Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 61 (2015) 322 – 327

Procedia Computer Science

Complex Adaptive Systems, Publication 5
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2015-San Jose, CA

Simulation Tool for Asynchronous Cortical Streams (STACS): Interfacing with Spiking Neural Networks

Felix Wang[a]*

[a]*The University of Illinois at Urbana-Champaign, Beckman Institute, 405 N Mathews Ave, Urbana, IL 61801, USA*

Abstract

We present a Simulation Tool for Asynchronous Cortical Streams (STACS) for studying spiking neural networks exhibiting adaptation in a closed-loop system. The goal is to develop a more complete understanding of the emergent behaviors at the network level, and attention is given to methods of analysis at this scale. In particular, STACS facilitates the development of network level metrics of spiking activity. At the same time, emphasis is placed on biologically faithful models of spiking and plasticity with respect to the underlying neural substrate. The essential component, however, is the ability of the neural system in interfacing with the environment. This is because behaviors such as learning and adaptation are inherently closed-loop processes that involve the interaction between an intelligent agent and its environment, here, embodied cognition. To this end, STACS utilizes a portable communication protocol, YARP, for interfacing and interacting with a wide range of external devices, both sensory and motor, as well as the ability to create user-defined methods. In doing so, we may capture and respond to real world input to a neural network, simulating experimentation of live cortical cultures such as on multielectrode arrays.

Peer-review under responsibility of scientific committee of Missouri University of Science and Technology

Keywords: simulation; spiking; neural network; asynchronous communication; closed-loop; embodied cognition; biological models

1. Introduction

Learning is a process that occurs over a lifetime. Yet, the mechanics of how it is realized in the brain leaves plenty to be explored. At the lowest level, learning occurs through the strengthening of the synaptic connections between neurons. Of particular interest to us are neural networks and their evolution in response to stimuli. There are many methods for studying the learning process in a neural network, multielectrode arrays having been of particular

* Corresponding author. Tel.: +1-785-317-2728
E-mail address: fywang2@illinois.edu

Peer-review under responsibility of scientific committee of Missouri University of Science and Technology
doi:10.1016/j.procs.2015.09.149

interest in the recent years. These devices allow for recording and stimulation over a network for an extended period of time, providing the ability to monitor the evolution of a network. Extending this to simulation allows for more in depth analysis than would be possible with the constraints of a live culture. We present a Simulation Tool for Asynchronous Cortical Streams (STACS), which aims to address several key concerns with respect to learning in a neural network. Most importantly, we provide a framework for embodiment and feedback through interfacing with the external environment. Computationally, the simulator is designed from the ground up to run in parallel, taking into consideration the unique communication patterns of a highly connected, spiking neural network.

1.1. Feedback in learning and memory

One of the most crucial aspects of the learning process that is typically overlooked in traditional methods from an engineering perspective is the motor-sensory feedback loop that enables grounding of the incoming signals [1]. In the biological system, neurons are never found in isolation to stimuli and some form of control, even in the most basic of organisms. Thus, experiments targeted at exploring the learning process necessitate the inclusion of such a feedback loop. For cell cultures plated on an MEA, this is straightforward as the electrodes may be programmed to provide stimulation to the network in addition to recording. With respect to simulated networks, there are several simulation tools as well as a handful of domain specific languages that exist to facilitate computational neuroscience research [2]. However, closing the loop on simulated networks is less studied as many tools fail to provide a way for the network to interact with a dynamic environment. In place of interfacing with real-world devices, various current or spike generators are used to provide predefined stimuli to the network.

1.2. Neural Simulation

Perhaps the most well known neural simulation tool is NEURON, which provides an environment for implementing biologically realistic cellular models of neurons and networks thereof [3]. While the tool provides graphical interfaces that are rather useful for instruction at small scale, because less emphasis is placed at the population level, the construction of complex networks consisting of many neurons quickly becomes intractable. This limits its ability to adequately describe networks, both from a constructional standpoint and in terms of analysis. To some extent, these limitations are addressed by the introduction of domain specific languages, such as NeuroML, that map onto a number of simulation backends [4]. More recently, simulation tools such as NEST have gained popularity due to their ability to efficiently simulate large networks at the expense of biological precision, concentrating on the dynamics of a large population as opposed to individual neurons [5]. GPGPUs have also been leveraged in newer tools such as HRLSim to achieve real-time simulation of large networks [6].

With respect to learning in a large spiking neural network, there are some shortcomings in the existing set of simulation tools. Although each simulator performs well for each of their specific focuses, as mentioned above, none are capable of adequately simulating large networks interacting in a realistic setting for embodiment. A few key features that are required of this task include the ability to store state data across simulation runs for continuous simulation, models of the spiking neuron and synapse which require interoperability in their dynamics, hardware support for interaction with the world in a closed-loop manner, and the ability to implement modules defined over neurons as a collective for population level analysis. In the development of STACS, in addition to taking the above items into consideration, we also pay attention to design strategies required for efficient parallelization.

2. Network representation

2.1. Network structure

In the construction of the neural network, we treat the system as a directed graph $G(V,E)$, where the spiking neurons form the vertex set V and the synaptic connections form the edge set E. The standard method for storing the edges of a graph is through an adjacency matrix $\mathbf{A}$ of size $n \times n$ where a non-zero entry at a_{ij} corresponds to the existence of an edge $e_1 : v_i \rightarrow v_j$. For a realistic network of 10^5 to 10^6 neurons with 10^3 to 10^4 outgoing synaptic connections, the overall connectivity is no greater than roughly 10%. This implies a sparse graph structure where the

majority of the entries in the adjacency matrix are zero. Due to its sparsity as well as the parallel nature of the problem, we chose the distributed compressed sparse row (dCSR) format for storage of the network topology, with each file containing the graph information for a partition of V. Here, each vertex is given an index corresponding to its line location in a dCSR file, which contains the edge list for a given vertex. Because of the parallel formulation, we have an additional vertex distribution file that is used to offset the indices of each partition.

2.2. Parallel partitioning

In order to keep communication between parallel processes at a minimum, the partitioning of V is chosen to minimize the number of edges across the partition boundaries. This is computed using the *parmetis* algorithm, which utilizes a coarsening and uncoarsening scheme [7]. The graph G_0 is transformed into a sequence of smaller graphs $G_1, G_2, \ldots, G_m$ such that the number of vertices are progressively reduced $|V_0| > |V_1| > \ldots > |V_m|$ using a heavy-edge-matching heuristic. The graph $G_m(V_m, E_m)$ is small enough such that the optimal partition may be computed combinatorially. This partitioning is then projected back onto the graphs $G_{m-1}, G_{m-2}, \ldots, G_0$ where at each stage of the projection, refinements are performed to include the reintroduced vertices. We may additionally bootstrap the partitioning process using information about the spatial distribution of vertices found from the neural network.

2.3. Substrate models

The dynamics of the network find their basis in neural substrate. In terms of their representation, the aim is to capture both the data-driven and event-driven state dynamics of the physical process. A natural mathematical formulation of this may be found in stochastic differential equations [8], and are generalized in the form of eq. (1).

$$dx = f(x)dt + \sum_{i=1}^{n} g_i(x)dN_i \tag{1}$$

The state, x, evolves continuously in time according to a drift process, $f(\cdot)$, as well as stochastically according to diffusion processes, $g_i(\cdot)$, indexed by the event type. Although the diffusion is commonly modeled by a Wiener process, giving rise to Brownian motion, we may model the stochastic behavior as a counting process, $N(t)$, that experiences jumps, dN, corresponding to event times.

Because the computation requirements of the neuron and synapse models fall between the drift and diffusion processes rather cleanly, we provide separate abstract classes tailored for each type of process, respectively. For modeling drift, we provide a time step for the dynamics to evolve over, dt, and for modeling the diffusion, we provide a timestamp of when the event occurred, dN. In terms of storage, the state data for the neurons and synapses are stored separately from the network structure, maintaining the same order. Experimentally, this enables the comparison between the effects of different substrate models on the network behavior. Due to the inter-dependence between neuron and synapse models, synaptic data is stored corresponding to the incoming edges for a vertex.

3. Simulation engine

Fundamentally, the architecture of the simulation engine is comprised of only a few key elements: the parallel infrastructure that embeds the network structure, the time-driven computation that evolves the network state, and the event-based protocol that determines the network communication both within the network and external to the system via asynchronous data streams. These elements provide the framework on top of which a multitude of network configurations may reside, and are elaborated below in turn.

3.1. Parallel infrastructure

The parallelization of the network by partitioning to a collection of parallel objects is realized by the Charm++ parallel programming system [9]. Charm++ establishes a programming paradigm of over-decomposition of an

application into logical work and data units called *chares*. Scheduling and execution is managed through an adaptive runtime system based on message-driven, migratable objects. That is, the parallel objects defined by the application may move across processors during the lifetime of the execution, enabling dynamic load balancing for more efficient utilization of resources. By allowing several parallel objects to share the same processor, Charm++ is able to mask network latency by encouraging the overlap of computation with communication.

The runtime system also manages the communication between parallel objects, which is asynchronous in nature. In particular, Charm++ employs a communication model through remote method invocations where computation occurs only when the required data dependencies are received. Using the *CkMulticast* libraries, Charm++ is capable of generating spanning trees that facilitate methods for broadcasting and reducing data to and from subsets of chares. Multicast has been shown to be optimal for neural architectures [10]. Moreover, the modular structure encourages the development of collective operations, such as network metrics or stimulation, on top of the parallel application.

3.2. Time-driven computation

With respect to the computation of network dynamics, we adopt a time-driven approach borrowing elements from NEST [11]. As described in section 2.3 we model the neural substrate generally as stochastic differential equations. We treat the occurrence of spike events as a counting process with a delay time equivalent to the axonal delay between the pre-synaptic neuron and post-synaptic neuron. Because the minimum axonal delay of the network as a whole corresponds to the minimum amount of time before a spike generated by one neuron may affect any other neuron, we are able to decouple the computation of the network accordingly.

Specifically, we are able to evolve the drift process of the neuron state independently of any communication within this minimum delay time. This enables the use of model specific integration schemes that allow for increased flexibility or precision of computation as necessary. In this way, the simulation of the network is able to evolve along a coarse time grid while still retaining precision of event times. With respect to performance, this also has an advantage over a global time step method where increased precision of events comes at the cost of increased computation required of the entire network. The communication overhead is also reduced as the number of synchronization points are minimized.

Whereas the neuron dynamics evolve alongside a potentially variable time step, the synaptic dynamics are computed at discrete times coinciding with the event timestamp. By evaluating these dynamics only at discrete times as opposed to integrating a large coupled system of differential equations, we lower the number of floating-point operations per second required per unit of simulated time by several orders of magnitude, and we reduce the amount of memory loads and stores by a similar amount. This is accomplished in an online fashion by representing the spiking history in terms of their traces at the synapse.

3.3. Event-based communication

Much of the computation that occurs in a spiking neural network rests on top of events, namely the spiking of a neuron. Although these events happen relatively infrequently when compared to the resting state of a neuron, their precise timings are important to the network level behaviors such as phase synchrony or polychronization [12, 13]. Timing is also important at the level of the neural substrate, for example, in the synapse where the timing between the pre-synaptic and post-synaptic spikes determines changes in strengthening or weakening.

The natural representation for handling this type of dynamic is through the use of event lists. As new events arrive on the network partition, they are placed on the list associated with its network element in order of their timestamps. However, implementing a single event list for a given element such as a neuron is impractical as the total number of events will be greater than the number to be processed in any given iteration. To accommodate this, we employ the use of calendar queues such that each ‘day’ is chosen to be the length of an iteration time interval and the number of days in a ‘year’ is chosen such that we may account for the maximum axonal delay without overlapping [14]. Any non-standard event delays that are greater than this may be placed in an additional buffer to be distributed at the beginning of each new year.

To reduce the amount of global communication that occurs as the parallel system scales, we employ a neighbor-only communication method whereby network partitions only communicate event data to partitions for which there

exists an edge between them. This local exchange of data falls in line with the partitioning scheme described in section 2.2. By using the broadcast libraries provided by the Charm++ runtime system as described in section 3.1, the communication overhead is further reduced.

3.4. Asynchronous streams

As an extension to the event-based communication protocols, the simulation tool also admits asynchronous data streams external to the spiking neural network. Unlike the network events where we must wait on neighboring partitions, however, there is no explicit data dependency on external information. In other words, the timing between the network and the environment is loosely coupled. Although the entry method invocations for Charm++ provide a good platform for asynchronous communication within the application, there is no way to natively invoke methods from outside of the application. In order to transmit external events to the simulation tool as the network evolves, we require a separate communication protocol that is capable of inter-process communication.

The YARP project, Yet Another Robot Platform, provides such a communication protocol [15]. The target application of YARP is in the infrastructure development for humanoid robotics where hardware operations include but are not limited to: audio, video, and tactile sensory, fine motor control, and the data processing in between. In line with the goal of studying neural networks in a closed-loop environment, the communication model supported by YARP is designed and built for closed-loop systems composed of many processes on many processors. In terms of stimulating the neural network, we use the YARP libraries to provide callback functions that may be triggered upon receiving data through remote procedure calls. These callback functions are responsible for transcribing the message into network events. In terms of recording from or driving control using the neural network, we simply reverse the process by defining where and how the network should send data generated through computation.

4. Network stimulation

Currently, STACS allows stimulation to the network in the form of applied current pulses. These may be applied to the entire network simultaneously, or, more realistically, to a localized region of the network. In particular, the researcher is capable of specifying a spherical volume by providing a center spatial coordinate along with a radius. The goal is to mimic how stimulation is typically applied to an electrode on an MEA. Stimulation to the simulated network is much more versatile, however, as we are not limited to specific locations in the network. Furthermore, each individual current pulse is defined by its amplitude, onset time offset, and duration, and may be combined to provide arbitrary signals. That is, we may provide arbitrary stimulation to arbitrary regions of the network. An example is provided below.

To emulate the scale of neural networks that are grown on MEAs, we construct a network consisting of 10,000 neurons distributed randomly over a circle of radius $800\mu m$, or area approximately $2mm^2$. Connectivity is spatially dependent, and the connection probability between any two neurons as a function of their separation distance is modeled according to [16]. This gives approximately 1,000 synaptic connections per neuron for an overall network connectivity of 10%. With respect to the neural substrate, we employ biologically faithful models. For the neuron, we use the phenomenological model developed by Izhikevich [17]. At the synapse, synaptic transmission is handled using the typical conductance-based model, and synaptic plasticity admits voltage dependencies in addition to the standard spike-timing-dependent component [18]. We implement online, event-based methods for both these models. Axonal delay between neurons is distributed between $1ms$ and $20ms$, and like the spatially dependent connectivity, is also determined as a function of separation distance. For the simulation, we generate a 16-way partition of the network into parallel objects according to section 2.2, preserving the spatial locality of the network.

For stimulation, we provide a simple square wave of duration $25ms$ with a period of $5ms$ (5 pulses) to a region first centered on the North-Eastern edge and then to the South-Western edge of our network (Fig. 1). The square wave has a duty cycle of 40% and an amplitude of $15\mu A/cm^2$, and the centers of the stimuli are located at ($500\mu m$, $500\mu m$) and (-$500\mu m$, -$500\mu m$) with a radius of $200\mu m$. To study the effects of the stimulation on the network, we observe the spike raster. From this, we immediately see an increase in spiking density in the regions of the network where the stimuli were applied, as well as a change in network activity from asynchronous firing to exhibiting slow oscillations in spiking density.

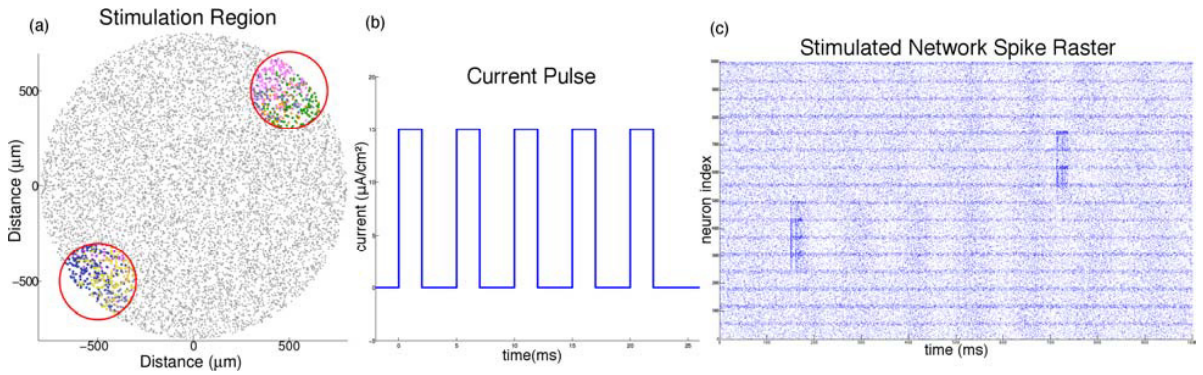

Fig. 1. (a) target stimulation regions and affected neurons; (b) applied square wave current pulse; (c) network spike raster for one second of biological time with multiple stimulation (as described in text)

5. Conclusion and future work

We have presented the construction of a simulation tool for the study of biologically faithful networks of spiking neurons with the ability to interface with the environment, STACS. Currently, there is significant room for expansion with respect to the implementation of additional modules that would lie on top of the core functionality and architecture. These include a wider variety of neuron and synapse models forming the neural substrate, methods for transcribing audio and video input into stimulation events onto the network, and the development of a number of metrics that operate at the network level such as detection of polychronous groups or the computation of phase synchrony. Support for birth and death processes with respect to both neurons and synaptic connections would also be beneficial for studies of structural plasticity at longer time scales, such as in the case of neurogenesis.

References

1. Wiener N. Cybernetics, or Control and Communication in the Animal and Machine. Cambridge: MIT Press; 1948.
2. Brette R et al. Simulation of networks of spiking neurons: A review of tools and strategies. J. Computational Neuroscience. 2007; 23:349-98.
3. Hines ML, Carnevale NT. The NEURON Book. Cambridge University Press; 1997.
4. Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO et al. NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. PLoS Computational Biology, 2010 June; 6.
5. Gewaltig MO, Diesmann M. NEST (NEural Simulation Tool). Scholarpedia, 2007; 2(4):1430.
6. Minkovich K, Thibeault CM, O'Brien MJ, Nogin A, Cho Y, Srinivasa N. HRLSim: A High Performance Spiking Neural Network Simulator for GPGPU Clusters. IEEE Trans. Neural Networks and Learning Systems, 2014; 25(2).
7. Karypis G, Kumar V. Parallel multilevel k-way partitioning scheme for irregular graphs. SIAM Review, 1999; 41(2):278-300.
8. Oksendal B. Stochastic Differential Equations. 5th ed. Springer-Verlag; 2002.
9. Kale LV, Krishnan S. Charm++: a portable concurrent object oriented system based on C++. In proc.: Object-oriented programming systems, languages, and applications, 1993 Oct; 28(10):91-108.
10. Vainbrand D, Ginosar R, Scalable network-on-chip architecture for configurable neural networks. Microprocessors and Microsystems, 2011; 35:152-66.
11. Plesser HE, Eppler JM, Morrison A, Diesmann M, Gewaltig MO. Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In: Euro-Par 2007 Parallel Processing. Springer-Verlag Heidelberg; 2007; 4641:672-81.
12. Engel AK, Fries P, Singer W. Dynamic predictions: oscillations and synchrony in top-down processing. Nature Reviews Neuroscience, 2001; 2:704–16.
13. Izhikevich EM. Polychronization: Computation with spikes. Neural Computation, 2006; 18:245-82.
14. Brown R. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. Communications of the ACM, 1998; 31(10):1220-7.
15. Metta G, Fitzpatrick P, Natale L. YARP: Yet Another Robot Platform. Int. J. Advanced Robotic Systems, 2006; 3(1)43-8.
16. Hellwig B. A quantitative analysis of the local connectivity between pyramidal neurons in layers 2/3 of the rat visual cortex. Biological Cybernetics, 2000; 82:111-21.
17. Izhikevich EM. Simple model of spiking neurons. IEEE Trans. Neural Networks, 2005; 94:3637-42.
18. Clopath C, Gerstner W. Voltage and spike timing interact in STDP - a unified model. Frontiers in Synaptic Neuroscience, 2010; 2:1–11.