaptic neurons connected to a post-synaptic neuron via synaptic elements with weights $w1$, $w2$ respectively. When a pre-synaptic neuron generates a spike, current is injected into the post-synaptic neuron, proportional to the product of the spike voltage and the conductance of the respective synapse. SNNs are trained by adjusting the

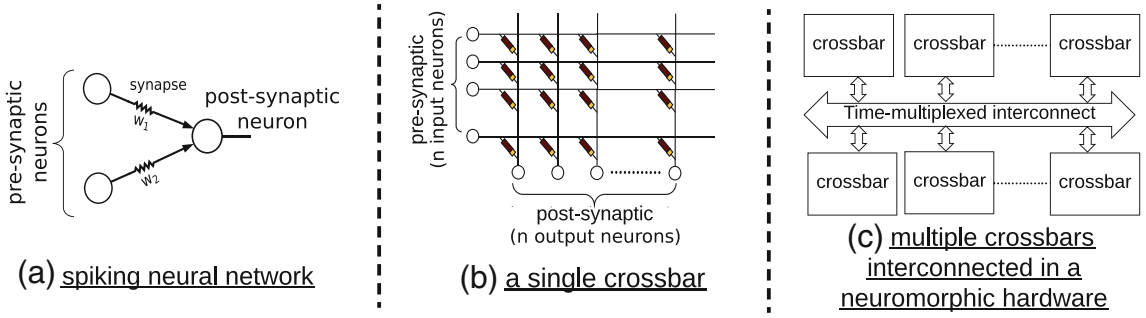


Figure 1 Overview of a SNN hardware: **a** connection of pre- and post-synaptic neurons via synapses in a spiking neural network, **b** a crossbar organization with fully connected pre- and post-synaptic neurons, and **c** a modern neuromorphic hardware with multiple crossbars and a time-multiplexed interconnect.

synaptic weights using a supervised, a semi-supervised, or an unsupervised approach [27, 28, 37].

Due to the ultra-low power footprint of neuromorphic hardware, several machine learning applications based on SNNs are implemented. In [18], the authors propose a multi-layer perceptron (MLP) based SNN to classify heartbeats using electrocardiagram (ECG) data. In [21], the authors propose the handwritten digit recognition using unsupervised SNNs. In [15], a spiking liquid state machine for heart-rate estimation is proposed. A SNN-based liquid state machine (LSM) for facial recognition is proposed in [24]. In [5], the authors propose a technique to convert a convolutional neural network (CNN) model for heartbeat classification into a SNN, with a minimal loss in accuracy.

Typically, SNNs are executed on special purpose neuromorphic hardware. These hardware can (1) reduce energy consumption, due to their low-power designs, and (2) improve application throughput, due to their distributed computing architecture. Several digital and mixed-signal neuromorphic hardware are recently developed to execute SNNs, such as Neurogrid [8], TrueNorth [1] and DYNAP-SE [35]. Although these hardware differ in their operation (analog vs. digital), they all support crossbar-based architectures. A crossbar is a two-dimensional arrangement of synapses (n^2 synapses for n neurons). Figure 1b illustrates a single crossbar with n pre-synaptic neurons and n post-synaptic neurons. The pre- and post-synaptic neurons are connected via synaptic elements. Crossbar size (n) is limited (<512) as scaling the size of the crossbar will lead to an exponential increase in dynamic and leakage energy. Therefore, to build large neuromorphic hardware, multiple crossbars are integrated using a shared interconnect, as illustrated in Figure 1c.

In order to execute an SNN on a neuromorphic hardware, the SNN is first partitioned into clusters of neurons and synapses. The clustered (local) synapses are then mapped to the crossbars and the inter-cluster synapses to the time-multiplexed interconnect. Several design time partitioning approach are presented in literature. In [44–46] the authors proposes techniques to efficiently map the neurons and synapses on a crossbar. The aim of these techniques is to maximize the utilization of the crossbar. NEUTRAMS partitions the SNN for crossbar-based neuromorphic hardware [26]. The NEUTRAMs approach also looks to minimize the energy consumption of the neuromorphic hardware executing the SNN. PyCARL [4] facilitates the hardware-software co-simulation of SNN-based applications. The framework allows users to analyze and optimize the partitioning and mapping of an SNN on cycle-accurate models of neuromorphic hardware. DFSynthesizer [42] uses a greedy technique to partition the neurons and synapses of an SNN. The SNN partitions are mapped to the neuromorphic hardware using an algorithm that adapts to the available resources of the hardware. SpiNeMap [6] uses a greedy partitioning technique to partition the SNN followed by a meta-heuristic-based technique to map the partitions on the hardware. PSOPART SNNs to a crossbar architecture [17]. The objective of SpiNeMap and PSOPART is to minimize the spike communication on the time-multiplexed interconnect in order to improve the overall latency and power consumption of the DYNAP-SE hardware. Table 1 compares our contributions to the state-of-the-art techniques.

As these partitioning approaches aim to find the optimal hardware performance, their exploration time is relatively large and therefore not suitable for partitioning and re-mapping of online learning SNNs. Run-time approaches are proposed for task mapping on multiprocessor systems. A heuristic-based run-time manager is proposed in [12]. The run-time manager controls the thread allocation and voltage/frequency scaling for energy efficient execution of applications on multi processor systems. In [30], the authors propose a genetic algorithm-based run-time manager to schedule real-time tasks on Dynamic Voltage Scaling (DVS) enabled processors, with an aim to minimize energy consumption. A workload aware thread scheduler is proposed in [20] for multi-processor systems. In [14], the authors propose a multinomial logistic regression model to partition the input workload in run-time. Each partition is then executed at pre-determined frequencies to ensure minimum energy consumption. In [13], the authors propose a technique to remap tasks run on faulty processors with a minimal migration overhead. A thermal-aware task scheduling approach is proposed in [11] to estimate and reduce the temperature of the multi processor system at

Table 1 Summary of related works.

Related works	Run-time mapping	Objective
[44–46]	×	Maximize single crossbar utilization
NEUTRAMS [25]	×	Minimize number of crossbars utilized
SpiNeMap [6]	×	Minimize spikes on time-multiplexed interconnect
PSOPART [16]	×	Minimize spikes on time-multiplexed interconnect
DFSynthesizer [42]	×	Optimize the hardware utilization in run-time
Proposed	√	Reduces energy consumption of online learning SNNs on hardware.

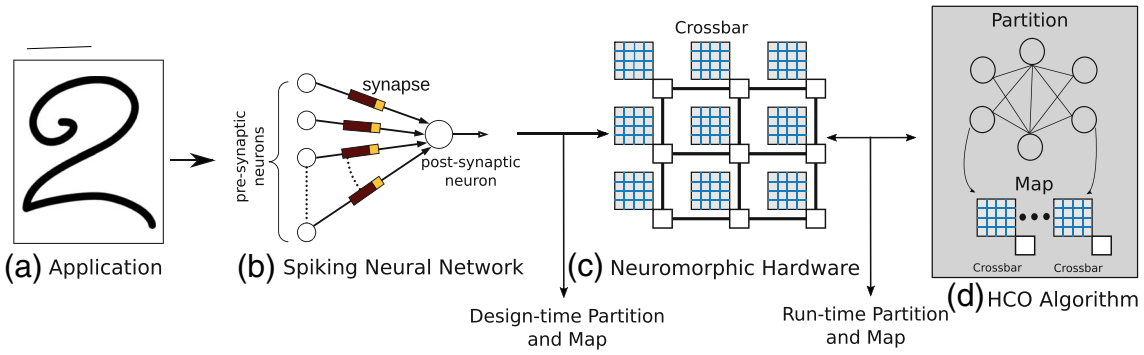


Figure 2 Mapping of online learning SNN on Neuromorphic Hardware.

run-time. The technique performs an extensive design-time analysis of fault scenarios and determines the optimal mapping of tasks in run-time. However, such run-time techniques to remap SNN on neuromorphic hardware are not proposed. To the best of our knowledge, this is the first work to propose a run-time mapping approach with a significantly lower execution time when compared to existing design-time approaches. Our technique reduces the spikes communicated on the time-multiplexed interconnect, therefore reducing the energy consumption.

3 Methodology

The proposed method to partition and map an SNN in run-time is illustrated in Figure 2 illustrates. The network model is built using a directed graph, wherein each edge represents a synapse whose weight is the total number of spikes communicated between the two SNN neurons. The input to the mapping algorithm is a list of all the neurons *(A)*, the total number of spikes communicated over each synapse and the size of a crossbar *(k)*. The mapping algorithm is split into two steps, as shown in Figure 3.

Figure 4 illustrates the partitioning of an SNN with 6 neurons into 3 sub-lists. The spikes communicated between the neurons is indicated on the synapse. First, we divide the input list of neurons into sub-lists (Section 3.1), such that each sub-list can be mapped to an available crossbar. Second, we reduce the number of spikes communicated between the sub-lists (Section 3.2), by moving the neurons between the sub-list (indicated in blue).

3.1 Building Sub-lists

Algorithm 1 describes the greedy partitioning approach. The objective is to greedily cut the input list of neurons *(A)* into s sub-lists, where s is the total number of crossbars in the given design. The size of a sub-list is determined by the size of the crossbars *(k)* on the target hardware. A variable *margin (line 3)* is defined to store the unused neuron slots available in each sub-list. The *mean (line 4)* number of spikes generated per crossbar is computed using the total number of spikes communicated in the SNN-based application. A *cost* function (Algorithm 2) is defined to compute the total number of spikes communicated (cost) between each of the sub-lists.

The algorithm iterates over the neurons (n_i) in the input list (A) and updates the slots in the current sub-list *(line 8)*. Neurons are added to the current sub-list until one of following two criteria are met - (1) the length of the sub-list equals k, or (2) the cost (number of spikes) is greater than the *mean* value and sufficient extra slots ($margin$) are still available. When the criteria is met, the current sublist is validated and its boundary stored. When the penultimate sub-list is validated, the execution ends because the boundary of the last sub-lists is already known (nth element in list). The list p contains the sub-lists boundaries.

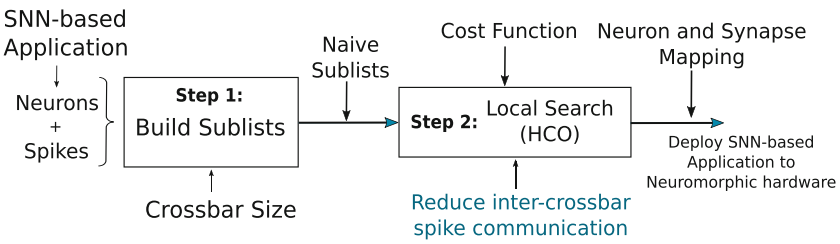


Figure 3 Overview of proposed partitioning algorithm.

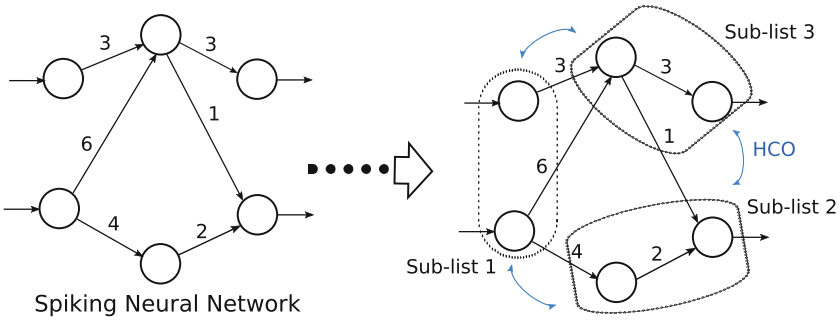


Figure 4 Partitioning of an SNN.

Algorithm 1 Building sublists.

```
1  procedure FUNCTION (A[1 → n])
2  foreach Crossbar s ∈ p do
       /* iterate over all crossbars in p   */
3      Input the variable margin;
       /* Mean spikes per crossbar          */
4      Compute Mean;
       /* iterate over all neurons in A     */
5      foreach n_i ∈ A do
           /* Cost is the number of spikes in
              current cluster                */
6          Compute Cost;
7          while Cost ≤ Mean do
8              Assign n_i to crossbar p;
9          end
10     end
11 end
```

Algorithm 2 Cost function.

```
1  procedure FUNCTION (A[1 → n], p[1 → s])
2  max ← 0;
3  foreach Cluster (p[i]) do
4      sum ← 0 ;
5      foreach n in p[i] do
           /* total spikes communicated      */
6          compute Sum;
7      end
8      if Sum > Max then
9          Max ← Sum;
10     end
11 end
```

Algorithm 3 Hill climbing algorithm.

```
1  procedure FUNCTION (A[1 → n], p[1 → s])
   /* compute the initial cost              */
2  compute Cost;
3  foreach n in A do
4      move n across cluster boundary;
5      compute new Cost C_n;
6      select min(C_n);
7  end
   /* end 2-part procedure                  */
```

3.2 Local Search

The solution obtained from Algorithm-1 is naive and not optimal. Although each sublist s obtained from Algorithm-1 meets the cost criteria, it is possible to have unevenly distributed costs across the sublists. We search for a better solution by performing multiple local searches to balance the cost. This is done by using the hill-climbing optimization technique to iterate through the sublist and *move* its boundary

Algorithm 3 describes the hill-climbing optimization technique. The technique relies on a *cost function (line 2)* to compute and evaluate a solution. The cost function used in the optimization process is shown in Algorithm 2. The cost function computes the maximum cost (number of spikes) for a chosen sub-list. The optimal solution should contain the lowest cost. The algorithm iterates through each subslist to search for the best solution (cost) of its neighbors. The algorithm begins by moving the boundary of a sub-list one position to the left or one position to the right. Each neuron (n_i) in the sublist is moved across the boundary to a neighboring sub-list and the *cost* of the neighbors are computed. The algorithm selects the solution with the local minimum cost. The process is repeated for every neuron in the list (A) until the sub-lists with the minimum cost is found.

4 Evaluation

4.1 Simulation Environment

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

- **CARLsim** [10] : A GPU accelerated simulator used to train and test SNN-based applications. CARLsim reports spike times for every synapse in the SNN.
- **DYNAP-SE** [36]: Our approach is evaluated using the DYNAP-SE model, with 256-neuron crossbars interconnected using a NoC. [47].

Table 2 Applications used for evaluating.

Category	Applications	Synapses	Topology	Spikes
synthetic	S_1000	240,000	FeedForward (400, 400, 100)	5,948,200
	S_2000	640,000	FeedForward (800, 400, 800)	45,807,200
realistic	EdgeDet [10]	272,628	FeedForward (4096, 1024, 1024, 1024)	22,780
	MLP-MNIST [21]	79,400	FeedForward (784, 100, 10)	2,395,300

4.2 Evaluated Applications

In order to evaluate the online mapping algorithm, we use 2 synthetic and 2 realistic SNN-based applications. Synthetic applications are indicated with an 'S_' followed by the number of neurons in the application. Edge detection (EdgeDet) and MLP-based digit recognition (MLP-MNIST) are the two realistic applications used. Table 2 also indicates the number of synapses (column 3), the topology (column 4) and the number of spikes for the application obtained through simulations using CARLsim [10].

4.3 Evaluated Design-time vs run-time Approach

In order to compare the performance of our proposed run-time approach, we choose a state-of-the-art design-time approach as the baseline. The crossbar size for both the algorithms is set to 256 (k=256).In this paper we compare the following approaches:

- *PSOPART* [16]: The PSOPART approach is a design-time partitioning technique that uses and instance of particle swarm optimization (PSO) to minimize the number of spikes communicated on the time-multiplexed interconnect.
- *HCO-Partitioning*: Our HCO-partitioning approach is a two-step layer-wise partitioning technique with a greedy partitioning followed by a HCO-based local search approach to reduce the number of spikes communicated between the crossbars.

5 Results

Table 3 reports the execution time (in seconds) of the design-time and run-time mapping algorithms for synthetic and realistic applications, respectively. We make the following two observations. *First*, on average, our HCO partitioning algorithm has an execution time 780x lower than that of the PSOPART algorithm. Second, the significantly lower run-time of the HCO partitioning algorithm (<50 seconds) allows for the online learning SNN to be re-mapped on the edge devices, before the start of the next training epoch.

Figure 5 shows the lifetime of an online learning application with respect to the execution times of each training epoch (t) and the HCO partitioning algorithm (h). The execution time of the partitioning algorithm needs to be significantly lower than the time interval between training epochs. This is achieved with the HCO-partitioning algorithm as its execution time is significantly (780x) lower than the state-of-the-art design-time approaches.

In Figure 6, we compare the number of spikes communicated between the crossbars while partitioning the SNN using the HCO partitioning algorithm when compared to the design-time PSOPART approach. We see that, on average, the PSOPART algorithm reduces the number of spikes by a further 6.25%, when compared to the HCO partitioning algorithm. The PSOPART will contribute to a further reduction in the overall energy consumed on the neuromorphic hardware. However, this outcome is expected as the design-time partitioning approach is afforded far more exploration time to minimize the number of spikes communicated between the

Table 3 Execution time of design-time and proposed run-time approach in seconds.

Category	Applications	PSOPART (sec)	HCO-Partition (sec)
synthetic	S_1000	20011.33	19.10
	S_2000	45265.00	24.68
realistic	EdgeDet	6771.02	45.62
	MLP-MNIST	5153.41	11.03

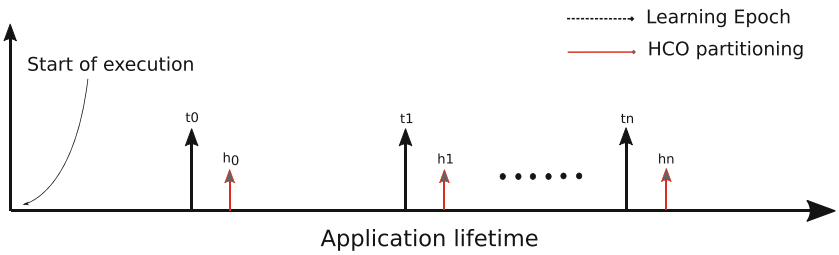


Figure 5 Life-time of online learning SNN.

crossbars. Also, the effects of *concept drift* will soon lead to the design-time solution becoming outmoded. Therefore, a run-time partitioning and re-mapping of the SNN will significantly improve the performance of the SNN on the neuromorhpic hardware and mitigate the effects of *concept drift*.

6 Conclusion

In this paper, we propose an algorithm to re-map online learning SNNs on neuromorphic hardware. Our approach performs the run-time mapping in two steps: (1) a layer-wise greedy partitioning of SNN neurons, and (2) a hill-climbing based optimization of the greedy partitions with an aim to reduce the number of spikes communicated between the crossbars. We demonstrate the in-feasibility of using a state-of-the-art design-time approach to re-map online learning SNNs in run-time. We evaluate the our approach using synthetic and realistic SNN applications. Our algorithm reduces SNN mapping time by an average 780x when compared to a state-of-the-art design-time approach with only 6.25% lower performance.

7 Discussion

In this section we discuss the scalability of our approach. Each iteration of Algorithm-1 performs basic math operations. The hill-climbing algorithm computes as many as 2x(s-2) solutions, and performs a comparison to find the minimum cost across all the solutions. In our case, the co-domain of the cost function are well-ordered positive integers. The cost function is also linear in n, however the hill-climb optimization algorithm only terminates when the local minimum cost function is computed. Therefore, it is in our interest to optimize the number of times the cost function is to be run.

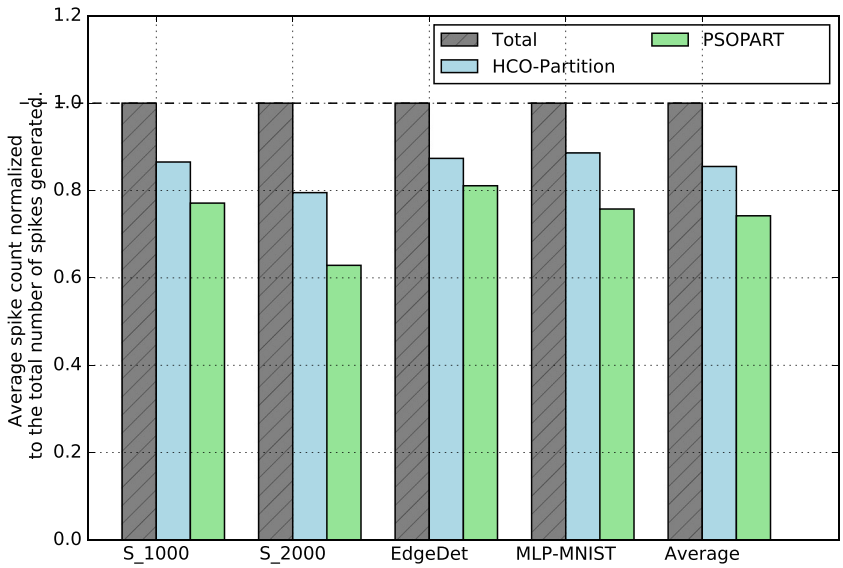


Figure 6 Number of spikes communicated on the time-multiplexed interconnect normalized to the total number of spikes generated.

Acknowledgments This work is supported by 1) the National Science Foundation Award CCF-1937419 (RTML: Small: Design of System Software to Facilitate Real-Time Neuromorphic Computing) and 2) the National Science Foundation Faculty Early Career Development Award CCF-1942697 (CAREER: Facilitating Dependable Neuromorphic Computing: Vision, Architecture, and Impact on Programmability).

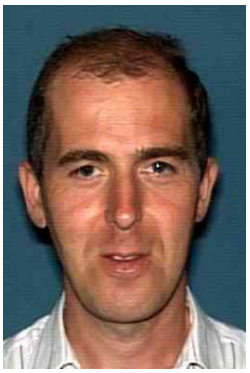
References

1. Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.J., et al. (2015). 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*, *34*(10), 1537–1557.
2. Akopyan, F., Sawada, J., et al. (2015). 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*, *34*(10), 1537–1557.
3. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M. (2015). Internet of things: a survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, *17*(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095.
4. Balaji, A., Adiraju, P., Kashyap, H.J., Das, A., Krichmar, J.L., Dutt, N.D., Catthoor, F. (2020). Pycarl: a pynn interface for hardware-software co-simulation of spiking neural network. In *2020 International joint conference on neural networks (IJCNN)*.
5. Balaji, A., Corradi, F., Das, A., Pande, S., Schaafsma, S., Catthoor, F. (2018). Power-Accuracy Trade-Offs for heartbeat classification on neural networks hardware. *Journal of Low Power Electronics*, *14*(4), 508–519.
6. Balaji, A., Das, A., Wu, Y., Huynh, K., Dell'Anna, F., Indiveri, G., Krichmar, J.L., Dutt, N., Schaafsma, S., Catthoor, F. (2019). Mapping Spiking Neural Networks on Neuromorphic Hardware. *IEEE transactions on VLSI systems*.
7. Benini, L., & De Micheli, G. (2002). Networks on chip: a new paradigm for systems on chip design. In *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition* (pp. 418–419): IEEE.
8. Benjamin, B.V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A.R., Bussat, J., Alvarez-Icaza, R., Arthur, J.V., Merolla, P.A., Boahen, K. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, *102*(5), 699–716.
9. Chicca, E., Badoni, D., Dante, V., et al. (2003). A vlsi recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory. *IEEE Transactions on Neural Networks, 14*.
10. Chou, T., Kashyap, H.J., Xing, J., Listopad, S., Rounds, E.L., Beyeler, M., Dutt, N., Krichmar, J.L. (2018). 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)* (pp. 1–8). https://doi.org/10.1109/IJCNN.2018.8489326.
11. Cui, J., & Maskell, D.L. (2012). A fast high-level event-driven thermal estimator for dynamic thermal aware scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *31*(6), 904–917.
12. Das, A., Al-Hashimi, B.M., Merrett, G.V. (2016). Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems. *ACM Trans. Embed. Comput. Syst 15*(2). https://doi.org/10.1145/2834120.
13. Das, A., & Kumar, A. (2012). Fault-aware task re-mapping for throughput constrained multimedia applications on noc-based mpsocs. In *International symposium on rapid system prototyping (RSP)*: IEEE.
14. Das, A., Kumar, A., Veeravalli, B., Shafik, R., Merrett, G., Al-Hashimi, B. (2015). Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems. In *Design, automation & test in europe conference & exhibition (DATE)*.
15. Das, A., Pradhapan, P., Groenendaal, W., Adiraju, P., Rajan, R.T., Catthoor, F., Schaafsma, S., Krichmar, J.L., Dutt, N., Van Hoof, C. (2017). Unsupervised heart-rate estimation in wearables with liquid states and a probabilistic readout. arXiv:1708.05356.
16. Das, A., Wu, Y., Huynh, K., Dell'Anna, F., Catthoor, F., Schaafsma, S. (2018). Mapping of local and global synapses on spiking neuromorphic hardware. In *Design, automation & test in europe conference & exhibition (DATE)* (pp. 1217–1222). https://doi.org/10.23919/DATE.2018.8342201.
17. Das, A., Wu, Y., Huynh, K., Dell'Anna, F., Catthoor, F., Schaafsma, S. (2018). Mapping of local and global synapses on spiking neuromorphic hardware. In *2018 Design, automation test in europe conference exhibition (DATE)* (pp. 1217–1222). https://doi.org/10.23919/DATE.2018.8342201.
18. Das, A.K., Catthoor, F., Schaafsma, S. (2018). Heartbeat classification in wearables using multi-layer perceptron and time-frequency joint distribution of ecg. In *2018 IEEE/ACM International conference on connected health: applications, Systems and Engineering Technologies (CHASE)* (pp. 69–74): IEEE.
19. Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99.
20. Dhiman, G., Ayoub, R., Rosing, T. (2009). PDRAM: a hybrid PRAM and DRAM main memory system. In *Proceedings of the Annual Design Automation Conference (DAC)* (pp. 469–664).
21. Diehl, P.U., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience, 9*.
22. Evans, D. (2011). The internet of things: How the next evolution of the internet is changing everything. *CISCO White Paper*, *1*(2011), 1–11.
23. Gama, J.a., Žliobaitundefined, I., Bifet, A., Pechenizkiy, M., Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Comput. Surv* 46(4). https://doi.org/10.1145/2523813.
24. Grzyb, B.J., Chinellato, E., Wojcik, G.M., Kaminski, W.A. (2009). Facial expression recognition based on liquid state machines built of alternative neuron models. In *2009 International joint conference on neural networks* (pp. 1011–1017): IEEE.
25. Ji, Y., Zhang, Y., Li, S., Chi, P., Jiang, C., Qu, P., Xie, Y., Chen, W. (2016). NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *International symposium on microarchitecture (MICRO)*: IEEE.
26. Ji, Y., Zhang, Y., Li, S., Chi, P., Jiang, C., Qu, P., Xie, Y., Chen, W. (2016). NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *International symposium on microarchitecture (MICRO)*.
27. Kasabov, N. (2001). Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *31*(6), 902–918.
28. Lee, J.H., Delbruck, T., Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*, 508.
29. Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671.

30. Mahmood, A., Khan, S.A., Albalooshi, F., Awwad, N. (2017). Energy-aware real-time task scheduling in multiprocessor systems using a hybrid genetic algorithm. *Electronics*, *6*(2), 40.
31. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B. (2017). Mobile edge computing: Survey and research outlook. arXiv:1701.01090.
32. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, *19*(4), 2322–2358.
33. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.
34. Mohammadi, M., Al-Fuqaha, A., Sorour, S., Guizani, M. (2018). Deep learning for iot big data and streaming analytics: a survey. *IEEE Communications Surveys & Tutorials*, *20*(4), 2923–2960.
35. Moradi, S., Qiao, N., Stefanini, F., Indiveri, G. (2018). A Scalable Multicore Architecture with Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs). *IEEE Transactions on Biomedical Circuits and Systems*, *12*(1), 106–122. https://doi.org/10.1109/TBCAS.2017.2759700.
36. Moradi, S., Qiao, N., Stefanini, F., Indiveri, G. (2018). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Transactions on Biomedical Circuits and Systems*, *12*(1), 106–122. https://doi.org/10.1109/TBCAS.2017.2759700.
37. Mostafa, H. (2018). Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(7), 3227–3235.
38. Pan, S.J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359.
39. Shafique, M., Hafiz, R., Javed, M.U., Abbas, S., Sekanina, L., Vasicek, Z., Mrazek, V. (2017). Adaptive and energy-efficient architectures for machine learning: challenges, opportunities, and research roadmap. In *2017 IEEE Computer society annual symposium on VLSI (ISVLSI)* (pp. 627–632). https://doi.org/10.1109/ISVLSI.2017.124.
40. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198.
41. Shi, W., & Dustdar, S. (2016). The promise of edge computing. *Computer*, *49*(5), 78–81.
42. Song, S., Balaji, A., Das, A., Kandasamy, N., Shackleford, J. (2020). Optimizing tensor contractions for embedded devices with racetrack memory scratch-pads. In *Proceedings of the 21st ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems, LCTES 2020*.
43. Thrun, S. (1998). Lifelong learning algorithms. In *Learning to learn* (pp. 181–209): Springer.
44. Wen, W., Wu, C.R., Hu, X., Liu, B., Ho, T.Y., Li, X., Chen, Y. (2015). An eda framework for large scale hybrid neuromorphic computing systems. In *2015 52Nd ACM/EDAC/IEEE design automation conference (DAC)* (pp. 1–6): IEEE.
45. Wijesinghe, P., Ankit, A., Sengupta, A., Roy, K. (2018). 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*, *2*(5), 345–358.
46. Xia, Q., & Yang, J.J. (2019). Memristive crossbar arrays for brain-inspired computing. *Nature Materials*, *18*(4), 309.
47. Zhao, W., & Cao, Y. (2006). New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Transactions on Electron Devices*, *53*(11), 2816–2823.

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 (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.



Francky Catthoor received the engineering degree and a Ph.D. in electrical engineering from the Katholieke Universiteit Leuven, Belgium in 1982 and 1987 respectively. Between 1987 and 2000, he has headed several research domains in the area of high-level and system synthesis techniques and architectural methodologies. Since 2000 he is also strongly involved in other activities at IMEC including co-exploration of application, computer architecture and deep submicron technology aspects, biomedical systems and IoT sensor nodes, and photo-voltaic modules combined with renewable energy systems, all at the Inter-university Micro-Electronics Center (IMEC), Heverlee, Belgium. Currently he is an IMEC fellow. He is part-time full professor at the EE department of the K.U. Leuven.

In 1986 he received the Young Scientist Award from the Marconi International Fellowship Council. He has been associate editor for several IEEE and ACM journals, like Trans. on VLSI Signal Processing, Trans. on Multi-media, and ACM TODAES. He was the program chair of several conferences including ISSS'97 and SIPS'01. He has been elected an IEEE fellow in 2005.

Affiliations

Adarsha Balaji[1] · Thibaut Marty[2] · Anup Das[1] · Francky Catthoor[3]

Thibaut Marty
thibaut.marty@ens-rennes.fr

Anup Das
anup.das@drexel.edu

Francky Catthoor
francky.catthoor@imec.be

1 Drexel University, Philadelphia, Pennsylvania, 19104, USA
2 ENS Rennes, Rennes, France
3 Neuromorphic Division, IMEC, 3001 Leuven, Belgium