Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

OPTIMIZATIONS FOR NEURON TIME WARP(NTW) FOR STOCHASTIC REACTION-DIFFUSION MODELS OF NEURONS

Mohammad Nazrul Ishlam Patoary
Carl Tropper
School of Computer Science
McGill University
Montreal, CANADA

Robert McDougal
Department of Neuroscience
Yale University
New Haven, Connecticut, USA

William W. Lytton, MD
SUNY Downstate Medical Center
450 Clarkson Avenue, MSC 31
Brooklyn, New York 11203, USA

ABSTRACT

The intracellular calcium signaling pathways of a neuron consist of biochemical reactions along with molecular diffusion. It is known that stochastic discrete event simulation of these pathways provides a more detailed understanding of the pathways than deterministic simulators because they capture behavior at a molecular level. Our research employs a parallel discrete event simulation simulator, Neuron Time Warp (NTW), which is intended for use for the simulation of neurons. In previous work we built a discrete event Ca^{2+} wave model. However, we did not achieve the expected performance because of an imbalance in the computation between the area of the neuron covered by the Ca^{2+} wave and the remaining area of the neuron. In this paper we describe a dynamic load balancing algorithm and a dynamic window control algorithm for NTW. We make use of Q-learning to determine the basic parameters of the algorithm. Using this algorithm we obtained an improvement in the performance of the simulator of up to 30%.

1 INTRODUCTION

It is well known that the neuron is the computational building block of the human brain. Neurons use a number of signaling pathways to regulate their internal activity. A signaling pathway is initiated when an extracellular molecule activates a specific receptor located on the cell surface. In turn, this receptor triggers a chain of reactions and diffusions inside the cell.

Intracellular signaling pathways can be simulated deterministically. Numerical methods are used to solve these equations. Calcium ions, Ca^{2+}, are the principal actors in these equations. The concentration of these ions are used by the deterministic simulators. However, deterministic simulations are not very accurate for a small number of ions, and as a result stochastic discrete-event simulation has emerged as a method to complement the use of differential equations for these models (Blackwell 2013).

A system consisting of a collection of chemical reactions can be modeled by a chemical master equation (Sterratt et al. 2011). Such an equation models the probability distribution of the chemical reactants in the system at each point in simulated time. In general, it is very difficult to solve this equation. In (Gillespie 1977) Gillespie introduced a Monte Carlo simulation algorithm for this model. Under the assumption that the molecules of the system are uniformly distributed, the algorithm simulates a single trajectory of the chemical system. Repeated simulation of these trajectories then gives a picture of the system.

Gibson and Bruck describes the Next Reaction Method (NRM) in (Gibson and Bruck 2000), which attempts to reduce the computation time of the propensities in Gillespie's algorithm. It makes use of a dependency graph of the reactions which is used to identify the reactions in need of an update. A number of other efforts aimed at improving the efficiency of the Gillespie algorithm have been made, including (Gillespie 2001), (Takahashi et al. 2004).

A key assumption in the Gillespie algorithm is that chemical reactions are distributed homogeneously in space. This assumption fails to hold when modeling a calcium signaling pathway because neurons are so large that diffusion cannot equilibrate their concentration. Molecular dynamic simulations (Leach 2001) (Bartol Jr et al. 1991) (Andrews et al. 2010) can be used to model individual particles but they take into account inter-particle forces, rendering them computationally expensive. Instead, we make use of a less expensive Monte Carlo algorithm, the Next sub-volume method (Elf and Ehrenberg 2004). The next sub-volume method (NSM) partitions space into cubes and represents the reactions and the diffusion of ions between these cubes by events. It makes use of the Gillespie algorithm to compute the next event times within individual cubes and relies upon the use of a priority queue to determine the next event time in the simulation.

It is well known that sequential stochastic discrete event simulation of a large scale reaction-diffusion system is quite slow. The use of parallel discrete event simulation (PDES) to overcome this difficulty appears in (Jeschke et al. 2008) (Dematt and Mazza 2008) (Wang et al. 2011).

This paper makes use of Neuron Time Warp (NTW). NTW is an optimistic simulator based on Time Warp (Jefferson 1985). It uses a multi-level queue for the pending event set and a single rollback message in place of individual anti-messages. The multi-level queue structure disperses contention for the priority queue which naturally occurs in a parallel simulator. Making use of single rollback messages decreases the overhead of Time Warp rollbacks. In this paper we use NTW to simulate a Ca^{2+} wave in a neuron. We describe a dynamic load balancing algorithm and a dynamic time window algorithm to improve the efficiency of NTW.

(Patoary et al. 2014) made use of a model of a dendrite branch in order to evaluate NTWs performance and accuracy for a spatial Lotka-Volterra model (Schinazi 1997). (Patoary et al. 2017) describes a simulation of a Ca^{2+} buffer and a Ca^{2+} wave model. The performance of these simulations was compared to the results obtained by NEURON (Carnevale and Hines 2006), a deterministic simulator which is widely used by the neuroscience community. We achieved a speedup in excess of 5 with 7 worker processes for the calcium buffer model. We derived a stochastic discrete event Ca^{2+} wave model from a deterministic model (Neymotin et al. 2015) using the stochastic model of an inositol 1,4,5 -trisphosphate receptor (IP_3R). The IP_3Rs are located on the surface of the endoplasmic reticulum (ER) of a neuron. These receptors are responsible for the release of Ca^{2+} from the ER into the cytosol of a neuron. We did not achieve good performance on this model due to the imbalance of event computation between area covered by the wave and the area which was not covered by the wave. In this paper we describe a dynamic load balancing algorithm and a dynamic window control algorithm to address these problems. We use Q-learning (Meraji and Tropper 2012a) to compute the values for the parameters of these algorithms.

The rest of the paper is structured as follows. Section 2 contains background and related work while section 3 focuses on the NTW simulation environment. Section 4 describes our load balancing algorithm and section 5 describes the calcium wave model along with our experimental results. Section 6 contains our conclusions.

2 BACKGROUND AND RELATED WORK

NSM is a discrete-event algorithm for the simulation of reactions and diffusion of species within a volume which contains an inhomogeneous distribution of particles (Elf and Ehrenberg 2004). The volume is divided into sub-volumes which are then assumed to be well mixed. A well-mixed sub-volume is one in which the reactants in the sub-volume are spatially homogeneous in the sub-volume. Well mixed sub-volumes are essential for the use of NSM.

There are two types of events used in NSM: 1) those which represent reactions inside a sub-volume and 2) those which represent diffusion between adjacent sub-volumes. NSM makes use of the Gillespie algorithm to compute the next event time within a sub-volume. Within the main loop of the algorithm the sub-volume with the smallest next event time is selected and the event type is then determined using the reaction rates. The event is then executed and the state is updated. Note that the update is done only in a small region. NSM relies upon priority queues for both the next event time and the diffusion time.

A parallel discrete event simulation is composed of a set of processes which are executed on different processors. The processes model different parts of a physical system. Each process is referred to as a logical process (LP). The *LP*s communicate with each other via time stamped messages. Events are stored in an event list. Each *LP* processes its events in increasing time stamp order. The efficient management of the event list has a significant effect on the performance of the parallel simulation. Although processing the events in a sequential simulation is done by using a centralized list, such an approach is not possible in a parallel simulation as it would create contentions among the multiple processes. Again errors can result in a parallel simulation from out of order event execution (causality errors). The problem of maintaining causality is referred to as the synchronization problem-the idea is to make sure that the execution of the parallel simulation produces the same sequence of events as if all of the events had been processed by a single process. There are two main approaches for solving the synchronization problem-conservative and optimistic. Time Warp (TW) synchronization is a widely used optimistic synchronization protocol and is the one which we make use of in NTW (Patoary et al. 2014).

2.1 Previous Work on Parallel Stochastic Simulation of Neurons

In order to parallelize NSM (Jeschke et al. 2008) divides space into sub-volumes and assigns several sub-volumes to each *LP*. Interactions between *LP*s involve the diffusion of events between neighboring sub-volumes. Preliminary results on small models were found to be encouraging. As pointed out by the authors, a number of areas including window management and state saving remain to be investigated.

Dematt and Mazza points out in (Dematt and Mazza 2008) that a conservative synchronization is not suitable for use with the NSM due to the zero lookahead property of the exponential distribution. An optimistic algorithm was implemented and preliminary results were obtained for a predator-prey model (Schinazi 1997). The performance of the simulation was found to be hampered by a lack of control over the window size.

A variant of NSM called the Abstract Next Sub-volume Method (ANSM) is presented at (Wang et al. 2009). Three optimistic synchronization algorithms were employed-Time Warp (TW) (Jefferson 1985), an optimistic approach with risk-free message sending called Breathing Time Bucket (BTB) (Steinman 1991) and a hybrid approach which combined these methods were used for the simulation of a predator-prey model. The moderate optimistic approaches resulted in better performance. Neither (Jeschke et al. 2008) nor (Wang et al. 2009) employed (1) a real neuronal geometry or (2) a real model (e.g. Ca buffering, Ca^{2+} wave model etc.) on which to evaluate their accuracy and performance.

XTW (Xu and Tropper 2006), an optimized version of TW, introduces a new optimistic synchronization mechanism to improve the performance of Time Warp. XTW consists of a new event scheduling algorithm, XEQ, and a new rollback mechanism, rb-message. XEQ has an O(1) cost bounded on the number of simulated entities (not on the number of events). Rb-message not only reduces the computing cost of annihilating previously sent messages, but also dramatically reduces the memory cost by eliminating the output queue in each LP. NTW inherits all of those features in XTW.

(Meraji and Tropper 2012b) and (Meraji and Tropper 2012a) make use of *Q*-learning techniques to develop a combined bounded window and dynamic load balancing algorithm for parallel digital logic simulation. The algorithm resulted in about a 60% improvement in simulation time. (Lin et al. 2016) employed a *Q*-learning algorithm for a dynamic load balancing algorithm for a multi-threaded parallel discrete event simulation. The authors simulate a Ca^{2+} wave model, obtaining a 21% improvement for inter-process LP migration.

We simulated two models, a calcium buffer and a Ca^{2+} wave model, in (Patoary et al. 2017) to verify NTW's correctness and to examine the performance. The performance of the simulation of the calcium buffer model is compared with a sequential deterministic simulation using NEURON. We also derived a discrete event Ca^{2+} wave model from a stochastic IP_3R model (Falcke 2003). While we obtained good speed-up for the calcium buffer model, we did not get satisfactory performance for the Ca^{2+} wave model. This was because there was a large computation imbalance between the areas in the model with high Ca^{2+} and those areas with low Ca^{2+}, resulting in a large number of rollbacks.

3 NTW

We have designed and implemented a parallel discrete event simulator, NTW, to simulate 3D stochastic reaction-diffusion systems. We used NEURON's front end as NTW's front end, making it possible for a group of mass action chemical reactions to be input by the user. This enables the user to experiment with different reactions and different concentrations of molecules and ions as well as their associated reaction rates. It is also possible to experiment with different diffusion rates between adjacent cells. NTW inherited a multi-level priority queue from XTW (Xu and Tropper 2006). Every sub-volume (SV) in NSM is an LP in NTW. Chemical reactions and diffusion between sub-volumes are events in NTW.

3.1 Architecture

LPs are grouped together into clusters which, in turn, can be distributed among separate physical computation units. We currently allocate one cluster per core. Each cluster processes the events belonging to its LPs in increasing time-stamped order. There is a special cluster (a controller), which is in charge of distributing the simulation workload, computing the global virtual time (GVT), and collecting the simulation results. We employ Mattern's algorithm (Mattern 1993) to calculate the GVT. Checkpointing is done upon the arrival of a diffusion event at a cluster. The architecture and communication overview of NTW is shown in figure 1.

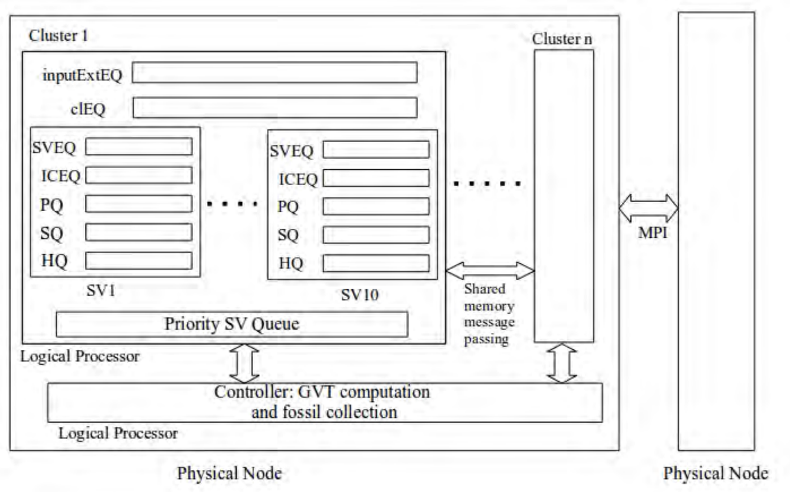


Figure 1: The architecture and communication overview of NTW.

In every cluster there are two event queues, the clEQ and the inputExtEQ. The clEQ is used to sort the events generated by local LPs (i.e. within the same cluster). Its top event is the lowest time stamped event in the cluster. All events from different clusters are temporarily put into inputExtEQ and are then forwarded to an input channel event queue, ICEQ. The priority SV queue is a container of SVs. The head of the SV priority queue points to the SV with the minimum local time in the cluster.

In a three-dimensional geometry, an SV can have at most six adjacent SVs as neighbors. Each SV has a SV event queue (SVEQ), an input channel event queue (ICEQ) which is used to hold diffusion events from neighbors, a processed event queue (PEQ), a state saving queue (SQ) and a history queue (HQ). See (Xu and Tropper 2006) for a detailed description of multi-level queuing. The history queue (HQ) is an addition to the structure described in XTW (Xu and Tropper 2006). A more detailed description of NTW's architecture along with a detailed description of event processing is contained in (Patoary et al. 2014).

4 THE DYNAMIC LOAD BALANCING ALGORITHM

A centralized algorithm is used for the dynamic load balancing of NTW. It is initiated every C GVT cycles, where C is a control parameter whose value is determined by a Q-learning algorithm. After C GVT computations we first estimate the workload for each worker process and detect a load imbalance using runtime statistics. The parameters which are made use of by the computation and communication algorithms are as follows:

LP computation Load (*LpLoad*): The computation load of each LP is defined as the number of processed events since the last load balance of the simulation.

Worker process computation Load (*PComp*): The computation load of each worker process is defined as the sum of the computation loads of the LPs within that worker process.

LP communications Load (*LpComm*[]: The communication load of each LP is an array of length $n-1$ where n is the number of processors in the system. Each element of this array is the number of messages that the LP sent to other processors since the last load balance of the simulation.

Processor to processor communication load (*PPComm*[]): The communication load of a processor is represented by an array of length $n-1$ where n is the number of processes. The elements of the array are the number of messages that the process sent to the other processors since the last load balance.

Processor communication load (*PComm*): the number of messages that each processor sent to other processors.

The computation-communication weight (λ): λ final load of the processor. Its value is between 0 to 1.

Processor load (*PLoad*): the weighted sum of the computation and communication loads:

$$PLoad = \lambda * PComp + (1-\lambda) * PComm. \tag{1}$$

4.1 Load Imbalance Detection

Every C GVT cycles controller detects the imbalance of workload among the worker processes. Cluster level local time, $cltLt$, is updated by the SV's local time, Lt, which is calculated as $Lt = Lt + \tau$ (τ is the time of next event). τ is inversely proportional to number of molecules in the process. When a Ca^{2+} wave is initiated in a process, its size increases very sharply, causing τ to decrease. When τ decreases, Lp level (SV's) local simulation time (Lt) progresses very slowly. In our experiments we observed that least valued Lt is always overloaded.

When the controller sends $loadDetectionSignal$ to all worker processes, all processes send back their cluster level local time, $cltLt$, to the controller. Controller determines the lowest $cltLt$ by $cltLt_{min} = min(cltLt_{p_1}, cltLt_{p_2}, ...cltL_{p_n})$ as well as the maximum difference between $cltLt_{min}$ and $cltLt_i$ by $differenceLt_i = max(cltLt_i - cltLt_{min})$ where $i = 1....n$ processes. If $differenceLt_i$ is greater than a threshold, $cltLt_{threshold}$ it is considered process p_i is overloaded.

When a load imbalance is detected, Controller sends $CollectLoadSignal$ to all worker processes. Worker processes compute their *PComp* (sum of all *LpComps* within the worker process). *PComm* (sum of all *LpComm*[] within the worker process) is sent to the controller. The controller computes λ_i as follows:

$$\lambda_i = PComp/(PComp + PComm). \tag{2}$$

Computes $PLoad_i$ of a worker process i using equation (1). Controller determines the highest loaded process, $PLoad_{max}$ and least loaded process, $PLoad_{min}$.

4.2 Computation and Communication Algorithms

The computation algorithm first balances the computation and then attempts to balance the communication between processors. Each processor sends $PLoad$ and $PComm[]$ to the controller every C GVT cycles. The controller selects the highest and lowest loaded processes $PLoad_{max}$ and $PLoad_{min}$ and sends $PLoad_{max}$ the address of $PLoad_{min}$ which in turn selects n LPs having the most communication with $PLoad_{min}$ to send to $PLoad_{min}$.

The communication load-balancing algorithm has the same structure as the computation load-balancing algorithm. The main difference is that it attempts to first balance the communication and then computation. The algorithm uses $PComp$, $LpLoad$, $PComm[]$ and $LpComm[]$ to balance load. Every C GVT cycle, each processor P_i sends its $PComp$ and $PComm_i[]$ values to the controller process. $PComm_i[j]$ contains the communication load between nodes i and j. The controller process finds the maximum value of $PComm_i[j]$ among the value of $PComm_i[]$ that it received from different worker processes. If worker process P_i and P_j had the most communication during the last C cycle the algorithm attempts to transfer LPs between these two processes. In order to take into account the effect of the computation, the process with the highest value of the $PComp$ is chosen as the sender process and $doMigration$ message is sent to it. Upon receipt of the $doMigration$ message at process P_i , it selects up to L LPs which have the most communication with the destination process.

4.3 Reinforcement Learning

Basically, we have used a single agent with two-state approach to formulate the dynamic load balancing approach. Here, a central node called controller process, gathers information from all worker processes, runts the Q-learning algorithm, finds new values of the control parameters and inform to all worker process about the new values. Our single agent is run in the controller process. We have two control parameters for the load balancing algorithm - L and C.

L: the number of LPs which are transferred from one worker process to another in each cycle of the dynamic load balancing algorithm. If L is large, the communication overhead increases the simulation time. If L is small, we may have an unbalanced load. We utilize 4 different values for L, $L = 10, 15, 20, 25$.

C: the number of GVT cycles between executions of the load balancing algorithm. If we run the algorithm too frequently, the communication cost increases, while if we run it infrequently, we may fail to balance the load. We utilize 3 different values for C $C = 2, 4, 6$.

If L has m values and C has n values there are $m*n$ combinations of these control parameters. Each (combined) value corresponds to an action of the Q-learning algorithm. There are two states for the algorithm-unbalanced and balanced. The state change diagram is contained in figure 2

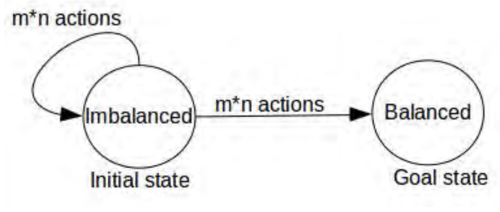


Figure 2: States and actions in Q-learning.

After C GVT cycles all of the worker processes send their data to the controller which then executes the RL algorithm. After computing new values for the control parameters, it broadcasts them to the worker processes.

The reward function and Q-learning algorithm: If the reward function does not reflect the main goal of the system (reducing the simulation time), the RL algorithm does not succeed. Hence we relate the reward to the event processing rates of the worker processes.

If t_i is the wall clock time at the i^{th} GVT cycle (GVT_i), we define the Event Commit Rate (ECR) of the i^{th} GVT interval (the interval from GVT_{i-1} to GVT_i) to be:

$$ECR_i = NC_i/(t_i - t_{i-1}) \tag{3}$$

where NC_i denotes the number of committed events as GVT_i. Committed events are the processed events whose times stamps are less then the current GVT.

In order to define a reward, we use a reference point. We define ECR_{ref} as the average event commit rate since the beginning of the simulation:

$$ECR_{ref} = (\sum_{i=0}^{D} ECR_i)/(t_D - t_0). \tag{4}$$

In the above formula, D is a small number between 5 to 10. The reward of the i^{th} GVT interval is then defined as:

$$R_i = ECR_i - ECR_{ref}. \tag{5}$$

From this definition, the reward is positive if the simulation is faster that the reference rate during the last GVT interval. Otherwise a punishment (negative reward) is effected. The event commit rate represents the speed of the simulation.

We employed the following Q-learning update rule

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Max_a Q(s_{t+1}, a)] \tag{6}$$

where α and γ are the learning step and the discount rate, respectively. In order to select an action in a state, we follow a straightforward approach - the highest rewarded action is selected first. After calculating ECR_{ref}, the reward is calculated every C GVT cycles and the value of the current state action pair (the Q matrix) is updated. At each cycle of the dynamic load balancing algorithm, we select the state action pair with the largest average reward. The algorithm 1 presents the general structure of the Q-learning dynamic load balancing algorithm. Algorithm 2 presents the basic Q-learning algorithm.

Algorithm 1 Q-learning and Dynamic load balancing

```
 1: In every C GVT cycle the controller checks for a load imbalance.
 2: if load imbalance is detected then
 3:    Controller identifies processes with maximum and minimum workload.
 4:    Controller calculates $\lambda$ using equation (2).
 5:    if $\lambda > 0.6$ then
 6:       Run the communication algorithm in process with maximum workload
 7:    else
 8:       Run the computation algorithm in process with maximum workload
 9:    end if
10:    Set the state of the simulation to $S_i$, $i = (1,2)$. $S_1$ is a balanced state and $S_2$ is an unbalanced state.
       In state $S_2$, take action $a_i$ ( LPs migrated from highest loaded process $PLoad_{max}$ to minimally loaded
       process, $PLoad_{min}$)
11:    Compute the reward $R_i$ by using equation (5).
12:    Run the $Q$-learning algorithm with $R_i$ as input.
13: end if
```

Algorithm 2 *Q*-learning

```
Require: reward, $R_i$, of the latest execution of the last action.
Ensure: $L$ and $C$
1: for Repeat for each episode do
2:     Update the reward of the last action via equation (5).
3:     Update the Q matrix via equation (6)
4:     Select the action with maximum reward in state $s_2$. If rewards are equal, then selection is made
       randomly.
5:     Compute $L$ and $C$ from the selected action.
6: end for
7: After computing new values for the control parameters $L$ and $C$, controller broadcasts them to all
   worker processes.
```

4.4 The Time Window

Rollbacks can be controlled by a time window. The time window is an interval $[T, T+W]$ in virtual time. If the next event at an LP has a time stamp larger then $GVT+W$, the LP is blocked. A blocked LP can receive events from other LPs but cannot execute them or send messages to other LPs. The LP remains blocked until the GVT is updated, after which the events within the new window can be executed. The objective of the time window is to prevent too great a disparity in the virtual time of LPs, thereby reducing the number of rollbacks. In our experiment we used 4 values for L(10, 15, 20, 25) and 3 values for $C$is effected(2, 4, 6), resulting in 12 actions. Each action for our time window algorithm is a choice of the window size. The window sizes are multiples of a fixed parameter U. The multiple is the product of the values chosen in the load balancing algorithm for L and C.

Algorithm 3 Time Window

```
Require: $L$ and $C$ of the latest execution of the last action.
Ensure: $W_i$
1: Compute the window size by using C and L obtained from algorithm 6.
2: $W_i$ = $C$x$L$x$U$. Here i = 1 to mxn actions. We use U = 1000.
```

5 MODEL AND EXPERIMENTAL RESULTS

5.1 Ca^{2+} Wave Model

Calcium signaling depends upon increases in the intracellular concentration of Ca^{2+} ions. In most cells the concentration of intracellular calcium oscillates with a period ranging from a few seconds to a few minutes. These oscillations often take the form of waves. At rest, the concentration of calcium in the cell cytoplasm is low while outside the cell and in the internal compartments of the cell (e.g. the endoplasmic reticulum (ER)) it is high-see figure 3. The ER acts as an internal warehouse for Ca^{2+} ions from which they can exit to the cytosol through channels inositol triphosphate receptor channels (IP_3R). It can also exit via RyR (ryanadone receptor) channels or via leak channels. The IP_3Rs are located on the surface of the ER. Ca^{2+} is pumped back from the cytosol to the ER by ATP pumps. When an IP_3R channel is triggered by both Ca^{2+} and IP_3, it is opened and allows the fast release of Ca^{2+} from the ER to the cytoplasm, as shown in figure 3.

Stochastic Discrete Event Ca^{2+} wave model: In the deterministic model, the Ca^{2+} concentration in the cytosol, $[Ca\_C]$, is calculated every time step from different (in/out) fluxes- J_{IP_3R} , J_{SERCA}, and J_{LEAK} (Neymotin et al. 2015). In the stochastic approach, events (Channel Open, Release, Pump back and Leak) occur at discrete points in time. The event frequency can be controlled by their (in/out) flux rates (Neymotin

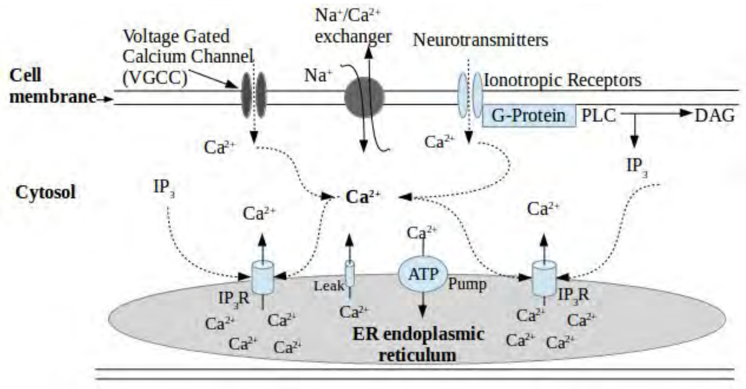


Figure 3: Ca^{2+} wave signaling in neuron.

et al. 2015). The parameter values were obtained from (Neymotin et al. 2015) and produced a Ca^{2+} wave propagation in a hippocampal pyramidal neuron (from NeuroMorpho.Org, consisting of 14749 SVs).

IP_3R releases Ca^{2+} from the ER to the cytosol. It consists of four identical sub-units, each of which is contains three binding sites (Falcke 2003)-an activating Ca^{2+} site, an inhibiting Ca^{2+} site and an IP_3 binding site. The three binding sites result in eight different states X_{ijk} for each subunit. i stands for the IP_3 site, j for the activating Ca^{2+} site, and k for the inhibiting Ca^{2+} site. An index is 1 if an ion is bound and 0 otherwise. Transition probabilities (per unit time) for transitions which involve the binding of a molecule are proportional to the concentration of that molecule. The channel is open if the subunit is in X_{110} , i.e., they have bound Ca^{2+} at the activation site and at the IP_3. Because the transition rates between the states X_{0JK} and X_{1JK} (IP_3 binding and dissociation) are two orders of magnitude faster than the other transition rates, the state probabilities for IP_3 will be stationary before the state probabilities for the Ca^{2+} binding and dissociation sites. Hence (Falcke 2003) only considered the Ca^{2+} binding and dissociation sites in his model, i.e. $X_{jk} = X_{0jk} + X_{1jk}$. We used the same approach and assumed that the channel can open if the subunits is X_{10}.

The binding and dissociation of Ca^{2+} at the activating and inhibition sites are stochastic events, hence the opening and closing of the channel is a stochastic process. This stochastic process is coupled to the concentration of cytosolic Ca^{2+} because the binding probabilities per unit time depend upon it and the number of open channels determines the concentration. The transitions corresponding to the reactions follow:

$$R_0 : IP_3R_{Inhibit} \xrightarrow{rc_1} IP_3R_{Not\_Inhibit} \qquad R_1 : IP_3R_{Not\_Inhibit} \xrightarrow{rc_2} IP_3R_{Inhibit}$$

$$R_2 : IP_3R_{Not\_Active} \xrightarrow{rc_3} IP_3R_{Active} \qquad R_3 : IP_3R_{Active} \xrightarrow{rc_4} IP_3R_{Not\_Active}$$

$$R_4 : Ca\_E \xrightarrow{rc_5} Ca\_C \qquad R_5 : Ca\_C \xrightarrow{rc_6} Ca\_E \qquad R_6 : Ca\_E \xrightarrow{rc_7} Ca\_C.$$

The, first two reactions, R_0 and R_1, correspond to Ca^{2+} binding and unbinding at the Ca^{2+} inhibition site. R_2 and R_3 are Ca^{2+} binding and unbinding at the Ca^{2+} activation site. Reaction R_4 is for Ca^{2+} release from the ER to the cytosol. R_5 and R_6 are for the Sarco/Endoplasmic Reticulum Ca^{2+}-ATP (SERCA) pump from the cytosol to the ER and leaking Ca^{2+} to the cytosol, respectively. Ca^{2+}in the cytosol is Ca_C and Ca^{2+} in the Endoplasmic Reticulum, ER, is Ca_E. Two mobile species in this model are Ca_C, for which the diffusion constant is $D_C = 0.75\mu m^2/ms$ and IP_3, for which the diffusion constant is $D_{IP_3} = 1.75\mu m^2/ms$. We assume that the Ca_E is not mobile, so its diffusion constant is $D_{CE} = 0$. For our experiment, the reaction constants were set as follows: $rc_1 = 1.0, rc_2 = 1.0, rc_3 = 1.0, rc_4 = 1.0, rc_5 = 10.0, rc_6 = 0.5$ and $rc_7 = 1.5$.

5.2 Experimental Results

Environment: A SW2 node, consisting of two Dual Intel(R) Sandy Bridge EP E5-2670 2.6 GHz CPUs, 8 cores per processor, 8 GB of memory per core, and a non- blocking QDR InfiniBand network with 40 Gbps between nodes. The node runs Linux 2.6.32-504.30.3.el6.x86_64.

Load imbalance detection: We did an experiment to observe how the load imbalance detection performs in our simulation. We used 4 processes (1 controller and 3 worker processes). Initially, we put IP_3 in process 1 to trigger a Ca^{2+} wave, resulting in a computation load imbalance among the worker processes. The cluster level (i.e. process level) local simulation times of the three worker processes are displayed in figure 4. We note that the cluster level local simulation time of process 1 is always less than that of the other worker processes because the Ca^{2+} wave was initiated in process 1.

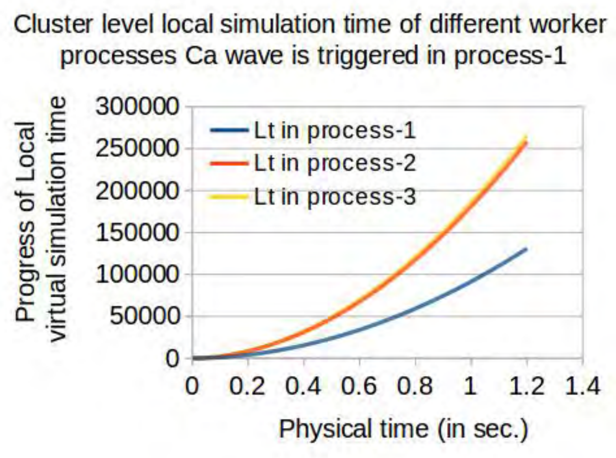
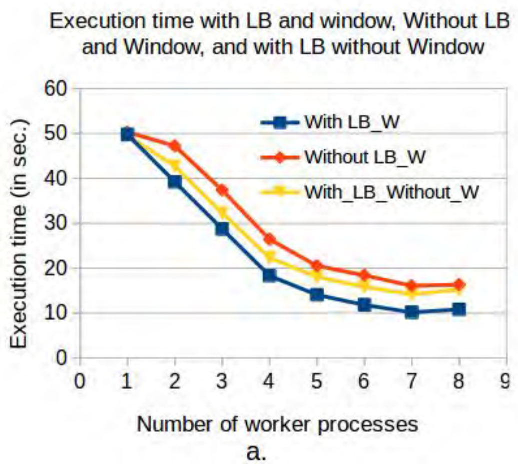


Figure 4: Cluster level local simulation time when Ca^{2+} wave is initiated in process 1.

Performance We employed the Ca^{2+} wave model in a hippocampal pyramidal neuron (from Neuro-Morpho.Org ,C91662, ModelDB:87284, 50 micron distance from soma and consisting of 14749 SVs). We executed about 72 Million events. The execution time and the number of roll-backs both decrease when load balancing along with a time window is used, as shown in figure 5. The maximum improvement (30% in execution time) takes place when 4 worker processes are used. Note that without the time window, the load balancing algorithm becomes less effective because of the increase in the number of rollbacks.

6 CONCLUSION

This paper describes a dynamic load balancing algorithm and a dynamic window control algorithm for use with NTW. NTW is used for the simulation of 3D reaction diffusion models in a neuron. The load balancing algorithm balances the computation and communication loads for these models. We employed these algorithms for the simulation of a discrete event Ca^{2+} wave model. Every C GVT cycles, a controller node computes the workload for each worker process. If there is a significant load imbalance among the worker processes, the controller initiates the algorithm. We use two tuning parameters, C and L, to determine (1) how often to run the algorithm and (2) the number of LPs to transfer between worker processes. In order to avoid an excess of optimism (which results in an excess of rollbacks) we made use of a dynamic time window which was computed using the values of C and L. The combined algorithm improved the simulation time of our Ca^{2+} wave model by up to 30 percent.



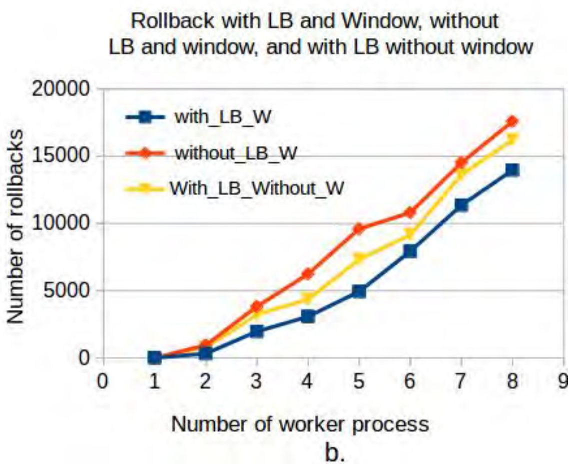


Figure 5: (a) Execution time 1) with load balancing and window, 2) with load balancing and without window, and 3) without load balancing and window. (b) Number of rollbacks) with load balancing and window, 2) with load balancing and without window, and 3) without load balancing and window.

REFERENCES

Andrews, S., N. J. Addy, R. Brent, and A. P. Arkin. 2010. “Detailed simulations of cell biology with smoldyn 2.1”. *PLOS Comput Biol* 6(3).

Bartol Jr, T., B. Land, E. Salpeter, and M. Salpeter. 1991. “Monte carlo simulation of miniature endplate current generation in the vertebrate neu- romuscular junction”. *Biophysical Journal* 59(6):1290–1307.

Blackwell, K. 2013. “Approaches and tools for modeling signaling pathways and calcium dynamics in neurons”. *Journal of neuroscience methods* (220:2): 131–140.

Carnevale, N. T., and M. L. Hines. 2006. *The NEURON book*. Cambridge University Press.

Dematt, L., and T. Mazza. 2008. “On parallel stochastic simulation of diffusive systems”. In *Computational methods in systems biology*, 191–210: Springer.

Elf, J., and M. Ehrenberg. 2004. “Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases”. *Systems biology* 1 (2): 230–236.

Falcke, M. 2003. “On the role of stochastic channel behavior in intra- cellular Ca^{2+} dynamics”. *Biophysical Journal* 84:4256.

Gibson, M. A., and J. Bruck. 2000. “Efficient exact stochastic simulation of chemical systems with many species and many channels”. *The journal of physical chemistry A* 104 (9): 1876–1889.

Gillespie, D. T. 1977. “Exact stochastic simulation of coupled chemical reactions”. *The journal of physical chemistry* 81 (25): 2340–2361.

Gillespie, D. T. 2001. “Approximate accelerated stochastic simulation of chemically reacting systems”. *The Journal of Chemical Physics* 115 (4): 1716–1733.

Jefferson, D. R. 1985. “Virtual time”. *ACM Transactions on Programming Languages and Systems* (3): 404–425.

Jeschke, M., R. Ewald, A. Park, R. Fujimoto, and A. M. Uhrmacher. 2008, March. “A parallel and distributed discrete event approach for spatial cell-biological simulations”. *SIGMETRICS Perform. Eval. Rev.* 35 (4): 22–31.

Leach, A. R. 2001. *Molecular modelling: principles and applications*. Pearson Education.

Lin, Z., C. Tropper, R. McDougal, M. Patoary, W. Lytton, and M. Hines. 2016. “Optimizations for multi-threaded parallel discrete event simulation of stochastic reaction-diffusion in neuron”. *Journal of Simulation*.

Mattern, F. 1993. "Efficient algorithms for distributed snapshots and global virtual time approximation". *Journal of parallel and distributed computing* 18 (4).

Meraji, S., and C. Tropper. 2012a. "On the scalability and dynamic load-balancing of optimistic gate level simulation". *IEEE Transaction on computer aided design of integrated circuits and systems* 29 (9): 1368–1380.

Meraji, S., and C. Tropper. 2012b. "Optimizing techniques for parallel digital logic simulation". *IEEE Transaction on parallel and distributed systems* 23 (6).

Neymotin, S. A., R. A. McDougal, M. A. Sherif, C. Fall, M. L. Hines, and W. W. Lytton. 2015. "Neuronal Ca^{2+} wave propagation varies with changes in endoplasmic reticulum parameters: a computer model". *Neural Computation* 27 (4): 898–924.

Patoary, M., C. Tropper, Z. Lin, R. McDougal, and W. W. Lytton. 2017. "Parallel stochastic discrete event simulation of calcium dynamics in neurons". *IEEE transactions on computational biology and bioinformatics*.

Patoary, M. N. I., C. Tropper, Z. Lin, R. McDougal, and W. Lytton. 2014. "Neuron time warp". 3447–3458: SCS.

Schinazi, R. B. 1997. "Predator-prey and host-parasite spatial stochastic models". *The Annals of Applied Probability* 7 (1): 1–9.

Steinman, J. 1991. "SPEEDES: synchronous parallel environment for emulation and discrete event simulation". *SCS western multiconference on advances in parallel and distributed simulation (PADS91)* 23:95–103.

Sterratt, D., B. Graham, A. Gillies, and D. Willshaw. 2011. *Principles of computational modelling in neuroscience*. Cambridge University Press.

Takahashi, K., K. Kaizu, B. Hu, and M. Tomita. 2004. "A multi-algorithm, multi-timescale method for cell simulation". *Bioinformatics* 20 (4): 538–546.

Wang, B., Y. Yao, B. Hou, Y. Zhao, and S. Peng. 2011. "Abstract next sub-volume method: a logical process-based approach for spatial stochastic simulation of chemical reactions". *Computational Biology and Chemistry* 35:5193–198.

Wang, B., Y. Yao, Y. Zhao, B. Hou, and S. Peng. 2009. "Experimental analysis of optimistic synchronization algorithms for parallel simulation of reaction-diffusion systems". In *High Performance Computational Systems Biology, 2009. HIBI'09. International Workshop on*, 91–100: IEEE.

Xu, Q., and C. Tropper. 2006. "Towards large scale optimistic vlsi simulation". *Simulation Modelling Practice and Theory* 14 (6): 695–711.

AUTHOR BIOGRAPHIES

MOHAMMAD NAZRUL ISHLAM PATOARY is a Ph.D. student of School of Computer Science at McGill University. His research area is parallel discrete event simulation. His email is mohammad.patoary@mail.mcgill.ca.

CARL TROPPER is a Professor in the Department of Computer Science at McGill University. His focus over the past several years has been on applying PDES techniques to stochastic discrete event simulations of neurons. His email address is carl@cs.mcgill.ca.

ROBERT A. McDOUGAL is an Associate Research Scientist in Neuroscience at Yale University with a PhD in Mathematics from The Ohio State University. His email address is robert.mcdougal@yale.edu.

WILLIAM LYTTON is a neurologist caring for the indigent at Kings County Hospital, and teaching and researching computational neuroscience at Downstate Medical Center, both in Brooklyn, NY. His email address is billl@neurosim.downstate.edu.