DNAsim: Evaluation Framework for Digital Neuromorphic Architectures

Sherif Eissa
Electronic Systems Group
Eindhoven University of Technology
Eindhoven, The Netherlands
s.s.b.eissa@tue.nl

Sander Stuijk
Electronic Systems Group
Eindhoven University of Technology
Eindhoven, The Netherlands
s.stuijk@tue.nl

Henk Corporaal
Electronic Systems Group
Eindhoven University of Technology
Eindhoven, The Netherlands
h.corporaal@tue.nl

***Abstract*—Neuromorphic architectures implement low-power machine learning applications using spike-based biological neuron models trained with bio-inspired or machine learning algorithms. Prior work on simulating Spiking Neural Networks (SNNs) focused on simulating emerging compute in-memory (CIM) architectures, while prior work on mapping SNNs focused mainly on minimizing inter-core communication or resource utilization and targeted either emerging CIM architectures or specific target platforms. SNN mapping choices on a neuromoprhic multi-processor platform can impact performance and energy consumption. In this paper, we introduce a simulation framework that evaluates application mapping on a user-defined NoC-based multi-core digital neuromorphic architecture. Our simulator evaluates latency and energy based on mapping and spike activity traces which indicate the firing of neurons at specific discrete timesteps defined in the application. We create two hardware models based on reported work in literature and show the evaluation of different mapping scenarios for a state-of-the-art SNN benchmark.**

***Index Terms*—neuromoprhic, simulator, spiking, SNN, modeling**

I. Introduction

With Moore's law and Dennard's scaling already ending, and the steep demand for edge processing for IoT devices, the computing community is looking to leverage new technologies and new computing architectures to further enhance performance at the edge.

The field of Artificial Intelligence (AI), particularly the field of Deep Learning (DL), has gained tremendous attention during the last decade and supporting their application on the edge is in rising demand. However, these edge solutions remain many orders of magnitude less energy efficient than any biological nervous system. This problem can be highlighted from both the algorithmic side as well as the architecture side.

With an intent to break the Von-Neumann architecture bottleneck and to achieve efficient edge processing, neuromorphic hardware attempts to mimic the structures and models of computation of biological nervous systems. The human brain is by far the most complicated and impressive computing system known to science; it can effectively perform over trillions of operations every second with a power budget of 20 watts.

This work has been funded by the Dutch Organization for Scientific Research (NWO) as part of P16-25 eDL project 7.

When viewed as a computing architecture, the human brain is in fact the complete opposite of a classical Von-Neumann computing architecture. In Von-Neumann architectures, the bandwidth and latency between computation and memory limits the system's performance and present a bottleneck. While in the brain, computation and memory are inseparable and intertwined in neurons and synapses respectively. The brain is a 3D spatial layout of an intricate mesh of these neurons and synapses, running independently and in paralell.

Neuromorphic architectures look to mimic the structure of the brain. However, with our current technology, we cannot build hardware on the same level of intricate details as of the brain. In digital technologies, we cannot merge single neurons and synapses together as in the brain. It is also impractical or even impossible to replicate the huge connectivity between neurons found in the brain with any technology [1].

In practice, neuromorphic architectures are a middle-of-the-road solution between Von-Neumann architectures and the brain. They are multi-core architectures of typically homogeneous processing elements (PEs) where each PE holds a subset of neurons and their synapses. PEs are usually connected by a high-speed NoC which enables shared communication [1].

In this paper, we present the following contributions:

- DNAsim, a spike trace-driven simulation framework for evaluating SNN applications mapped to digital neuromorphic multi-core architectures.
- An abstract and modular design that encapsulates user-defined hardware parameters and simulates any SNN application at the abstract level of its discrete timesteps using its spiking trace.
- Demonstrating DNAsim using an SRNN [2] application benchmark mapped on a multi-core mesh architecture using models inspired by Loihi [3] and MorphIC [4].

II. Background

Spiking Neural Networks (SNNs) are the third generation of deep learning algorithms [5]. They come in networks and structures similar to that of Artificial Neural Networks (ANNs) like feed-forward, recurrent, convolutional and fully connected. They can also be viewed as dynamical systems [6].

SNNs take inspiration from biological neurons by computing their states over time using bio-inspired neuron models and

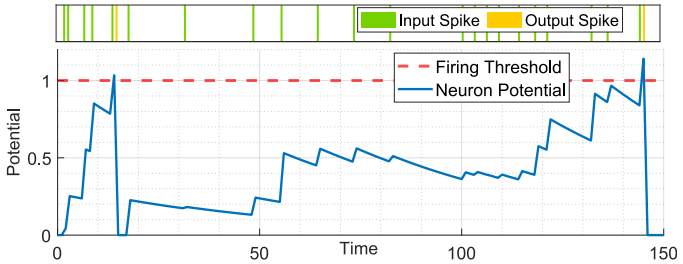

Fig. 1: The response of an LIF neuron (bottom) to incoming random spikes (top). The LIF neuron potential leaks exponentially in time and it fires when it crosses firing threshold.

by communicating using binary spikes. Similar to biology, a spike injects charge into a neuron's soma (body). This charge is modulated by the strength of the synapse mediating the spike, similar to weight kernels in ANNs. When a neuron's soma accumulates sufficient charge, it releases its charge as a spike which travels to destination neurons, while the soma resets back to its resting potential. The neuron model implemented by an SNN varies from very simple models like the integrate-and-fire models to much more complicated models like the Izhikevich neuron model [7].

To solve a neuron model's ordinary differential equations, digital SNN computation is performed over discretized timesteps, which we refer to in this paper as *algorithmic timesteps*. During an algorithmic timestep, a neuron integrates all incoming spikes and, if it exceeds a certain threshold potential, it fires and releases its potential as an output spike. Note that in this discrete model, we assume that each neuron can only spike at-most once per algorithmic timestep.

Figure 1 shows the discretized behavior of a leaky-integrate-and-fire (LIF) neuron responding to randomly weighed incoming spikes. The LIF neuron immediately integrates incoming spikes into its potential and it loses its potential exponentially according to a decay time constant.

Some digital neuromorphic architectures, such as Loihi [3] and TrueNorth [8], implement time synchronization between its cores to buffer spikes between algorithmic timesteps. This *synchronized execution* can easily match applications correctly and has more predictable behavior and performance.

Other digital architectures, such as ODIN and μBrain [9] [10], ignore inter-core synchronization and rely on explicit decay commands for LIF models, as synchronization has a periodic overhead cost in energy and latency which scales with the number of tiles. In this *fully event-driven execution*, spikes are produced and consumed instantaneously without any concept of algorithmic timesteps.

In this work, we consider modeling synchronized execution due to its predictable behavior. We also see that our simulation, which is abstracted at the level of algorithmic timesteps and cost models, can estimate the predictable execution of synchronized execution platforms.

While most hardware architectures implement periodic leakage, event-based leakage can be beneficial for highly sparse application. Accurate leakage calculation has been shown to be unnecessary and can be approximated with cheaper circuits even for highly dynamical systems [11].

Implementing SNNs involves mapping neurons on a multiprocessor neuromorphic system. Such a mapping can be constrained by the capacity of each tile in terms of neurons, synapses or input/output axons. Mapping decisions can have a tremendous effect on performance and sometimes even accuracy. Similar to typical multiprocessor mapping problems, digital implementations have a trade-off between spreading and balancing computation load per tile and the overall communication volume. Finding good partitions between tasks in not an obvious problem. In fact, task mapping in multicore processors is considered an NP-hard problem [12].

Unlike analog and mixed-signal architectures, digital neuromoprhic architectures often deploy time-shared neuron compute units which serialize the execution of synaptic operations (SOPs) inside a single core [3] [8]. Hence, the more SOPs a digital core has to execute during an algorithmic timestep, the more time it will need to finish said timestep.

Since analog and CIM architectures have completely dedicated spatial layouts, they have fully parallel functional neurons that do not mind the amount of incoming spike traffic. This is why existing research in SNN mapping for CIM crossbar-based architectures focuses only on minimizing traffic, maximizing crossbar utilization and/or technology related issues like endurance to read and write operations [13] [14].

However, for digital time-multiplexed implementations, load balancing of SOPs balances the computation latency between cores, which reduces the latency of the busiest core and increases overall system throughput. The topic of SOP load balancing for digital neuromorphic implementations is underexplored in SNN mapping literature.

III. Related work

In this section, we cover some of the literature in mapping and simulating SNN implementations on a multiprocessor neuromorphic architecture.

In some works on SNN mapping from the group of Anup Das [15], they develop a novel method to simulate and evaluate SNN partitions using Synchronous Dataflow Graph (SDFG) models [16]. In these models, which use SDF^3 software [17], they can model buffer sizes and measure system throughput. However, one can argue that their models assume fully event-driven execution, not the synchronized execution which we are targeting.

PyCarl [18] is a PyNN-based [19] common python programming interface for hardware-software cosimulation of SNN. In this work, the authors created an interface between PyNN and CARLsim [20] as well as an interface between CARLsim and cycle-accurate hardware models. This enables performance evaluation as well as accuracy evaluation by incorporating hardware information such as latency. This framework works at a lower level of abstraction than our work, simulating SNNs behaviorally and simulating its implementation at cycle-level rather than algorithmic timestep level.

Noxim++ [13] is an extension of Noxim [21], a cycle-accurate NoC simulator, tailored for simulating SNN implementations.

PACMAN [22] is a partitioning tool for mapping applications on the SpiNNaker system [24]. It fits an application using bin-packing heuristics without any considerations to performance.

In NEUTRAMS [23], authors apply Kernighan-Lin (KL) graph partitioning technique to minimize inter-core communication by mapping strongly connected nodes together. For simulation, the authors implemented a modular hardware model simulator similar to DNAsim. Although they develop a generic methodology, they limit their simulator to crossbar synaptic memory structures.

NxTF [25] is a mapping tool for Loihi. Its heuristic iterates the network from output to input. It maps each layer according to a multi-objective function based on minimizing resource usage to reduce energy consumption. Hence, this work fails to address throughput/latency related mapping issues and focuses only on energy and resource/area optimization.

In [26], an older mapping tool was developed for Loihi. The goal of this tool was to maximize input axon sharing. This peculiar objective was chosen based on the Loihi energy breakdown. The main source of energy consumption was attributed to trace variables related to online learning. Sharing input axons meant sharing these input trace variables which resulted in reduced energy consumption.

There are many recent SNN compiling and mapping tools from Anup Das et al. such as [27] [28] [29] [30] [31] [32]. In these tools, mapping is done based on mapping heuristics like particle swarm optimization, graph partitioning or bin packing. However, they mainly consider analog and crossbar architectures. Their objective functions focus only on minimizing communication for reduced power consumption and crossbar utilization for resource efficiency. They do not consider load balancing issue which is not an issue for dedicated or analog architectures.

SNEAP [33] is a tool for SNN partitioning and mapping. It applies a two-step heuristic to partition and then map SNNs. They use k-way multi-level graph partitioning and use three different mapping heuristics with the goal of minimizing overall spike traffic and spike hops. They use Noxim++ [13] to evaluate latency and energy of their mapping.

[34] is a software-hardware co-exploration framework. On the software side, spikes are dropped to save latency at the expense of accuracy and traces are extracted. While on the hardware side, we map the SNN and evaluate it using a hardware simulator that can estimate performance. Their hardware simulator leverages Booksim 2.0, a NoC simulation tool [35].

In MigSpike [36], authors use min-cut graph-based mapping algorithms to minimize inter-cluster communication volumes. In addition to that, they consider faults in CIM-based neurons and apply a novel heuristic to migrate neurons from a cluster to another.

In [37], authors create a framework for NoC design space exploration. They apply a seach heuristic based on simulated anealing and its objective function looks to maximize system throughput. They use Booksim 2.0 [35] to simulate their NoC design and SNN mapping.

In TAMA [38], meta-heuristics based on Genetic algorithm and Ant Colony Optimization algorithm are used to map neurons while minimizing spike hops (distance) as well as spike turns inside routers for more efficient implementation. They use Garnet2.0 [39] for evaluation.

To our knowledge, our work is the first to recognize computational load balancing as one of the issues to consider in SNN mapping on digital multi-core architectures. This paves the way for multi-objective mapping optimization problems that try to balance between system energy consumption and throughput [40].

IV. DNASIM

DNAsim is a framework for simulating the implementation of SNNs on a multi-processor NoC as well as estimating performance. It is separated into well-defined modules that enable plugging in different user-requirements according to the user-defined hardware and system-level software models.

It consists of 5 modules: Application, Spike trace, Hardware model, Mapper, and Simulator. Figure 2 shows a simple diagram of our framework. DNAsim is open-source available at https://github.com/TUE-EE-ES/DNAsim.

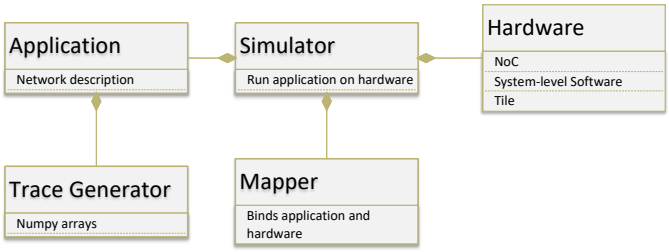

Fig. 2: DNAsim framework.

A. Application

The application definition module describes the SNN application to be deployed. In general, neurons can be represented as task nodes that sparsely and sporadically communicate using spikes. The SNN spiking activity is described over discretized algorithmic timesteps. Hence, we decided to use a directed graph model to represent our application. This also enables the use of graph-theortic mapping techniques which are widely used in the domain of multi-processor task mapping and allows us to put the problem of SNN mapping in the context of generic multi-processor mapping problems which have been studied extensively in the literature [40].

In this version of DNAsim, we only support fully and recurrently connected layers of neurons. However, we plan to introduce other connections in our future releases such as convolution layers, pooling operations and sparse connections.

B. Spike Trace

We rely on spike traces as input for our simulator. These traces describe the spiking activity of each neuron at each algorithmic timestep. We have opted for a trace-based simulator for multiple reasons:

- **Application-independent**: No need to worry about implementing the application's model as this has already been done on the application's framework (e.g Pytorch).
- **Complexity**: A trace-based evaluation is far simpler than a cycle accurate hardware simulator. This comes at the cost of accuracy, however accuracy loss can be minimized in synchronized execution platforms by appropriately modeling the system-level software.
- **Abstraction**: Our DSE methodology relies on statistics regarding spiking activity only at the granularity of a discrete timestep. This level of abstraction is in line with MP-SoC mapping techniques [40].

Each node in the application graph is associated with a firing rate vector that indicates its firing rate at each algorithmic timestep, which has a value between 0 and 1. The firing rate vector corresponds to data obtained from inference runs on the application. These values can correspond to average cases or corner-cases that exhibit the highest neuron activity.

The volume of communication between neurons as well as the SOPs performed by each neuron can be derived from spiking activity traces. We use numpy arrays [41] for our trace files due to its prominence among DL applications developed on Python.

C. Hardware model

The hardware model can be described as a two-layer module. At the top layer, the NoC architecture is described as well as the system-level software which describes things like the communication, routing, and synchronization protocols. At the bottom layer, the router and core are described in terms of routing costs as well SOP, neuron update, and neuron spiking costs, in addition to idle power and leakage power.

In DNAsim, we have implemented a 2-D mesh topology, static XY routing, as well as uni-cast and multi-cast communication protocols and a configurable spike packet header cost.

Each tile implements user-defined cost model that describes the cost of different operations. The latency of an algorithmic timestep is determined by the slowest core in the mesh while the energy of an algorithmic timestep is the aggregate energy consumed by all cores.

In addition to latency and energy, the tile module also records statistical information such as spike traffic, spikes produced and consumed, and number of SOPs performed at each timestep which provide insights to what is affecting performance in the system.

D. Mapper

The mapper module is responsible for binding the application to tiles in the hardware model. Mapping must be constrained according to hardware resources.

The mapping process can be automated with the help of heuristics, which have been extensively studied in MPSoC mapping problems, with hardware constraints encoded. [12]

For this work, however, we demonstrate a few manually generated mappings and we plan to automate mapping in future work using graph-theoretic mapping techniques.

E. Simulator

The simulator runs the mapped application model on top of the hardware model by issuing transfer transactions between nodes of the hardware model according to the trace files.

There are two modes of simulation depending on the hardware model. For simple hardware models that assume no spike propagation delay or congestion, we have *coarse-grained* simulator that runs transactions in parallel, across different tiles, across different algorithmic timesteps, and in one shot from source to destination. Another *fine-grained* simulator runs transactions in a chronological order and step by step fashion which slower, memory demanding and has limited parallelism but enables modeling of spike delay and router congestion.

V. Experiments

We report several mapping experiments demonstrating the functionality of our simulator using a state-of-the-art SRNN benchmark and two approximated hardware models.

For our application, we use a network called SRNN [2] which provides state-of-the-art benchmarks for recurrent SNNs. We use the Spiking Heidelberg Dataset (SHD) [42] benchmark application from SRNN. This network consists of 700 input neurons, 20 output neurons and 2 hidden layers of size 128 neurons each. The network is fully connected, from input to output. Each hidden layer is also fully connected recurrently with itself. Figure 3 shows the structure of this network.

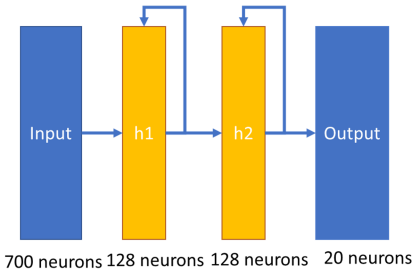

Fig. 3: SRNN SHD network.

For our traces, we extracted a corner-case with the most spiking activity within the test dataset. After analyzing the corner-case and average traces, we concluded that the input layer spikes dominate the total spiking activity and thus invoke a high number of SOPs on the first hidden layer. Figure 4 shows the spikes produced and synaptic operations performed by each spiking layer for our corner-case under study.

Our hardware model is a 2-D mesh structure with static XY routing and multi-cast communication. We ignore the cost of

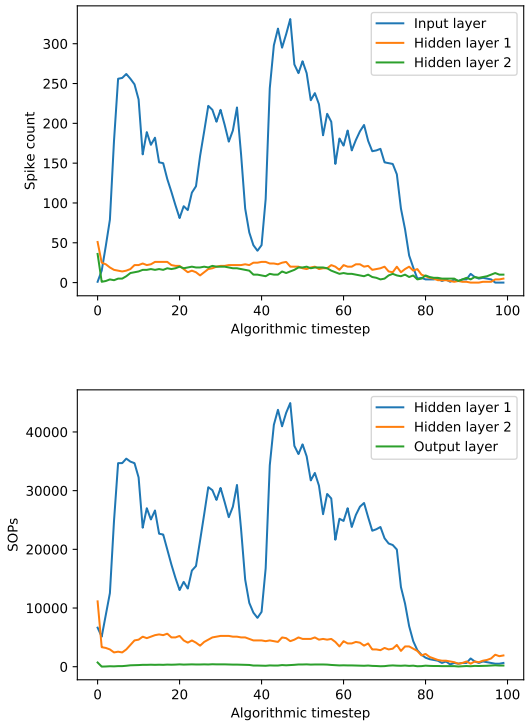

Fig. 4: Spikes produced (top) and Synaptic operations performed (bottom) by each layer of the SHD network. Input layer spikes dominate the activity.

synchronization in our experiments. We use numbers reported from Loihi [3] and MorphIC [4] to create an approximate model of our tile. Table I shows values extracted for our hardware models. These values are used to generate performance estimates. Energy per Hop corresponds to the energy needed to transfer one spike from tile to tile. Energy and latency per SOP correspond to the energy and time consumed due to execution of an SOP on a tile respectively while energy and latency per active neuron is the added overhead needed to update (leak) a neuron at each algorithmic timestep. We ignore idle power and leakage power in this cost model. We also assume no delay due to spike propagation or congestion, hence we use our fast coarse-grained simulator.

Our model estimates energy based on the total amount of spike hops, SOPs and neuron updates while it estimated latency based on the distribution of SOPs and neuron updates between cores. The two missing components in our model are the effects of the NoC size and dimensions as well as the effects of spike propagation delays and traffic congestion. The latter component requires more intricate simulations using the fine-grained simulator. As our model ignores these effects, our results should be rather taken with a pinch of salt.

For our mapper, we consider 4 different scenarios. Firstly, we consider cores having a synaptic memory of size 16K. For these 16K cores, we need a 3x3 mesh to map our application. For such a scenario we create two different mappings; one

TABLE I: Hardware model parameters

Model Parameters	Hardware Model	
	Loihi	***MorphIC***
Energy per Hop	4.0 pJ	9.0 pJ
Energy per SOP	24 pJ	30 pJ
Latency per SOP	3.5 ns	36 ns[a]
Energy per active neuron	52 pJ	N/A
Latency per active neuron	5.3 ns	N/A

[a]Deducted from reported core bandwidth of 27.5MSOP/s/core [4].

having the two hidden layers separated on different cores, and another having the two hidden layers merged. Secondly, we consider cores having synaptic memory of size 8K. For these 8K cores, we need a 4x4 mesh to map SHD efficient. For such a scenario we again consider two different mappings where we separate and merge the two hidden layers on the cores.

Figures 5 and 6 show the synaptic operations and spike hops per tile for the two 3x3 mappings. We notice that merging layers results in more balanced SOP load between tiles, but higher spike communication volume.

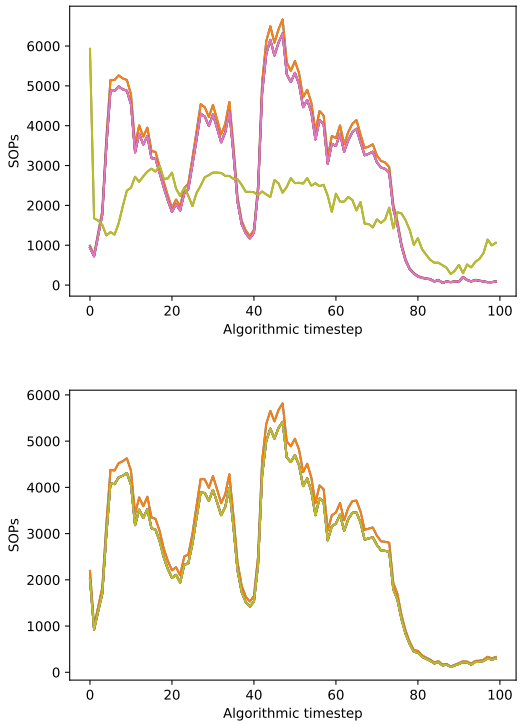

Fig. 5: Synaptic operations performed by each tile for 3x3 mesh mapping with different layers separated (top) and with different layers merged together (bottom). Merging hidden layers together on all cores results in more balanced SOP load.

Figures 7 and 8 show the synaptic operations and spike hops per tile for the two 4x4 mappings. We notice that merging layers results in more balanced SOP load between tiles but higher spike communication volume. Additionally, compared to 3x3 mappings, spreading our application on a 4x4 NoC

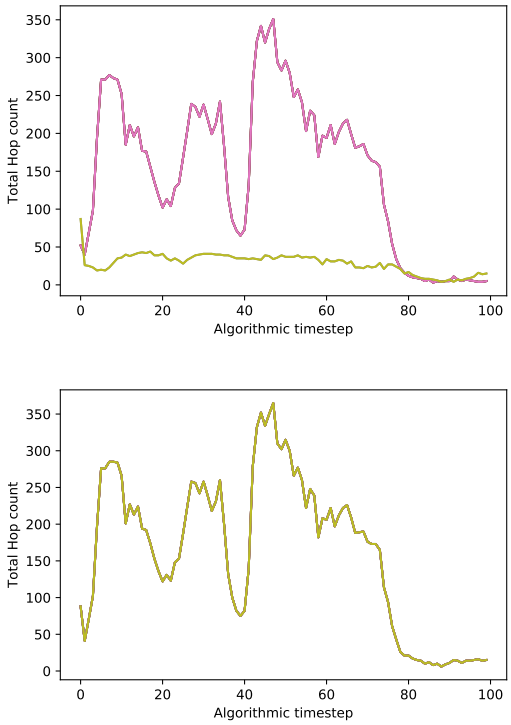

Fig. 6: Spike hops passing through each tile for 3x3 mesh mapping with different layers separated (top) and with different layers merged together (bottom). Merging hidden layers together on all cores results in increased communication.

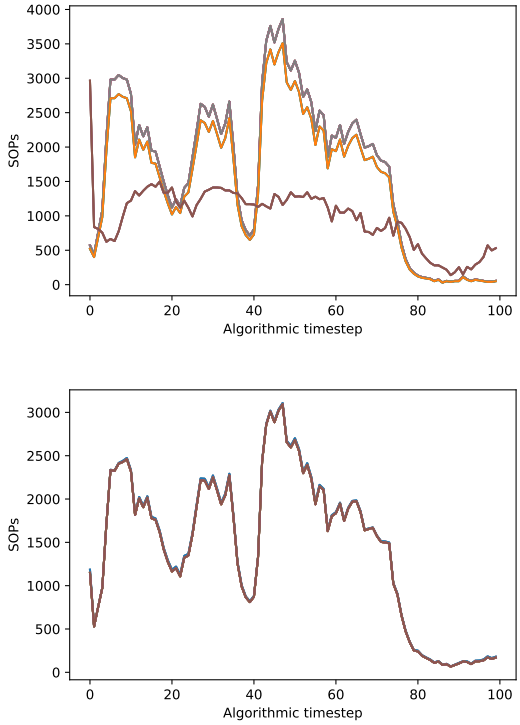

Fig. 7: SOPs performed by each tile for 4x4 mesh mappings with different layers separated (top) and merged together (bottom). Merging layers balances SOP load. 4x4 mappings perform less SOPs per tile compared to 3x3 mappings.

results in lower SOP load per tile but higher communication volume.

Figures 9 and 10 compare the latency and energy costs of different mappings for the MorphIC and the Loihi models respectively. We notice in our models that the cost of synaptic operations heavily outweigh the cost of spike propagation, which results in heavily favoring the merging of layers. We also contribute the favorable performance of spread-out mappings to the fact that we ignored overhead costs in latency and energy of synchronization and turning on extra cores.

Table II shows total performance estimates, based on table I, produced by our simulator for all mapping experiments. We notice how the contribution of increased spike traffic is negligible in our model. Increasing the number of cores from 3x3 to 4x4 resulted in roughly doubling the system throughput.

TABLE II: Inference performance of different mappings of SRNN SHD benchmark on Loihi and MorphIC models.

Mapping Configuration	**Total Inference Performance**			
	Loihi		**MorphIC**	
	Energy	Time	Energy	Time
3x3 Separated	566 μJ	1.12 ms	704 μJ	11.2 ms
3x3 Merged	568 μJ	972 μs	707 μJ	9.81 ms
4x4 Separated	569 μJ	631 μs	711 μJ	6.37 ms
4x4 Merged	572 μJ	521 μs	718 μJ	5.25 ms

VI. Discussion

Spreading an application on more cores results in less computational load which can lead to better system throughput. However, as typical to MPSoC mapping problems, this results in increased inter-core traffic which increases energy consumption. Having more cores also results in higher energy consumption attributed to the extra idle power and leakage power consumed. Overspreading an application can even result in reduced system throughput or accuracy due to explosive increase in communication traffic and possible spike dropping as well as the steady increase in synchronization costs with increasing mesh dimensions [3].

Mixing groups of neurons within an application can balance the load of synaptic operations on each core which can lead to better system throughput. However, mixing them results in increased inter-core communication which directly influences energy consumption and can indirectly influence system throughput through spike congestion.

Our experiments verify, although not precisely, the traditional trade-offs found in MPSoC mapping problems; a trade-off between balancing loads and limiting communication volumes as well as the trade-offs of parallelizing execution by spreading an application. In our models and for our application, we notice that synaptic operations dominate both energy consumption and system latency.

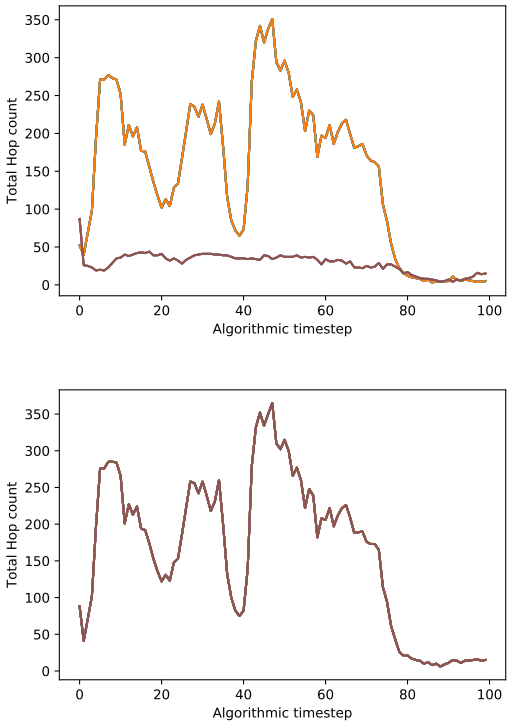

Fig. 8: Spikes routed by each tile for 4x4 mesh mappings with different layers separated (top) and merged together (bottom). Merging layers together on cores increases spike traffic. 4x4 mappings have more spike traffic compared to 3x3 mappings.

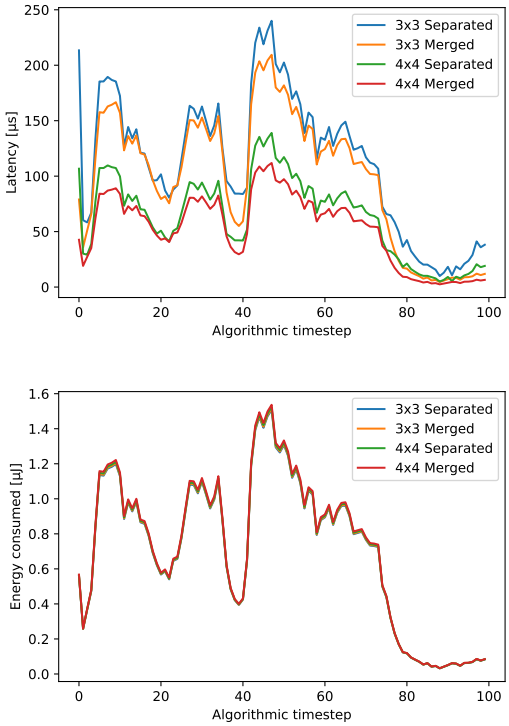

Fig. 9: Latency (top) and energy (bottom) estimates per algorithmic timestep for MorphIC model. We notice performance is overwhelmingly dominated by SOPs and not spike hops.

VII. Conclusion and Future work

In this work, we present a framework for simulating and analyzing SNNs implemented on a multi-processor digitial neuromorphic system. Our simulator is based on application-independent traces that indicate spiking activities. We have conducted experiments based on approximate models inpired from values reported for Loihi and MorphIC and using SRNN SHD benchmark. Our experiments highlighted the importance of mapping for SNNs and multi-processor neuromorphic systems. We highlight issues relevant to MP-SoC mapping problems: load balancing versus communication volume reduction as well as application spreading trade-offs.

For our future work, we intend to implement an automated mapping module that applies multi-objective partitioning and mapping heuristics that are novel to SNNs.

We also plan to enhance the accuracy of our model by incorporating factors like synchronization, idle power and leakage power. We would also like to incorporate a spike delay and congestion model within our future models.

We will also work on applications involving larger deeper SNNs with convolutional connections as well as structural sparsity. We believe such applications can bring different observations and create more interesting mapping problems than a non-sparse fully connected application.

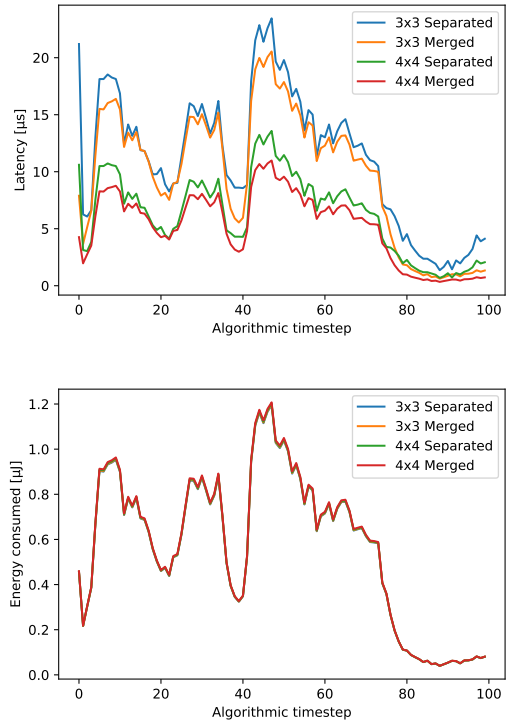

Fig. 10: Latency (top) and energy (bottom) estimates per algorithmic timestep for Loihi model.

REFERENCES

[1] M. Bouvier et al, "Spiking Neural Networks Hardware Implementations and Challenges: A Survey". J. Emerg. Technol. Comput. Syst. 15, 2, Article 22 (April 2019), 35 pages. doi: 10.1145/3304103

[2] B. Yin, F. Corradi, and S. Bohté, "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks". Nature Machine Intelligence, 3(10), pp.905-913, 2021.

[3] M. Davies et al, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," in IEEE Micro, vol. 38, no. 1, pp. 82-99, January/February 2018, doi: 10.1109/MM.2018.112130359.

[4] C. Frenkel et al, "MorphIC: A 65-nm 738k-Synapse/mm^2 Quad-Core Binary-Weight Digital Neuromorphic Processor With Stochastic Spike-Driven Online Learning," in IEEE Transactions on Biomedical Circuits and Systems 2019, vol. 13, no. 5, pp. 999-1010, Oct. 2019, doi: 10.1109/TBCAS.2019.2928793.

[5] Wolfgang Maass, "Networks of spiking neurons: The third generation of neural network models", Neural Networks, 10, 9, 1997, p.p 1659-1671, ISSN 0893-6080, doi: 10.1016/S0893-6080(97)00011-7.

[6] Wolfgang Maass, "Liquid State Machines: Motivation, Theory, and Applications" from "Computability in Context", pp. 275-296, 2011, doi:10.1142/9781848162778_0008

[7] W. Gerstner, W. Kistler, R. Naud, and L. Paninski, "Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition," 2014, Cambridge University Press, USA.

[8] F. Akopyan et al., "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 10, pp. 1537-1557, Oct. 2015, doi: 10.1109/TCAD.2015.2474396.

[9] C. Frenkel et al, "A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS," in IEEE Transactions on Biological Circuits and Systems, vol. 13, no. 1, pp. 145-158, Feb. 2019, doi: 10.1109/TBCAS.2018.2880425.

[10] J. Stuijt et al, "μBrain: An Event-Driven and Fully Synthesizable Architecture for Spiking Neural Networks," in Frontiers in Neuroscience, vol. 15, 2021, doi:10.3389/fnins.2021.664208

[11] S. Eissa, S. Stuijk and H. Corporaal, "Hardware Approximation of Exponential Decay for Spiking Neural Networks," 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2021, pp. 1-4, doi: 10.1109/AICAS51828.2021.9458560.

[12] M. Gupta et al, "Mapping techniques in multicore processors: current and future trends". J Supercomput 77, 9308–9363 (2021). https://doi.org/10.1007/s11227-021-03650-6

[13] A. Balaji et al,"Mapping Spiking Neural Networks to Neuromorphic Hardware," in IEEE Transactions on VLSI Systems, vol. 28, no. 1, pp. 76-86, Jan. 2020, doi: 10.1109/TVLSI.2019.2951493.

[14] T. Titirsha et al, "Endurance-Aware Mapping of Spiking Neural Networks to Neuromorphic Hardware," in IEEE Transactions on Parallel and Distributed Systems, vol. 33, 2022, doi: 10.1109/TPDS.2021.3065591.

[15] A. Balaji and A. Das, "A Framework for the Analysis of Throughput-Constraints of SNNs on Neuromorphic Hardware," in 2019 IEEE Computer Society Annual Symposium on VLSI, pp. 193-196, doi: 10.1109/ISVLSI.2019.00043.

[16] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," in Proceedings of the IEEE, vol. 75, no. 9, pp. 1235-1245, Sept. 1987, doi: 10.1109/PROC.1987.13876.

[17] S. Stuijk, M. Geilen and T. Basten, "SDF^3: SDF For Free," Sixth International Conference on Application of Concurrency to System Design (ACSD'06), 2006, pp. 276-278, doi: 10.1109/ACSD.2006.23.

[18] A. Balaji et al., "PyCARL: A PyNN Interface for Hardware-Software Co-Simulation of Spiking Neural Network," In 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1-10, doi: 10.1109/IJCNN48605.2020.9207142.

[19] A. Davison et al, "PyNN: a common interface for neuronal network simulators," Frontiers in Neuroinformatics, 2009.

[20] T. Chou et al., "CARLsim 4: An Open Source Library for Large Scale, Biologically Detailed SNN Simulation using Heterogeneous Clusters," In 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489326.

[21] V. Catania et al, "Noxim: An open, extensible and cycle-accurate network on chip simulator," 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2015, pp. 162-163, doi: 10.1109/ASAP.2015.7245728.

[22] F. Galluppi et al, "A hierachical configuration system for a massively parallel neural hardware platform". In Proceedings of the 9th conference on Computing Frontiers (CF '12), p.p 183–192. https://doi.org/10.1145/2212908.2212934

[23] Y. Ji et al, "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1-13, doi: 10.1109/MICRO.2016.7783724.

[24] S. B. Furber et al, "The SpiNNaker Project", in Proceedings of the IEEE, vol. 102, no. 5, pp. 652-665, May 2014, doi: 10.1109/JPROC.2014.2304638.

[25] B. Rueckauer et al, "NxTF: An API and Compiler for Deep Spiking Neural Networks on Intel Loihi". J. Emerg. Technol. Comput. Syst. 18, 3, Article 48 (July 2022), 22 pages. https://doi.org/10.1145/3501770

[26] Chit-Kwan Lin et al, "Mapping spiking neural networks onto a manycore neuromorphic architecture", In Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018). Association for Computing Machinery, New York, NY, USA, 78–89. https://doi.org/10.1145/3192366.3192371

[27] S. Song et al, "A Design Flow for Mapping Spiking Neural Networks to Many-Core Neuromorphic Hardware". In 2021 IEEE/ACM ICCAD. IEEE Press, 1–9. doi: 10.1109/ICCAD51958.2021.9643500

[28] A. Balaji et al, "NeuroXplorer 1.0: An Extensible Framework for Architectural Exploration with Spiking Neural Networks". In International Conference on Neuromorphic Systems 2021 (ICONS 2021), Article 10, 1–9. doi: 10.1145/3477145.3477156

[29] S. Song et al, "DFSynthesizer: Dataflow-based Synthesis of Spiking Neural Networks to Neuromorphic Hardware."ACM Trans. Embed. Comput. Syst. 21, 3, Article 27 (May 2022), 35 pages. https://doi.org/10.1145/3479156

[30] A. Balaji et al, "Run-time Mapping of Spiking Neural Networks to Neuromorphic Hardware". J. Signal Process. Syst. 92, 11 (Nov 2020), 1293–1302. https://doi.org/10.1007/s11265-020-01573-8

[31] S. Song et al. "Compiling Spiking Neural Networks to Neuromorphic Hardware". In The 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '20). Association for Computing Machinery, New York, NY, USA, 38–50. https://doi.org/10.1145/3372799.3394364

[32] M. L. Varshika, A. Balaji, F. Corradi, A. Das, J. Stuijt and F. Catthoor, "Design of Many-Core Big Little μBrains for Energy-Efficient Embedded Neuromorphic Computing," 2022 Design, Automation Test in Europe Conference Exhibition (DATE), 2022, pp. 1011-1016, doi: 10.23919/DATE54114.2022.9774613.

[33] S. Li et al, "SNEAP: A Fast and Efficient Toolchain for Mapping Large-Scale Spiking Neural Network onto NoC-based Neuromorphic Platform". In Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI '20). Association for Computing Machinery, New York, NY, USA, 9–14. https://doi.org/10.1145/3386263.3406900

[34] S. Wang et al, "A Software-Hardware Co-exploration Framework for Optimizing Communication in Neuromorphic Processor". In Advanced Computer Architecture (ACA) 2020. Communications in Computer and Information Science, vol 1256. Springer, Singapore. https://doi.org/10.1007/978-981-15-8135-9_7

[35] N. Jiang et al, "Booksim 2.0 user's guide". Standford University (2010)

[36] K. Dang et al, "MigSpike: A Migration Based Algorithms and Architecture for Scalable Robust Neuromorphic Systems," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 2, pp. 602-617, 1 April-June 2022, doi: 10.1109/TETC.2021.3136028.

[37] Z. Kang et al. "Application-specific network-on-chip design space exploration framework for neuromorphic processor". In 17th ACM International Conference on Computing Frontiers (2020).

[38] R. Aligholipour et al, "TAMA: Turn-aware Mapping and Architecture – A Power-efficient Network-on-Chip Approach". ACM Trans. Embed. Comput. Syst. 20, 5, 44, 2021, 24 pages. doi: 10.1145/3462700

[39] Niket Agarwal et al, "GARNET: A detailed on-chip network model inside a full-system simulator", In IEEE International Symposium on Performance Analysis of Systems and Software, 2009. IEEE, 33–42.

[40] M. Deveci et al, "Hypergraph partitioning for multiple communication cost metrics: Model and methods", J. of Parallel and Distributed Computing, V. 77, 2015, p,p 69-83, doi: 10.1016/j.jpdc.2014.12.002.

[41] C. Harris et al, "Array programming with NumPy". Nature (2020).

[42] B. Cramer et al, "The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks". IEEE Transactions on Neural Networks and Learning Systems 1–14 2020.