SENSIM: An Event-driven Parallel Simulator for Multi-core Neuromorphic Systems

***Abstract*—In this paper, we present SENSIM, which is an open-source simulator designed specifically for the SENECA neuromorphic processor. This simulator is unique in that it combines features from both hardware-specific and hardware-agnostic spiking neural network simulators, resulting in a hybrid event-driven and time-step-driven simulation approach. This allows for flexibility between accuracy and speed during different stages of simulation. Our work highlights the open-source SENSIM platform, which enables the mapping of large-scale SNN/DNN models to the SENECA cores, as well as the benchmarking of crucial KPIs such as power and latency estimations[1].**

***Index Terms*—Hardware-aware Simulation, Neuromorphic, SENECA, SENSIM, SNN/DNN inference**

I. Introduction

Brain-inspired neuromorphic processors have the potential to greatly enhance the performance and energy efficiency of AI execution engines in both cloud and edge computing. However, their potential is yet to be fully realized due to limited accessibility, limited capabilities, and high costs associated with scaling them up to multi-chip platforms. To address these challenges, a flexible simulator that offers reliable and comparable KPIs, such as energy, latency, area, and accuracy, for neuromorphic algorithms and hardware architectures may be a viable solution.

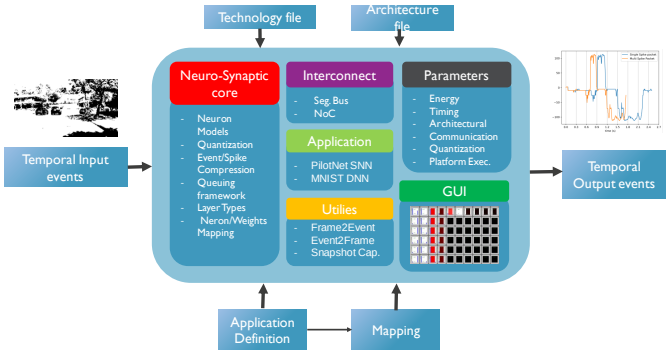

Fig. 1. SENSIM Simulation framework

Existing neuromorphic simulators can be categorized into hardware-specific simulators such as those mentioned in [1], [18] and hardware-agnostic spiking neural network simulators such as [3], [5], [10], [19]. Specifically, hardware-specific simulators provide information on energy and latency for the hardware implementation of an algorithm. In this paper, we propose SENSIM (Fig.1), an open-source hardware-aware simulator designed specifically for the SENECA [21], a flexible neuromorphic architecture that can be used for extensive neuromorphic architecture search. SENSIM is a hardware-specific neuromorphic simulator that has been fine-tuned using circuit-level power and latency measurements of SENECA. It can be used for hardware-aware training, architectural search, and mapping. Additionally, SENSIM allows users to perform experiments on scaled-up systems that contain thousands of cores.

Neuromorphic processors have an asynchronous and event-driven nature. As a result, many neuromorphic hardware-aware simulators are designed to be truly event-driven. This requires processing events one by one in a queue, regardless of whether they come from sensors or are generated internally. Therefore, true event-driven simulations cannot be parallelized because the order of the events affects the final result. In SENSIM, we developed a hybrid event-driven and time-step-driven simulation platform that can be adjusted for more accuracy or speed in different simulation phases.

The main contributions of the paper are as follows:

- Open-source SENECA Simulation Platform: SENSIM provides a simulation platform that allows large SNN/DNN models with varying sizes of neurons and parameters to be mapped to cores of the SENECA architecture and provides the target KPI benchmarks.
- Hybrid event-driven Design: SENSIM is designed to be both event and time-step-driven. This design choice enables exploiting all event-driven characteristics and helps overcome problems such as limited accuracy, large empty loads, and invalid calculations [15].
- Scaling up the Flexible SENECA Neuro Synaptic Core: SENSIM demonstrates how energy and timing parameters extracted from basic hardware RTL simulation can emulate a scaled-up hardware platform effectively.
- Customization and Future Research: SENSIM is customizable and can be used by the research community by incorporating different neuro synaptic cores and SNN/DNN applications for future development and research.

The remainder of the paper will cover topics including the SENECA architecture, the design and implementation of SENSIM, the scalability aspects, energy optimizations, accuracy considerations, and practical examples of architectural enhancements.

[1]Source code of the simulator can be found in https://anonymous.4open.science/r/SENSIM_paper_submission-4E4F

II. Related Works

With the expansion of the neuromorphic and neuroscience community, there is a simultaneous advancement in the development of emulators and simulation environments in the industry and research community. Table I briefs on the key parameters for comparing SENSIM with other simulators in the industry/research community.

The Lava framework [1] is a simulator for evaluating applications on Intel Loihi [7] and Loihi2 neuromorphic processors. Lava supports various neuron models such as Leaky integrate and fire (LIF), adaptive LIF, Sigma Delta, etc.

Brian [10] [19] is a Python-based spiking neural network simulator allowing the neuromorphic community to implement new SNN models. Brian is hardware-agnostic and targets the neuroscience community for simulating large-scale SNN models. Brian2Loihi [14] is an emulator designed for implementing spiking neural networks on Loihi. It differs from SENSIM as it focuses on single small-scale recurrent neural networks with on-chip learning, while SENSIM primarily concentrates on inference emulation of large-scale SNN models.

PyCARL [3] is a Python interface enabling hardware-software co-simulation of detailed spiking neural networks (SNN). It combines PyNN with CARLsim [5], a GPU-accelerated SNN simulator. PyCARL integrating cycle-accurate models of advanced neuromorphic hardware like TrueNorth [2], Loihi [7], and DynapSE [17]. This facilitates accurate modeling of hardware latencies, which impact machine learning model performance. PyCARL allows users to analyze and optimize performance differences between software-based and hardware-oriented simulations, expediting the time to market for neuromorphic products. It also includes NOXIM++ to integrate hardware models into the loop, extracting metrics like latency, throughput, power, ISI distortion, and disorder spike count. Unlike PyCARL, SENSIM achieves faster simulation by estimating timing through packet time stamping on event generation, eliminating the need for a cycle-accurate trace-driven simulator.

Nengo [4] is a platform-independent Python tool to model large-scale brain models. In terms of simulation, Nengo provides a time simulation of the SNN models. Nengo is a Python-based trace-driven SNN simulator that makes SNN model emulation more intuitive. Nengo is hardware-agnostic.

BindsNET [11] is a user-friendly framework for rapidly building and simulating spiking networks. It utilizes PyTorch for implementation, allowing for CPU and GPU computation. Additionally, BindsNET can be adapted to other computing backends like TensorFlow and SpiNNaker. It integrates with OpenAI gym for reinforcement learning training and evaluation of spiking networks.

NEURON [13] is a hardware-agnostic simulator that has been developed over the past three decades and is widely used by neuroscientists to model the electrical activity of neuronal networks with complex branched anatomy and biophysical membrane properties. A high level of morphologically detailed models can be simulated using NEURON and represented using the spatial diversity of electrical and biophysical properties of neurons.

NEST [9] is a hardware-agnostic neural simulation tool designed for heterogeneous networks of point neurons or neurons with few compartments. It specializes in simulating large neural systems with over 10,000 neurons and 10^7 to 10^9 synapses. PyNEST [8] is built upon NEST's efficient simulation kernel and offers the simplicity and flexibility of Python. Compared to NEST's native simulation language SLI, PyNEST simplifies the setup of simulations, stimulus generation, and result analysis.

III. SENECA Architecture

SENECA architecture is illustrated in Fig.2. SENECA is a multi-core neuromorphic processor where each core contains:

- Data memory: It is the main memory of the core, which can be used to keep all the parameters.
- Neuron Processing Elements (NPEs): 8 NPEs in a core are used to perform neural processing. NPEs support 4b, 8b and 16b data types. In this work, we used the 4b fixed-point type for weights and the 16b BrainFloat type for neuron states.
- RISC-V controller: In SENECA, we use a double-controlling system. RISC-V is the most flexible but, at the same time, energy-consuming controller. This RISC-V implements the RV32IMC instruction set, which can perform 32b integer operations for controlling tasks such as the translation of the address for events.
- Loop Buffer: This is our custom cache memory designed to keep neural network loop instructions local and fetch/decode them in order to the NPEs. This logic block is offloading a significant amount of neural network-specific tasks from RISC-V. Due to limited scope and flexibility, this controller is 10x more power efficient than RISC-V.
- Event Generator: This logic block converts the spike pulses to a packet of data containing the address of the source neuron and an optional value (for graded spike support).
- Network on Chip (NoC): NoC connects the SENECA cores using a compressed multi-casting routing algorithm.

For a more extensive description of the SENECA architecture, the readers are referred to [21].

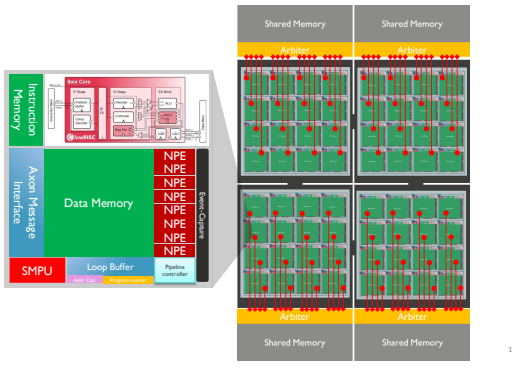

Fig. 2. A 64-core SENECA architecture with shared memory [21]

TABLE I
COMPARING SENSIM WITH OTHER SIMULATORS AND SOFTWARE PACKAGES

Simulator	Programming language support	Execution Platform	Emulation Platform	Learning support	Event/Time-step driven	energy/power estimates	Latency estimates
Lava [1]	Python	CPU, GPU, Loihi, Loihi2	NA	Yes	Event	No	Yes
Carlsim [5]	C/C++	CPU,GPU	TrueNorth, Loihi, Dynapse	Yes	Time-step	Yes	Yes
PyCarl [3]	Python	CPU, GPU	TrueNorth, Loihi, Dynapse	Yes	Time-step	Yes	Yes
Brian2Loihi [14]	Python	CPU, GPU	Loihi	Yes	Time-step	No	Yes
SpyNNaker [18]	Python	CPU, GPU	SpiNNaker	NA	Event	No	Yes
BindsNet [11]	Python	CPU, GPU	NA	Yes	Time-step	No	No
Py(NEST) [8]	Python, SLI	CPU	NA	Yes	Time-step	No	Yes
Core(NEURON) [12]	Python, NMODL	CPU, GPU	NA	Yes	Hybrid	No	Yes
Nengo [4]	Python	CPU, GPU	NA	Yes	Time-step	Yes	Yes
Brian/2 [10] [19]	Python	CPU, GPU	NA	Yes	Time-step	No	No
EvtSNN [16]	Python	CPU	NA	Yes	Event	No	Yes
EDHA [15]	Java	CPU (cross-platform)	NA	Yes	Event	No	Yes
SENSIM	**Python**	**CPU**	**SENECA**	**No**	**Hybrid**	**Yes**	**Yes**

IV. SENSIM

SENSIM is an open-source Python-based parallelized HW-aware simulator for event-based neuromorphic processors. The input and outputs of SENSIM are shown in Fig.1. Currently, SENSIM is specifically designed to model SENECA. However, it has a modular architecture that can serve other event-based dataflow architectures. SENSIM provides detailed and temporal power/latency measurements based on counting the number of operations and memory accesses for a specific application.

SENSIM requires technology and architectural parameters, a definition of the neural network application, and the mapping descriptor to know where each neuron is mapped. At run time, it reads the input event list and generates the output event list. In addition, it makes periodic snapshots of the cores' status (like energy, idle times, FIFO occupations, ...).

A. SENSIM Software Framework

The software framework of SENSIM primarily comprises the neuromorphic core, representing a single neuromorphic processor that allows custom neuron models. Neurons from different layers are mapped to specific cores based on their names and locations. Real-time events from the sensor unit are sent to the processing unit, and their timestamps are recorded at each stage within the core. The simulator allows for various packaging configurations of spikes as events, depending on the communication setup between cores. Timing and synchronization parameters can be adjusted for different read and write times between cores. SENSIM handles the buffering of input data and encapsulates them as events/flits/spikes, considering the communication parameters. SENSIM offers the flexibility to choose between source-based multicast and unicast routing. In addition to providing methods to adjust hardware parameters, SENSIM offers a GUI to visualize data transfers within the hardware, enhancing the user experience.

SENSIM uses multi-thread processing to speed up large-scale simulations. Users can select one or a few SENECA cores and assign them to a thread, allowing a SENECA system with N cores to be parallelized up to N threads. Each thread has its own input and output event queues and is entirely event-driven. However, the threads need to synchronize periodically, which is where the simulation time step is used. Using a long time step may cause the results to deviate slowly from a single-thread simulation, as the order of processing events will be disturbed. On the other hand, using a very short synchronization period time-step increases the overall idle times of the threads, resulting in an increase in simulation time. SENSIM allows users to trade-off between accurate simulation and speed by allowing users to opt for longer time-steps for fast and less accurate simulation in the beginning or shorter time-steps during the fine-tuning phase.

B. Hardware and Simulation Parameters

SENSIM allows for customizing various parameters, including energy, communication, flow control, and architecture. These parameters are selected based on either technology node or architectural requirements. Table II displays the energy parameters along with their calibrated relative energy numbers obtained from circuit simulation tools such as Cadence Joules [20].

Table III are timing parameters that depend on the technology node and are estimated by hardware verification software. All the parameters are estimated in cc (clock cycles).

TABLE II
NORMALIZED ENERGY PARAMETERS FOR SENSIM

Parameter	Value	Description
e_{con}	3	energy unit per each controller operation and instruction memory read
e_{npe}	1	energy unit for each NPE operation
e_{dmem_rd}	3	energy unit for every local data memory bit read
e_{dmem_wr}	3	energy unit for every local data memory bit write
$e_{ext\_mem\_rd}$	300	energy unit for every shared memory bit read
$e_{ext\_mem\_wr}$	300	energy unit for every shared memory bit write
e_{fifo_rd}	1.5	energy unit per each event queue bit read
e_{fifo_wr}	1.5	energy unit per each event queue bit write
$e_{interconnect}$	6	energy unit for sending a bit of data on a interconnect

TABLE III
TIME PARAMETERS FOR SENSIM

Parameter	Value	Description
t_{npe}	1	time units (in cc) per each NPE operation
t_{fifo}	1	time units (in cc) per each event queue access
$t_{interconect}$	1	time units (in cc) for sending a flit of data on the interconnect
t_{ext_mem}	100	time units (in cc) for accessing the shared memory

Table IV are communication and spike packaging parameters that define how the data traverses from one core to another core.

TABLE IV
COMMUNICATION PARAMETERS FOR SENSIM

Parameter	Value	Description
max event flit	9	Maximum limit the max number of flits per event
flit width (fw)	32	Bandwidth (bits) per flit
bw_{ext_mem}	32	Shared memory bandwidth (bits per cycle)
$F_{flowcontrol}$	Strict	The flow control can be controlled by dropping packets in the interconnect or ensuring complete delivery of packets.

Table V details the architectural parameters that can be varied by the architect when designing the computing system.

TABLE V
ARCHITECTURAL PARAMETERS OF SENSIM

Parameter	Value	Description
Event queue depths	[128,128]	Sizes of the input and output queues (in flits) of every core on the chip.
N_{npe}	8	a global parameter that sets the total number of processing elements on every core on the chip.
Mesh size	[64,64]	a parameter which decides to total number of cores (x,y) on the neuromorphic chip in the simulation
Clock Freq	200	Clock frequency at which each processor is running (in MHz)
$Core_{dmem}$	∞	local data memory of each core (in bits)
$N_{spikereg}$	8	number of spike registers for every processing element
$F_{multicast}$	True	Flag to control the multicasting and unicasting of events in the network on chip

C. Graphical User Interface

The SENSIM graphical user interface enables the real-time monitoring of simulation progress. Each SENECA core is represented as a square, displaying its input/output event queues and instantaneous utilization. The colors of the cores indicate their power consumption, starting from white, followed by red and then black, exhibiting heavily utilized cores. A snapshot of the SENSIM GUI is illustrated in Fig.3.

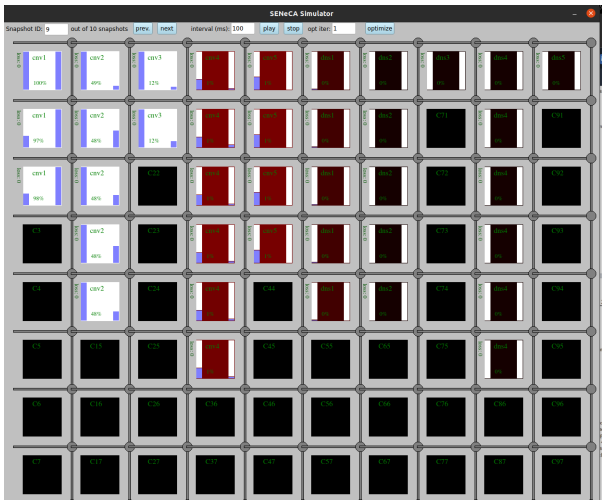

Fig. 3. SENSIM graphical user interface

V. EXPERIMENTS

In order to showcase the effectiveness of the tool, we have employed the PilotNet application [6]. By utilizing our DeltaDNN toolbox [23], we have successfully sparsified the PilotNet for asynchronous inference in event-based hardware. Through the experiments conducted, we have gained valuable insights into how slight modifications in a neuromorphic system can impact the energy consumption of the scaled-up system.

A. Experiment 1: Varying number of neural processing elements (NPEs) per core

Increasing the number of Neural Processing Elements (NPEs) can enhance the processing speed of the neuro-synaptic cores while also reducing energy consumption due to sharing other core elements, such as the controller and loop buffer. However, a higher number of NPEs requires more silicon area and could lead to potential bottlenecks with other core components depending on the application. In this experiment, each layer of PilotNet was assigned to one core, and all cores had the same number of NPEs per neuro-synaptic core. SENSIM measured the normalized consumed energy (Fig. 4) and processing time (Fig. 5) of each core for an event.

B. Experiment 2: Spike compression

Event communication and spike packaging are important sources of latency and energy consumption in large-scale neuromorphic systems. In this experiment, we tried to use a form of compressed sparse column (CSC) representation for

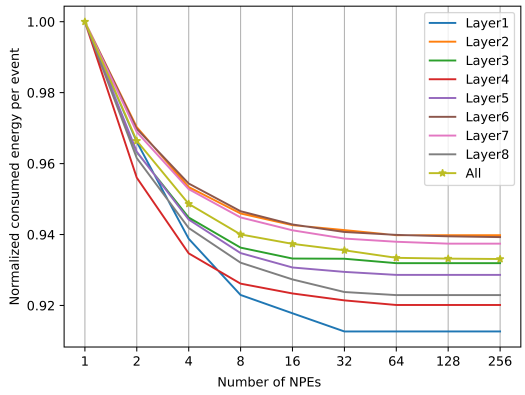

Fig. 4. Normalized total cores' energy consumption per each event.

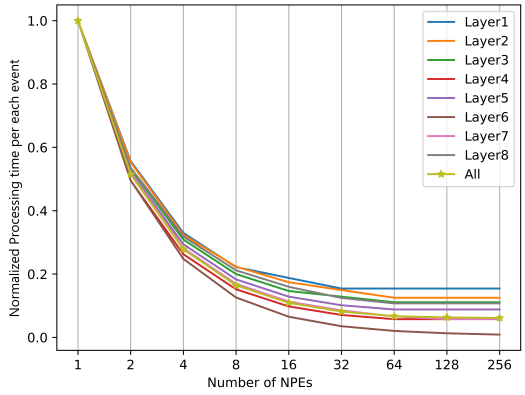

Fig. 5. Normalized total cores' processing time per each input event.

the convolutional layers versus the baseline AER packetization [22] which is used in most neuromorphic chips. Our experimental results in Fig.6 and Fig.7 show major improvements in total communicated bits and latency.

C. Experiment 3: Spike grouping

In neuromorphic systems, memory read and write requires a lot of energy. To improve performance, it's helpful to reuse the data that's already been read from memory. A technique called spike grouping, mentioned in [21], involves updating the neuron state memory with multiple spikes before storing it back. The amount of reuse depends on the number of available spikes to be processed and the size of the level-0 memory (register file).

In this section, we conducted experiments to demonstrate how increasing the size of the register file can enhance the system's energy efficiency. Our results, as shown in Fig.8, indicate that having enough memory to process 4 spikes in parallel can save over 40% of energy.

D. Experiment 4: Varying shared memory bandwidth

In order to overcome the limitations of the technology, SENECA's neuro-synaptic cores are designed to use shared memory as an extension of their internal data memory.

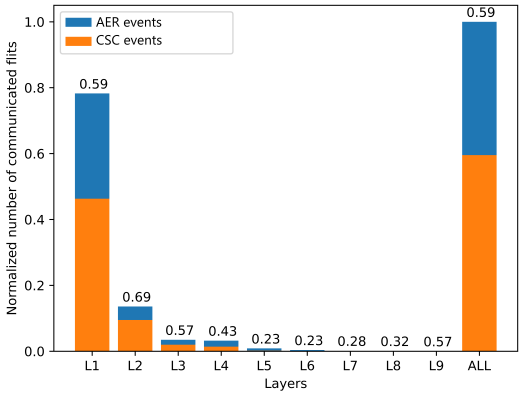

Fig. 6. The normalized number of output flits from each layer of PilotNet when using AER events and CSC event packetization. The number on top of the bars shows the compression ratio for each layer.

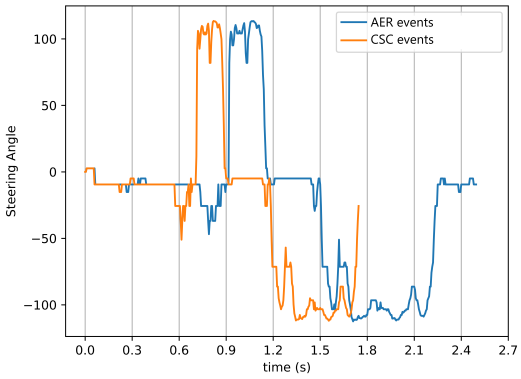

Fig. 7. Output steering angle for PilotNet inference in AER events and CSC event configurations shows significant latency improvements.

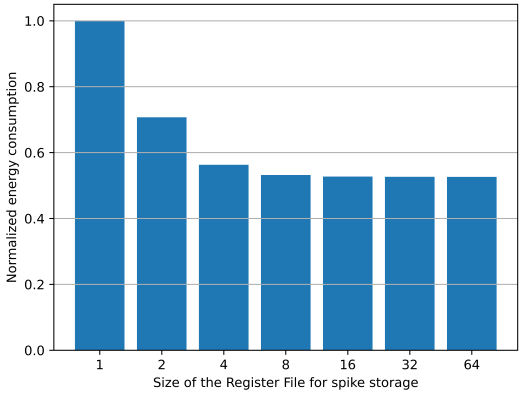

Fig. 8. Normalized energy consumption when processing multiple spikes (spike grouping) in one go. The size of the register files for storing spikes defines the maximum possible number of spikes to be processed together.

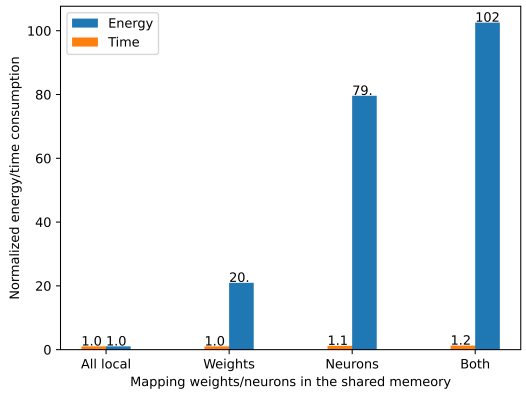

Fig. 9. Comparison of normalized energy consumption when using the shared memory for neuron states, weights, or both in PilotNet.

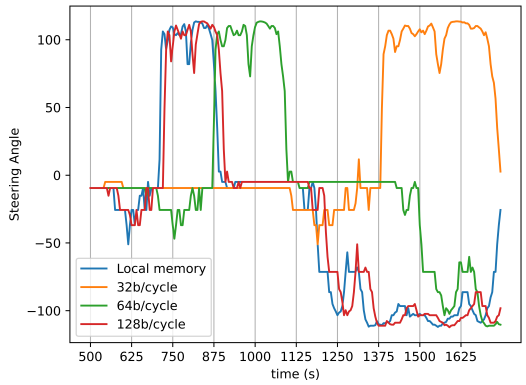

Fig. 10. Output latency of PilotNet for several shared memory bandwidths, compared to the output when not using the shared memory.

However, accessing this memory incurs a higher energy and latency cost compared to local data memory (100× in this experiment). Energy consumption of inference is depicted in Fig.9 when mapping neuron states, weights or both to the shared memory. Compared to local memory, shared memory results in up to a 100 times increase in energy consumption and a 20% increase in latency. Fig.10 illustrates the impact of memory bandwidth on latency when both neuron states and weights are mapped into shared memory. As shown in this figure, when memory bandwidth is not a bottleneck, the latency can be controlled by pre-fetching the data from shared memory proactively for future events waiting in the queue.

Compared to local memory, shared memory results in up to a 100 times increase in energy consumption and 20% increase in latency.

VI. Conclusion

The SENSIM tool offers a convenient way to simulate a larger platform before its realization in silicon. With SENSIM, one can explore various architectural parameters, interconnects, synchronization, flow control, and neuron models. In addition, SENSIM provides a means of benchmarking emerging memory technologies, such as non-volatile and high-bandwidth memory, while allowing for early optimization and development of a scalable platform. The tool also estimates relative power, energy, and latency by approximating memory access and operations on the chip. It also delivers status updates for cores, event queues, and idle processor times for any application at each time step. Experiments show that SENSIM helps make architectural decisions that can reduce processing time by up to 85% and energy consumption by up to 40%, depending on the available hardware for the system.

References

[1] Lava Software Framework — Lava documentation.

[2] Filipp Akopyan, et al. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. 34(10):21, 2015.

[3] Adarsha Balaji, et al. PyCARL: A PyNN Interface for Hardware-Software Co-Simulation of Spiking Neural Network, May 2020. arXiv:2003.09696 [cs].

[4] Trevor Bekolay, et al. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7, 2014.

[5] Michael Beyeler, et al. CARLsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015. ISSN: 2161-4407.

[6] Mariusz Bojarski, et al. The nvidia pilotnet experiments. *arXiv preprint arXiv:2010.08776*, 2020.

[7] Mike Davies, et al. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, PP:1–1, January 2018.

[8] Jochen Eppler, et al. PyNEST: a convenient interface to the NEST simulator. *Frontiers in Neuroinformatics*, 2, 2009.

[9] Marc-Oliver Gewaltig et al. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[10] Dan Goodman et al. Brian: a simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics*, 2, 2008.

[11] Hananel Hazan, et al. BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python. *Frontiers in Neuroinformatics*, 12, 2018.

[12] Michael Hines, et al. NEURON and Python. *Frontiers in Neuroinformatics*, 3, 2009.

[13] Pramod Kumbhar, et al. CoreNEURON : An Optimized Compute Engine for the NEURON Simulator. *Frontiers in Neuroinformatics*, 13, 2019.

[14] Carlo Michaelis, et al. Brian2Loihi: An emulator for the neuromorphic chip Loihi using the spiking neural network simulator Brian. *Frontiers in Neuroinformatics*, 16, 2022.

[15] Lingfei Mo, et al. EDHA: Event-Driven High Accurate Simulator for Spike Neural Networks. *Electronics*, 10(18):2281, January 2021.

[16] Lingfei Mo et al. EvtSNN: Event-driven SNN simulator optimized by population and pre-filtering. *Frontiers in Neuroscience*, 16, 2022.

[17] Saber Moradi, et al. 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, February 2018.

[18] Oliver Rhodes, et al. sPyNNaker: A Software Package for Running PyNN Simulations on SpiNNaker. *Frontiers in Neuroscience*, 12, 2018.

[19] Marcel Stimberg, et al. Brian 2, an intuitive and efficient neural simulator. *eLife*, 8:e47314, August 2019.

[20] Guangzhi Tang, et al. Open the box of digital neuromorphic processor: Towards effective algorithm-hardware co-design, March 2023. arXiv:2303.15224 [cs, eess].

[21] Guangzhi Tang, et al. SENECA: building a fully digital neuromorphic processor, design trade-offs and challenges. *Frontiers in Neuroscience*, 17, 2023.

[22] Amirreza Yousefzadeh, et al. *Multiplexing AER Asynchronous Channels over LVDS Links with Flow-Control and Clock-Correction for Scalable Neuromorphic Systems*. May 2017.

[23] Amirreza Yousefzadeh et al. Delta Activation Layer exploits temporal sparsity for efficient embedded video processing. July 2022.