GPGPU Accelerated Simulation and Parameter Tuning for Neuromorphic Applications

Kristofor D. Carlson[1], Michael Beyeler[2], Nikil Dutt[2], Jeffrey L. Krichmar[1,2]

Department of Cognitive Sciences[1]
School of Social Sciences
University of California, Irvine
Irvine, CA 92697
{kdcarlso}{jkrichma}@uci.edu

Department of Computer Science[2]
Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697
{mbeyeler}{dutt}@uci.edu

Abstract - Neuromorphic engineering takes inspiration from biology to design brain-like systems that are extremely low-power, fault-tolerant, and capable of adaptation to complex environments. The design of these artificial nervous systems involves both the development of neuromorphic hardware devices and the development neuromorphic simulation tools. In this paper, we describe a simulation environment that can be used to design, construct, and run spiking neural networks (SNNs) quickly and efficiently using graphics processing units (GPUs). We then explain how the design of the simulation environment utilizes the parallel processing power of GPUs to simulate large-scale SNNs and describe recent modeling experiments performed using the simulator. Finally, we present an automated parameter tuning framework that utilizes the simulation environment and evolutionary algorithms to tune SNNs. We believe the simulation environment and associated parameter tuning framework presented here can accelerate the development of neuromorphic software and hardware applications by making the design, construction, and tuning of SNNs an easier task.

I Introduction

Neuromorphic systems are gaining importance as traditional scaling in CMOS technology begins to reach its physical limits. These systems aim to mimic the biological structure of the nervous system; potentially both for solving engineering applications as well as understanding neural computation, which is one of the grand challenges of the 21st century [1], [2]. Biological information processing systems operate at performance levels set by fundamental physical limits, and do so under severe constraints of size, weight, and energy resources. As a result of these constraints, biological nervous systems are extremely energy-efficient (8–9 orders of magnitude better than digital computation [3]). In addition, brain networks employ learning at all levels of computation, are capable of adapting to complex environments, and possess remarkable fault tolerance by maintaining excellent performance even after the loss of many neurons. Thus investigating the computational mechanisms and engineering strategies that give rise to these system properties may not only further our understanding of the brain, but may also lead to novel algorithmic and architectural approaches that can overcome the limits of Moore's law.

A key aspect to the computing power of brain circuits is their massively parallel architecture. Brain systems are organized into highly interconnected modules that operate in concert with one another to carry out intrinsically parallel algorithms, rather than parallelizations of inherently serial procedures. For example, a certain computation in the brain might be carried out by millions of low-precision processing elements (neurons) in less than 100 serial steps [4]. On the other hand, parallelization of serial code typically leads to very limited speedup due to Amdahl's law [5]. The resulting energy efficiency of brain architectures is remarkable: For example, although there are approximately 20 billion neurons and 240 trillion synapses in the human cortex alone, the power consumption of the human brain is estimated to be no more than 13–15 watts [6].

Another key aspect to the computing power of brain circuits is the use of an event-driven communication protocol. Generally speaking, neurons employ relatively infrequent (~10–100 Hz) brief electrical pulses (called action potentials or spikes) as their main communication means. These pulses travel along a wire (axon) to the connection site (synapse) of another neuron, where they cause a small change in the electric potential of the receiving neuron. Neurons integrate these small changes and spike when their voltage reaches a threshold value [7]. Neuromorphic systems have successfully modeled this type of communication using address-event representation (AER), which is a communication protocol that represents each spike by its location (that is, the neuron that fired; explicitly encoded as an address) and the time at which it occurred (implicitly encoded) ([8], [9]). Using AER, it is possible to emulate massive connectivity in an efficient way.

A powerful framework in the development of neuromorphic applications that captures both of these key aspects is the use of spiking neural network (SNN) models in combination with highly parallel, off-the-shelf graphics processing units (GPUs). SNN models provide detailed neuronal dynamics [10] while utilizing the digital AER protocol for efficient communication, which makes them amenable to hardware application development. Furthermore, recent developments in high-performance GPUs enable the simulation of large-scale SNNs in real-time on affordable, programmable platforms. In order for the field of neuromorphic engineering to produce results and applications of practical value, such large-scale networks will be necessary. However, the tuning and stabilization of these large-scale dynamical systems is challenging, due to the large number of state variables and open parameters. Incorrect values in the parameter landscape lead to unstable, chaotic or undesired network operations (e.g.,

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

epileptic oscillations). Thus there is a need for both hardware and software tools that can aid the modeler in the otherwise extremely tedious and possibly error-prone process of tuning complex, large-scale dynamical systems.

In this paper we introduce a software environment for the efficient simulation of SNN models on general-purpose graphics processing units (GPGPUs) as well as an automated parameter tuning framework that uses evolutionary algorithms (EAs) to tune SNN models in parallel. The remainder of the paper is organized as follows. In Section II, we briefly discuss recent hardware and software efforts aimed at building neuromorphic applications. Section III then introduces a GPU-accelerated SNN simulator called CARLsim, and the accompanying parameter tuning interface (PTI) for use in the neuromorphic engineering community.

We believe that the simulation environment and parameter tuning framework presented here will allow neuromorphic engineers to more easily construct larger, more complex SNNs, leading to the development of more powerful neuromorphic applications that may offer practical solutions to currently unsolved real-world problems.

II. Neuromorphic Hardware Devices and Software Tools

A. *Neuromorphic Hardware Devices*

Neuromorphic engineers have made significant progress developing neuromorphic devices to emulate both sensory systems and cognitive architectures. We briefly review recent advances in the development of neuromorphic cognitive architectures from research teams in the USA and Europe. We refer the reader to the following review on neuromorphic sensory systems [11] and focus our discussion on neuromorphic devices that emulate cognitive architectures.

The construction of neuromorphic chips and devices is an active area of research, and has spawned major research initiatives. The Neurogrid board at Stanford University is a neuromorphic device that emulates ion channels with analog circuit components but handles synaptic addressing with digital circuit components. It is capable of simulating 1 million neurons and 6 billion synaptic connections in real-time using only 5 watts and is an impressive example of the speed and power that can be achieved with neuromorphic devices [12]. Both IBM and HRL Laboratories, LLC (HRL) participated in the DARPA-funded systems of neuromorphic adaptive plastic scalable electronics (SyNAPSE) project, where the goal was to build highly scalable neuromorphic devices. The Cognitive Computing Group at IBM recently unveiled its True North architecture, which features a hierarchical design of neurosynaptic cores, each with 256 neurons and approximately 256k synapses [13]. These neurosynaptic cores are built from silicon neurons that can perform many realistic biophysical behaviors but lack synaptic plasticity (and thus lack learning capabilities). HRL also released a general purpose neural chip which has 576 neurons and 70k time multiplexed virtual synapses. The neural chip implements simple spiking neural models and plasticity in the form of spike-timing-dependent plasticity (STDP) [14], a learning paradigm which modulates the weight of synapses according to their degree of causality.

European researchers have also made advances in the design of neuromorphic chips due to funding from a number of initiatives that include two projects called fast analog computing with emergent transient states (FACETS) and brain-inspired multiscale computation in neuromorphic hybrid systems (BrainScaleS). The FACETS/BrainScaleS projects have produced two neuromorphic hardware devices to date. The first neuromorphic device is a single chip system called Spikey, which simulates 384 spiking neurons and 256 synapses per neuron [15]. The second neuromorphic device is more ambitious and is referred to as a wafer-scale neuromorphic hardware system. This system is constructed from 352 separate analog network cores (ANCs), in which each contains 512 spiking neurons and 16k synapses per neuron. The 352 ANCs can fit onto a single 20 cm wafer with a total of 180k neurons and $4 \cdot 10^7$ synapses [16].

Neuromorphic architectures can be either digital, analog, or a hybrid. The SpiNNaker (a contraction of spiking neural network architecture) project uses a digital design that has resulted from a unique collaboration between UK universities and industry partners. The ultimate goal of the SpiNNaker project is to build a computing engine that consists of 1,036,800 ARM9 processor cores capable of simulating 1 billion neurons in real-time. The SpiNNaker computing engine consists of an array of nodes, each containing 18 ARM9 cores, which communicate via packets using a custom interconnect fabric. Each ARM9 core can model 100 neurons and approximately 1M synapses or 10,000 inputs per neuron [17]. The projected power dissipation for the full million-core machine is 90 kW. SpiNNaker can implement spiking and non-spiking neurons models and synaptic plasticity with a novel form of STDP [18]. On the other hand, research teams from the University and ETH Zurich have recently developed a hybrid analog/digital VLSI implementation of an SNN with programmable synaptic weights. The chip has 32 silicon neuron circuits, 128 virtual synaptic weights per neuron, and STDP learning. Because the synaptic weights can be changed on-line and saved offline, the chip can be used to explore spike-based learning rules [19].

B. *Neuromorphic Software Tools: SNN Simulators*

Many research groups have developed SNN simulators to study brain function [20]–[22] or develop neuromorphic applications [23], [24]; we briefly review SNN simulators produced from these research efforts. Both NEURON [25] and GENESIS [26] began as simulation environments originally designed for detailed neuronal modeling at the ionic channel level, but both have the capability to run network models. These simulation environments have a large user-base, extensively-tested code, and parallelized versions that can be executed using MPI on super-computing clusters. However, the computation cost for solving all the equations governing channel activity and propagation of signals makes these models difficult to use in large network applications. Other simulation environments like CSIM/PCSIM, NCS, XPPAUT, SPLIT, Brian, NEST, and Mvaspike were built specifically to run SNNs and have been optimized with SNNs in mind; see [27] for more information. Of particular note are Brian [20] and NEST [21], which have Python interfaces, multiple spiking neuron model implementations, and distributed

parallel implementations. Brian is flexible and easily extensible, partially because it is written in Python. However, there is a slight performance penalty, as it is about 25% slower than similar C implementations. NEST has an optional Python front-end but avoids the potential decrease in performance by using a kernel written in C++. Although NEST has an MPI implementation, it does not currently have a parallel GPU implementation. Because NEST is a larger, more mature software project, the implementation of new features like accelerated GPU implementations may take more time.

There are SNN environments implemented for specific hardware and simulation environments. For example, IBM's C2 SNN simulator is massively parallelized and designed to run on powerful Blue Gene/P supercomputing clusters [28]. The C2 simulator ran a large-scale SNN simulation that consisted of 1.6 billion neurons and 8.87 trillion synapses on a Blue Gene/P with 147,456 central processing units (CPUs) and 144 TB of memory, which is one of the largest SNN simulations to date [29]. NENGO is an SNN simulator that uses a control theory oriented approach called the neural engineering framework (NEF) to specify the synaptic weights required to achieve a desired computation and has been used to build a large-scale brain model with impressive functionality [22]. There are also SNN simulators designed to mimic neuromorphic hardware computing architectures, such as HRLsim [23] developed at HRL and Compass [24] developed at IBM. However, these classes of simulators are currently not available for public use.

C. GPU-Enabled SNN Simulators

Many SNN modelers are turning to GPGPUs for developing application software. Modern GPUs are a low-cost alternative to traditional supercomputing clusters for applications in scientific computing [30] and theoretical neuroscience [31]. A number of groups have developed parallel implementations of SNN simulators that run on GPUs [23], [32]–[38] . For a more comprehensive review see [39].

Whereas some of these software tools are still under heavy development, there are a number of fully functional SNN simulators that feature a whole range of detailed neuronal and synaptic dynamics (such as spike-rate adaptation, plasticity, homeostasis, and specific ion channels), routines that enable the construction of arbitrary connection topologies, and an optimized GPU implementation. Among them are HRLsim [23], NeMo [33], and CARLsim [36], [37]. Although HRLsim is the only simulator to offer parallelization across a GPU cluster using MPI and CUDA, it is currently unavailable for public use. NeMo is a C++ library that simulates networks of Izhikevich neurons on multiple CUDA-enabled GPUs, with a frontend in C/C++, Matlab, and Python. NeMo also features synaptic plasticity in the form of STDP and axonal delays. CARLsim has specifications that are similar to NeMo, in that it simulates networks of Izhikevich neurons on a single CUDA-enabled GPU. Multi-GPU support is planned in the future. Additionally, CARLsim offers optimal computational efficiency by utilizing a reduced AER protocol, different kinds of synaptic plasticity, a method to stabilize synaptic dynamics, the simulation of specific ion channels, and the option to either run the network on a CPU or a GPU. Moreover, CARLsim has been used in a variety of computational studies to simulate detailed large-scale models of cortical processing. The next section will discuss these features in more detail.

III. CARLsim: An SNN Simulator

The Cognitive Anteater Robotics Laboratory Simulator (CARLsim) was designed to make large-scale SNN modeling readily available and is intended for use by the computational neuroscience and neuromorphic engineering communities [36], [37]. CARLsim is written in C/C++ and has both a single-threaded CPU implementation and parallelized GPU implementation. To maximize accessibility, CARLsim runs on both generic x86 CPU architectures and the widely used NVIDIA CUDA GPU architecture under both the Windows and Linux operating systems. We provide a user-friendly programming interface similar to that of PyNN [40] and allow the user to specify a number of preprogrammed connection topologies along with a mechanism to allow for user-defined connection topologies. CARLsim is publicly available at http://www.socsci.uci.edu/~jkrichma/CARLsim/.

A. CARLsim Features

CARLsim uses the Izhikevich spiking neuron model [41], which is well-suited for large-scale neuromorphic applications, because it is computationally efficient yet allows for complex neuronal dynamics that closely mimic biological neurons. CARLsim includes expressions to model specific ion channels (such as AMPA, GABA, and NMDA), which play an important role in neuronal excitability and plasticity.

CARLsim employs a reduced AER protocol for efficient encoding of neuronal communication, which helps reduce both memory usage and memory bandwidth limitations. Recall that AER stores a spike event by representing it as an address-time pair. If many neurons have the same time step, however, this approach leads to high memory overhead due to duplicate storing of time for each address. We overcome this limitation by removing the duplicate time entry for each address, and instead store the cumulative count of fired neurons during each time step [37]. Additionally, SNN state variables are compactly stored in memory and ordered in a manner that minimizes the state update process.

CARLsim includes descriptions of synaptic plasticity at different time scales. Short-term plasticity (STP) occurs over timescales of milliseconds to minutes, whereas long-term plasticity (LTP) occurs over time steps of minutes or longer [42]. STP changes the synaptic weight over time in a way that reflects the history of presynaptic activity. It can be used to model phenomena such as synaptic facilitation or synaptic depression (fatigue). LTP can be induced through STDP, which has been shown to play a crucial role in unsupervised learning [43], forming sparse representations of temporal sequences [44], and computing with neural synchrony [45]. CARLsim offers routines to implement both STP and LTP.

Learning rules in SNNs can often lead to unstable, runaway synaptic dynamics and completely disrupt learning and neural function [46]. To cope with this challenge, CARLsim implements a biologically plausible weight update rule that promotes stable learning and homeostasis that mimics

experimental observations [47]. For more information on the details of the STDP and homeostasis update rules please see reference [48].

B. CARLsim Parallel GPU Implementation

The GPU-accelerated NVIDIA CUDA implementation of CARLsim is an important aspect of the simulation framework and allows for a significant speed up over the single-threaded CPU implementation, as we will see later. The basic structure of the CUDA GPU architecture is shown in Figure 1. Each GPU consists of multiple streaming multiprocessors (SMs) and a global memory accessible by all SMs. Each SM is built from multiple floating-point scalar processors, a cache/shared memory, and one or more special functions units (SFU), which execute transcendental functions such as sine, cosine, and square root operations. CUDA groups parallel threads into 'warps', where the number of threads per warp varies depending on the specific CUDA architecture. Each SM also has at least one warp scheduler that is built to maximize the number of threads running concurrently. The warp scheduler monitors threads within a warp and automatically switches to another warp when a thread makes a time-consuming memory access.

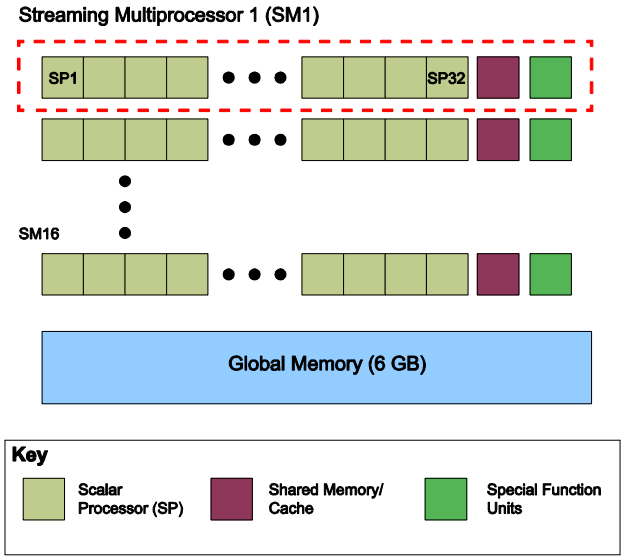

Fig. 1. Simplified diagram of NVIDIA CUDA GPU architecture.

CARLsim simulations are primarily run on the M2090 Tesla GPU that utilizes the CUDA Fermi architecture. The Tesla M2090 has 6 GB of global memory, 512 cores (each operating at 1.30 GHz) grouped into 16 SMs, and a single precision compute power of 1331.2 GFLOPS. Each SM is composed of 32 SPs, 16 load/store units, 4 special function units, and two warp schedulers [49].

The GPU implementation of CARLsim utilizes a number of approaches to maximize the degree of parallelization, minimize memory usage, reduce the effects of memory bandwidth limitations, and avoid thread/warp divergence. There are number of ways to assign SNN calculations to the GPU threads for parallelization. N-parallelism describes organizing the computations assigned to GPU threads by neuron while S-parallelism organizes the computations assigned to GPU threads by synapse. CARLsim uses both of these approaches by using N-parallelism for neuron state variable updates and S-parallelism for synapse variable updates. Thread/warp divergence occurs when a thread executes a different operation than other threads in a warp causing the other threads to wait for its completion. CARLsim prevents thread/warp divergence by buffering data until all threads are ready to execute the same operation. More information about the CARLsim GPU implementation can be found in reference [36].

C. CARLsim Applications

CARLsim has been used to construct large-scale simulations of cognitive processes on the order of 10k–100k neurons and millions of synapses, with examples that include models of visual processing [37], neuromodulation [50], and neural plasticity [48]. Although their efficient implementation is challenging due to the associated computational cost, investigating large-scale models of cortical networks in more biological detail is widely regarded as crucial in order to understand brain function [51].

One of our most recent works [52] concerned the ability of a large-scale spiking neural network model to rapidly categorize highly correlated patterns of neural activity such as handwritten digits from the MNIST database [53]. Although many studies have focused on the design and optimization of neural networks to solve visual recognition tasks, most of them either lack neurobiologically plausible learning rules or decision-making processes. In contrast, our model demonstrated how a low-level memory encoding mechanism based on synaptic plasticity could be integrated with a higher-level decision-making paradigm to perform the visual classification task in real-time.

The model consisted of Izhikevich neurons and conductance-based synapses for realistic approximation of neuronal dynamics, an STDP-like synaptic learning rule for memory encoding (previously described in [54]), and an accumulator model for memory retrieval and categorization [55]. Grayscale input images were fed through a feed-forward network consisting of visual cortical areas V1 and V2 (selective to one of four spatial orientations, in 45° increments), which then projected to a layer of downstream classifier neurons through plastic synapses that implement the STDP-like learning rule mentioned above. Decision neurons were equally divided into ten pools, each of which would develop selectivity to one class of input stimuli from the MNIST dataset (i.e., one of the ten digits) as a result of training. Population responses of these classifier neurons were then integrated over time to make a perceptual decision about the presented stimulus.

The model constitutes an important proof of concept; that is, (i) to show how considerably hard problems such as visual pattern recognition and perceptual decision-making can be solved by general-purpose neurobiologically inspired cortical models solely relying on local learning rules that operate on the abstraction level of a synapse, and (ii) to do it in real-time. The network achieved 92% correct classifications on MNIST in 100 rounds of random sub-sampling, which provides a conservative performance metric, yet is comparable to other SNN approaches ([54], [56]). Additionally, the model correctly predicted both qualitative and quantitative properties of reaction time distributions reported in psychophysical

experiments. The full network, which comprised 71,026 neurons and approximately 133 million synapses, ran in real-time on a single NVIDIA Tesla M2090, which demonstrates the efficiency of the CARLsim implementation. Moreover, because of the scalability of the approach and its neurobiological fidelity, the model can be extended to an efficient neuromorphic implementation that supports more generalized object recognition and decision-making architectures found in the brain.

III. CARLsim Parameter Tuning Interface

The size and complexity of the SNN models used by the neuromorphic engineering community has been steadily increasing in an effort to capture more biological realism [57] and underlying functionality [22]. This complexity has taken the form of more detailed neuron models, the inclusion of plasticity rules, and connection topologies that include recurrent connections. The integration of these features into SNN models comes at a cost: it produces SNNs with less stable neuronal and synaptic dynamics. Both the size and instability of this new generation of SNNs have made the task of constructing and tuning them a difficult one. To meet this challenge, we have developed an automated parameter tuning framework that uses evolutionary algorithms (EAs) and CARLsim to construct and tune SNNs in parallel with GPUs.

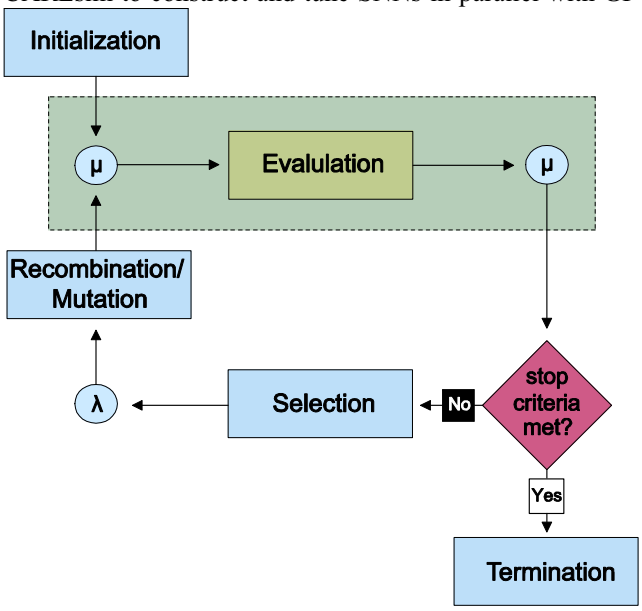

Fig. 2. Diagram illustrating the overall approach of the parameter tuning framework using evolutionary algorithms. The dotted gray box indicates processes that are done in parallel using the GPU implementation of CARLsim. The rest of the processes are done sequentially using Evolving Objects.

The process of tuning an SNN with the automated parameter tuning framework is detailed in Figure 2: (1) a population of SNNs, each with a different set of parameters, is created. (2) Each SNN is evaluated by a fitness function and assigned a fitness score based on how well the behavior of the SNN matches a target behavior. (3) The highest scoring SNNs are selected to produce the next generation of offspring via recombination and mutation while the remaining individuals are discarded. (4) This evolutionary process continues until the desired fitness is reached or another termination condition has been met. Using this approach, an SNN can be tuned to produce suitable firing patterns, stable learning dynamics, and behaviors that closely match experimental data.

The automated parameter tuning framework has three components: the CARLsim SNN simulator, an open source EA library called Evolving Objects [58], and a program to pass information between CARLsim and Evolving Objects we call the parameter tuning interface (PTI). Evolving Objects handles all EA computations shown with light blue boxes in Fig. 2. CARLsim executes the most computationally expensive portion of the tuning algorithm, evaluating the fitness of each SNN, in parallel indicated by the light brown box. During each new EA generation, Evolving Objects assigns parameters from each offspring individual in the population to CARLsim SNNs using the PTI. CARLsim then runs these SNNs, each with a different set of parameters, in parallel and assigns them a fitness which is passed backed to Evolving Objects via the PTI. Evolving Objects then selects those individuals with the best fitness and produces the new parent generation via recombination and mutation until a termination condition is reached.

A. Preliminary Tuning Framework Results

As a proof of concept, an SNN with 4,104 Izhikevich neurons, two forms of synaptic plasticity, and feedback connections was successfully tuned to reproduce neuronal responses found in the visual cortex. Each generation consisted of 10 SNNs and the simulation framework took 127.2 hours of wall-clock time to complete 287 generations. The parallelized GPU CARLsim implementation was performed on an NVIDIA Tesla M2090 GPU card with 6 GB of memory and 512 cores. The single-threaded CPU implementation was performed on a system with an Intel Core i7 2.67 GHz quad-core processor with 6 GB of memory.

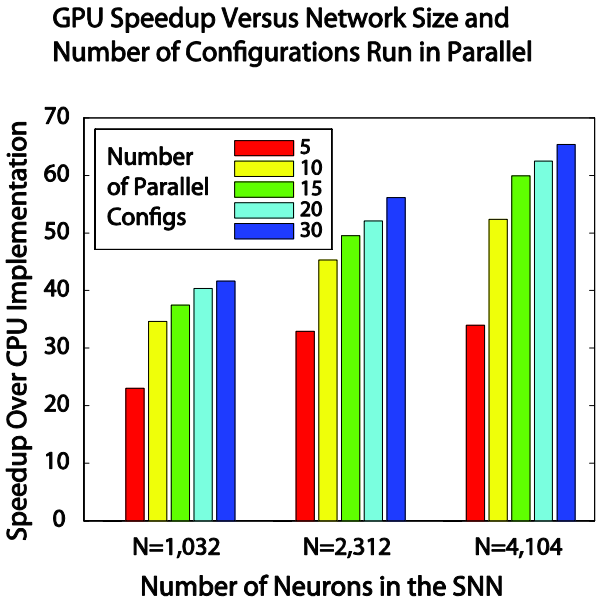

Fig. 3. Comparison of speedups of the parallel GPU CARLsim implementations over the single-threaded CPU CARLsim implementation. The different colored bars represent the number of configurations run in parallel for each SNN network size. The maximum speedup occurs at the largest network size (4,104 neurons) that ran 30 SNN configurations in parallel.

To further characterize the relationship between the speedup of the GPU implementation over the CPU implementation, both the SNN network size and number of SNN configurations were varied. The results are shown in Figure 3. Three SNNs with 4104, 2312, and 1032 neurons were run for 10 simulated minutes each. The number of configurations each SNN ran in parallel was varied from 5 to 30. There were significant speedups over the single threaded CPU implementation for all three network sizes when 5 or more SNNs were executed in parallel. The largest speedup was approximately (65x) and occurred when 30 configurations of 4,104-neuron SNN were simulated in parallel, as shown in Figure 3. The automated parameter tuning framework presented here efficiently searches SNN parameter spaces using EAs and performs fitness evaluations in parallel using GPUs. The tuning approach presented here uses these two techniques to build a powerful tool for researchers to design, construct, and test complex SNNs for neuromorphic applications.

IV. Summary and Conclusions

Neuromorphic applications have the potential to provide insight into brain function and create low-power, fault tolerant neuromorphic devices for use in sensory systems and cognitive computing architectures. However, due to the massive number of computing elements and the unstable neuronal and synaptic dynamics inherent in these models, the design and construction of neuromorphic applications is a difficult task. We presented the CARLsim SNN simulation environment, which features Izhikevich spiking neurons, three plasticity mechanisms, and GPU acceleration for use in the computational neuroscience and neuromorphic engineering communities. We also presented an automated parameter tuning framework which integrates CARLsim and an EA library to efficiently tune SNNs in parallel using GPU acceleration. We believe these software tools will accelerate the design and construction of large-scale neural models and neuromorphic applications, potentially offering practical solutions to currently unsolved real-world problems.

Acknowledgments

This work was supported by the Defense Advanced Research Projects Agency (DARPA) subcontract 801888-BS and by NSF Award IIS/RI-1302125. We thank Micah Richert for his work developing CARLsim 2.0 and Jayram Nageswaran for his work developing CARLsim 1.0 and the parameter tuning framework.

References

[1] J. M. Nageswaran, M. Richert, N. Dutt, and J. L. Krichmar, “Towards reverse engineering the brain: Modeling abstractions and simulation frameworks,” in *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, 2010, pp. 1–6.

[2] “NAE Grand challenges for engineering -www.engineeringchallenges.org.”

[3] J. Hasler and B. Marr, “Finding a roadmap to achieve large neuromorphic hardware systems,” *Front. Neuromorphic Eng.*, vol. 7, p. 118, 2013.

[4] G. Lynch, G. S. Lynch, and R. Granger, *Big Brain: the origins and future of human intelligence*. Macmillan, 2008.

[5] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[6] C. Koch, *Biophysics of computation: information processing in single neurons*. Oxford university press, 2004.

[7] E. R. Kandel, J. H. Schwartz, T. M. Jessell, and others, *Principles of neural science*, vol. 4. McGraw-Hill New York, 2000.

[8] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, “Silicon auditory processors as computer peripherals,” *Neural Netw. IEEE Trans. On*, vol. 4, no. 3, pp. 523–528, 1993.

[9] M. Mahowald, “An Analog VLSI System for Stereoscopic Vision,” 1994.

[10] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[11] S.-C. Liu and T. Delbruck, “Neuromorphic sensory systems,” *Curr. Opin. Neurobiol.*, vol. 20, no. 3, pp. 288–295, Jun. 2010.

[12] R. Silver, K. Boahen, S. Grillner, N. Kopell, and K. L. Olsen, “Neurotech for neuroscience: unifying concepts, organizing principles, and emerging tools,” *J. Neurosci.*, vol. 27, no. 44, pp. 11807–11819, 2007.

[13] S. K. Esser, A. Andreopoulus, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico, S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. A. Kusnitz, T. M. Wong, W. P. Risk, E. McQuinn, T. K. Nayak, R. Singh, and D. S. Modha, “Cognitive Computing Systems: Algorithms and Applications for Networks of Neurosynaptic Cores,” in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013.

[14] J. M. Cruz-Albrecht, T. Derosier, and N. Srinivasa, “A scalable neural chip with synaptic electronics using CMOS integrated memristors,” *Nanotechnology*, vol. 24, no. 38, p. 384011, 2013.

[15] T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier,

"Six networks on a universal neuromorphic computing substrate," *Front. Neuromorphic Eng.*, vol. 7, p. 11, 2013.

[16] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 1947–1950.

[17] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Trans. Comput.*, vol. 99, no. PrePrints, 2012.

[18] S. Davies, F. Galluppi, A. D. Rast, and S. B. Furber, "A forecast-based STDP rule suitable for neuromorphic implementation," *Neural Netw.*, vol. 32, pp. 3–14, Aug. 2012.

[19] S. Moradi and G. Indiveri, "An Event-Based Neural Network Architecture With an Asynchronous Programmable Synaptic Memory," *IEEE Trans. Biomed. Circuits Syst.*, vol. Early Access Online, 2013.

[20] D. Goodman and R. Brette, "Brian: a simulator for spiking neural networks in Python," *Front. Neuroinformatics*, vol. 2, p. 5, 2008.

[21] M.-O. Gewaltig and M. Diesmann, "NEST (NEural Simulation Tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[22] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A Large-Scale Model of the Functioning Brain," *Science*, vol. 338, no. 6111, pp. 1202–1205, Nov. 2012.

[23] C. M. Thibeault, "Computational Neuroscience: Theory, Development and Applications in Modeling The Basal Ganglia," Ph.D., University of Nevada, Reno, United States -- Nevada, 2012.

[24] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: a scalable simulator for an architecture for cognitive computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Los Alamitos, CA, USA, 2012, pp. 54:1–54:11.

[25] M. L. Hines and N. T. Carnevale, "The NEURON simulation environment," *Neural Comput.*, vol. 9, no. 6, pp. 1179–1209, 1997.

[26] J. M. Bower, D. Beeman, and A. M. Wylde, *The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System*. Telos Santa Clara, Calif, 1998.

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

[28] R. Ananthanarayanan and D. S. Modha, "Anatomy of a cortical simulator," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2007, pp. 3:1–3:12.

[29] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, "The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009, pp. 63:1–63:12.

[30] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," in *Computer graphics forum*, 2007, vol. 26, pp. 80–113.

[31] J. Baladron, D. Fasoli, and O. Faugeras, "Three Applications of GPU Computing in Neuroscience," *Comput. Sci. Eng.*, vol. 14, no. 3, pp. 40–47, Jun. 2012.

[32] D. Yudanov, M. Shaaban, R. Melton, and L. Reznik, "GPU-based simulation of spiking neural networks with real-time performance & high accuracy," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, 2010, pp. 1–8.

[33] A. K. Fidjeland, E. B. Roesch, M. P. Shanahan, and W. Luk, "NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, 2009, pp. 137–144.

[34] V. K. Pallipuram, M. C. Smith, N. Raut, and X. Ren, "Exploring Multi-level Parallelism for Large-Scale Spiking Neural Networks," 2012.

[35] T. Nowotny, "Flexible neuronal network simulation framework using code generation for NVidia(R) CUDATM," *BMC Neurosci.*, vol. 12, no. Suppl 1, p. P239, 2011.

[36] 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 Netw. Off. J. Int. Neural Netw. Soc.*, vol. 22, no. 5–6, pp. 791–800, Aug. 2009.

[37] M. Richert, J. M. Nageswaran, N. Dutt, and J. L. Krichmar, "An efficient simulation environment for modeling large-scale cortical processing," *Front. Neuroinformatics*, vol. 5, no. 19, 2011.

[38] "NeoCortical Simulator - http://www.cse.unr.edu/brain/ncs." .

[39] R. Brette and D. F. M. Goodman, "Simulating spiking neural networks on GPU," *Netw.-Comput. Neural Syst.*, vol. 23, no. 4, pp. 167–182, 2012.

[40] A. P. Davison, D. Bruderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "PyNN: A Common Interface for Neuronal Network Simulators," *Front. Neuroinformatics*, vol. 2, Jan. 2009.

[41] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569 – 1572, Nov. 2003.

[42] P. Dayan and L. F. Abbott, *Theoretical neuroscience*, vol. 31. MIT press Cambridge, MA, 2001.

[43] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput. Biol.*, vol. 3, no. 2, p. e31, 2007.

[44] S. Byrnes, A. N. Burkitt, D. B. Grayden, and H. Meffin, "Learning a sparse code for temporal sequences using STDP and sequence compression," *Neural Comput.*, vol. 23, no. 10, pp. 2567–2598, Oct. 2011.

[45] R. Brette, "Computing with neural synchrony," *PLoS Comput. Biol.*, vol. 8, no. 6, p. e1002561, Jun. 2012.

[46] L. F. Abbott and S. B. Nelson, "Synaptic plasticity: taming the beast," *Nat. Neurosci.*, vol. 3, no. 11, pp. 1178–1183, Nov. 2000.

[47] G. Turrigiano, "Homeostatic synaptic plasticity: local and global mechanisms for stabilizing neuronal function," *Cold Spring Harb. Perspect. Biol.*, vol. 4, no. 1, Jan. 2012.

[48] K. D. Carlson, M. Richert, N. Dutt, and J. L. Krichmar, "Biologically Plausible Models of Homeostasis and STDP: Stability and Learning in Spiking Neural Networks," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Dallas, Texas, 2013.

[49] "NVIDIA's Next Generation CUDA Compute Architecture: Fermi," NVIDIA Corp., white paper, 2009.

[50] M. C. Avery, D. A. Nitz, and J. L. Krichmar, "Simulation of cholinergic and noradrenergic modulation of behavior in uncertain environments," *Front. Comput. Neurosci.*, vol. 6, p. 5, 2012.

[51] C. J. Honey, R. Kötter, M. Breakspear, and O. Sporns, "Network structure of cerebral cortex shapes functional connectivity on multiple time scales," *Proc. Natl. Acad. Sci.*, vol. 104, no. 24, pp. 10240–10245, Jun. 2007.

[52] M. Beyeler, N. D. Dutt, and J. L. Krichmar, "Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule," *Neural Netw. Off. J. Int. Neural Netw. Soc.*, vol. 48C, pp. 109–124, Aug. 2013.

[53] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[54] J. M. Brader, W. Senn, and S. Fusi, "Learning Real-World Stimuli in a Neural Network with Spike-Driven Synaptic Dynamics," *Neural Comput.*, vol. 19, no. 11, pp. 2881–2912, Sep. 2007.

[55] P. L. Smith and R. Ratcliff, "Psychology and neurobiology of simple decisions," *Trends Neurosci.*, vol. 27, no. 3, pp. 161–168, Mar. 2004.

[56] D. Querlioz, O. Bichler, and C. Gamrat, "Simulation of a memristor-based spiking neural network immune to device variations," in *The 2011 International Joint Conference on Neural Networks (IJCNN)*, 2011, pp. 1775–1781.

[57] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 105, no. 9, pp. 3593–3598, Mar. 2008.

[58] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer, "Evolving objects: a general purpose evolutionary computation library," in *Artficial Evolution*, vol. 2310, P. Collet, C. Fonlupt, J. K. Hao, E. Lutton, and M. Schoenauer, Eds. Berlin: Springer-Verlag Berlin, 2002, pp. 231–242.