A Novel Multi-GPU Neural Simulator

C. M. Thibeault[1,2,3,*], R. Hoang[2], and F. C. Harris Jr.[2,3]

[1]*Department of Electrical and Biomedical Engineering, University of Nevada, Reno. Reno, NV.*
[2]*Department of Computer Science and Engineering, University of Nevada, Reno. Reno, NV.*
[3]*Brain Computation Lab, University of Nevada, Reno. Reno, NV.*

Abstract

Between the biophysical and behavioral studies of the brain lies computational neuroscience. The goal of which, among other things, is to help bridge the gap in our knowledge and provide alternative or complimentary theories to other neurological studies. As more information is provided and more complex theories are developed, the size and computational cost of neural models continues to increase. This is an obvious impediment to the field and something that developers are constantly attempting to overcome. Presented here is a unique simulator design aimed at leveraging advances in hardware for the simulation of biologically realistic neural models. This proof-of-concept design offers an example of a high-performance environment that utilizes multiple general purpose graphical processing units in a novel configuration. The result is a scalable system that offers the promise of both performance and biophysical faithfulness.

1 Introduction

There are many different levels of neuroscience research attempting to clarify the function of the brain. From the single molecule studies of biophysics to the behavioral research of cognitive neuroscience, the mysteries of the brain are enjoying elucidation from both bottom-up and top-down approaches. Computational Neuroscience can be described as a complimentary field that spans the spectrum of neuroscience. There are obvious physiological barriers to gathering detailed information of most complex neuronal structures. Beyond the lack of connection information, is the lack of non-invasive measurement equipment. Computational neuroscience provides unique and unrivaled access to both the deep structures of the brain as well as the molecular information of single cells.

Presented here is a parallel modeling environment written for large-scale neural simulations. The overarching goal of this project is to prove that a generalized simulation architecture could have both extensibility and high-performance. This is achieved through the use of general-purpose graphical processing hardware and a unique software design.

The remainder of this section continues with an overview of neural simulation, the Izhikevich neuron, the CUDA programming environment and a short review of similar simulation environments. The paper continues with the simulator design, an explanation and results of the testing, and finally with the a discussion and future directions for this project.

*Corresponding Author. Corey@cmthibeault.com

1.1 Neural Simulation

Biologically realistic neural simulations generally begin with a model of a single neuron. Of which there are a large number available to computational neuroscientists, each with different levels of computational complexity and biological realism. There is a constant balance between execution time and biophysical plausibility. A balance that more often than not leans in favor of execution time. Even as neuron models are simplified and approximated, the neural structures of interest may require a computationally unreasonable amount of them. In order to further drive the neuroscience research, engineers are creating more optimized simulation environments that take advantage of the latest hardware advances.

Izhikevich Neuron Model

Although simple, the Izhikevich neuron model [1–3] is quite powerful. It is capable of replicating much of the dynamic phenomena observed in neurons from almost all regions of the brain. The mathematical formulation of this model derives from the treatment of a neuron as a dynamical system. The result is a very simple membrane voltage expression, Equation 1, and a simple recovery variable, Equation 2.

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I \tag{1}$$

$$\frac{du}{dt} = a(bV - u) \tag{2}$$

$$\text{if V} \geq 30\text{, then} \begin{cases} V \leftarrow c \\ u \leftarrow u + d \end{cases}$$

The use of this model depends heavily on the hypothesis of the research. In some instances more detail is required to completely elucidate the neural structure. However, researchers often mistake detail for biological realism. Model parameters can be obtained from physiological experiments and used directly in many detailed models. Problems arise when the natural variability in real neuronal parameters is lost through both tuning and measurement errors. The result is a detailed model that doesn't accurately replicate the dynamics observed in the actual system.

1.2 GPGPU Programming

Recently, the utilization of graphics processing units (GPUs) for scientific computing has increased dramatically. Originally intended as a means of offloading graphics and visualization tasks away from the central processing units (CPUs), the single instruction, multiple data architecture of GPUs lends itself to many scientific computing problems. As one of the leaders in graphics chip design, NVIDIA has invested considerable resources in providing the scientific community with both hardware and software solutions aimed at leveraging their products for just such applications. The Compute Unified Device Architecture (CUDA) created by NVIDIA provides developers with a relatively simple instruction set as well as comprehensive tools for working in a GPU environment.

1.3 Similar Simulators

There are a wide variety of neural simulators available to researchers today and the addition of a another may seem excessive. Although many share similar approaches and features, most have unique qualities that separate them. This diversity of simulators has proven to be a benefit for the field [4]. Ideally, the diversity provides a way to explore different levels of model abstraction, which can help determine the level of realism needed in future research, and helps researchers validate model results by using different tool sets. This is difficult achieve in practice, however, as the simulators use different and often incompatible data formats. For a comprehensive review of the more popular simulators see [5].

Within the community there has been a surge in neural simulator development for execution on GPUs. All of these simulators have shown significant improvement over their CPU only counterparts. However, unlike the design presented here, these approaches are all limited to single GPU simulations.

Tiesel et al. [6], created a simulator for spiking integrate-and-fire neurons. A single planar network was constructed without axonal delays or synaptic learning. This simulator is unique in that it exploits the GPU hardware through the OpenGL graphics API rather than a specific computational library such as the CUDA interface described above. Although the simulator out-performed the author's CPU implementation, it is difficult to make an accurate comparison as the CPU version was written in the interpreted language MATLAB (R).

Nageswaran et al. [7], developed a system for modeling networks of Izhikevich neurons on a single GPU utilizing the CUDA programming API. With a C++ interface for network creation and execution this simulator provides a more generalized option than those above. Performance testing for the simulator demonstrated a speed-up of 26 over a comparable CPU version.

Taha et al. [8], constructed a simulator for the modeling of both Hodgkin-Huxley and Izhikevich neuron models. Based on the CUDA API the system was specific to a two-layer input-output network used for image recognition. with the Izhikevich model the authors measured a speed-up or 5.6 over the CPU implementation and with the Hodgkin-Huxley neuron they found a speed-up of 84.4.

2 Simulator Design

The proof-of-concept simulation code described here is presented as an illustration of both the design's scalability and performance potential once integrated to the existing environment. As an unoptimized prototype, it is in many ways a worst-case scenario. Currently only GPUs within a single compute node are supported. However, even in this immature state, the design lends itself to the addition of message passing between compute nodes. This will become more apparent below.

The prototype supports a simple input file format that at present is generated by a separate program. A converter of the NeoCortical Simulator (NCS) [5] input file format has been proposed and development of a NeuroML [9] interpreter has begun. Once the input file has been read in the program executes the steps outlined in Figure 1.

The simulation setup begins with a redistribution of the input model. The neurons are sorted based on the number of synaptic connections. These are then distributed to the respective GPUs in a round-robin fashion. This provides a first-pass load balancing of the model. Once the neurons have been distributed each GPU forms a local indexing and representation of its neurons. The new indexing scheme is shared amongst GPU threads and is used to develop the local neuron structure array and the Cell Firing Bit Vector, as shown in Figure 2. In this implementation the Cell Firing Bit Vector is a representation of the entire neural model at the current simulation time tick. In future version this will be restricted only to Neurons

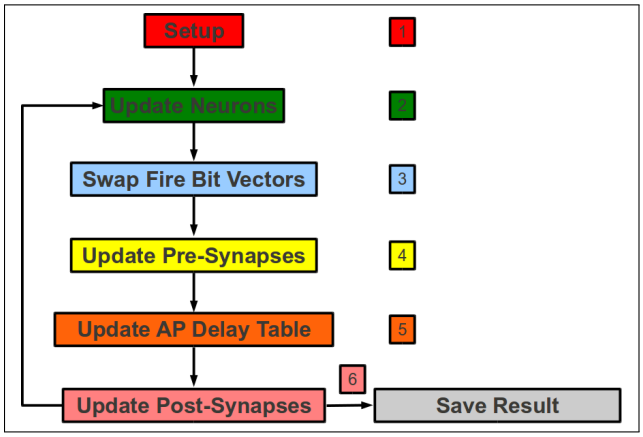


Figure 1: Simulator Execution Flow

that are of interest to a particular GPU thread.

The Local Synapse array is constructed in a similar manner with the synapses being grouped by their presynaptic neural connection. This layout provides a contiguous region of memory that can be accessed with minimal overhead within the GPU architecture.

Finally, the Action Potential Delay Table is constructed. This is a bit vector that provides a mechanism for simulating the propagation delay of action potentials. As seen in Figure 2, the X axis represents the local synapse's axonal connection. The Y axis is a circular buffer that is the size of the maximum propagation delay.

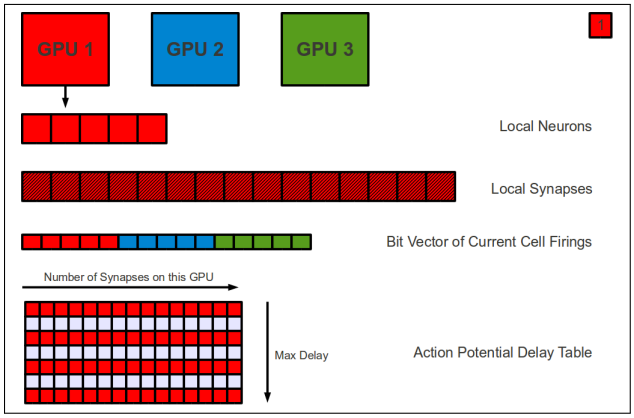


Figure 2: System Setup

After setup the simulation begins by updating (numerically integrating) the neurons. The appropriate region of the Action Potential Delay Table is read and the number of "1" bits are noted. In this context, a "1" bit represents an action potential that has arrived at that particular synapse. The neuron code then samples the electrical current contributed by that synapse. After the entire Delay Table for the current time tick has been read, the voltage of the cells are computed numerically using a forward Euler method. If the cell reaches threshold and fires an action potential, its corresponding bit in the Current Cell Firings Bit Vector is set high.

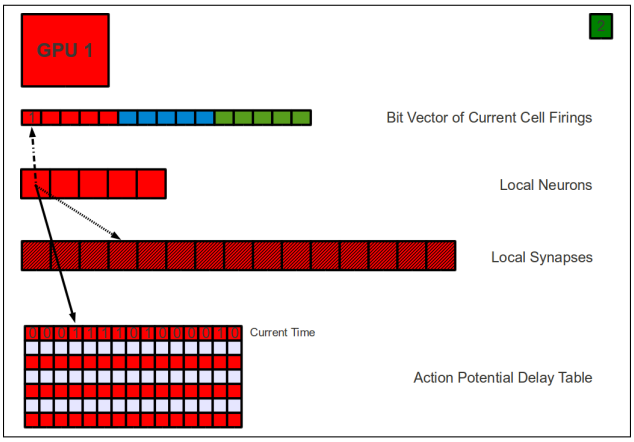


Figure 3: Update Neurons

Once the neurons have been updated and the Cell Firings Bit Vector has been filled in, the GPU threads will pass a copy of the vector to the other GPU threads. This is illustrated in Figure 4. The layout of this vector for each of the respective GPUs should be noted. This was described in the setup above and is a result of the redistribution of neurons.

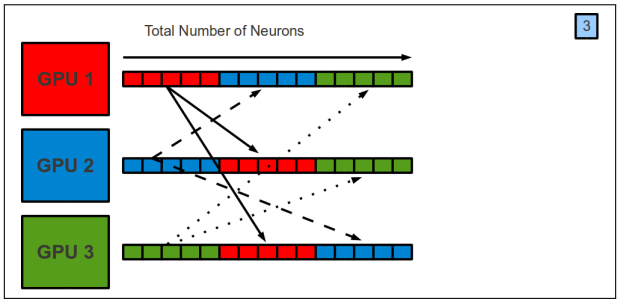


Figure 4: Swapping of the Current Cell Firings Bit Vector

The synaptic updates begin by reading the respective Current Cell Firings Bit Vector, shown in Figure 5. Negative synaptic learning can be calculated and the Action Potential Delay Table can be updated at this time.

Shown in Figure 6, the Action Potential Delay Table is updated for the Neurons that have fired. The appropriate bit, based on the delay specified by the model, is set high.

Finally, the synapses sample their post synaptic

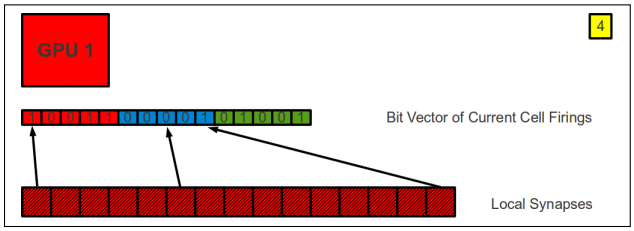


Figure 5: Update Pre-Synapses

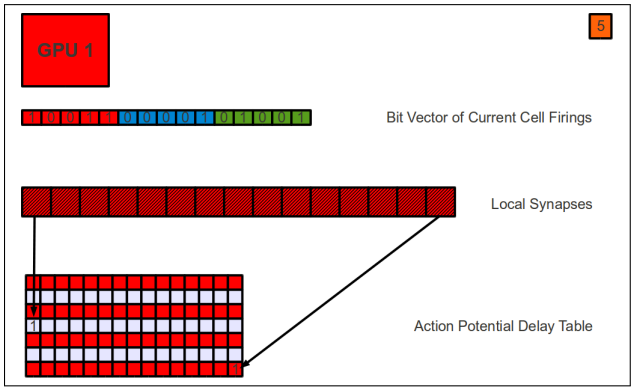


Figure 6: Update Action Potential Table

neuron and use the result to calculate positive learning if needed.

At this point A producer-consumer thread model will grab the current cell firings vector and begin writing it to the output file. Concurrently, if needed, the Neuron Update step starts the process all over again.

3 Testing

Some basic benchmarks were run to illustrate the scalability and functionality of the design. The test network was based on the polychronization models from Izhikevich et al. [10] and Szat et al. [11]. This was also used by Nageswaran et al. [7]. This network utilizes a ratio of four excitatory neurons to 1 inhibitory neuron. The excitatory to excitatory connections were made with spike timing dependent plasticity enabled synapses [12]. It should be noted, that one aspect of this design that negatively impacts the performance was the treatment of learning in synapses. Here, all of the synapses were polled whether they fired or not. Similarly, learning was calculated for all synapses regardless of connection. The model is illustrated in Figure 8.

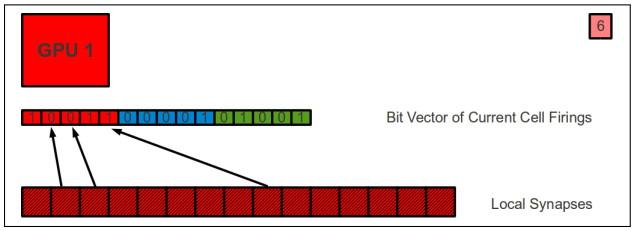


Figure 7: Update Post Synapses

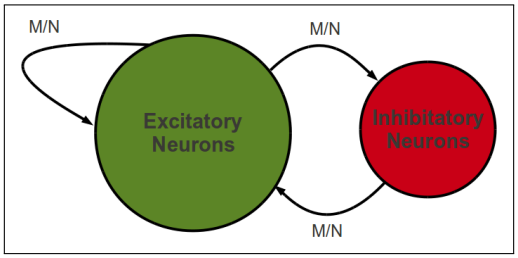


Figure 8: Neuron Network Model. M/N is the probability ratio. M is the number of connections per Neuron. N is the number of Neurons.

Trials were run for 50 seconds of simulation time with the number of neurons and number of connections per neuron modified between trials. Rather than allowing the synaptic weights to reach a steady-state, the model timing was calculated from the beginning. A random input current was injected to 1 of every 1000 neurons. For hardware, two identical Fermi based GeForce 480 graphics cards were utilized.

4 Results

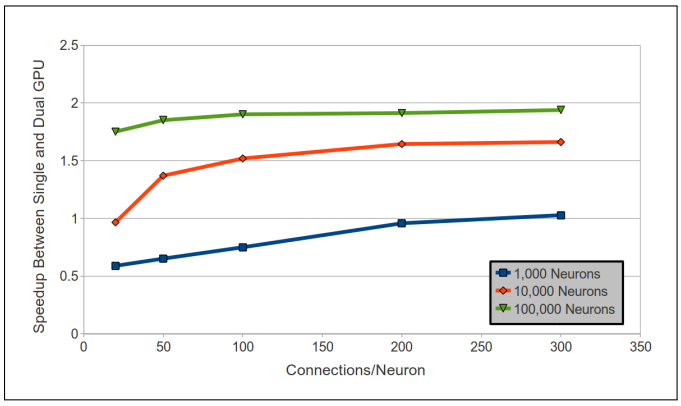


Figure 9: Speedup vs. Connections/Neuron

Presented in Figure 9 are the speedup results between one and two GPU simulations. For networks of only 1000 neurons there is no advantage to moving the

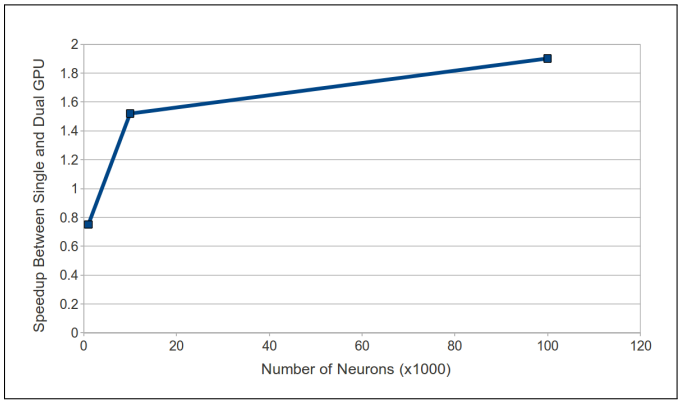


Figure 10: Speedup vs. Number of Neurons (100 Connections per Neuron)

simulation off of a single card. As the network size increases there is a non-linear increase in speedup that can be seen in Figure 10. As the Number of neurons and connections increases the advantage of two cards approaches the ideal speedup of two.

Also of note is the amount data transferred between GPUs. Table 1 illustrates the small amount of information required at each time step. Based on the small bandwidth requirements of the design and the linear dependence on the number of neurons, the addition of hardware should provide a near linear increase in speedup. This is of course dependent on the model sizes as illustrated by these benchmarks.

Table 1: Total Data Transfer between GPUs at each time step.

Neurons	Data/Time step (Bytes)
1,000	125
10,000	1,250
100,000	12,500
1,000,000	125,000

Figure 11 illustrates the real-time capabilities of the prototype simulator. Once again as the number of connections per neuron is increased the advantage of multiple GPUs is enhanced. From this a model with 100,000 Neurons and 50 connections per neuron can run at about 1.2 times real time. We are confident with some basic optimizations that these numbers will be drastically improved.

5 Discussion and Future Directions

Although the performance of this simulator was notable, the potential extensibility is even more important. This was a first-pass implementation and as mentioned before, a worst-case scenario. There are many areas that lend themselves to optimization. There are also additional features that will be included with the intention of making this a more appealing option to researchers.

Clustered GPU Development

This initial offering runs on multiple GPUs within a single node. The next step is to extend the simulator to multiple nodes. As illustrated in this paper the design lends itself to this kind of extension. Additionally, support for heterogeneous clusters will be built-in. This paradigm will provide a flexible platform for many different research groups who may not have access to homogeneous clusters of GPUs.

Standard Input Format

The ability to exchange and extend neural models from different research groups using different simulator packages is one of the main goals of the Neuroscience community [4]. This requires the use of a standard design language. To further enhance the generalization of this simulator a set of input parsers will be constructed. As mentioned above, a NeuroML parser is under construction but additional formats will also be explored as this project matures.

Data Output

An obvious consequence of larger network models is the amount of data produced during a particular experiment. This is further exacerbated as these simulators are incorporated into real-time interactive systems. To overcome the performance requirements, a parallel communication mechanism is under development. This will be incorporated into the UNR Brain Computation Lab's current research utilizing Virtual Neurorobotics (VNR) [13, 14]. Additionally, this is intended for use in real-time visualization of neural simulations.

Acknowledgements

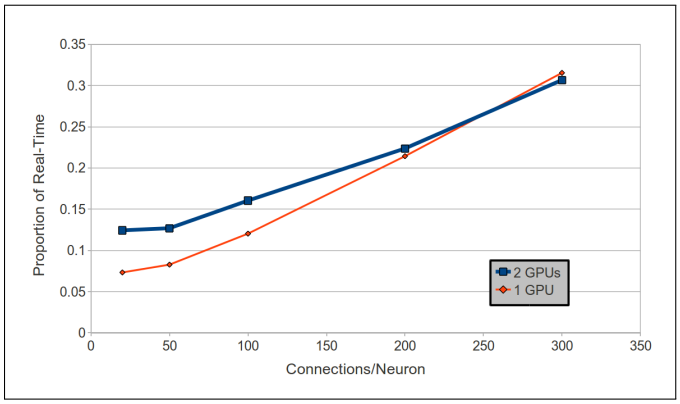
This work was supported in part by grants from the U.S. Defense Advanced Research Projects Agency (HR001109C001) and the U.S. Office of Naval Research (N000140110014).

References

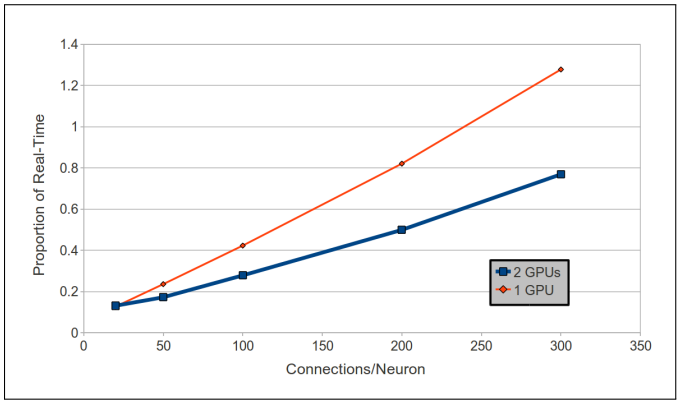
[1] E. Izhikevich, "Simple model of spiking neurons," *Ieee Transactions On Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[2] E. M. Izhikevich, *Dynamical Systems in Neuroscience*. Cambridge, MA: The MIT Press, 2007.

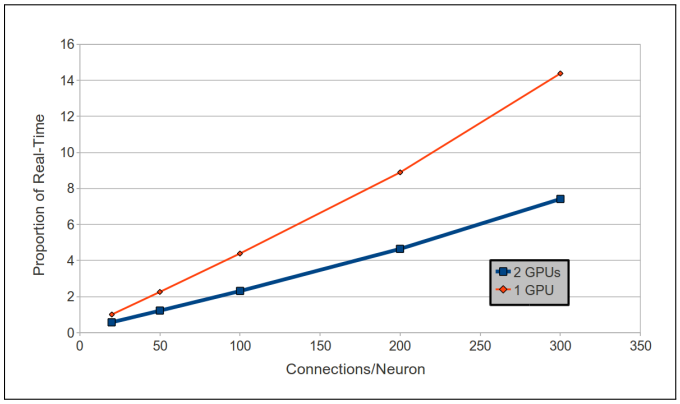
[3] E. Izhikevich, "Which model to use for cortical spiking neurons?," *Ieee Transactions On Neural Networks*, vol. 15, no. 5, pp. 1063–1070, 2004.



(a)



(b)



(c)

Figure 11: Timing comparison for number of neurons vs. number of connections per neuron. (a) 1,000 Neurons., (b) 10,000 Neurons., (c) 100,000 Neurons.

[4] M. Djurfeldt and A. Lansner, "Workshop report: 1st incf workshop on large-scale modeling of the nervous system.," *Available from Nature Precedings*, 2007.

[5] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris, M. Zirpe, T. Natschlager, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. Davison, S. El Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: a review of tools and strategies," *Journal of computational neuroscience*, vol. 23, no. 3, pp. 349–398, 2007.

[6] J.-P. Tiesel and A. S. Maida, "Using parallel gpu architecture for simulation of planar i/f networks," *Neural Networks, IEEE - INNS - ENNS International Joint Conference on*, vol. 0, pp. 3118–3123, 2009.

[7] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural Networks*, vol. 22, no. 5-6, pp. 791 – 800, 2009. Advances in Neural Networks Research: IJCNN2009, 2009 International Joint Conference on Neural Networks.

[8] B. Han and T. M. Taha, "Acceleration of spiking neural network based pattern recognition on nvidia graphics processors," *Appl. Opt.*, vol. 49, no. 10, pp. B83–B91, 2010.

[9] P. Gleeson, S. Crook, R. C. Cannon, M. L. Hines, G. O. Billings, M. Farinella, T. M. Morse, A. P. Davison, S. Ray, U. S. Bhalla, S. R. Barnes, Y. D. Dimitrova, and R. A. Silver, "NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail.," *PLoS computational biology*, vol. 6, pp. e1000815+, June 2010.

[10] E. Izhikevich, "Polychronization: Computation with spikes," *Neural Computation*, vol. 18, no. 2, pp. 245–282, 2006.

[11] B. Szatmáry and E. M. Izhikevich, "Spike-timing theory of working memory," *PLoS Comput Biol*, vol. 6, p. e1000879, 08 2010.

[12] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neuroscience*, no. 9, pp. 919–926, 2000.

[13] P. H. Goodman, S. Buntha, Q. Zou, and S.-M. Dascalu, "Virtual neurorobotics (vnr) to accelerate development of plausible neuromorphic brain architectures," *Frontiers in Neurorobotics*, 2007.

[14] P. H. Goodman, Q. Zou, and S.-M. Dascalu, "Framework and implications of virtual neurorobotics," *Frontiers in Neuroscience*, p. 5, 2008.