Parallel Event-Driven Neural Network Simulations Using the Hodgkin-Huxley Neuron Model

Collin J. Lobb[1], Zenas Chao[2], Richard M. Fujimoto[1], Steve M. Potter[2]

[1] *College of Computing*
Georgia Institute of Technology
{collin,fujimoto)@cc.gatech.edu

[2] *Laboratory for Neuroengineering*
Georgia Institute of Technology
zenas@neuro.gatech.edu
stpotter@ece.gatech.edu

Abstract

Neural systems are composed of a large number of highly-connected neurons and are widely simulated within the neurological community. In this paper, we examine the application of parallel discrete event simulation techniques to networks of a complex model called the Hodgkin-Huxley neuron[1]. We describe the conversion of this model into an event-driven simulation, a technique that offers the potential of much greater performance in parallel and distributed simulations compared to time-stepped techniques. We report results of an initial set of experiments conducted to determine the feasibility of this parallel event-driven Hodgkin-Huxley model and analyze its viability for large-scale neural simulations.

1. Introduction

Much work in parallel and distributed simulation focuses on synchronizing time in hundreds or thousands of entities. Researchers have been able to create scalable algorithms that efficiently coordinate the temporal activities of these entities. However, there are many domains in which one must synchronize a massive number of entities. Many of these domains have not been investigated. In this paper, we will examine a relatively new domain in parallel and distributed simulation whose scale could eventually reach billions of entities making trillions of connections: the nervous system.

In parallel and distributed simulation, there are two main techniques for incrementing time: a time-stepped approach and an event-driven approach. Event-driven simulations have been shown to offer better performance. Thus, many efforts are devoted to creating suitable event-driven models. This allows one to abstract out the partitioning and synchronizing algorithms. In this paper, we will discuss the transformation of the Hodgkin-Huxley model [1] of a neuron, hence referred to as the "HH model", into an event-driven process and discuss some of our preliminary findings in simulating a large scalable number of these neurons.

We are investigating how network scale and interconnectivity affect the emergence of a large synchronous burst of neural activity called a "superburst" [2]. In order to investigate these network-level questions, we need the ability to run large network simulations. The computational requirements for such networks are much greater than those currently available on typical desktop computers. The effective use of parallel and distributed computation can help address these ever-increasing computational demands.

This paper is organized into eight sections. In section 2, we provide a brief introduction to neural systems. In section 3, we discuss previous work in simulating these neuron models. In section 4, we describe how we converted the continuous HH model into a discrete event-driven simulation model. In section 5, we discuss the implementation of the HH model using a parallel discrete event simulation engine called μsik [3] and the methods by which we create networks of these HH neurons. In section 6, we describe efforts to verify this model. In section 7, we discuss some initial scalability experiments on this event-driven HH model. Finally in section 8, we provide some concluding thoughts on these results and how one might go about improving them.

2. Neural Systems

Like most biological systems, neural systems are typically comprised of cells and the fluid that they inhabit (extracellular medium). These cells are able to communicate with one another. For many cells, these communication mechanisms involve the movement of ions, which are atoms with an electric charge. Most cells, including neurons, have a charged membrane but only a few types of cells (e.g. neurons and cardiac cells) are

Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS'05)
1087-4097/05 $20.00 © 2005 **IEEE**

considered to be "electrically excitable" and are able to produce "electrical" responses that can be transmitted from cell to cell. It is important to remember that these "electrical" responses are in the form of ion flows, not flows of electrons.

A neuron (depicted below in Figure 1) is an electrically-excitable cell found in the nervous system. A neuron lives in a medium similar to seawater, which contains various ions. The *soma* is the cell body and has a charge due to the presence of these *ions*. The soma is surrounded by a selectively permeable *cell membrane*. The soma of a "pre-synaptic" neuron can produce an electrical response called an "action potential" that is transmitted down its *axon* to its *synaptic terminals* with some transmission delay. This "output" charge is then received from the *synaptic cleft* by the "post-synaptic" neuron via its *dendrites*.

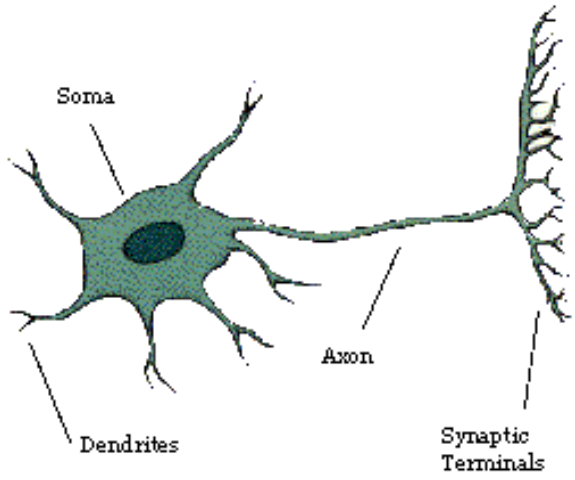

Figure 1: A schematized neuron

In the absence of any outside forces (i.e., at equilibrium), there will be differences in the amount of each ion on either side of the neuron's membrane. When we account for all of the forces due to (a) differences in concentration of each type of ion (i.e., "diffusion") and (b) the differences in the amount of positive and negative ions on each side (i.e., "electrostatic pressure"), we find that typically a neuron at equilibrium has a negative "membrane potential" (the charge difference on the neuron's membrane) in the range of -65 mV to -90 mV.

In the presence of outside forces, e.g., due to an injected current or the reception of a transmitted electrical response from another neuron, ions will flow into or out of the neuron through pores in the membrane called "ion channels". One can think of these ion channels as small gates that are each operated by a gatekeeper according to the laws of chemistry and physics. A gatekeeper will only allow the flow of certain types of ions at certain times at certain rates and in certain conditions. The membrane potential of the neuron is governed by a combination of how open each of these gates is and thus what ion flow is permitted across the membrane (i.e. its permeability). This "combination" is modeled by a set of differential equations that many experimenters such as Hodgkin and Huxley have quantified.

One of the most important processes that a neuron undergoes is the action potential, shown in Figure 2 (henceforth called an "AP"). It is characterized electrically as a very large increase in the membrane potential (on the order of 100mV) followed by a sharp decrease in potential that sends the membrane potential below its equilibrium value, followed by a slow return to equilibrium. This also defines the shape of the AP. Consider this in terms of gates and gatekeepers. The gatekeepers manipulate these gates, much like an operator controlling floodgates at a reservoir, trying to maintain equilibrium. For the neuron, there are two main types of gatekeepers: sodium gatekeepers that permit sodium flow into the neuron, and potassium gatekeepers that permit potassium flow out of the cell. Since both sodium and potassium ions are positive, all of these gatekeepers must work together in order to maintain equilibrium.

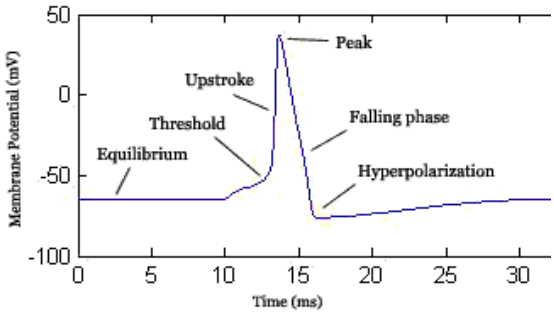

Figure 2: An action potential generated from a HH neuron in our simulation

In response to a sufficiently large positive input (e.g., injecting current into the cell) the sodium ion gatekeepers begin to open their gates to let sodium ions into the cell. This starts to increase the membrane potential, which in turn tells the sodium gatekeepers to let more sodium ions in. Meanwhile, there is a large number of potassium gatekeepers that prior to this sodium influx have been inactive and kept their gates closed. One of their jobs is to monitor the membrane potential and open up their gates when the membrane potential becomes too high. As the sodium level (and thus the membrane potential) begins to rise these potassium gatekeepers begin to take notice and open their gates to let potassium out of the cell. This would lower the membrane potential and return the cell to equilibrium. However, these potassium gatekeepers (letting potassium out) operate more slowly than the sodium gatekeepers (letting sodium in) so the membrane

potential keeps rising. When the sodium ion gatekeepers let in enough sodium ions to raise the membrane potential above some threshold, a flood of additional sodium ions rush into the cell and quickly raise the membrane potential (the "upstroke"). The sodium gatekeepers try to quickly close the gates to control the flood of incoming sodium ions and are only able to effectively do so after letting in a large number of sodium ions. The potassium gatekeepers try to compensate by opening their gates further. Through the combination of the sodium gates closing and the potassium gates opening the large influx of positive ions is controlled and the membrane potential stops rising (the "peak" of the AP). However, at this point much potassium efflux is being allowed through the open potassium gates and the membrane potential begins to drop dramatically (the "falling phase"). The potassium gatekeepers slowly begin to notice that the membrane potential is dropping rapidly (this is their other responsibility) and that they need to close their gates. By the time they close their gates, the membrane potential has shot well below its equilibrium value ("hyperpolarization"). The sodium gatekeepers, however, quickly recover and begin to open their gates up just enough to increase the membrane potential to its equilibrium value. It is this process that experimenters such as Hodgkin and Huxley have mathematically modeled with a set of differential equations.

Neuroscientists have created a wide range of neuronal models but are split concerning how to deal with the action potential. One class of models called integrate-and-fire (henceforth called "IF") or "spiking" neuron models [4, 5], does not model the AP itself. These models effectively model the relationship between the neuron and its responses to various input currents but when this "threshold" is crossed they assume that a "spike" is generated and the neuron is instantaneously reset to some reset voltage. Think of this spike as taking the action potential from Figure 2 and compressing it from both ends until it approaches a vertical line. They are unable to capture the shape of the AP or accurately capture how the neuron responds to input during the action potential and during recovery. Despite being biologically unrealistic, the differential equations that govern their behavior are much simpler, easier to analyze, and are solvable. Models such as the HH model that are more biologically realistic are typically governed by a more complex set of differential equations that cannot be solved explicitly and therefore must be approximated using numerical analysis. To put it more quantitatively, 1 ms. of simulation time of the IF model requires around 5 FLOPS while the HH requires 1200 FLOPS [6]. For reasons such as these, large network simulations have been primarily limited to IF models. How might we be able to efficiently run large simulations of these more complex models? This is the driving question behind this work.

3. Related Work

Since IF models are primarily used in large scale network models, many iteratively-stepped parallel simulators have been created to efficiently simulate them: NEOSIM[13], the "Neocortex" simulator[11], SPIKELAB[14], SPIKENET [15] and Brettle and Niebur [16] to mention a few.

Using an iterative approach, one would iterate the governing differential equations for each dt and determine if its threshold has been crossed. If so, it would fire a spike. However, since the governing equations of the IF model can be solved, one can determine when the next time the IF neuron will fire. One could just keep an estimate of this "firing time" and update it if any more input is received. This is an event-driven approach and can be used to dramatically speed up the simulation[10, 17-19].

There are several existing parallel simulators capable of running networks of HH neurons: PNEURON, a parallel version of NEURON [7], PGENESIS [8], PARALLEL NEUROSYS[9], and the "Neocortex" simulator [10, 11]. Thomas [12] completed some work on running HH networks, which parallelizes time-stepped subsystems through waveform relaxation. As far as the authors are aware, only the "Neocortex" simulator has looked into partitioning algorithms. All of these simulators use an iterative approach. This paper explores the use of an event-driven framework on such HH networks.

4. Model Design

Muresan [10] states that ideally a large-scale neural simulator would support a high degree of interoperability in both the types of neurons that can be used and the techniques that each neuron employs. It would allow for both "implicit" and "explicit" synaptic representations. An explicit representation is one that says that neuron A connects to neuron B; in contrast, an implicit representation is one where neuron A connects to some other neuron as defined by some "rule of connectivity" [10]. This is in addition to the stereotypical desires of high accuracy, computational efficiency (speed), and minimal memory requirements.

Two major, interrelated aspects that one must consider when designing a parallel, event-driven HH simulation are (a) how to define the distinct entities making up the network and (b) the time intervals inherent in the model that can be used to extract lookahead. It is well known that good lookahead properties are desirable for optimistic synchronization mechanisms and essential for conservative techniques. The means by which network entities are defined will determine the types of events that are exchanged between entities, greatly affecting the

partitioning strategy, and the types of optimizations that might be later applied.

One can define the neural system as being composed of at least four components: the soma, the axon, the synapse (i.e., the synaptic terminals, the dendrites and the synaptic cleft) and the extracellular medium (environment). One important question is how one would represent each of these neural components in such a way that performance is maximized for the types of experiments that will be performed. There are two things that we should keep in mind as we go about defining these components. First, for scalability purposes, a fairly compact representation for these components is desirable as memory requirements will be large. It is also important that we keep in mind that there are many simplifying assumptions that one makes in order to investigate network behavior. These can be made for a general class of neural systems or can be made purely on the nature of the experiment at hand.

Foremost, we describe several general simplifications that one might make when looking at network-level questions. The first simplification is that we are not modeling changes in the environment. We assume isopotentiality in the neuron, i.e., space is uniform and does not have to be explicitly considered in our equations. We are not modeling the interactions of the AP as it travels down the axon (e.g. by cable equations). We characterize the axon purely in terms of the delay in which an AP is transmitted. This delay is typically much larger than the timesteps that are used in these types of simulations. This transmission is also one-way. These two facts suggest that the axon would provide a good means of separating the neuron components from the synapse components.

Other important considerations arise when we consider a model for our synapses. The interactions between the post-synaptic neuron and the synapse itself are assumed to be instantaneous; i.e., the dendrites are not explicitly modeled. After an AP has been transmitted down the axon with some delay, it is considered to be instantly transmitted to the receiving neuron. Learning mechanisms inherent in a typical synaptic model (e.g., [20, 21]) describe instantaneous messages to be passed between the post-synaptic neuron and the synapse model. Unfortunately, this means that separating the neuron model from the synapse model would result in a zero-lookahead simulation. To avoid this and to improve the efficiency of interaction between these components, the representations of the neuron model and the synaptic model are combined into a single "synapse-neuron" simulation entity (i.e., are mapped to a single logical process), allowing instantaneous messages to be implemented as manipulations of shared state. However, combining these representations can be potentially disastrous if one has to replicate and coordinate multiple synaptic models. Because of our desire to obtain good lookahead, we wish to partition the network by splitting a neural connection by its axon and base our lookahead on the axonal transmission time. Fortunately, the standard synaptic models employed here are unidirectional, meaning that the synapse only affects the post-synaptic neuron and does not need to pass messages back to the pre-synaptic neuron. Therefore, we incorporate the synaptic model into the post-synaptic neuron's representation. The pre-synaptic neuron only needs to know to which synapses it should fire and what delays are involved in these firings. On the other hand, the post-synaptic neuron doesn't need to know this delay since it will receive this action potential as an event. The only replication of the synapse that is needed is the unique synaptic identifier.

An important consideration that arises from this combined representation is a hierarchical notion of variable scope. A neuron "owns" many synaptic representations. Thus, variables that are common to all of its synapses may be abstracted from the synaptic representation and put in the neuron representation. As this is a one-to-many relationship this can result in large memory savings. Computationally, a common means of speeding up various synaptic calculations (e.g. how much current is being received by a neuron at this moment) is to allow these calculations to be performed in batch for all of the synapses thus making the calculation more computational efficient (see [22] for an excellent example of this technique). One can extend this variable abstraction technique two more levels and say that state common to all neurons in a simulation process can be abstracted to that simulation process itself and that variables common to all simulation processes can become "constants" in the simulation. Such abstraction is a powerful means for limiting memory consumption.

One of the most challenging issues concerns how to define the "action potential event" and how to determine not only what it looks like but exactly when it occurred. Typically, they can be defined in terms of timing, kinetic models, and sometimes their "waveforms" (i.e., shape). Here we have chosen to use the synaptic model described in [22], which uses kinetic models to exhibit STDP[21] and short-term depression[20], but are not currently using any of the optimizations that they describe. STDP is a Hebbian learning mechanism based purely on timings of received and fired action potentials. Short-term depression describes a synapse as having limited resources and its output is affected by recent synaptic activity.

We can effectively model the AP here as a singular event where we only need to know the time it is received. The advantage of this is that we are effectively modeling the relationship between the voltage on the soma and any input current at all times. The disadvantage of this is that any small variation in AP shape is not transmitted to a receiving neuron. We could effectively transmit a

waveform to a receiving neuron by knowing where it "starts" and "stops". This event-driven approach can theoretically handle all three types of synaptic models and thus promotes interoperability.

Since we are measuring the AP purely in terms of timing, we need to determine when the AP needs to be received. If we know the axonal delay then we need to determine when the AP was fired. This is essentially the same issue as the problem of finding the exact moment threshold crossing for an IF neuron (e.g. see the work in [17, 19, 24, 25]). For most IF models this is determined by explicitly solving the governing equations (see Brette [19] for an unsolvable IF model).

The HH model does not explicitly have any notion of a "threshold". However, this notion is quite useful here. For an event-driven HH model, one could set the "threshold" to be a voltage on the action potential with the idea that if this "threshold" was reached then a neuron must be firing an action potential (by definition, we must because the threshold is *on* the action potential). By experimentally varying the amount of a constant input current we can determine that the HH neuron will fire if the voltage is increased above -50mV. Thus we can say that -50mV is our threshold. This leaves us with the problem of determining the exact moment in which the threshold was crossed. One possible solution lies in the fact that the APs generated from a neuron generally have a narrow range of possible shapes. Given this, it seems plausible that we could better interpolate the exact crossing from the range of slopes that will occur at threshold crossing.

As the reader may have noticed it is not absolutely necessary to determine when this HH threshold was crossed. In the above paragraphs, this notion of "threshold" was used because it is a convenient place to define the moment when the action potential event occurred. However, one must keep in mind that the focus of events in discrete event simulation is not when an event actually is "sent" but when it is received. Instead of calculating the threshold and using that to determine when the action potential event should be received, one could determine this received time from the current stepped value of time directly. This would conflate the issue of what lookahead would be used but it is indeed possible especially for optimistic protocols. This type of approach may be very useful for more complex neurons where the shape of the AP is not as well defined.

The above description of an event-driven HH model affords several advantages. One such advantage is that it allows one to make various modeling optimizations concerning when the state of the neural components need not be updated. As an illustrative example, consider the 'synapse-neuron' representation at equilibrium. In the absence of noise, none of the values are changing so there is no point in updating it during these times. This is essentially the same type of rationale that led to the creation of event-driven IF models. The next time that it must be updated is when it comes out of equilibrium. Or do we? If the input is small and enough time would pass that the system would quickly respond back to equilibrium, one might be able to ignore it completely. This type of optimizations would be more acceptable for many network-level questions where the concern is typically about the generation of action potentials and not minute changes in a neuron's membrane potential.

A vital point to be made here is that in these neural systems there is a significant amount of noise. This noise is accounted for in many of the models as a "noise current". How might we be able to efficiently have random noise in the system? Does this make it a highly irregular problem and inherently more difficult to solve using this type of an approach? Does this dismiss the possible optimizations that are discussed above? These types of questions deserve further exploration (see [18] for an account of how to deal random noise in IF neurons).

5. Model Implementation

We chose to implement the neural system using the simulation engine μsik [3] (pronounced "Musik"), which is written in C++. μsik was developed to be able to efficiently simulate large-scale discrete-event simulations while allowing for maximum flexibility in its synchronization algorithm. It uses a micro-kernel design analogous to the one employed in operating systems. These inherent advantages are reminiscent of the design goals that were discussed in the previous section. As such, μsik is well-suited as a simulation engine for modeling this neural system. It also will provide a seamless way for testing different synchronization methodologies and comparing the performance advantages and disadvantages of each.

In μsik, a physical process is referred to as a 'Simulator' and is composed of many logical processes. Each logical process has its own synchronization algorithm. Logical processes communicate via discrete events. For conservative approaches, time is synchronized by determining the next event for each logical process via a global lower bound timestamp (LBTS) computation. Lookahead is important in these approaches for efficient synchronization as it allows the simulation engine to know when the federate will not schedule any new events.

The neural simulation is implemented using the 'synapse-neuron' representation as the logical process. Presently, a conservative protocol is used for synchronization. LPs send "action potential" events when their threshold is crossed. In order to meet the constraints of the conservative approach, each neuron updates its state at regular intervals. The reasoning behind this is outlined below.

In μsik, the physical process, called *NeuralSimulator,* is defined as a subclass of 'Simulator'. Inside this federate, there are pointers to output files for recording data. The logical processes are implemented as the class *Neuron*, a subclass of 'NormalSimProcess'. Inside *Neuron,* several state variables are defined including the current timestep, the threshold of excitation, the membrane potential, the time the neuron was last updated, and two ids. One id serves as the unique LP identifier for a *NeuralSimulator* instance. The other id is defined to be unique over the entire simulation and is used for determining noise current. In the previous section, the state of the synapse was described as being an explicit component in the representation for the post-synaptic neuron and only an identifier and a delay for the pre-synaptic neuron. We have implemented this as two C++ maps to which *Neuron* holds a pointer: one that holds a list of pointers to this *Synapse* class and the other that holds a list of neurons, their federate ids and the appropriate axonal transmission delay (in a *ConnectionInfo* class) to whom this neuron fires. Both maps are keyed on the unique synapse id.

The *Neuron* class sends and receives two types of messages. The first is an event called an "AP_ARRIVAL", that signals the neuron that an action potential has arrived at the neuron. When this event is received, the neuron will update its time from the last time it was updated and then update the appropriate state variables. An adaptive Runge-Kutta timestep method with error control, as described in [23], is used to perform the necessary numerical approximations. This algorithm allows one to set a tolerance on the calculated error and lets the timestep grow or shrink as needed in order to minimally meet this tolerance. This timestep is bound to a certain range and will grow/shrink at a constant rate. Each neuron LP has its own timestep value which is retained between events. This adaptive timestep method is also consistent with the event-driven philosophy.

One challenge is determining when this action potential event will actually be received. If the exact "firing time" were known then the AP event would need to be received at the firing time plus the axonal transmission time. For this implementation we simply see if the threshold was crossed during some timestep dt. If it was crossed then a neuron will fire an AP_ARRIVAL event at the end of this dt. We perform no interpolation to get a more exact crossing moment (within dt).

Unlike the IF event models, it is not known deterministically when this threshold will be crossed and one cannot employ a simpler LBTS-maintained system-wide "spike estimate" priority queue with conservative synchronization. To meet the constraints of the conservative protocol that we are currently using, we update the neuron periodically with a "WAKEUP" event. If the threshold has not been crossed then we could theoretically fire at any time (e.g. a supra-threshold input current arrives). In order to ensure that the received event occurs in the LPs future and meets the constraints of the conservative protocol we require that the WAKEUP event occurs at one-half the minimum axonal transmission delay. As a result of this, we must define our lookahead as also being one-half the minimum axonal transmission delay.

However, if a threshold has been crossed and we are currently generating the AP we know that we cannot fire again until the AP finishes and a period of time that the neuron cannot fire again (called the "absolute refractory period") has passed. We have experimentally calculated a very conservative estimate of the sum of these two periods as 10ms. Thus, one-half the minimal transmission time plus 10ms is the next time that our neuron would have to be updated.

The use of this WAKEUP event has an interesting consequence in that the more the neuron fires the less often we need to update its state via a WAKEUP event. This suggests that the simulation may get better performance if there is a moderate amount of activity. We investigate this quantitatively in section seven.

To model noise in the system, we added a noise current to the HH model. The value used is obtained by a Gaussian distribution with a mean of 700nA and standard deviation of 7500 and is sampled upon every time increment. This was defined with the intuition that most of the noise will generate sub-threshold responses but occasionally a neuron will receive supra-threshold noise, generating an action potential. Experimentally, if these large noise currents did not exist then the dynamics of our ion channels would simply converge to a steady-state range of a few millivolts above equilibrium.

The creation of these networks and its partitioning is done by a separate C++ program. This program will generate a network configuration based on [26]. The network generated is consistent with biological data in that synaptic connections between two neurons are much more likely if those neurons are near each other. It is important to note that the number of synapses generated will grow as the amount of neurons in the simulation grows! This relationship is shown in Figure 3. The current version of this program is extremely memory intensive and limits us to creating a network below 10,000 neurons.

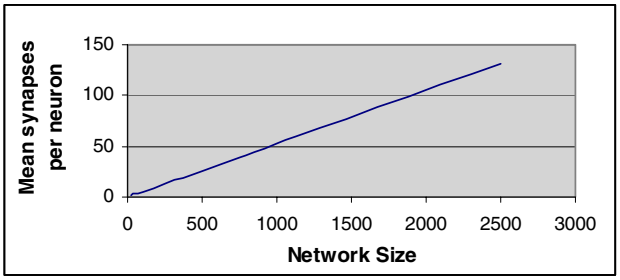

Figure 3: Neural connectivity in the generated HH networks

The partitioning of these networks is currently done through a simple scoring system where each synapse is given a weighted score based on the transmission delay and the number of synapses that are firing to the same neuron. Qualitatively, this weighted score is a measure of how often events will be received by the post-synaptic neuron. We iterate through the synapses from highest score to lowest score assigning neurons to processors. If both neurons for a synapse are unassigned then both are assigned in a round robin order to the next processor. If only one of the neurons is unassigned then it is either assigned to the same processor as the already assigned neuron or to the next processor. The greater the score of the synapse the more likely it is that it will be assigned to the same processor. The partitioned network for each processor is written to a unique input file that a *NeuralSimulator* reads in when it is initialized.

6. Model Verification

It is very difficult to verify that the network that we have created will generate reasonable neuronal activity in the absence of experimental data. Such verification requires a fair amount of parameter tuning. In the present paper, we are more concerned with if the basic mechanisms are correct and how different parameters (e.g. those that increase the chance that a neuron will fire an AP) will affect simulator performance. In order to investigate this, we have conducted several efforts to verify the neuron and synaptic models used in our implementation.

In order to verify our model of the HH neuron, we compared the event-driven model to a standard time-stepped HH model. The event-driven model appears to generate a normal action potential shape (see Figure 2) and respond similarly to sub-threshold input currents. It is worth noting here that since we are using numerical approximation the results of the comparison will not be exact.

We have completed some preliminary verification on our synaptic model. Initial testing on the mechanisms of STDP in our synaptic model and each of these variables indicates it appears to be responding correctly. We have done no testing with our short-term depression implementation so we have not included it in the experiments below. Verification of the biological accuracy of the networks that we generated remains an area of future work.

7. Initial Results

An initial set of simulation runs were conducted in order to test the feasibility of this event-driven Hodgkin-Huxley model and to access its scalability and performance. The simulations were run on a single cluster node consisting of 8 550 MHz PIII Xeon SMP processors with an available 4GB RAM. All simulations were conducted on a single node and thus there is no overhead due to intern-node communications.

Networks of size 25, 50, 100, 200, and 400 neurons were created according to the description in section five and simulated for 15 seconds of simulation time using 1, 2, 4, and 8 processors. Each neuron initially received between zero and four action potentials at random times in the first twenty milliseconds of the simulation. These inputs were designed to augment the spontaneous activity of these networks. All synapses in these networks were excitatory, meaning that input currents increase the membrane potential of the neuron. The speedups for these networks are shown in Figure 4.

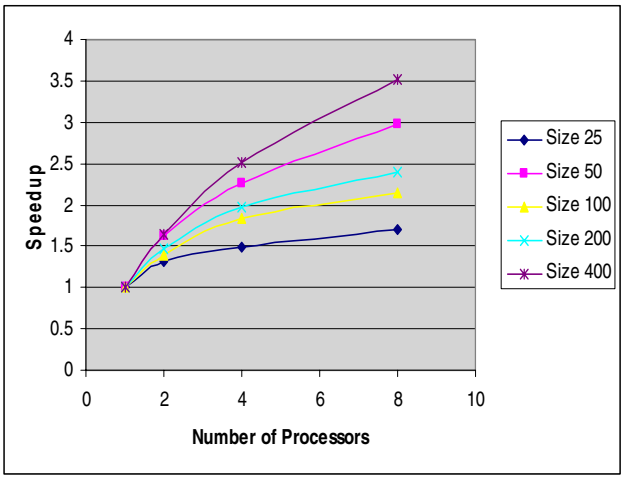

Figure 4: Speedup for networks of 25, 50, 100, 200, and 400 neurons on 1, 2, 4 and 8 processors.

As can be seen from the results in Figure 4, we are able to achieve good speedup in our neural networks. The network of 50 neurons offers particularly good speedup here because there are fewer remote events due to the way the network is partitioned. One might be able to achieve better speedup with other networks with better partitioning. These results suggest that one may be able to effectively run larger networks of these event-driven Hodgkin-Huxley neurons.

Next, we conducted an experiment to determine how simulation performance changes as network activity increases. In order to test this, we varied the maximal synaptic conductance (g_{max}) in a set of 15 second simulations on a network of size 100 on a single processor. The results are shown in Table 1. This experiment demonstrates that we are able to achieve better performance when there is a moderate amount of activity.

Table 1: Effects of Network Activity on Execution Time of a 100 HH Neuron Network

g_{max}	APs Fired	Total Elapsed Time (s)
0.010	402	11761.8
0.015	425	11745.2
0.030	398	11747.2
0.050	399	11699.6
0.065	412	11708.8
0.075	461	11445.6
0.090	417	11403.5
0.100	427	11518.3
0.130	481	10772.0

Important factors that determine the execution time are the number of events that must be processed and the amount of computation required to process each event. In our model there are two types of events: a WAKEUP event and an AP_ARRIVAL event. In Table 2 below, we list the percentage of each event that we process in our simulations. As one can see the number of WAKEUP events is far greater than the number of AP_ARRIVAL events. Efforts to reduce the number of WAKEUP events needed or the regularity in which they occur would greatly reduce the number of events that need to be processed and could yield much better performance.

Table 2: Percentage of Each Event in Simulations

	% AP_ARRIVAL Events	% WAKEUP Events
25	0.0677	99.9323
50	0.0787	99.9213
100	0.0604	99.9396
200	0.1046	99.8954
400	0.1567	99.8433

It is important to be able to characterize the computation time of each of these events. In order to determine this, the time taken to process each type of event was recorded in a separate set of simulations that were setup as described above but ran for only 1s of simulation time.

In Figure 5, we can see the amount of time to process each event increases as the size of our network increases. The WAKEUP event updates the state of the neuron to the timestamp in that event. The time to process this event grows roughly linearly with network size because the number of synapses for which we must calculate the synaptic input grows linearly (see Figure 3). Optimizations that try to pull out computations from individual synapses may greatly help speed up the computation here (e.g., see [22]).

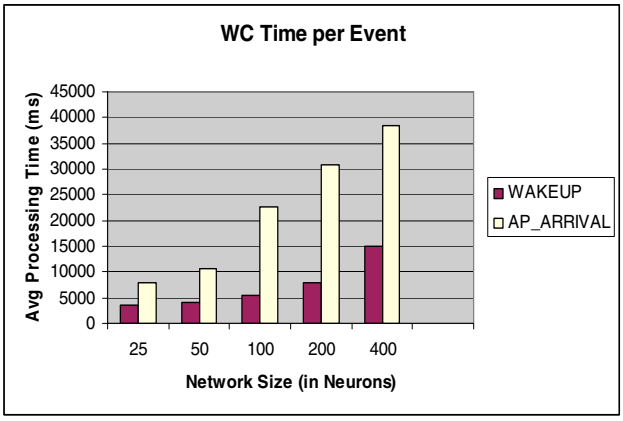

Figure 5: Wall clock processing time for WAKEUP and AP_ARRIVAL events for networks of size 25, 50, 100, 200, and 400 neurons

The AP_ARRIVAL event updates the state of our neuron to the timestamp of that event and then updates the state of the synapses for the arrival of an action potential. The time to process an AP_ARRIVAL event will grow similar to the WAKEUP event because of linear synaptic growth. In addition, we must find and update the appropriate synapse in our synaptic representation (a C++ map). This computation is logarithmic and accounts for the difference between the two types of events.

8. Conclusion

In this paper, we have described an event-driven Hodgkin-Huxley neuron that offers promise for effective parallel simulation of larger networks. This technique offers good performance as the networks size increases, better performance when there is moderate activity in the network, and meets several of the design criteria described in [10] for large scale neural simulators.

From the simulation side, there are many future directions that could be pursued. These include: improving the partitioning algorithm; investigating the use of other synchronization algorithms, experimentation on larger networks with a larger number of processors; achieving a lookahead based on action potential events only; characterizing and limiting the memory consumption of the simulation; and performing a comparison with a time-stepped approach to determine what, if any, savings result from using this event-driven approach.

From the modeling side, there are equally as many directions that deserve further investigation. These include: further verification of the neuron and synapse models and networks for biological realism and parameter sensitivity; a better characterization of the noise current and the questions that it begs; and investigating other techniques and optimizations that may be employed to

improve the computational speed in which events are processed.

Appendix: Model Details

The Hodgkin-Huxley model is a set of four differential equations (1-4). The synaptic model is also a set of four differential equations (5-8).

(1) $-C * dV/dt = I_{noise} - (I_{Na} + I_K + I_{leak})$
   a. $I_{Na} = g_{na} * m(t)^3 * h(t) * (V - E_{Na})$
   b. $I_K = g_K * n(t)^4 * (V - E_K)$
   c. $I_{leak} = g_{leak} * (V - E_{leak})$

(2) $dm/dt = (m_{inf} - m) / \tau$

(3) $dn/dt = (n_{inf} - n) / \tau$

(4) $dh/dt = (h_{inf} - h) / \tau$

(5) $dg_{ex}/dt = -g_{ex} / \tau_{ex}$ (Excitatory Synapses)

(6) $dg_{in}/dt = -g_{in} / \tau_{in}$ (Inhibitory Synapses)

(7) $dM/dt = -M/\tau_-$

(8) $dP_a/dt = -P_a/\tau_+$

Equation 1 was solved using adaptive Runge-Kutta using a tolerance of 10^{-11}. Equations 2-8 were formulated in terms of an update rule (of the form $x(t + dt) = \ldots$) and evaluated for the appropriate value of V. These parameters resulted in average timesteps in the range of 0.001 ms. to 0.00004 ms. for the first set of experiments described in section seven.

Parameter values for these equations were given in their respective papers. Similar to [21], we set the initial value of g_a to g_{max} in order to maximize network activity (g_{max} = 150 pS).

Acknowledgements

This research was supported in part by NSF grant ATM-0326431.

References

[1] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve." *Journal of physiology*, vol. 117, pp. 500-544, 1952.

[2] Nadasdy Z. , W. D. A., Potter S.M. (2003). "Attractor dynamics of superbursts in living neural networks." SFN 2003, New Orleans, LA.

[3] K.Perumalla, "Musik – A Micro-kernel for Parallel/Distributed Simulation Systems", PADS 2005.

[4] L. Lapicque, "Recherches quantitatives sur l'excitation electrique des nerfs traitée comme une polarisation," vol. 9, pp. 620-635, 1907.

[5] W. Gerstner and W. M. Kistler, *Spiking neuron models: single neurons, populations, plasticity*. New York: Cambridge University Press, 2002.

[6] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE transactions on neural networks*, vol. 15, pp. 1063-1070, 2004.

[7] M. Hines and N. T. Carnevale, "The NEURON Simulation Environment," *Neural Computation*, vol. 9, pp. 1179-1209, 1997.

[8] J. M. Bower and D. Beeman, *The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System*. Santa Clara, Calif.: Telos, 1995.

[9] P. Pacheco, M. Camperi, and T. Uchino, "PARALLEL NEUROSYS: a system for the simulation of very large networks of biologically accurate neurons on parallel computers," *Neurocomputing, July 1999*, vol. 32-33, pp. 1095-1102, 2000.

[10] I. I. Muresan R.C., "Principles of Design for Large Scale Neural Simulators," presented at International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, 2004.

[11] I. I. Muresan R.C., "The "Neocortex" Neural Simulator. A Modern Design," presented at International Conference on Intelligent Engineering Systems, Cluj-Napoca, 2004.

[12] E. A. Thomas, "A parallel algorithm for simulation of large neural networks," *Journal of Neuroscience Methods*, vol. 98, pp. 123-34, 2000.

[13] N. Goddard, G. Hood, F. Howell, M. Hines, and E. De Schutter, "NEOSIM: portable large-scale plug and play modelling," *Neurocomputing*, vol. 38-40, pp. 1657-61, 2001.

[14] C. Grassman and J. K. Anlauf, "Fast digital simulation of spiking neural networks and neuromorphic integration with SPIKELAB," *International Journal of Neural Systems*, vol. 9, pp. 473-8, 1999.

[15] A. Delorme, J. Gautrais, R. van Rullen, and S. Thorpe, "SpikeNET: a simulator for modeling large networks of integrate and fire neurons," *Neurocomputing*, vol. 26-27, pp. 989-996, 1999.

[16] D. Brettle and E. Niebur, "Detailed parallel simulation of a biological neuronal network," *IEEE Computational Science & Engineering*, vol. 1, pp. 31-43, 1994.

[17] M. Mattia and P. Del Giudice, "Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses," *Neural Computation*, vol. 12, pp. 2305-29, 2000.

[18] J. Reutimann, M. Giugliano, and S. Fusi, "Event-driven simulation of spiking neurons with stochastic dynamics," *Neural Computation*, vol. 15, pp. 811-30, 2003.

[19] R. Brette, "Event-driven simulation of integrate-and-fire neurons with exponential synaptic conductances," *Submitted*, 2005.

[20] H. Markram, Y. Wang, and M. Tsodyks, "Differential signaling via the same axon of neocortical pyramidal neurons," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, pp. 5323, 1998.

[21] S. Song, Miller, K.D. and Abbott, L.F., "Competitive Hebbian Learning Through Spike-Timing Dependent Synaptic Plasticity," *Nature Neurosci*, vol. 3, pp. 919-926, 2000.

[22] M. Giugliano, M. Bove, and M. Grattarola, "Fast calculation of short-term depressing synaptic conductances," *Neural Computation*, vol. 11, pp. 1413-26, 1999.

[23] W. H. Press, *Numerical recipes in C: the art of scientific computing*, 2nd ed. Cambridge; New York: Cambridge University Press, 1992.

[24]M. J. Shelley and L. Tao, "Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks," *Journal of Computational Neuroscience*, vol. 11, pp. 111-19.
[25]T. Makino, "A discrete-event neural network simulator for general neuron models," *Neural Computing & Applications*, vol. 11, pp. 210-23, 2003.
[26]R. Segev and E. Ben-Jacob, "Generic modeling of chemotactic based self-wiring of neural networks," *Neural Networks*, vol. 13, pp. 185-99, 2000.