SuperNeuro: A Fast and Scalable Simulator for Neuromorphic Computing

Prasanna Date
datepa@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Chathika Gunaratne
gunaratnecs@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Shruti R. Kulkarni
kulkarnisr@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Robert Patton
pattonrm@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Mark Coletti
colettima@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Thomas Potok
potokte@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

ABSTRACT

In many neuromorphic workflows, simulators play a vital role for important tasks such as training spiking neural networks, running neuroscience simulations, and designing, implementing, and testing neuromorphic algorithms. Currently available simulators cater to either neuroscience workflows (e.g., NEST and Brian2) or deep learning workflows (e.g., BindsNET). Problematically, the neuroscience-based simulators are slow and not very scalable, and the deep learning–based simulators do not support certain functionalities that are typical of neuromorphic workloads (e.g., synaptic delay). In this paper, we address this gap in the literature and present SuperNeuro, which is a fast and scalable simulator for neuromorphic computing capable of both homogeneous and heterogeneous simulations as well as GPU acceleration. We also present preliminary results that compare SuperNeuro to widely used neuromorphic simulators such as NEST, Brian2, and BindsNET in terms of computation times. We demonstrate that SuperNeuro can be approximately 10×–300× faster than some of the other simulators for small sparse networks. On large sparse and large dense networks, SuperNeuro can be approximately 2.2×–3.4× faster than the other simulators, respectively.

CCS CONCEPTS

• **Hardware** → **Biology-related information processing**; • **Software and its engineering** → **Software libraries and repositories**; • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

ICONS'23, August 1–3, 2023, Santa Fe, NM

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/3589737.3606000

KEYWORDS

Neuromorphic Computing, Neuromorphic Simulator, Neuromorphic Algorithm, Neuromorphic Software, Spiking Neural Networks, General-Purpose Neuromorphic Computing

ACM Reference Format:
Prasanna Date, Chathika Gunaratne, Shruti R. Kulkarni, Robert Patton, Mark Coletti, and Thomas Potok. 2023. SuperNeuro: A Fast and Scalable Simulator for Neuromorphic Computing. In *Proceedings of International Conference on Neuromorphic Systems (ICONS'23).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3589737.3606000

1 INTRODUCTION

Neuromorphic computing is a promising computing paradigm for low-power applications [17], including neuroscience simulations, autonomous vehicles, anomaly detection, graph algorithms, epidemiological modeling, and may other scientific applications [3, 4, 10, 11, 15, 16]. If used in a machine learning setting, training spiking neural networks (SNNs) is a critical step in the neuromorphic workflow, and this is usually accomplished on CPUs or GPUs [2]. If used in another setting, it is important to design, implement, and test novel neuromorphic algorithms efficiently and rapidly [1]. Both cases benefit from an efficient simulator.

Unlike CPUs and GPUs, neuromorphic hardware is not readily available off-the-shelf [14] and is typically found in research settings as research-grade prototype devices (e.g., Intel Loihi, IBM TrueNorth, and SpiNNaker) [5, 6, 8]. As a result, it is difficult to work with the neuromorphic hardware directly, especially for researchers who design novel neuromorphic algorithms or train SNNs for their applications. An efficient neuromorphic computing simulator is needed to mitigate this lack of access to hardware.

Neuromorphic algorithms inform the design of the neuromorphic hardware in many ways [17]. For instance, some neuromorphic algorithms require dense connectivity, whereas others require sparse connectivity. Some algorithms require floating point synaptic weights, whereas others require binary (0, 1) weights. Simulators play a vital role in enabling this co-design between neuromorphic algorithms and hardware. They establish the requirements for the neuromorphic algorithms, thereby making it possible to realize these requirements on the hardware.

Current neuromorphic computing simulators are primarily focused on either neuroscience workloads or deep learning workloads [14]. The neuroscience-based simulators, such as NEST (based on

Table 1: SuperNeuro Modes: Matrix Computation (MAT) Mode and Agent-Based Modeling (ABM) Mode

MAT Mode	ABM Mode
Homogeneous simulations	Heterogeneous simulations
Built-in STDP learning	Built-in STDP learning
CPU execution	GPU acceleration
Fast and scalable	Custom neurons and synapses

discrete event simulation) and Brian2 (based on solving systems of ordinary differential equations), are slow for SNN-based machine learning and general-purpose computing applications [9, 14, 18]. On the other hand, the deep learning–based simulators, such as snntorch and BindsNET, do not provide all the necessary features (e.g., synaptic delays) needed for neuroscience and general-purpose neuromorphic computing applications [7, 12]. To this end, we present SuperNeuro, a fast and scalable Python-based simulator for neuromorphic computing with GPU acceleration capability.

2 SUPERNEURO

SuperNeuro models the neuromorphic simulation problem by using two different approaches: a matrix computation-based approach (MAT) and an agent-based modeling (ABM) approach. To the best of our knowledge, the neuromorphic simulation problem has never been modeled using these approaches. SuperNeuro provides a development framework for accelerating neuromorphic simulations with the flexibility to define custom neuron and synapse models while utilizing computationally efficient algorithms and hardware for highly scalable simulations. SuperNeuro also provides the AI practitioner with the capability to study and optimize large-scale SNNs, regardless of neuromorphic hardware availability. The practitioner can implement general-purpose computing algorithms (e.g., graph algorithms) and non-neural, network-based machine learning algorithms as well as to design neuromorphic primitives for data encoding and data structures. Moreover, SuperNeuro lends itself to different types of workloads, including neuroscience workloads, deep learning workloads (based on SNNs), and general-purpose computing workloads.

Table 1 shows the features of the two SuperNeuro modes. More detailed descriptions are provided in Section 2.1 for MAT mode and Section 2.2 for ABM mode, although the mathematical and algorithmic details are beyond the scope of this paper and will be explored in future work. The SuperNeuro code is open-source and available on GitHub.[1] The goal of this paper is to highlight preliminary computational results for SuperNeuro and compare the new framework with other widely used simulators. The results of this comparison are described in Section 3.

2.1 The MAT Mode

SuperNeuro's MAT mode models neuromorphic simulations by using matrices and vectors. It supports homogeneous simulations (i.e., in which all neurons and all synapses are of the same type). All neurons in the MAT mode are leaky integrate and fire (LIF) neurons, and each neuron has four parameters: threshold, leak, reset state, and refractory period. The leak used in the MAT mode is a constant. Specifically, at each time step, a constant value is subtracted from (or added to) the internal state of each neuron to bring the internal state closer to the reset state. So, if the internal state is greater than the reset state but less than the threshold, then we subtract the leak; else, we add the leak. The internal states of all neurons at a given time step in the simulation are represented as a vector. The neuron thresholds, leaks, reset states, refractory periods, and spikes are also represented as vectors.

Each synapse in the MAT mode has two parameters: weight and delay. The weights are represented in a square matrix such that the weight of the synapse going from neuron i to neuron j is captured at the element located at the i^{th} row and j^{th} column in the matrix. The delay is computed by adding proxy neurons in the simulations such that each and every synapse in the simulation always has a delay of unity. For example, say that a synapse from neuron i to neuron j must have a delay of 3. We implement this functionality by adding two proxy neurons, k_1 and k_2, both having a threshold of 0 and infinite (very high) leak. Next, we connect neuron i to neuron k_1, neuron k_1 to neuron k_2, and neuron k_2 to neuron j such that each of these synapses has a weight and delay of unity. The effective delay going from neuron i to neuron j is thus 3 as required. Although this implementation of synaptic delay sidesteps any temporal computational overheads, it does introduce spatial overheads by adding proxy neurons. As a result, this approach could be inefficient for neuromorphic simulations that contain significant delays on the synapses. In general, all neuron and synapse operations are represented as matrix or vector operations in the MAT mode and thus can be easily parallelized.

To compute the internal states of all neurons at the current time step, we first start with the internal state vector from the previous time step and apply the constant leak to it. Second, the vector of spikes from the previous time step is multiplied by the weight matrix and then added to the internal state vector from the previous time step. Third, if there are any external spikes at the current time step, they are added to the internal state vector of the current time step. Next, the spike vector at the current time step is computed by comparing the current internal state vector to the vector of neuron thresholds. Last, for neurons in their refractory periods, the spikes are zeroed out. All of the above operations are implemented in numpy, which is a highly efficient, CPU-based, numerical computation library in Python.

When the problem is formulated in this fashion, it allows for both speed and scalability for homogeneous simulations. The MAT mode also has a built-in spike-time dependent plasticity (STDP) learning mechanism, which can be used for training SNNs. The STDP mechanism is also implemented by using matrix and vector operations. At present, MAT mode only supports CPU execution, but GPU acceleration will be provided in the future. The MAT mode should be used when speed and scalability are important.

2.2 The ABM Mode

In SuperNeuro's ABM mode, we take a complex adaptive systems perspective to SNN design by simulating each neuron as an individual agent. ABM is a widely used modeling and simulation

[1] https://github.com/ORNL/superneuro

technique for complex adaptive systems, and accordingly, neurons are autonomous entities, thereby allowing for heterogeneous simulations; i.e., neurons may be of different breeds, each supporting a unique spiking mechanism. The simulation is clock driven, with step functions defining the actions taken by agents at each time step. Heterogeneous simulation is a useful feature for exploring neuron and synapse mechanisms that have yet to be realized on the hardware (e.g., stochastic neurons, stochastic synapses).

The ABM mode is implemented on SAGESim, a GPU-capable, ABM framework developed at the Oak Ridge National Laboratory. While using GPUs, SAGESim works by assigning a GPU thread to each agent. SAGESim supports the execution of multi-breed simulation and multiple step functions ordered by priority for each breed. Each neuron in the ABM mode is an agent within the SAGESim simulation. Each neuron agent is provided with two step functions: the neuron step function and the synapse step function. The neuron step function aggregates external spikes at the current time step with the current internal state of the neuron and subtracts the leak. If the new internal state is greater than the neuronal threshold, then the neuron spikes, and the delay registers at outgoing synapses are updated accordingly. In the synaptic step function, the synaptic delay registers are updated, and the final elements of the delay registers are used to update the internal states of the postsynaptic neurons.

The ABM mode supports both CPU and GPU execution, although the GPU device must meet NVIDIA compute capability 6 or higher. In fact, the agent-based form is well suited for GPU acceleration and greatly improves speed and scalability. ABM also supports the LIF mechanism for the neuron. Furthermore, the neuron and synapse parameters supported include threshold, leak, refractory period, axonal delay, synaptic weights, and synaptic delays. The notion of leak supported in the ABM mode is the same as the one for MAT mode. The ABM mode also has a spike-time dependent plasticity (STDP) learning mechanism implemented for on-simulation SNN training. The STDP mechanism is implemented such that each neuron updates the weights of each of its outgoing synapses to postsynaptic neurons by comparing spike time co-occurrences over a specified number of past time steps. The degree of weight change follows an exponential decay with relation to the difference in time between two postsynaptic and presynaptic spikes, and considers both positive (postsynaptic neuron firing after presynaptic neuron) and negative (postsynaptic neuron firing before presynaptic neuron) instances [13].

3 RESULTS

We compare the performance of both SuperNeuro modes against three widely used neuromorphic simulators: NEST, Brian2, and BindsNET. We generated random networks by using a graph and network algorithms Python library called *networkx*. Using the Erdős-Rényi graph generation algorithm in networkx, we generated random graphs such that each node of the graph would correspond to a neuron, and each edge of the graph would correspond to a synapse in our neuromorphic simulations. Networks of 100, 1,000, and 10,000 neurons were generated because networks of these sizes can be efficiently run on a desktop workstation. For each size of the network, we varied the sparsity by changing the connection probabilities of the synapses. We chose the following values for the synapse connection probabilities: 0.25, 0.5, 0.75, and 1. With three values for the number of neurons and four values for the synapse connection probabilities, we had 12 network configurations. Each of these network configurations was used to initialize the neurons and synapses within each of the five simulators. Each simulation was run for 1,000 time steps, and 3 input neurons were chosen at random. Each input neuron was externally spiked at every 10 time steps. All neuron thresholds were set to 1, reset states were set to 0, refractory periods were set to 0, and the axonal delays (if applicable) were set to 0. All synaptic weights and delays were set to 1.

Table 2 lists the total execution times in seconds for all simulator runs. Runs that exceeded 1 hour were terminated and are listed in the table with values of *>1 h*. All ABM and BindsNET runs leveraged GPU acceleration. The MAT mode in SuperNeuro was the fastest across all network configurations. For the smaller jobs (100 neurons), MAT obtained a speedup of 9× for the 0.25 connection probability over the next best simulator, which was NEST in this case. With increasing connection probabilities, MAT obtained increasing speedups of 15×, 20×, and 27× over NEST. Compared with some of the slower simulators, such as Brian2, MAT obtained speedups of 150×, 290×, 406×, and 530× for the four sparsity configurations. The total execution times for ABM were comparable to NEST for the configuration with 100 neurons and a 1.0 connection probability. For this configuration, ABM obtained a speedup of 19× over Brian2 and 2× over BindsNET.

For the medium-sized networks (1,000 neurons), BindsNET performed better than all other simulators except MAT, which obtained speedups of 10×, 7×, 6×, and 6× over BindsNET for the four sparsities. MAT also obtained speedups ranging from 1,695× to 3,174× over Brian2 for the different sparsities. ABM was 7×, 11×, 11×, and 12× faster than NEST for the four sparsities. Compared with Brian2, ABM's speedup ranged from 149× to 279× for the four sparsities. For the large-sized networks (10,000 neurons), BindsNET was once again faster than other simulators, except MAT. In this case, MAT obtained a speedup of 2×–3× over BindsNET for the four different configurations. For the 10,000 neuron and 0.25 connection probability configuration, MAT was 107× faster, and ABM 19× faster, than NEST. At this neuron count, NEST runs for higher connectivity and all Brian2 runs exceeded 1 hour and were terminated, speedups were therefore not calculated. For these larger configurations, ABM took roughly a third of the time as BindsNET. Overall, on smaller-sized networks, MAT was the fastest, followed by NEST, ABM, BindsNET, and finally Brian2. On medium and large-sized networks, MAT was the fastest, followed by BindsNET, ABM, NEST, and finally Brian2.

4 DISCUSSION

An efficient neuromorphic simulator allows us to work with customizable neuron and synapse mechanisms that go beyond the typical deferential equation specifications found in traditional neuroscience. This enables engineering and computer science focused design of SNNs for neuromorphic devices in an accelerated manner. Unlike existing neuromorphic simulators, SuperNeuro enables the practitioner to include heterogeneous neuron and synapse spiking mechanisms within the same SNN.

Number of neurons		100				1,000				10,000			
Connection probability		0.25	0.50	0.75	1.0	0.25	0.50	0.75	1.0	0.25	0.50	0.75	1.0
SuperNeuro	MAT	**0.04**	**0.04**	**0.05**	**0.05**	**0.36**	**0.52**	**0.68**	**0.80**	**30.37**	**40.34**	**55.38**	**71.62**
	ABM	1.10	1.16	1.18	1.26	4.11	5.59	7.22	9.08	166.72	325.30	488.83	641.36
NEST		0.41	0.65	0.91	1.23	28.05	61.42	81.54	112.82	3242.83	>1 h	>1 h	>1 h
Brian2		6.63	12.43	18.36	24.23	612.43	1,249.31	1,892.19	2,529.77	>1 h	>1 h	>1 h	>1 h
BindsNET		2.44	2.34	2.46	2.38	3.45	3.67	4.19	4.64	63.73	117.78	181.04	230.07

Table 2: Total execution times in seconds for SuperNeuro vs. other state-of-the-art simulators for different configurations of network sizes (given by number of neurons) and network sparsities (given by synapse connection probabilities).

SuperNeuro can simulate larger SNNs in less time with higher computational efficiency than any other neuromorphic simulator available today. It can even simulate networks at a scale comparable to some living organisms studied in neuroscience. For example, we have simulated networks with 100,000 neurons (lobster-sized brain) with all-to-all connectivity on a desktop computer in approximately 5 minutes when using MAT mode. By contrast, current simulators such as NEST and Brian2 take more than 1 hour for such tasks. By leveraging high-performance computing resources at Oak Ridge National Laboratory, SuperNeuro could potentially simulate networks with a few million neurons, such as those found in bees and lizards.

SNNs have been used to realize various cognitive and machine learning algorithms, including control, reinforcement learning, classification, decision trees, and regression. Having access to a high-performance simulator is crucial for the rapid development and benchmarking of these algorithms in the neuromorphic domain. By using a high-performance simulator such as SuperNeuro, we can train SNNs for deployment on edge platforms for autonomous vehicles, industrial robotics, autonomous drones, and high energy physics, among others. This enables AI practitioners to rapidly develop and prototype new SNN architectures significantly faster, thereby enabling the co-design of neuromorphic hardware.

5 CONCLUSION

Neuromorphic computing suffers from a lack of fast, highly scalable, and flexible neuromorphic simulators for designing and training SNNs. SuperNeuro provides AI practitioners with a neuromorphic simulator in Python that is both fast and scalable and also provides the option of simulating the user's own spiking mechanisms. SuperNeuro is capable of leveraging GPU acceleration and can provide superior performance compared with existing simulation platforms. SuperNeuro can easily integrate with learning and optimization tools for SNN optimization. This opens many possibilities for the successful co-design of neuromorphic circuits to enable intelligent edge computing device design while also facilitating large-scale AI experimentation on accelerated computing infrastructure.

ACKNOWLEDGMENTS

This material is based in part upon work supported by the US Department of Energy Office of Science's Advanced Scientific Computing Research program under award number DE-SC0022566.

REFERENCES

[1] Aimone, J., Date, P., Fonseca-Guerra, G., Hamilton, K., Henke, K., Kay, B., Kenyon, G., Kulkarni, S., Mniszewski, S., Parsa, M., et al. A review of non-cognitive applications for neuromorphic computing. *Neuromorphic Computing and Engineering* (2022).

[2] Bhuiyan, M. A., Pallipuram, V. K., Smith, M. C., Taha, T., and Jalasutram, R. Acceleration of spiking neural networks in emerging multi-core and gpu architectures. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)* (2010), IEEE, pp. 1–8.

[3] Cong, G., Lim, S.-H., Kulkarni, S., Date, P., Potok, T., Snyder, S., Parsa, M., and Schuman, C. Semi-supervised graph structure learning on neuromorphic computers. In *Proceedings of the International Conference on Neuromorphic Systems 2022* (2022), pp. 1–4.

[4] Date, P., Carothers, C. D., Hendler, J. A., and Magdon-Ismail, M. Efficient classification of supercomputer failures using neuromorphic computing. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)* (2018), IEEE, pp. 242–249.

[5] Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro 38*, 1 (2018), 82–99.

[6] DeBole, M. V., Taba, B., Amir, A., Akopyan, F., Andreopoulos, A., Risk, W. P., Kusnitz, J., Otero, C. O., Nayak, T. K., Appuswamy, R., et al. Truenorth: Accelerating from zero to 64 million neurons in 10 years. *Computer 52*, 5 (2019), 20–29.

[7] Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., and Lu, W. D. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894* (2021).

[8] Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. The spinnaker project. *Proceedings of the IEEE 102*, 5 (2014), 652–665.

[9] Gewaltig, M.-O., and Diesmann, M. Nest (neural simulation tool). *Scholarpedia 2*, 4 (2007), 1430.

[10] Hamilton, K., Date, P., Kay, B., and Schuman D, C. Modeling epidemic spread with spike-based models. In *International Conference on Neuromorphic Systems 2020* (2020), pp. 1–5.

[11] Hamilton, K., Mintz, T., Date, P., and Schuman, C. D. Spike-based graph centrality measures. In *International Conference on Neuromorphic Systems 2020* (2020), pp. 1–8.

[12] Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., and Kozma, R. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in neuroinformatics 12* (2018), 89.

[13] Iakymchuk, T., Rosado-Muñoz, A., Guerrero-Martínez, J. F., Bataller-Mompeán, M., and Francés-Víllora, J. V. Simplified spiking neural network architecture and stdp learning algorithm applied to image classification. *EURASIP Journal on Image and Video Processing 2015* (2015), 1–11.

[14] Kulkarni, S. R., Parsa, M., Mitchell, J. P., and Schuman, C. D. Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing 447* (2021), 145–160.

[15] Markram, H. The blue brain project. *Nature Reviews Neuroscience 7*, 2 (2006), 153–160.

[16] Patton, R., Schuman, C., Kulkarni, S., Parsa, M., Mitchell, J. P., Haas, N. Q., Stahl, C., Paulissen, S., Date, P., Potok, T., et al. Neuromorphic computing for autonomous racing. In *International Conference on Neuromorphic Systems 2021* (2021), pp. 1–5.

[17] Schuman, C. D., Kulkarni, S. R., Parsa, M., Mitchell, J. P., Date, P., and Kay, B. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science 2*, 1 (2022), 10–19.

[18] Stimberg, M., Brette, R., and Goodman, D. F. Brian 2, an intuitive and efficient neural simulator. *Elife 8* (2019), e47314.